CPC945 Bridge and Memory Controller

User Manual

A15-6010-02a

Preliminary

February 1, 2008

IBM	PowerPC
ibm.com	PowerPC Architecture
IBM Logo	

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN “AS IS” BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM System and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**®

The IBM Semiconductor solutions home page can be found at **ibm.com/chips**

A15-6010-02a
February 1, 2008

Contents

List of Figures

List of Tables

About This Manual

This book describes the CPC945 Bridge and Memory Controller, the frontside bus controller companion chip that is compatible with the PowerPC® 970MP and 970FX RISC microprocessors. A general overview of the device, which is also called the CPC945, and functional descriptions of the subsystems are provided, as well as register descriptions.

Who Should Read This Manual

Software designers and system architects of PowerPC 970xx applications should use this document to effectively incorporate the IBM CPC945 Processor Bridge and Memory Controller architecture into a design. Readers should be familiar with the PowerPC Architecture and the general constraints and requirements of developing personal computers. Information about IBM CPC945 Processor Bridge and Memory Controller hardware considerations and IBM CPC945 Processor Bridge and Memory Controller signal descriptions are contained in the *CPC945 Bridge and Memory Controller Datasheet*.

Related Publications

PowerPC Microprocessor Documentation

The latest version of this manual, errata, and other IBM documents listed below can be found at: ibm.com/chips/techlib.

- *CPC945 Bridge and Memory Controller Datasheet*
- *PowerPC 970FX RISC Microprocessor Datasheet*
- *PowerPC 970MP RISC Microprocessor Datasheet*
- *PowerPC 970FX RISC Microprocessor User’s Manual*
- *PowerPC 970MP RISC Microprocessor User’s Manual*

Architecture and Specifications

- *Double Data Rate (DDR2) SDRAM Specification*, JESD79-2B
  www.jedec.org
- *HyperTransport I/O Link Specification*, revision 1.04
  www.hypertransport.org
- *I^{2}C-Bus Specification*, version 2.1
  www.semiconductors.philips.com
- May, Cathy, et. al., eds. *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition. San Francisco: Morgan-Kaufmann, 1994.
- *PCI Express® Base Specification*, version 1.0a.
  www.pcisig.com

Conventions Used in This Book

- A lowercase x precedes hexadecimal values. For example: x‘0B00’.
- Binary values in text are either spelled out (zero and one) or appear in single quotation marks. For example: ‘10101’. In tables, these quotation marks are omitted.
- Differential pairs of signals are designated by _P for the positive signal and _N for the negative signal. For example: DDR_CK_A_P and DDR_CK_A_N.
- Overbars designate active-low, nondifferential signals. For example: $\overline{\text{PI\_CSTP}}$.

1. Overview

1.1 Introduction

The IBM CPC945 Bridge and Memory Controller is a frontside bus controller that is compatible with the PowerPC 970FX and 970MP RISC microprocessors.

The CPC945 provides a 5-way interconnection among:

- Two PowerPC 970MP or 970FX processor interfaces
- A DDR2 SDRAM memory subsystem
- PCI Express bus
- A HyperTransport host bridge

The boot ROM is connected to an I/O device attached to the system via the HyperTransport bus. The CPC945 includes two master-only I^{2}C interfaces for configuring the memory subsystem. An I^{2}C slave interface connects the CPC945 to a single-chip microcontroller for power management and processor interface configuration. A multiprocessor-capable interrupt controller collects and distributes system interrupts from the PCI Express and HyperTransport blocks.

To software, the CPC945 appears as:

- A processor interface to the PCI Express root complex
- A processor interface to the HyperTransport host bridge
- A memory controller
- An interrupt controller
- A set of control registers

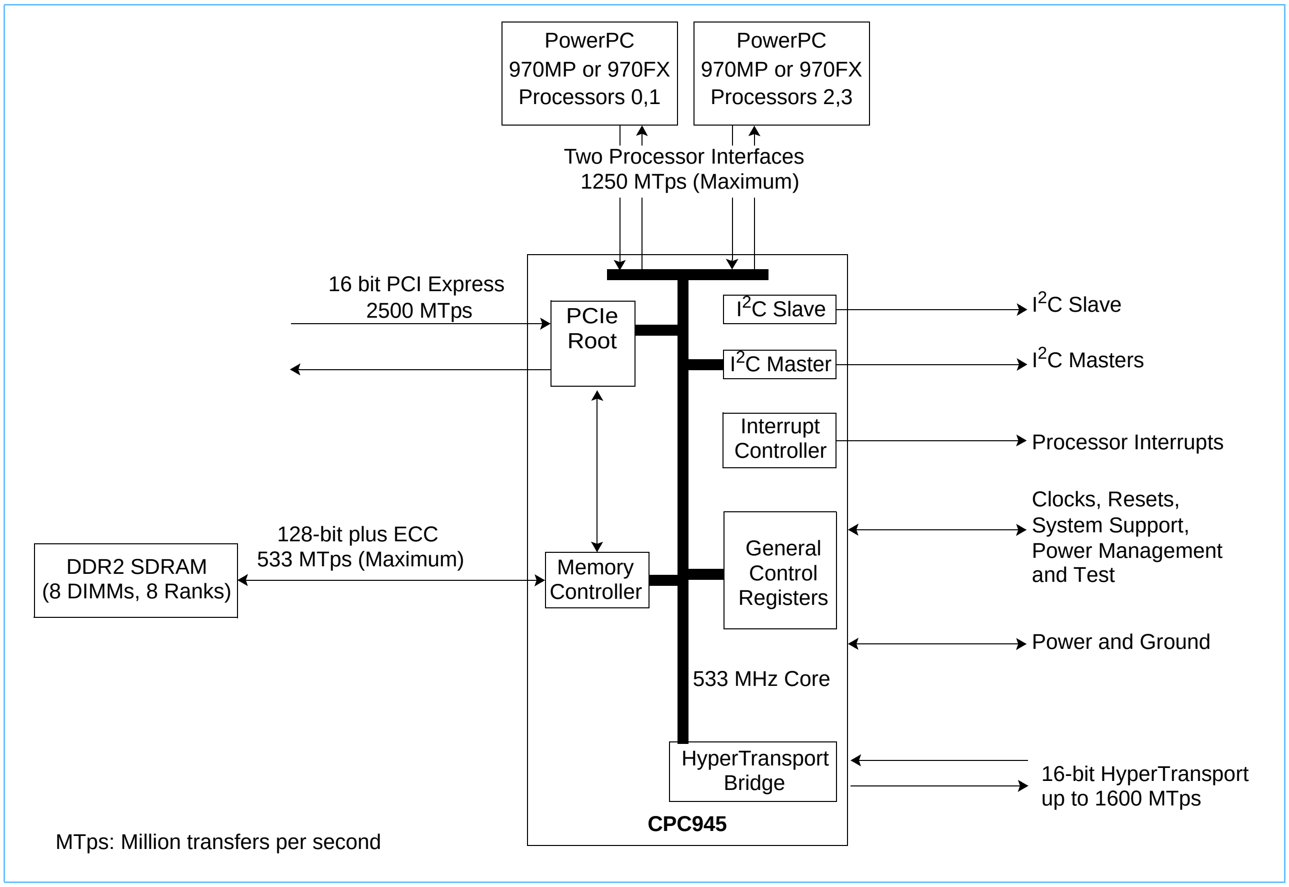
1.2 Features

- Dual PowerPC® 970MP or 970FX processor interface buses with cache coherency and snooping protocols supporting up to four processor cores.
- PCI Express x1, x4, x8, or x16 interface
- 128/144-bit, 533-Mega Transfers per second (MTps) double data rate 2(DDR2) synchronous DRAM (SDRAM) controller and interface with ECC
- 800 MHz (1600 Mtps DDR2) HyperTransport host bridge with 16-bit wide data interface and flash ROM support
- One slave and two master I^{2}C interfaces
- Interrupt controller
- Point-to-point internal architecture provides high-speed non-blocking performance
- 40-bit logical/36-bit physical memory address space
- 1182-pin flip-chip plastic ball grid array (FC-PBGA) single chip implementation
- Processor interface interconnection speed of up to 625 MHz (1250 Mbps double data rate)
- Support for eight outstanding transactions per master
- Support for read pipelining and write combining from the processor interface or PCI bus
- Support for read-around-write on the processor interface with PCI ordering
- Support for eight 64/72-bit double-sided dual inline memory modules (DIMMs) arranged in pairs
- PCI and HyperTransport transactions snooped on the processor interface

- Power management support for the CPC945 through clock control
- DMA address relocation table (DART) provides flexible I/O memory space relocation and consolidation

1.3 Block Diagrams

Figure 1-1. CPC945 Bridge and Memory Controller Block Diagram



2. Functional Description

2.1 External Interfaces

The CPC945 core logic has nine basic interfaces:

- Two processor interfaces for 2-way (single-core processor) or 4-way (dual-core processor) connection
- A DDR2 SDRAM interface for system memory connection
- A PCI Express interface
- A HyperTransport host bridge
- Two port I^{2}C master interface
- A I^{2}C slave interface
- A system interrupt controller (MPIC)

The primary function of the core logic is to convey transactions between these interfaces.

2.1.1 Processor Interconnect (PI)

The CPC945 has two Processor Interconnects each capable of directly connecting to a single or dual core PPC970xx. The PI bus consists of a 35-bit (logical)/44-bit (physical) multiplexed address/data (AD) bus segment, a 1-bit (logical/physical) transfer handshake bus segment, and a 2-bit (logical)/4-bit (physical) snoop response bus segment in each direction per directly connected processor. In total, each PI interface consists of 76 bits (logical)/100 bits (physical), wherein each bus segment uses point-to-point connections. A system consisting of a single PI simply leaves the second PI unconnected.

The PI supports split transactions for read and write operations using a packet-passing protocol between the processors and CPC945. Addresses and data are transmitted and received in packets over the AD busses. Acknowledgements are sent over the transfer handshake bus segment, and snoop responses are sent over the snoop response bus segment.

Memory/cache coherency is maintained by using a global snooping mechanism. The CPC945 PI reflects inbound command packets back to the processors for global snooping. As all connections are point-to-point, any interaction between processors and other processors, memory, PCI, etc., must be routed through the CPC945. The CPC945 contains sufficient buffering to hold eight outstanding requests and data to keep the processors from stalling.

The PI interface is asynchronous from the CPC945 core clock frequency. In initial implementations, the Processor Interconnect runs at up to 625 MHz DDR2. For more information, see *Section 3.5 Processor Interconnect Interface Microarchitecture* on page 61.

2.1.2 SDRAM Interface

CPC945 supports a memory subsystem with up to eight 64-bit wide double-sided DDR2 DIMMs organized in pairs. (See *Section 7 DDR2 Memory Controller* on page 129.) This interface is asynchronous from the PI interface at up to 266 MHz for the Memory Controller (533 MHz for the DDR2 DIMMs). With 4-bank DDR2 SDRAM parts, it is possible to construct a memory subsystem with 32 internal banks. The memory controller provides support to utilize up to 32 open banks. This further increases bandwidth and reduces the latency of a DDR2 SDRAM-based memory subsystem. The SDRAM interface uses two of the DDR2 DIMMs to produce a 128 bit wide data bus. Data can only be accessed in a set of four data beats (4 × 128 / 8 = 64 bytes) or two sets of 4 beats (128 bytes = 1 PPC970xx cache line).

2.1.3 PCI Express Interface

The CPC945 implements a PCI Express root complex interface for use with peripheral endpoints. This PCI Express interface consists on a single point-to-point link that can be configured as a x1, x4, x8, or x16 lane and operates at 2.5 GHz. For additional details on this interface, see *Section 5 PCI Express* on page 95.

2.1.4 HyperTransport Host Bridge

The CPC945 has a 16-bit wide HyperTransport host bridge. (See *Section 6 HyperTransport* on page 115.) This interface is used to attach I/O devices to the CPC945. In addition the HyperTransport bridge is used to access the boot ROM during system boot-up. The HyperTransport channel is also used to communicate interrupt information between the I/O subsystem and CPC945. HyperTransport can only perform 64 byte accesses to memory. For details on HT address mapping see *Section 6.7 HyperTransport Address Mapping* on page 119.

2.1.5 DRAM I^{2}C Master Interface

The CPC945 provides a simple Master-only I^{2}C interface based on *"The I2C-Bus Specification," Version 2.1, January 2000.*, for use in reading and writing the DRAM memory DIMM SPD configuration registers. The interface operates at the standard mode frequency of 100 kbps. It can also be programmed to run at 50 kbps and 25 kbps. This interface on the CPC945 has two pairs of clock and data pins SYS_ISCL0/SYS_ISCA0 and SYS_ISCL1/SYS_ISCA1. Only one pair can be active at a time. When the interface is idle, software can switch between these two pairs. (See *Section 8.3 I2C Master Interface* on page 275.)

2.1.6 I^{2}C Slave Configuration Register Interface

The CPC945 provides a standard two wire I^{2}C slave interface as defined in *"The I2C-Bus Specification," Version 2.1, January 2000.*, and is capable of transferring data in both standard (100kbps) and fast (400kbps) modes. The bidirectional line, PI_ISCA is used for reading and writing data. The clock line, PI_ISCL is always driven from the external master device however on read cycles the CPC945 can hold off the clock until data is ready. Both lines are “open drain” and require an external pullup. This interface is used by the microcontroller that manages the system configuration to control the setup of the Processor Interconnect physical interface, to access any CPC945 control register, and to generate read or write cycles to any address in the system by accessing a pair of system command registers and the associated data registers. (See *Section 8.2 I2C Slave Interface* on page 267.)

2.1.7 Interrupt Controller

The CPC945 includes a Multi Processor Interrupt Controller (MPIC) based on the OpenPIC standard. Interrupt information from the southbridge is communicated to the CPC945 using the HyperTransport. The MPIC in the CPC945 supports 124 I/O interrupt sources and four interprocessor interrupts, and delivers interrupts to four processors.

2.2 Implementation

2.2.1 CPC945 Clocking

CPC945 has five basic clock domains:

- The processor bus clock and internal advanced Processor Interconnect bus clock
- The DDR2 memory clock
- The PCI Express related clocks
- The HT clock
- The power manager unit clock.

2.2.1.1 Processor Interconnect Clock

Section 12.5.6 PLL1 Control Register on page 346 describes the dividers provided for different Processor Interconnect bus operation frequencies. All internal Advanced Processor Interconnect (API) interface clocks are generated from PLL1. The Processor Interconnect interface requires an externally generated APsync signal to establish “time zero” with the processor chips. This is required for system synchronization, and allows the processor and bus clocks to be synchronized to identify a unique cycle.

2.2.1.2 DDRClk

The internal interconnect memory clock, DDRClk, provides a maximum of 533 MHz distributed to the PI bus, the Memory Controller, and portions of the PCI Express interfaces (for synchronization). The clock is generated by PLL2 and the phase reference is used to remove the insertion delay of the clock tree. Insertion delay removal is required to shorten the input hold time and the output delay. The SDRAM controller generates the 266 MHz clock for the SDRAM system. The DDR_REFCLK clock input to the CPC945 is a 66 MHz clock, which is multiplied by the on-chip PLL.

2.2.1.3 PCI Express Clocking

The PCI Express bus operates at the fixed rate of 2.5GHz. The PCI-E_REFCLK clock input frequency is 100 MHz which is used for PLL3. The output of PLL3 multiplies this up to 625 MHz as an intermediate frequency for the PCI Express PHY. Within the PHY is a second PLL which steps the frequency down to 250 MHz for use within the PCI Express protocol stack.

2.2.1.4 HyperTransport Clock

The HyperTransport interface is capable of sending data at up to 800 MHz DDR2 or 1600 MTps. The clock for HyperTransport is generated from a 66.7 MHz HT_REFCLK clock input. The on-chip PLL4 multiplies this up to 800 MHz and the PLL outputs drive the HyperTransport logic. The same 66.7 MHz input must be applied to all HyperTransport devices connected to CPC945 so that the synchronous mode of operation can be used with the HyperTransport interface. Synchronous mode is a HyperTransport term which means that all devices operate at the same frequency, but the phase angle between devices is unknown.

2.2.1.5 Power Manager Clock

The power manager clock is not under the control of a PLL. It is a direct input to the CPC945 so that it is functional even if all else on the module is not operational. In addition to driving the power manager logic, the PMR_CLK is used to clock the I^{2}C slave logic and an internal bus that accesses all of the internal registers (registers accessed at addresses F8xxxxxx). It is essential that the PMR_CLK is running. Only a 300 MHz frequency is supported for the PMR_CLK.

2.2.2 ROM Controller

The boot ROM for the CPC945 is accessed through the HT interface. Typically, it is attached to a southbridge module.

PPC970xx ROM accesses are in the address space 0x‘FFxxxxxx’, a space of 16 Mbytes. The on-chip HyperTransport interface (HT) decodes addresses in the ROM space and forwards those transactions out the HyperTransport interface. If there are HyperTransport tunnel devices between CPC945 and the southbridge, the transactions must be forwarded by the tunnel devices. This should be the case at power on because at power on all tunnels forward transactions other than configuration transactions that are directed to their device number. Once the system has been initialized, the ROM address space should be assigned to the southbridge and transactions will be forwarded to it.

The PPC970xx initially fetches from address 0x‘00000100’, but on bootup, its hypervisor address can be set to 0x‘FF000000’. The CPC945 is hardcoded to make the initial fetch come from 0x‘FF000100’ in the HT space.

2.2.3 CPC945 Core Interface (API Interface)

The CPC945 core interface (Advanced Processor Interface or API) is basically responsible for interfacing between the following major blocks:

- HyperTransport Interface
- PCI Express Interface (PCIe™)
- General Control Registers (GCR)
- Processor Interconnect
- Memory Controller (DDR2)

In addition, it also contains an address re-mapping table (DART) for HT and PCI Express coherent requesters as well as snoop logic to maintain/enforce memory coherency requirements of the CPU.

The API connects the processors and the rest of the system. The interface receives and decodes all address transactions flowing across the processor bus. These transactions can originate from other primary units or the processors residing on the PI. The received address transactions are decoded and queued in the PI and then forwarded to the target unit.

The data read or written by these address transactions flows directly between the requester and the target unit. Therefore, the API has no involvement in the data transfers between other primary interface units of the CPC945. However, the API does contain the data path required to support a processor's access to all system address space.

2.2.4 PCI Express and HyperTransport Bus Interfaces

2.2.4.1 PCI Express and HyperTransport Transactions

CPC945 generates the following PCI Express transactions:

- Configuration Read (Type 0 and 1)
- Configuration Write (Type 0 and 1)
- I/O Read
- I/O Write
- Memory Read
- Memory Read Line
- Memory Write

2.2.4.2 Processor Interconnect to PCI Express Transaction Mapping

The Processor Interconnect slave interface can receive read and write transactions of all legal sizes destined for addresses on the PCI Express bus interface. Note that these transactions only map to PCI Express.

2.2.4.3 Processor Interconnect to HyperTransport Transaction Mapping

The Processor Interconnect Slave interface will receive read and write transactions of all legal sizes destined for addresses on the HT bus interface. HyperTransport follows the same rules as PCI for transaction mapping.

2.2.4.4 PCIe/HT-to-Processor Interconnect Virtual Transaction Mapping

PCI Express receives Memory Read and Memory Write transactions destined for addresses in system memory space. Likewise the HyperTransport interface receives these transactions from the HyperTransport bus. All addresses sent to Processor Interconnect on the PCI Express interface or the HyperTransport interface are re-mapped using the DART, described in *Section 3.4 DMA Address Relocation Table (DART)*, before accessing system memory.

The PCI Express or HyperTransport Bus interface is bridged to the Processor Interconnect in such a way that it appears that CPC945 contains a Processor Interconnect-to-PCI Express Bus Bridge and a Processor Interconnect-to-HyperTransport Bus Bridge. Since there are no Processor Interconnect slaves other than CPC945, the only reason to bridge I/O transactions to the Processor Interconnect is to maintain main memory coherency.

CPC945 does not pass any write data from the PCI Express or HT buses across the Processor Interconnect. The data to satisfy a PCI Express or HT read request comes from the memory subsystem or the other PCIe/HT bus interface directly. In the case of intervention cycles, the data returned by the intervention cycle is treated like a write flush. The data is flushed to its destination before the PCIe/HT read request are completed.

For more detail on PCI Express to Processor Interconnect Transactions see *Section 5 PCI Express* on page 95.

For more detail on HyperTransport to Processor Interconnect Transactions see *Section 6 HyperTransport* on page 115.

2.2.4.5 PCIe Transaction Ordering

See the *“PCI-SIG, "PCI Express Base Specification Revision 1.0a", PCI Express, April 15, 2003.”*

2.2.4.6 Data Consistency/Memory Coherence

CPC945 does not provide any coherency mechanism for memory located on the PCI Express or HyperTransport buses. Any processor that wishes to cache memory located on the PCI Express bus will need to use a software coherency mechanism to ensure proper operation.

2.2.4.7 Endianess

The Processor Interconnect is defined to be big-endian byte ordering while the PCI Express and HyperTransport buses are defined to be little-endian byte ordering. CPC945 performs the appropriate byte conversions. Byte addressing is similarly numbered in conflicting order on either side of the bridge. CPC945 supports one mode of byte swapping: big-endian mode (ByteSwap=1) as shown in *Figure 2-1*. *Table 2-1* and *Table 2-2* detail the mapping of Processor Interconnect to PCI Express Bus addresses in big-endian mode.

In the following figures and tables, “MSB” and “LSB” are denoted with respect to how the data appears in the microprocessor’s internal register, which is always in big-endian format.

Figure 2-1. PI to PCIe Bus Data Byte Swapping in Big-Endian Mode

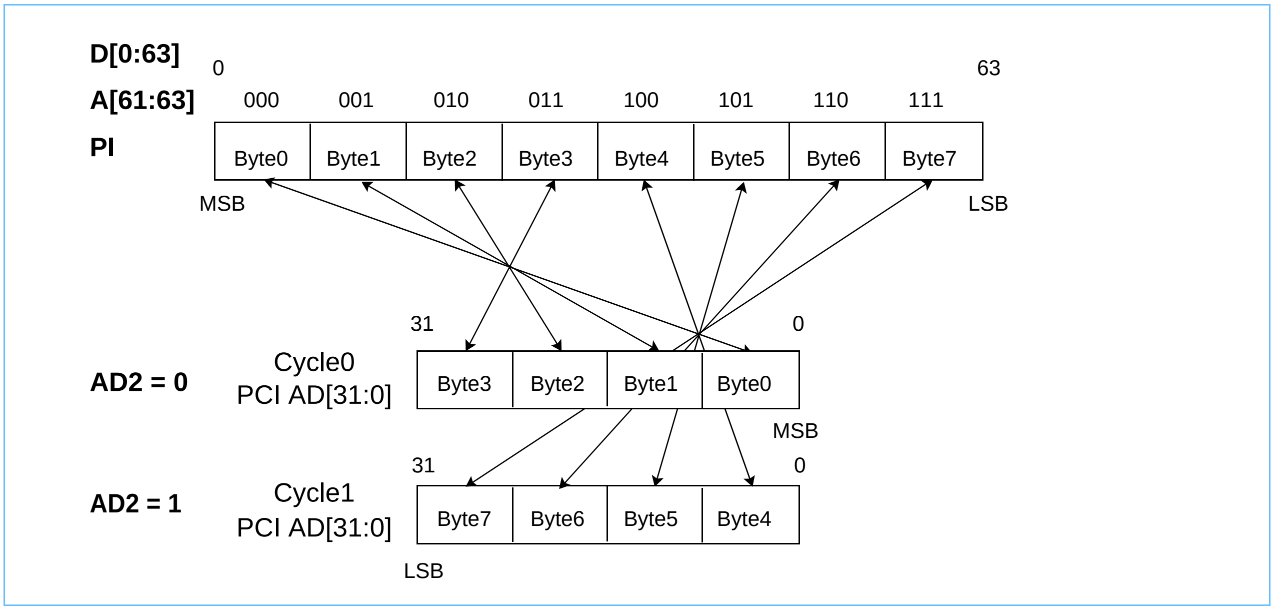


Table 2-1. PI to PCIe Bus address mapping in Big-Endian Mode for Memory Space .

Processor Interconnect (PI) Address	PCI Express Address	Comment
[32]	[31]	
[33]	[30]	
..	..	
[60]	[3]	
[61]	[2]	
[62]	[1] = 0, BEs encoded	Linear Incrementing Mode
[63]	[0] = 0, BEs encoded	Linear Incrementing Mode

Table 2-2. PI to PCIe Bus Address Mapping in Big-Endian Mode for I/O Space .

Processor Interconnect Address	PCI Express Address
[32]	[31]
[33]	[30]
..	..
[62]	[1], BEs encoded
[63]	[0], BEs encoded

2.2.5 Exceptions

The exceptions that can occur on the PI interface are caused by three fundamental errors:

- Invalid addresses
- Invalid burst transactions
- Invalid transaction types

Invalid addresses can also result in Master Abort and Target Abort terminations on HyperTransport or downstream PCI Bus interfaces. Accesses to an unmapped address will generally be reflected back to the PI interface if a processor was the source of the bad address. Some exceptions are normal events that occur in the course of system initialization, but others are gross errors that must be communicated to the system software, and in some cases, corrected.

2.2.5.1 Invalid Addresses

An invalid address is defined as an address that is not mapped, or is not responded to by a downstream slave device (Master Abort), or does not result in a normal completion status (Target Abort).

All I/O transactions will be acknowledged by the Processor Interconnect Slave interface of CPC945 with an Sresp Null. If CPC945 later determines that the address associated with that transaction is not mapped to an interface or device, then CPC945 will terminate the data tenure of that transaction by asserting DERR.

A Master Abort that occurs during a configuration read cycle initiated on the Processor Interconnect interface will result in normal completion on Processor Interconnect, however the read data returned will be forced to all 1’s and the RcvdMAbort status bit will not be set in the Legacy PCI Status Register, which is part of the PCI Express interface register set. If a configuration write cycle is terminated by a Master Abort, the data will be discarded. A Master Abort termination during any other PCI Express Bus access from Processor Interconnect will result in the RcvdMAbort bit being set in the Legacy PCI status register. If a read transaction results in a Master Abort, then the Processor Interconnect data tenure will be terminated with DERR. If a write transaction results in a Master Abort, then the write data will be discarded.

A Target Abort termination during a PCI Express Bus access from Processor Interconnect will result in the RcvdTAbort bit set in the Legacy PCI status register. If a read transaction results in a Target Abort, then the Processor Interconnect data tenure will be terminated by a DERR. If a write transaction results in a Target Abort, then the write data will be discarded.

For more detail on Legacy PCI status registers, see*Section 12.11 PCI Express Registers* on page 505.

2.2.5.2 Invalid Burst Transactions

If any type of burst access is attempted to CPC945’s control registers’ space, the transaction data tenure will be terminated by DERR.

2.2.5.3 Invalid Transaction Types

Only two transaction types are considered invalid: the special graphics operations **ecowx** and **eciwx**. The generation of either of these transaction types in a CPC945-based system results in a software error. The transaction data tenure will be terminated by DERR.

2.2.6 Interrupts

The CPC945 MPIC (Multiprocessor Interrupt Controller) supports 8 internal I/O interrupts, 116 external I/O interrupts, and 4 IPI (Interprocessor Interrupts). It forwards these interrupts to 4 external IRQ signals, which connect to up to 4 processors. The 116 external I/O interrupts are a mixture of interrupts which have been forwarded to the CPC945 on the PCI Express and HyperTransport buses.

PCI Express supports two types of interrupts: legacy PCI (INT A, B, C, D), and Message Signaled Interrupts (MSI). The legacy interrupts are presented a single level sensitive interrupt, one of the 8 internal interrupts, to the MPIC controller. The legacy interrupts are automatically assigned to INT 3. The MSI interrupts are presented to MPIC as 1 of the 116 external edge triggered interrupts. The assignment of MSI interrupt level is always done by software.

HyperTransport supports one type of interrupt: the Interrupt Packet. The HT Interrupt Packets are presented to MPIC as 1 of the 116 external edge triggered interrupts. Assignment of HT Interrupt Packet level is always done by software.

Software must coordinate interrupt source assignment among all interrupting devices on the PCI Express that use MSI interrupts and downstream devices on the HyperTransport interface.

Interrupts from devices on CPC945 (for example, error interrupts related to PCI Express link errors or HyperTransport link error), generate different internal MPIC interrupts and do not use messages carried on HyperTransport. (See *Section 9 MPIC* on page 279.)

3. CPC945 Core Interface (API Interface)

The CPC945 core interface (Advanced Processor Interface or API), as illustrated in *Figure 3-1* on page 40, basically serves as a crossbar switch for CPU and I/O requests to and from memory.

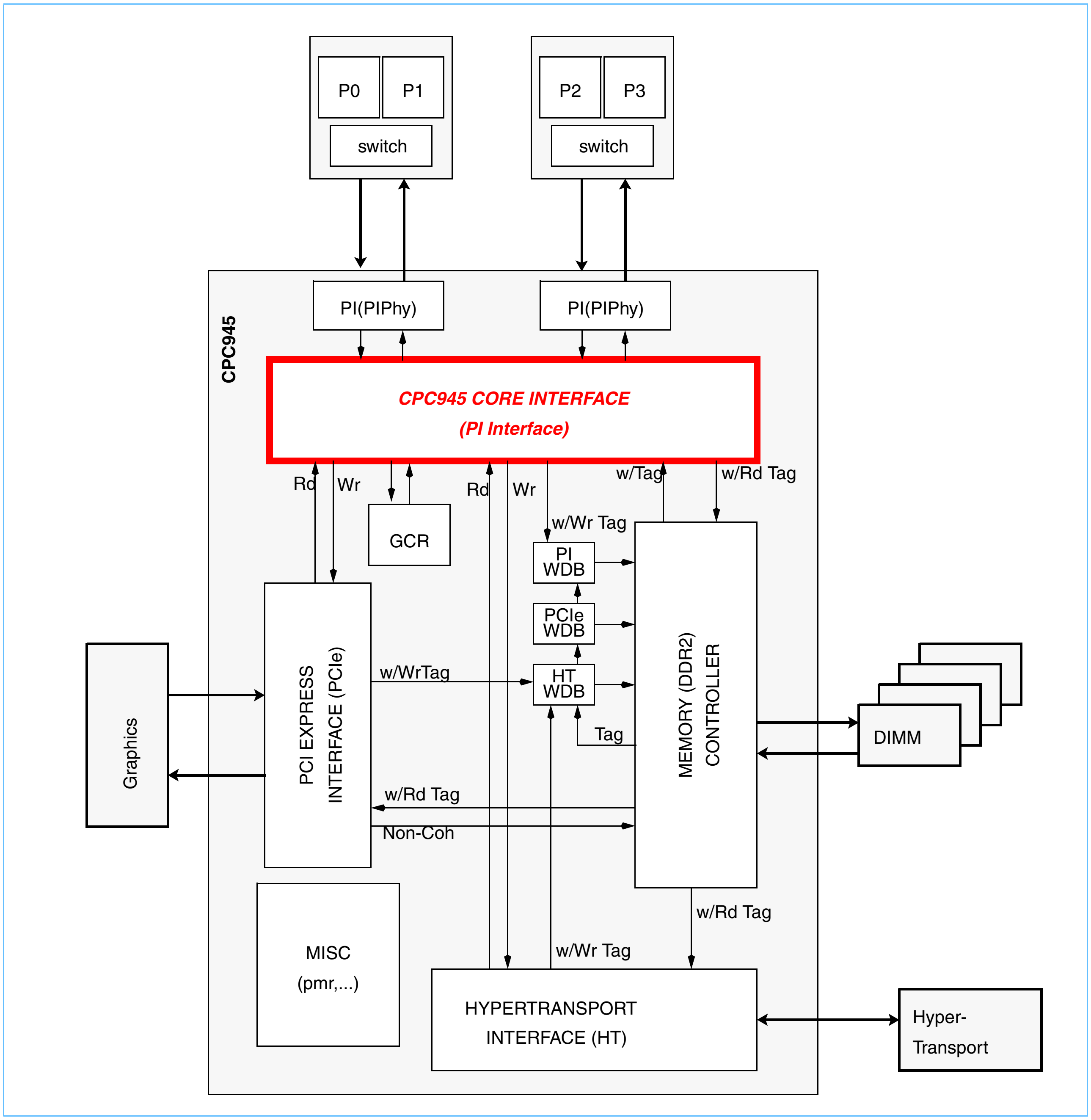
The API interface consists of various queues and data buffers that hold both requests and data respectively from its originators and is responsible for dispatching these to their intended destinations. In addition, it also contains a DMA address relocation table (DART) for HT and PCIe coherent requesters as well as snoop logic to maintain/enforce memory coherency requirements of the CPU.

3.1 CPC945 Core Interface Overview

The CPC945 core interface (API) is responsible for interfacing between the following major blocks:

- HyperTransport Interface (HT)
- PCI Express Interface (PCIe)
- General Control Registers (GCR)
- Processor Interconnect (PI)
- Memory Controller (DDR2)

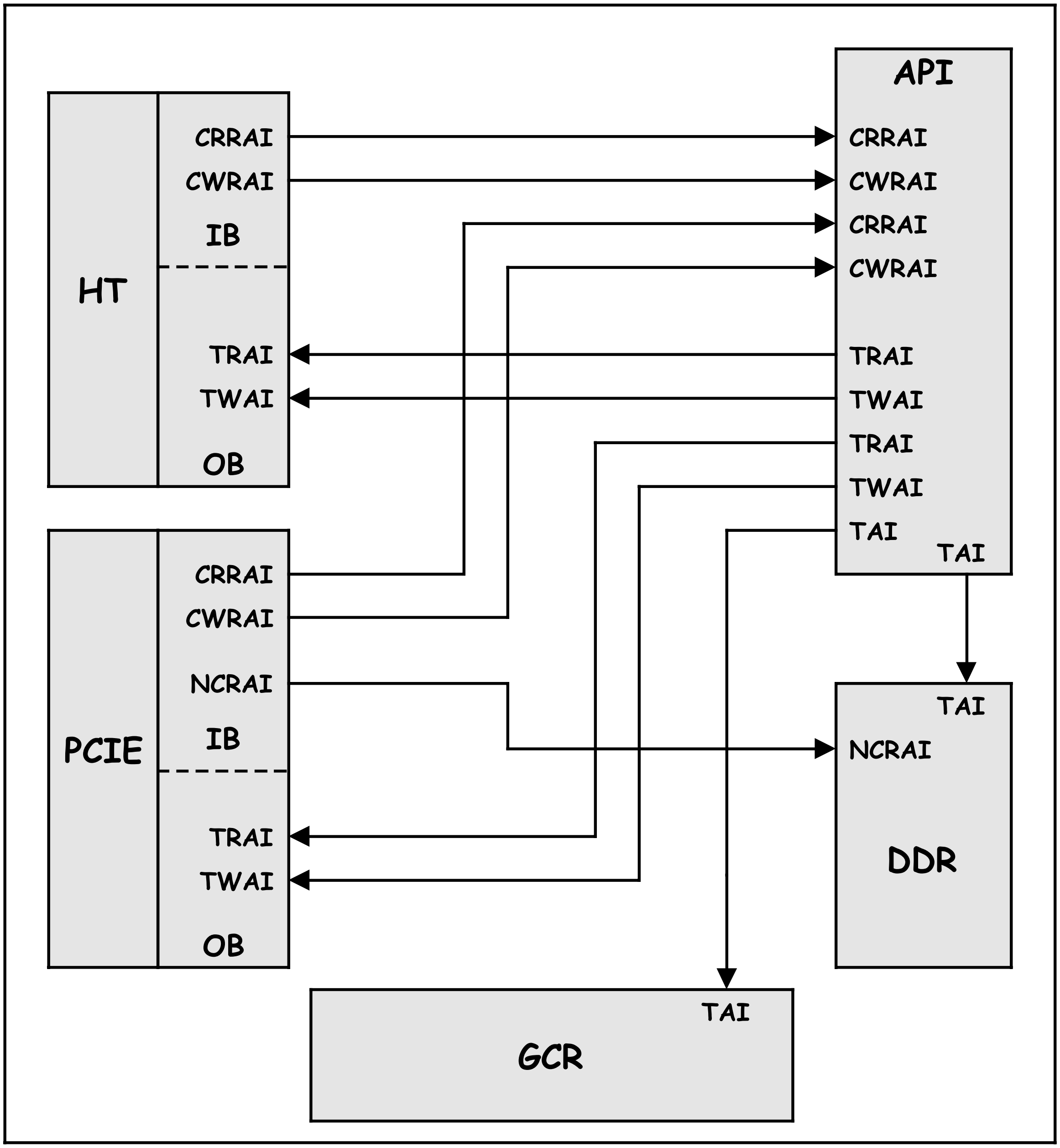
Figure 3-1. CPC945 Core Interface Block Diagram



3.1.1 General Description of the Request Interfaces

Figure 3-2 shows the request interfaces between the various internal blocks. The different types of request interfaces are described in the following subsections.

Figure 3-2. CPC945 Core Request Interfaces



3.1.1.1 CRRAI / CWRAI – Coherent Read/Write Request Address Interface

This interface is used for coherent inbound read/write requests from HT and PCIe. The requests are sent to the API logic block to be snooped by the processor's caches before they are sent to the various targets.

The interface contains address, cycles, tag, and control information. There are separate interfaces for reads (CRRAI) and writes (CWRAI). The tag provided is used to specify the source of the data for writes, and the destination of the data for reads.

All requests not going to or from memory are done in order. Requests to memory can be reordered with respect to the data. On reads to memory the tag is returned with the data on the Memory Read Data Interface. On writes to memory, the tag is used to read the write data from the Write Data Buffer. (See *Section 3.1.2 General Description of the Data Interfaces* on page 42.)

3.1.1.2 NCRAI – Non-Coherent Request Address Interface

This interface is used for non-coherent inbound read and write requests from PCIe. The requests are sent directly to the Memory Controller without being snooped by the processor's caches.

The interface contains address, cycles, tag, and control information. There is one interface for both reads and writes.

3.1.1.3 TRAI / TWAI – Target Read/Write Address Interface

This interface is used for sending outbound read/write requests to HT and PCIe after they have been snooped. These requests could have originated from the processors or as inbound requests from HT or PCIe.

The interface contains address, cycles, and control information. There are separate interfaces for reads (TRAI) and writes (TWAI).

3.1.1.4 TAI – Target Address Interface

The TAI is the same as TRAI and TWAI except there is a common interface for both reads and writes.

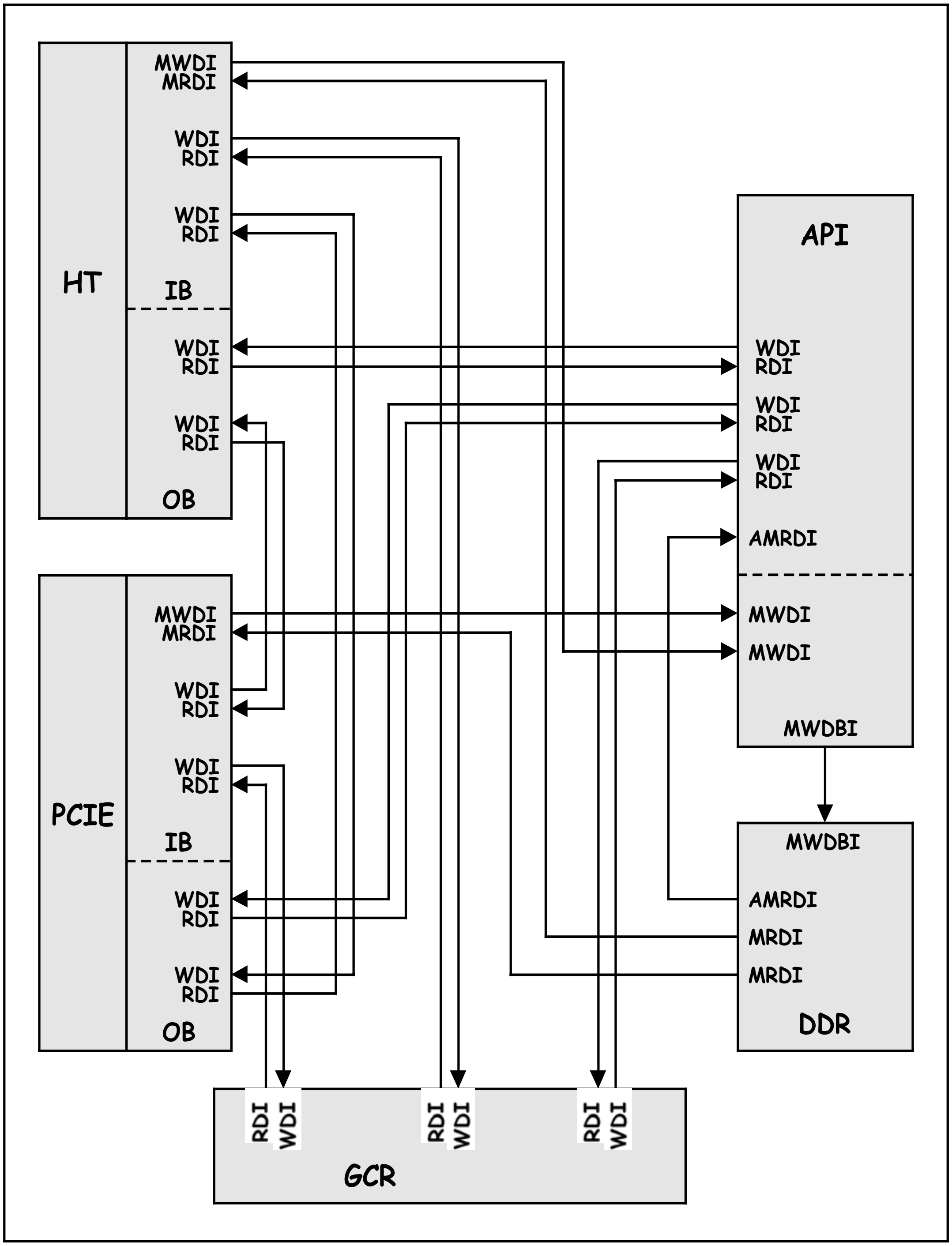
This interface is used for sending read and write requests to DDR2 and GCR after they have been snooped. These requests could have originated from the processors or as inbound requests from HT or PCIe.

The interface contains address, cycles, and control information.

3.1.2 General Description of the Data Interfaces

Figure 3-3 on page 43 shows the data interfaces between the various internal blocks. These data interfaces are described in the following subsections.

Figure 3-3. CPC945 Core Data Interfaces



3.1.2.1 WDI – Write Data Interface

The Write Data Interface is used for transferring write data between the various blocks except for memory. It is used for HT inbound write operations to PCIe and GCR, PCIe inbound write operations to HT and GCR, and PI write operations to PCIe, HT, and GCR. Except for GCR, the interface consists of 128 bits of data along with 16 bits of byte enables and control. The interface to GCR is only 64 bits of data with 8 bits of byte enables. The write data is expected to be available by the source when the request is issued. The target of the request pulls the data when it is able to execute the write operation.

3.1.2.2 RDI – Read Data Interface

This interface is used for transferring read data between the various blocks except for memory. It is used for HT inbound read operations to PCIe and GCR, PCIe inbound read operations to HT and GCR, and PI read operations to PCIe, HT, and GCR. Except for GCR, the interface consists of 128 bits of data along with 16 bits of byte enables. The interface to GCR is only 64 bits of data with 8 bits of byte enables. The source is expected to provide a buffer for the read data when the read operation is issued. The target of the request pushes the read data to this buffer when it has the data available.

3.1.2.3 MWDI – Memory Write Data Interface

This interface is used for transferring write data to memory (DDR2) from HT and PCIe inbound write operations. The interface consists of 128 bits of write data, 16 bits of byte enables, a Write Data Buffer (WDB) address where the write data is to be written, and control. The write data is written to the WDB before the request is issued to insure that the data is available in the WDB when the write request is executed. A Tag is sent with the write request that indicates where the write data from this operation is located in the WDB. The WDBs reside in the API block. The write data is sent across to the DDR2 block on the AMWD interface.

3.1.2.4 MRDI – Memory Read Data Interface

This interface is used for transferring read data from memory (DDR2) for HT and PCIe inbound read operations. The interface consists of 128 bits of read data and a tag that was carried along with the request because the memory controller might have issued the read out-of-order.

3.1.2.5 AMRDI – API from Memory Read Data Interface

Read data from memory is sent on the AMRD interface. This interface is similar to the MRDI except that part of the ECC correction is done in the API block to save latency.

3.1.2.6 MWDBI – Memory Write Data Buffer Interface

Write data from the various requesters is sent first on the requesters MWDI (see above) to the API block where it is buffered in Write Data Buffers. When the write to memory is actually performed, the memory controller requests the data from the WDBs and the data is sent to the memory controller on the MWDBI Interface.

3.2 Illustration of Requests and Dataflow within PI

This section illustrates the request and dataflow for HT and PCIe inbound requests and PI target requests.

3.2.1 HT Inbound Request

Figure 3-4 on page 46 shows the request data interfaces for HT inbound read and write requests to DDR2, PCIe, and GCR.

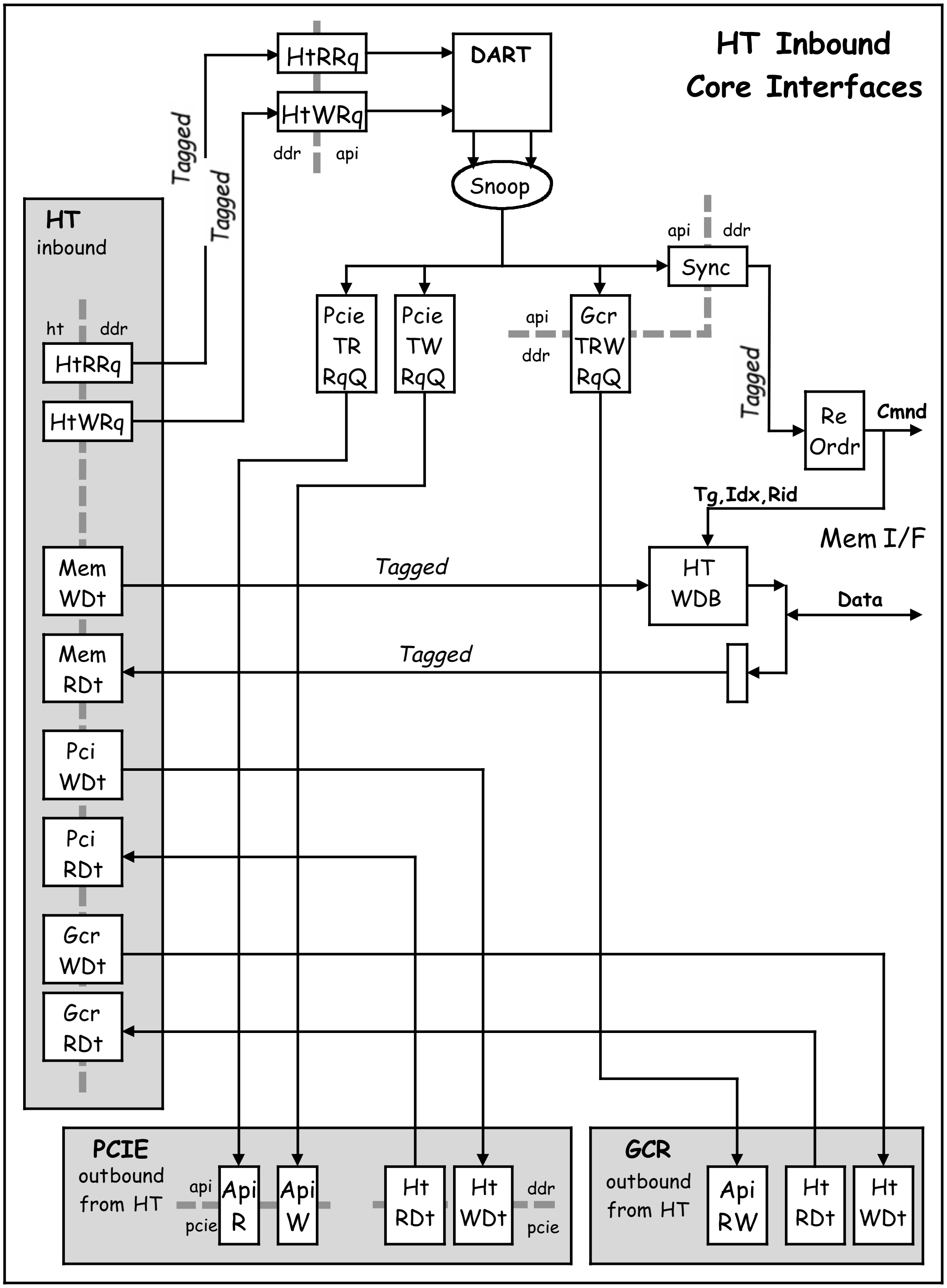
The HT block sends inbound requests to the API block to be translated by the DART and snooped by the processors. The requests are then mapped and sent to a target (DDR2, PCIe, or GCR).

The Target request contains a Requester ID to tell the Target where to return data on reads and where to get the data on writes.

Requests to PCIe and GCR are in order. Read data is returned and write data is received in the same order as the requests.

Requests to memory can be reordered in the memory controller. To keep track of the reorder, a Tag is provided on memory read requests that is returned with the read data. On memory write requests a Tag is provided that specifies the WDB location for write data.

Figure 3-4. HT Inbound Request Core Interfaces



3.2.2 PCIe Inbound Request

Figure 3-5 on page 48 shows the request data interfaces for PCIe inbound read and write requests to DDR2, PCIe, and GCR.

The PCIe block sends coherent inbound requests to the API block to be translated by the DART and snooped by the processors. The requests are then mapped and sent to a target (DDR2, HT, or GCR).

The Target request contains a Requester ID to tell the Target where to return data on reads and where to get the data on writes.

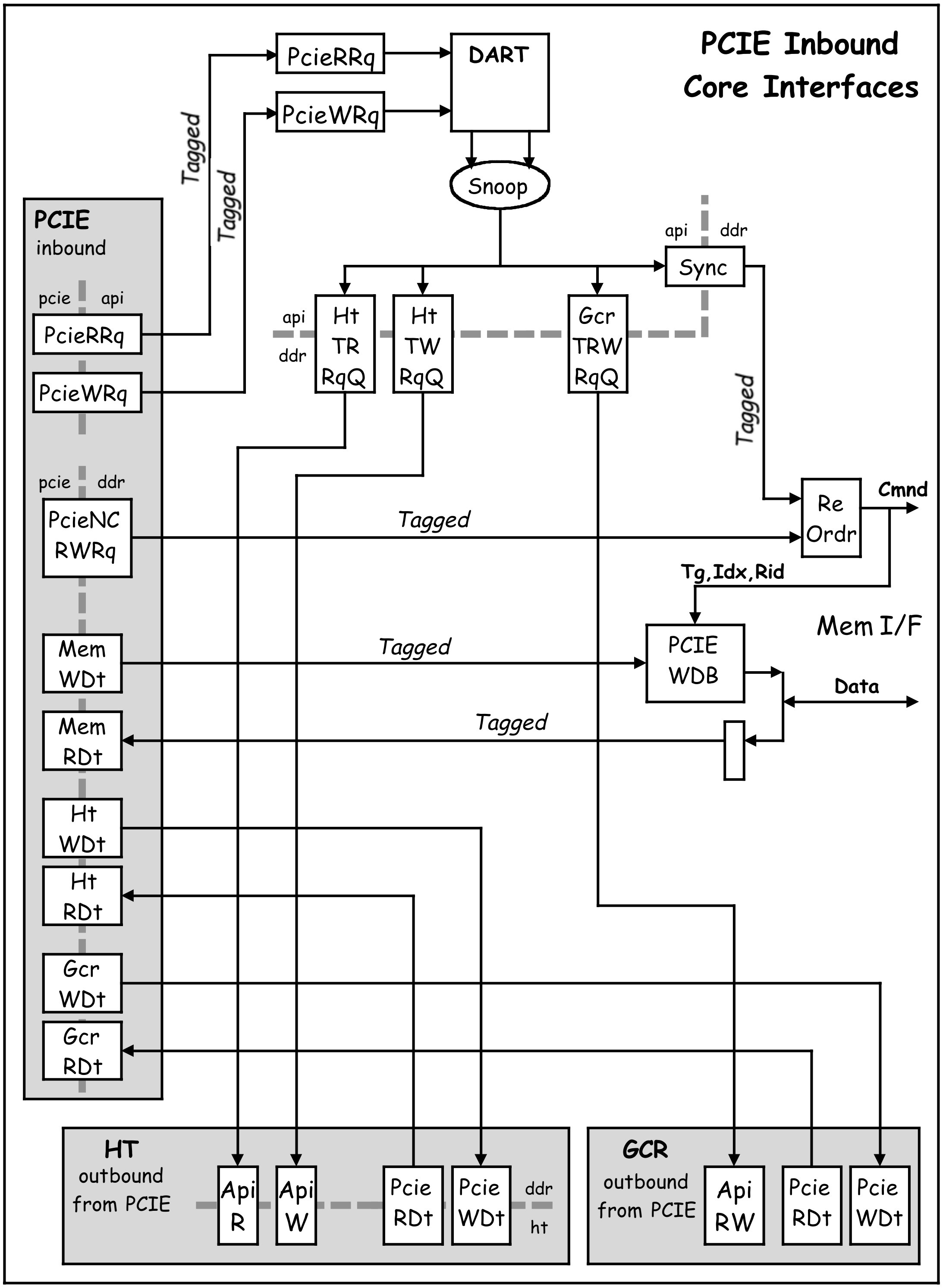
The PCIe block can also send non-coherent inbound requests directly to DDR2.

Requests to HT and GCR are in order. Read data is returned and write data is received in the same order as the requests.

Requests to memory can be reordered in the memory controller. To keep track of the reorder, a Tag is provided on memory read requests that is returned with the read data. On memory write requests a Tag is provided that specifies the WDB location for write data.

The same data buses are used to memory for both coherent and noncoherent requests.

Figure 3-5. PCIe Inbound Request Core Interfaces



3.2.3 PI Target Requests

Figure 3-6 on page 50 shows the request data interfaces for Processor Interface (PI) Requests to DDR2, PCIe, HT, and GCR.

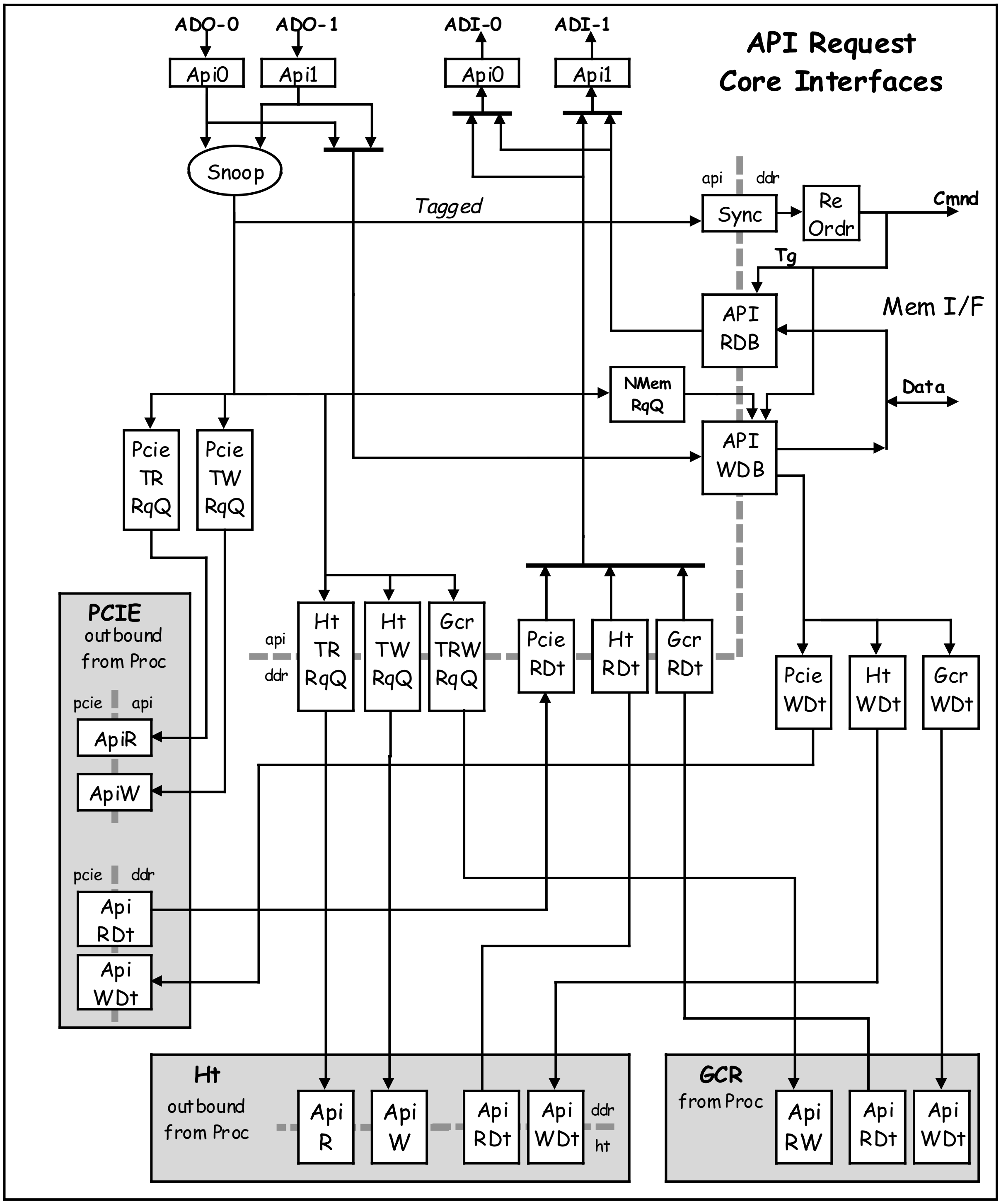
The API block receives requests from the processors through the Address DataOut (ADO) side of the APIPhy Bus. These requests are snooped by the processors, mapped and sent to a target (DDR2, PCI, HT, or GCR).

The Target request contains a Requester ID to tell the Target where to return data on reads and where to get the data on writes.

Requests to PCIe, HT, and GCR are in order. Read data is returned and write data is received in the same order as the requests.

Requests to memory can be reordered in the memory controller. To keep track of the reorder, a Tag is provided on memory read requests that is returned with the read data. On memory write requests a Tag is provided that specifies the WDB location for write data.

Figure 3-6. PI Target Request Interfaces



3.3 PI Bus Timing Parameters

The example in *Figure 3-7* on page 52 (for illustrative purposes only) shows the relationship between the various bus configuration timing parameters using a hypothetical CPU access. The numbers in brackets are provided for illustrative purposes only, and represent latency/propagation delay in terms of the number of bus beats, which is half of the api_clock period.

The CPU issues a command on the bus and this makes its way to the “ADO” pins of the CPU. Accounting for a board delay of 1 data beat in this example, the CPC945 sees this “ADO” signal internally after 5 beats, accounting for the 4 data beats latency through the Processor Interface (PI) for this particular target time setting. The CPC945 then takes 4 data beats to generate the required transfer handshake and it takes another 4 data beats to leave the CPC945 for the CPU. If the PI inbound delay for the CPU is 3 beats, then the STATLAT seen from the CPU’s perspective would be 18 in total.

Concurrently, when the CPC945 receives the “ADO” signals, it queues up the request in its various slots and queues and waits for the correct moment to send out the reflection packet for snooping by the processors. This can take a varying amount of time and depends on the queue conditions of the various FIFOs inside the API block.

When the CPC945 is ready to reflect the command packets, it begins its countdown of its own STATLAT counter, which differs from that of the CPU. Referring to the various delays shown in *Figure 3-7*, the CPC945 STATLAT is the round trip delay from the time the CPC945 first generates its reflection packet (internally) to the time it receives its transfer handshake (internally) from the CPU. As shown in *Figure 3-7*, this includes the latency through the PI for both the CPC945 and CPU, the I/O delays as well as the wire delays on the board. In this example the total delay is 25 bus beats.

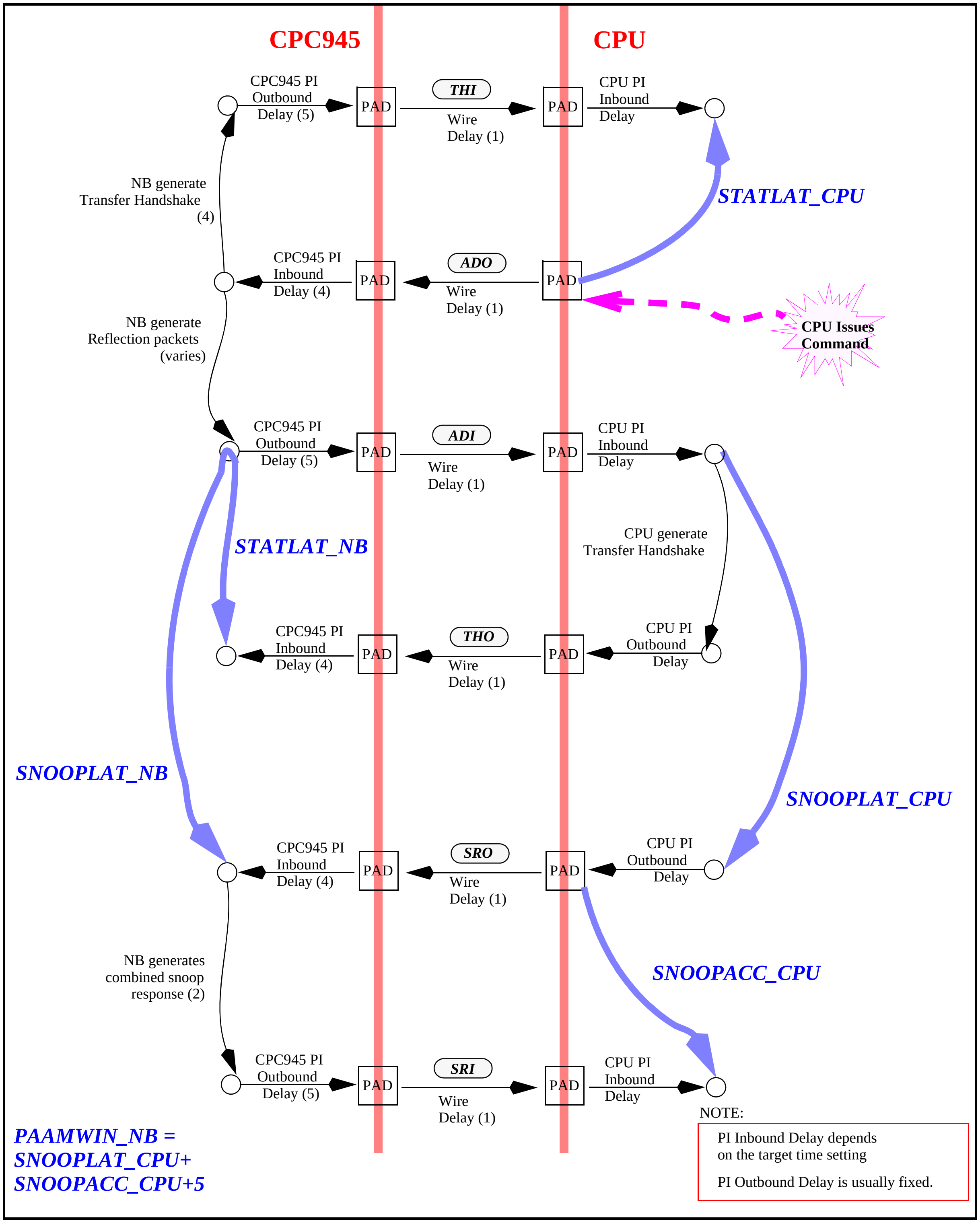
In addition to the STATLAT countdown that is initiated, the CPC945 also initiates the countdown of its SNOOPLAT counter. Referring to the various delays shown in *Figure 3-7*, the CPC945 SNOOPLAT is the round trip delay from the time the CPC945 first generates its reflection packet (internally) to the time it receives its snoop response (internally) from the CPU. As shown in *Figure 3-7*, this includes the latency through the PI for both the CPC945 and CPU, the I/O delays as well as the wire delays on the board. In this example the total delay is 23 bus beats.

The CPC945 takes the snoop responses from all the CPU attached and generates a combined/accumulated snoop response back to the CPU. From the CPU’s perspective, the SNOOPACC is the round trip delay from the time the individual snoop response leaves its “SRO” pins to the time it receives the “SRI” signal internally as shown in *Figure 3-7*. In this example the total delay is 16 bus beats.

The SNOOPLAT parameter of the CPU depends on the type of processor family that is attached and typically is 5 for the 970FX and 6 for the 970MP.

The PAAMWIN parameter of the CPC945 is calculated based on the formula shown in *Figure 3-7*.

Figure 3-7. Processor and CPC945 Bus Timing Parameters


CPC945
CPU
CPC945 PI Outbound Delay (5)
PAD
THI
Wire Delay (1)
PAD
CPU PI Inbound Delay
NB generate Transfer Handshake (4)
STATLAT_CPU
CPC945 PI Inbound Delay (4)
PAD
ADO
Wire Delay (1)
PAD
CPU Issues Command
NB generate Reflection packets (varies)
CPC945 PI Outbound Delay (5)
PAD
ADI
Wire Delay (1)
PAD
CPU PI Inbound Delay
STATLAT_NB
CPU generate Transfer Handshake
CPC945 PI Inbound Delay (4)
PAD
THO
Wire Delay (1)
PAD
CPU PI Outbound Delay
SNOOPLAT_NB
SNOOPLAT_CPU
CPC945 PI Inbound Delay (4)
PAD
SRO
Wire Delay (1)
PAD
CPU PI Outbound Delay
SNOOPACC_CPU
NB generates combined snoop response (2)
CPC945 PI Outbound Delay (5)
PAD
SRI
Wire Delay (1)
PAD
CPU PI Inbound Delay
PAAMWIN_NB = SNOOPLAT_CPU+ SNOOPACC_CPU+5
NOTE:
PI Inbound Delay depends on the target time setting
PI Outbound Delay is usually fixed.

3.4 DMA Address Relocation Table (DART)

Requests from PCI and HT undergo a mapping from their 40-bit logical address to a 36-bit physical address needed by memory.

Not all 40-bit addresses are mapped. Address bits 24:32 define ranges where mapping does not occur. This is described in the following table.

Table 3-1. DART Mapping Range.

High Order Bits of 40-bit Address [24:32] Binary	Address Range [24-63] Hexadecimal	DART Enabled	Mapping
0000 0000 0	0x00 0000 0000 - 0x00 7FFF FFFF	----	Mapping occurs if DART enabled.
0000 0000 1	0x00 8000 0000 - 0x00 FFFF FFFF	----	Mapping disallowed.
0000 0001 - 0000 001- - 0000 01-- - 0000 1--- -	0x01 0000 0000 - 0x0F FFFF FFFF	----	Mapping occurs if DART enabled.
0001 ---- - 001- ---- - 01-- ---- -	0x10 0000 0000 - 0x7F FFFF FFFF	DART enabled	Mapping occurs.
		DART disabled	Undefined Space. Addressing Exception reported.
1000 ---- -	0x80 0000 0000 - 0x8F FFFF FFFF	----	Mapping disallowed. Direct access by dropping bits 24:27.
1001 ---- - 101- ---- - 11-- ---- -	0x90 0000 0000 - 0xFF FFFF FFFF	----	Undefined Space. Addressing Exception reported.

The mapping consists of accessing a table in memory to translate the Logical 4 KB Page Number (LPN) (27 bits, address [25:51]) to a Physical 4 KB Page Number (PPN) (24-bits, address [28:51]). This table is referred to as the DMA Address Relocation Table (DART).

When bit [24] is a one, mapping is disallowed. An access beyond the range of physical memory causes an address exception. See *Section 12.9.22 DART Exception Status Register (DARTEXCP)* on page 443.

3.4.1 DART Format in Main Memory

The DART is a table in main memory (DRAM) that contains one 32-bit entry for every page that it maps. *Table 3-2* shows the format for a DART entry.

Table 3-2. Format of a DART Entry in Main Memory .

Bits	Field	Description
0	V	Valid Entry (1: Valid)
1	R	Read Protection Bit (1: Page is protected)
2	W	Write Protection Bit (1: Page is protected)
3:7		Undefined.
8:31	PPN	24-bit Physical Page Number representing address [28:51]

The Valid bit (V) indicates whether this entry has a correctly enabled translation. If this bit is zero and used for a translation, a DART Entry Exception is reported. See *Section 12.9.22 DART Exception Status Register (DARTEXCP)* on page 443.

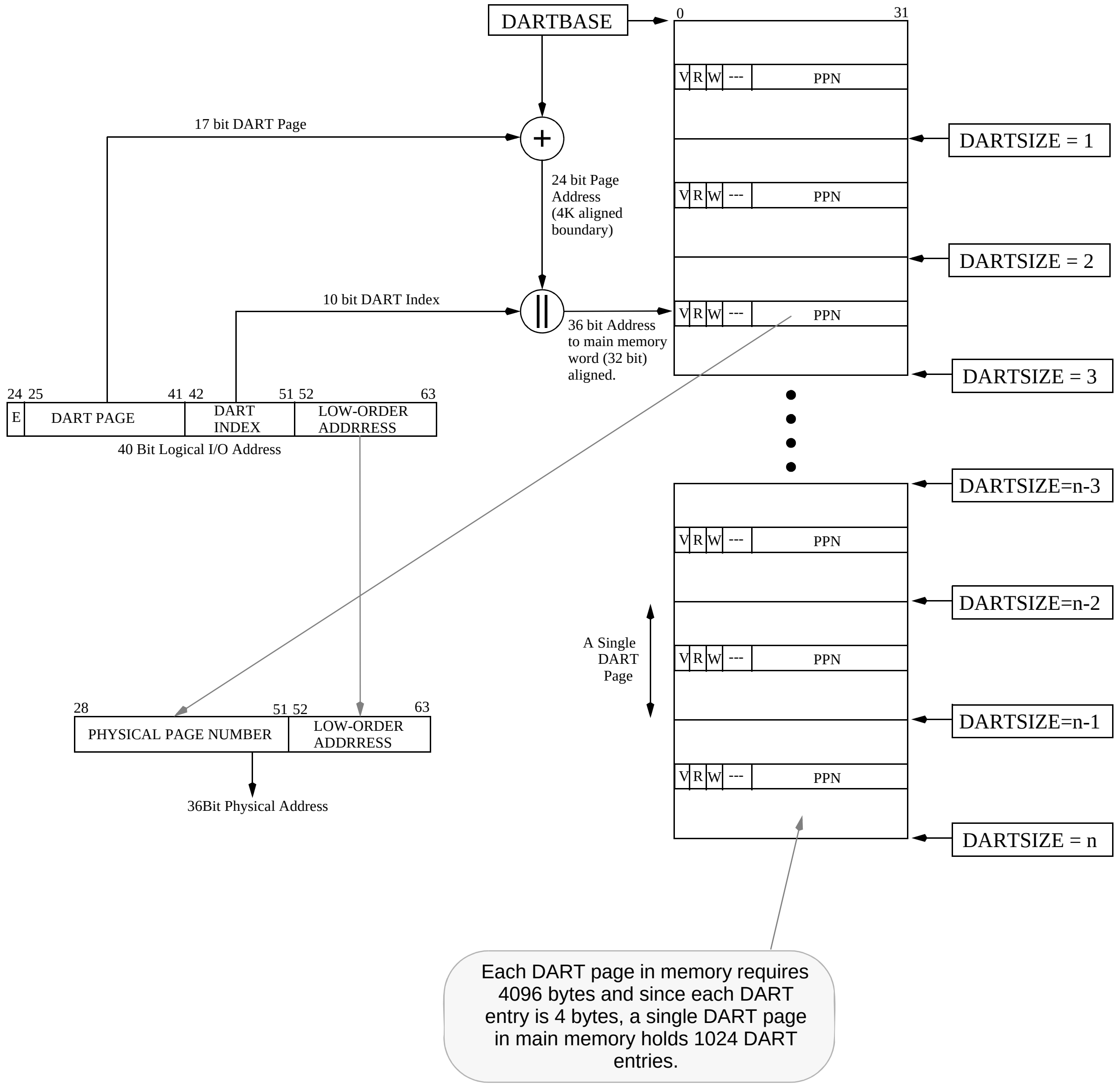
The Read Protection bit (R) indicates whether this entry is protected against read operations. If this bit is zero then translations due to read operations are allowed to access this table entry. It this bit is one (and the Valid is one) and the entry is being used to translate an address for a read operation then a DART Read Protection Exception is reported. See *Section 12.9.22* on page 443.

The Write Protection bit (R) indicates whether this entry is protected against write operations. If this bit is zero then translations due to write operations are allowed to access this table entry. If this bit is one (and the Valid bit is one) and the entry is being used to translate an address for a write operation then a DART Write Protection Exception is reported. See *Section 12.9.22* on page 443.

3.4.2 DART Translation Process

The DART translation consists of taking the 40-bit logical address (bits 24:63) from the I/O and examining the 27-bit (bits 25:51) Logical Page Number (LPN) as illustrated in *Figure 3-8*.

Figure 3-8. The DART Translation Process



A DART page in main memory (DRAM) is defined to be 4K in size and holds 1024 DART entries. See *Table 3-2 Format of a DART Entry in Main Memory* on page 54 for the bit definitions in memory.

This 27-bit LPN is then divided into 2 fields, the DART_Page and the DART_Index as follows:

DART_Page = LPN[25:41] (17 bits)

DART_Index = LPN[42:51] (10 bits)

For DART size checking, the DART_Page is compared to the DARTSIZE value to see if the mapping will fit in the DART.

If it does not fit, a DART Out-of-bounds Exception is set. A DARTSIZE of zero indicates that the entire address space is mapped and no OutOfBounds Exception will be issued.

OutOfBoundsException = (Size != 0) & (Size <= DART_Page)

If it fits (Size > DART_Page), the DART_Page is added to the DART_BASE to determine what page is accessed. The word index into the page comes from the remaining bits, DART_Index. The DART entry address in the main memory is computed as follows:

DART Entry Address = (DART_BASE + DART_Page) || DART_Index || b00

The Valid bit from the fetched DART entry is checked and if found to be not valid, a DART Entry Exception is set. See *Section 12.9.22 DART Exception Status Register (DARTEXCP)* on page 443.

If Valid, the Read and Write Protection bits are checked. If there are no exceptions, the 24-bit Physical Page Number (PPN) that was fetched from memory is concatenated with the 12 low-order address bits 52:63 to form the 36-bit physical address. This is illustrated in *Figure 3-8 The DART Translation Process* on page 55.

Recently used DART table entries are cached in the DART TLB (translation look-aside buffer).

3.4.3 DART TLB Format

The DART Translation Look-aside buffer (TLB) is a cache of recently used DART Table Entries and is implemented as internal SRAMs.

The DART TLB consists of a 256-entry DART TLB Tag RAM (DARTTAG) in a 4-way set-associative arrangement (64 sets of 4 ways per set) and a corresponding 1024-entry DART TLB Data RAM (DARTDATA).

Each TLB entry is capable of mapping 4 logical addresses using 4 entries of the TLB Data Ram. This gives the TLB the capability of mapping a total of 1024 pages.

To access the TLB, the 27-bit (bits 25:51) LPN is divided into three parts:

- TLB Logical Page Tag (TLBTAG),
- TLB Index (TLBIDX), and
- TLB Quad Select (TLBSEL).

Since there are 4 DART entries for each TLB entry, the 2 low order bits LPN[40:41] are used for the TLBSEL. The next six bits are used to index the 64 sets of the TLB. The remaining 19 bits (bits 25:43) are used for the TLBTAG. This is illustrated in *Figure 3-9*.

Figure 3-9. The TLB Translation Process

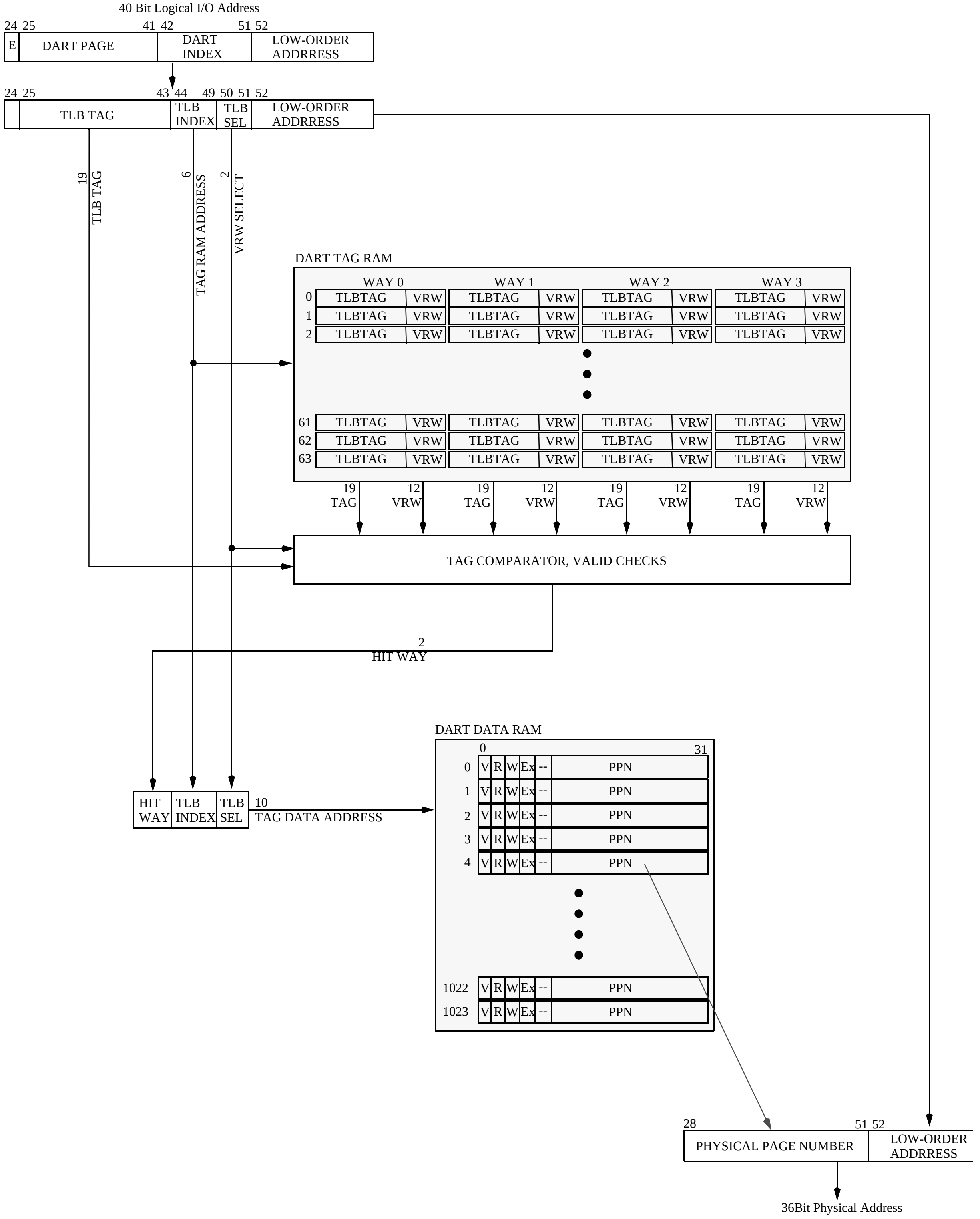


Table 3-3. LPN Address to access the DART TLB.

Bits	Field	Description
0:24	-	Unused
25:43	TLBTAG	Used as tag entries, TLB logical page tag
44:49	TLBIDX	Used as a lookup into the cache, TLB index
50:51	TLBSEL	Used as the way select. TLB quad select
52:63	unmapped	These are not considered during lookup

Each entry of the DARTTAG consists of the TLBTAG and 4 sets of control bits corresponding to the 4 Page Numbers of the DART Entry.

Each set of control bits consist of a Valid bit (V), and Read (R) and Write (W) Protection bits.

Each entry of the TLBData RAM consists of one Physical Page Number (PPN).

The DART TLB is accessible through memory-mapped addresses.

Table 3-4. Format of a DART Tag Entry in TLB (cache).

Bits	Field	Description
0	–	Unused. Read and write access. (Initial value undefined)
1:19	TLBTAG	19–bit TLB logical page tag (Initial value undefined)
20:22	V0, R0, W0	Valid 0 (1: Valid), Read 0, Write 0 (1: Protected) (Initial value: 0b000)
23:25	V1, R1, W1	Valid 1 (1: Valid), Read 1, Write 1 (1: Protected) (Initial value: 0b000)
26:28	V2, R2, W2	Valid 2 (1: Valid), Read 2, Write 2 (1: Protected) (Initial value: 0b000)
29:31	V3, R3, W3	Valid 3 (1: Valid), Read 3, Write 3 (1: Protected) (Initial value: 0b000)

Table 3-5. Format of DART Data Entry in TLB (cache).

Bits	Field	Description
0	V	Copy of valid bit for this entry. (used by hardware) (Initial value undefined)
1	R	Copy of read bit for this entry. (used by hardware) (Initial value undefined)
2	W	Copy of write bit for this entry. (used by hardware) (Initial value undefined)
3	Ex	Exception flag. (used by hardware) (Initial value undefined)
4:7	–	Unused
8:31	PPN	24-bit physical page number, Ad[28:51] (Initial value undefined)

3.4.4 DART TLB Translation

The translation process using the TLB consists of using the TLBIDX to access the 4 ways of DARTTAG entries.

The DARTTAGs from the 4 ways are compared to the TLBTAG field. On entries that match, the Valid bit is selected based upon the TLBSEL. If the entry is valid, the read and write protection bits selected by TLBSEL are checked.

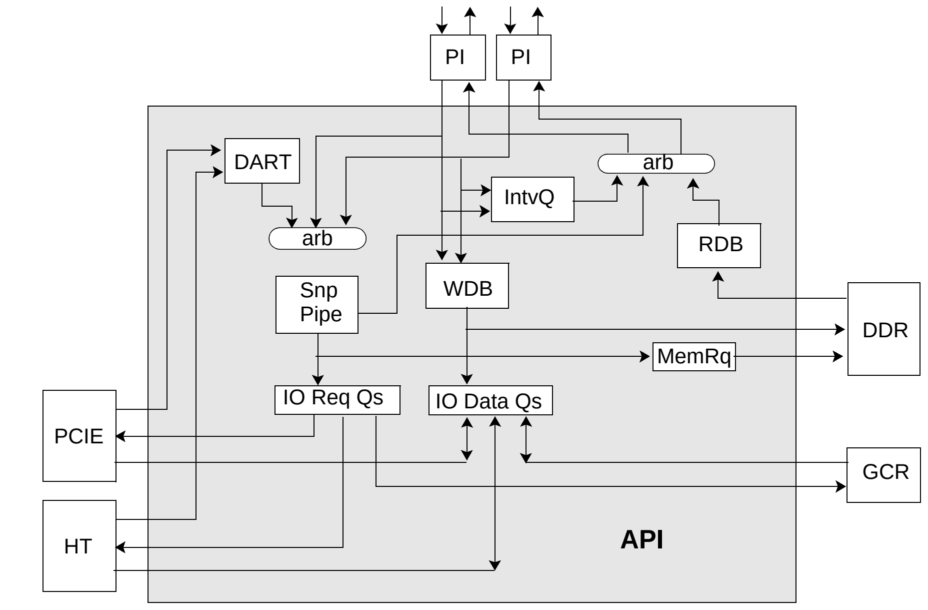
If the operation is allowed on this page, the way number is encoded and used along with the TLBIDX and the TLBSEL to access the 1024-entry DARTDATA cache RAM. The 24-bit Physical Page Number (PPN) is fetched from the DARTDATA cache RAM and is concatenated to the 12 low-order address bits 52:63 to complete the physical address. This process is illustrated in *Figure 3-9* on page 58.

If no match is found in the DART TLB, the DART Table entry in memory (*Figure 3-8* on page 55) is fetched and used to complete the mapping. The TLB entry in the cache is then updated with this new entry.

A LRU (Least-Recently-Used) replacement mechanism is used to determine which entry is updated. An entry is considered “used” when a valid mapped request hits in the entry. The initial state (after the TLB is invalidated) of a set of 4 entries is the order (0, 1, 2, 3) where way-0 is the least-recently-used (LRU) and way-3 is the most-recently-used (MRU).

3.5 Processor Interconnect Interface Microarchitecture

Figure 3-10. CPC945 Processor Interface Environment



3.5.1 System Overview

The Processor Interconnect Interface (ApiIf) Block interfaces the two processor interfaces to the rest of the agents on the CPC945 chip (*Figure 3-10*).

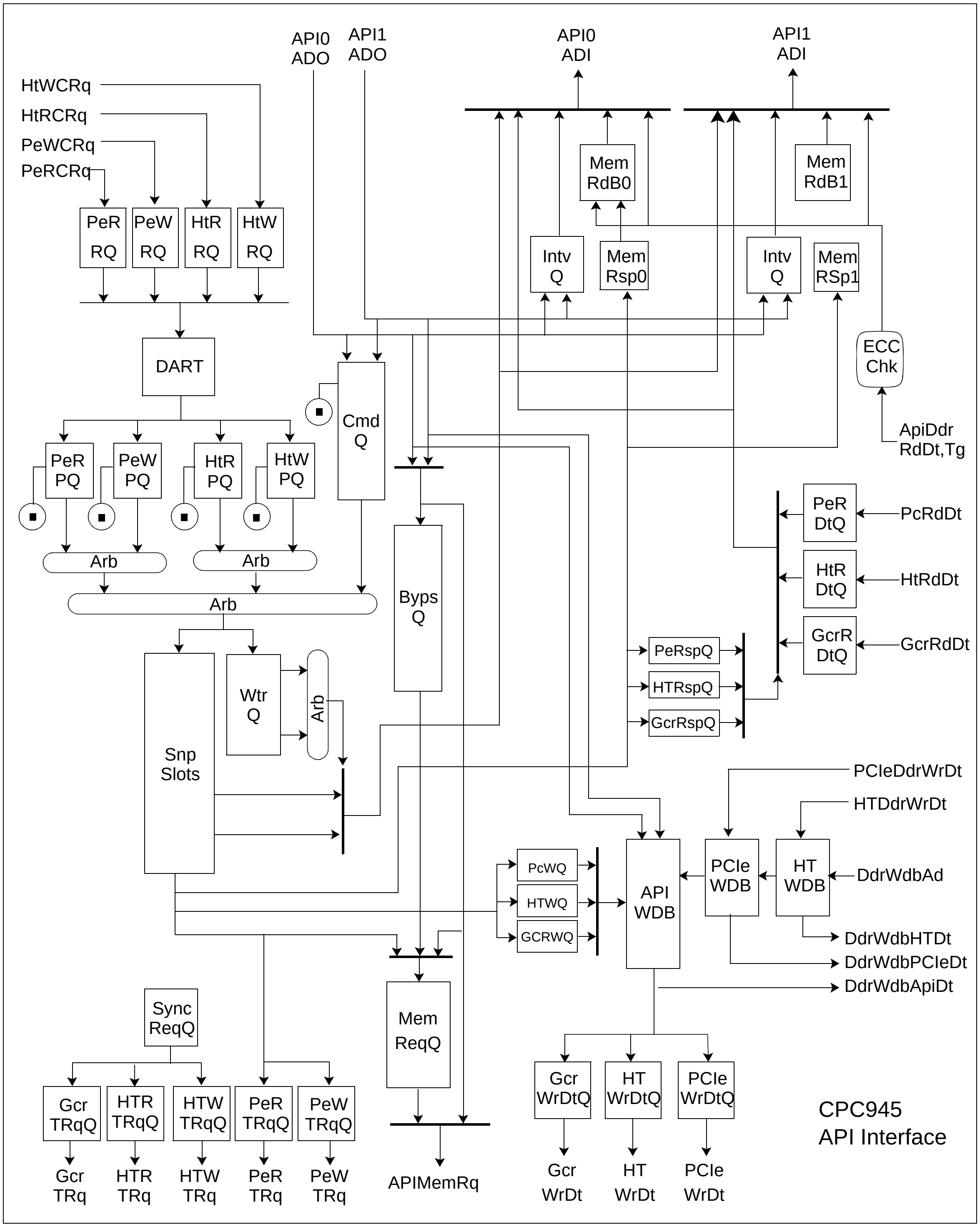
The main functions of the PI block are:

- Collecting requests from the processor and coherent requests from the other agents and dispatching them to their respective targets within CPC945.
- Ensuring cache coherency by reflecting the requests from the processors and other agents back to the processors so that the processor's caches can be snooped.
- Translating requests from PCI Express and HyperTransport interfaces from logical to physical addresses through a DART.

3.5.2 ApiIf Operation

Figure 3-11 shows CPC945's major functional PI blocks and is to be used with the descriptions of PI operation contained in this section.

Figure 3-11. CPC945 PI Unit



3.5.2.1 Commands from Processor

Table 3-6 lists CPC945’s response to Bus Operations from the processor. All operations reflect the command back to the processors and receive an Accumulated Snoop Response. The “PI Action” indicates what happens after the snoop response has been received. “Target” in the table refers to Memory, HT, and/or PCIe.

Table 3-6. Response to Bus Operations from the Processor.

Bus Operation	Acc Snoop Response	PI Action	Note
Read	Clean (Null), Shared	Send Read Request to Target	
	Retry	Do nothing	
	Modified Intervention	Send Write Request to Target Wait for Read Data Response from Exclusive Processor Send Read Data to Requesting Processor Send Data to Target	1
	Shared Intervention	Wait for Read Data Response from Shared Processor Send Read Data to Requesting Processor	1
	Anything Else	Generate an Exception	
RWITM	Clean (Null)	Send Read Request to Target	
	Retry	Do nothing	
	Modified Intervention, Shared Intervention	Wait for Read Data Response Send Read Data to Requesting Processor	1
	Anything Else	Generate an Exception	
Write with Flush, Kill, Clean	Clean (Null)	Send Write Request to Target	
	Retry	Do nothing	
	Anything Else	Generate an exception	
Sync EIEIO	Clean (Null)	Add indicator to PCIe, HT Interfaces to prevent combining	
	Retry	Do nothing	
	Anything Else	Generate an exception	
Power tuning	Any	See section on power tuning	
Other	Any	Do nothing	

Note:

1. Intervention responses are only allowed on accesses to Memory. Intervention responses to non-memory generate an exception.

Processor Read [and read with intent to modify (RWITM)] Commands to Memory

1. A command to read a block comes from a processor’s ADO interface.
2. A read data buffer is allocated, if available. If not available, then the command is retried with the Transfer Handshake signals.
3. The command is loaded into the CmdQ, if there is space. If there is no space, then the command is retried with the Transfer Handshake signals.
4. All read commands (and RWITM) are also “bypassed” to memory. This means the read request to memory is seen before the command is reflected and snooped by the processors.
5. To insure cache coherence, the read address is compared to addresses of the commands that are in front of it waiting to be reflected (including I/O commands). If there is a match to the same cache line, there is a potential for the commands to be reflected in a different order than the order of the requests going to memory. This could lead to incorrect data being read from the bypassed read request. A command is marked as having a “conflict” if this occurs and the bypassed read is effectively aborted allowing the normal read to occur after it has been snooped (See *Snoop Bypass* on page 75).
6. The command waits for arbitration between processor commands and other agents (PCIe and HT).
7. When granted, the command proceeds to a SnoopSlot and is removed from the CmdQ.
8. Upon entering a SnoopSlot, the command is checked against the other commands to see if it is targeting the same cache line. This information is used to insure that the PAAM window constraints are met (see PAAMWIN).
9. The command in a SnoopSlot reflects its command to the other processors at an appropriate time according to priority with other commands, the Snoop Window, and the PAAM Window. Also, before reflecting, the command waits in a SnoopSlot until its resources (Request Queues, Write Data Buffer, etc.) are available (see *Table 3-8 Resources Required for Reflecting* on page 73). Any resources that have a potential to be needed are allocated.
10. After reflecting its command, the SnoopSlot waits a fixed amount of time (SNOOPLAT) before receiving its accumulated Snoop Status.
11. The command proceeds depending on the accumulated snoop response value:
    - If the status is **retry**, the command is completed and will be reissued again by the processor.
    - If the status is **clean** or **shared**, then a request is sent to the MemRqQ and the MemRsp state is updated (See *Snoop Bypass* on page 75).
    - If the status is **shared intervention**, the intervening processor will return the data. No memory request is needed. Only an Intervention Queue entry is needed.
    - If the status is **modified intervention**, the intervening processor will return the data, but unless the command is a RWITM, the data also needs to be written to memory. On a RWITM, no write to memory is needed because the requesting processor becomes the owner of the line with it in a Modified state. The read command generates a write request to the memory (See Intervention Processing below).
12. Any resources that were allocated before reflecting the command and are not needed after the Snoop Response are de-allocated (see *Table 3-8 Resources Required for Reflecting* on page 73).
13. The read (or a write in the case of a Modified Intervention) is sent to the memory controller.
14. Some time later the read data is returned with a tag indicating which read data buffer the data corresponds to.

15. The data is loaded into the read data buffer and then sent to the processor after arbitrating for the ADI bus.

Processor Read (and RWITM) Commands to I/O

1. A command to read a block comes from the processor's ADO interface.
2. The command is loaded into the CmdQ, if there is space. If there is no space, then the command is retried with the Transfer Handshake signals.
3. The command waits for arbitration between processor commands and other agents (PCIe and HT).
4. When granted, the command proceeds to a SnoopSlot and is removed from the CmdQ.
5. Upon entering a SnoopSlot, the command is checked against the other commands to see if it is targeting the same cache line. This information is used to insure that the PAAM window constraints are met (See PAAMWIN).
6. The command in a SnoopSlot reflects its command to the other processors at an appropriate time according to priority with other commands, the Snoop Window, and the PAAM Window. Also, before reflecting, the command waits in a SnoopSlot until its resources (Request Queues, Write Data Buffer, etc.) are available (see resource table, *Table 3-8* on page 73). Any resources that have a potential to be needed are allocated.
7. After reflecting its command, the SnoopSlot waits a fixed amount of time (SNOOPLAT) before receiving its accumulated Snoop Status.
8. The command proceeds depending on the accumulated snoop response value:
   - If the status is **retry**, the Command is completed and will be reissued again by the processor.
   - If the status is **clean** or **shared**, then a request is sent to the appropriate Target's Read Request Queue and Response.
9. Any resources that were allocated before reflecting the command and are not needed after the Snoop Response are de-allocated (see resource table, *Table 3-8*).
10. The read operation in the Target Read Request Queue waits for space in the Read Data Queue. When the space is ready, a read is sent to the target.
11. The returning read data is put into the appropriate Read Data Queue according to the target of the request. The next entry in the corresponding Response Queue is fetched. The Response Queue entry determines who was the original requestor and contains the header for the ReadDataResponse to the processor. The data is then sent to that requestor over the appropriate ADI bus.

Processor Write Commands to Memory

1. A command to write a block comes from the processor's ADO interface.
2. The command is loaded into the CmdQ, if there is space. If there is no space, then the command is retried with the Transfer Handshake signals.
3. A Write Data Buffer (WDB) is also allocated to the command, if available. If not available, then the command is retried with the Transfer Handshake signals.
4. If the command is accepted, the data following it is written to the assigned WDB location.
5. While the data is being written, the command waits for arbitration between processor commands and other agents (PCIe and HT).
6. When granted, the command proceeds to a SnoopSlot and is removed from the CmdQ.

7. Upon entering a SnoopSlot, the command is checked against the other commands to see if it is targeting the same cache line. This information is used to insure that the PAAM window constraints are met (See PAAMWIN).
8. The command in a SnoopSlot reflects its command to the other processors at an appropriate time according to priority with other commands, the Snoop Window, and the PAAM Window. Also, before reflecting, the command waits in a SnoopSlot until its resources (Request Queues, Write Data Buffer, etc.) are available (see*Table 3-8 Resources Required for Reflecting* on page 73). Any resources that have a potential to be needed are allocated.
9. After reflecting its command, the SnoopSlot waits a fixed amount of time (SNOOPLAT) before receiving its accumulated Snoop Status.
10. The command proceeds depending on the accumulated snoop response value.
    - If the status is **retry,** the command is completed and will be reissued again by the processor.
    - If the status is not **retry,** then a request is sent to MemReqQ
11. Any resources that were allocated before reflecting the command and are not needed after the Snoop Response are de-allocated (see *Table 3-8 Resources Required for Reflecting* on page 73).
12. The write is sent to the memory controller.
13. Some time later the write command is issued to memory. The memory controller fetches the write data from the Write Data Buffer and sends it to memory at the appropriate time. When the write is complete, the WDB tag is sent back to the PI unit indicating that the WDB location can now be freed and used for another command.

Processor Write Commands to I/O

1. A command to write a block comes from the processor’s ADO interface.
2. The command is loaded into the CmdQ, if there is space. If there is no space, then the command is retried with the Transfer Handshake signals.
3. A Write Data Buffer (WDB) is also assigned to the command, if available. If not available then the command is retried with the Transfer Handshake signals.
4. If the command is accepted, the data following it is written to the assigned WDB location.
5. While the data is being written, the command waits for arbitration between other processor commands and other agents (PCIe and HT).
6. When granted, the command proceeds to a SnoopSlot and is removed from the CmdQ.
7. Upon entering a SnoopSlot, the command is checked against the other commands to see if it is targeting the same cache line. This information is used to insure that the PAAM window constraints are met (See PAAMWIN).
8. The command in a SnoopSlot reflects its command to the other processors at an appropriate time according to priority with other commands, the Snoop Window, and the PAAM Window. Also, before reflecting, the command waits in a SnoopSlot until its resources (Request Queues, Write Data Buffer, etc.) are available (see *Table 3-8* on page 73). Any resources that have a potential to be needed are allocated.
9. After reflecting its command, the SnoopSlot waits a fixed amount of time (SNOOPLAT) before receiving its accumulated Snoop Status.

10. The command proceeds depending on the accumulated snoop response value.
    - If it is a **retry**, the command is completed and will be reissued again by the processor.
    - If the status is not **retry,** then a request is sent to the appropriate Target’s Request Queue and a WDB Read is issued to the WDB. A WDB read causes the write data to be fetched from the WDB and written to the appropriate Write Data Queue.
11. Any resources that were allocated before reflecting the command and are not needed after the Snoop Response are de-allocated (see *Table 3-8 Resources Required for Reflecting* on page 73).
12. The write operation in the Target Write Request Queue waits for its data to appear in the write data queue. When the data is ready, a write is sent to the target. (The target request queue is only used for nonmemory writes in CPC945)

Intervention Processing

1. Intervention processing occurs when a processor returns a modified or shared intervention accumulated snoop response to a command to memory space. I/O transactions do not get an intervention. Instead the processor’s read data response goes direct to memory and the I/O transaction reads the data from memory.
2. After the snoop response, the ApiIf waits for the intervening processor to send the data with a read data response (RDR) header.
3. The RDR contains the transfer tag of the original requesting processor. This transfer tag is used to steer the intervention data to the appropriate processor through the intervention queue.
4. The intervention data is sent to the requesting processor as soon as it is received from the intervening processor. If a command is interjected from the intervening processor, then null cycles might be inserted into the data stream to the requesting processor.
5. If a write to memory is also needed (see *Table 3-6 Response to Bus Operations from the Processor.* on page 63), the intervention data is also written to the write data buffer (WDB). The transfer tag is used to select which WDB entry to use. The tag matches the transfer tag stored with the WBD state.
6. When the read data response has completed and the intervention data has been completely written into the WDB, a valid indication for that buffer is sent to the memory controller. The write command for this data can now be sent to the memory interface. The write data is then fetched from the WDB and sent to memory at the appropriate time.

Operations from External Agents (PCIe, HT)

Table 3-7 lists the coherent operations from other agents, how they translate into a command for reflection to the processors and the action taken after the Accumulated Snoop response is received.

Table 3-7. Operations (I/O Commands) from Other Agents .

Operation from PCIe, HT	Bus Operation[2]	Acc Snoop Response	ApiIf Action
Read	Read	Nonretry	Send Read Request to Target
		Retry	See Note 1
Write with WrAll and TSize of 128 bytes.	Write with Kill	Nonretry	Send Write Request to Target
		Retry	See Note 1
Write without WrAll or TSize less than 128 bytes	Write with Flush	Nonretry	Send Write Request to Target
		Retry	See Note 1

Notes:

1. On a retry, the Agent’s Requesting Queue is backed up to the command that was retried. Any commands that were reflected are forced to retry, also. The Request Queue then reissues Commands starting at the original retried command. For PCIe and HT Writes, both the Write Request Queue and Read Request are backed up and retried. For PCIe and HT reads just the Read Request Queue is backed up and retried.
2. The Address Modifier bits for all operations are: N=0 (No Intervention), S=1 (Non-Cacheable), M=1 (Coherent), G=0 (non-guarded), R=0 (No rerun), and P=1 (Pipeline Snoops).

I/O Read or Write Command Processing

1. An I/O operation to read or write a block comes from the I/O Agent’s Read or Write Coherent Request Interface and is written to its respective request queue.
2. The Request Queue entries are read, arbitrated, passed through the DART to translate into a 36-bit Physical Address, converted into commands, and written to their respective pending queues.
3. The command waits for arbitration between processor commands and other agents (PCIe and HT). The command has its N bit reset to insure that an Intervention Snoop Response does not occur from the Processor.
4. When granted arbitration, the command proceeds to a SnoopSlot but is not removed from the pending queue.
5. Upon entering a SnoopSlot, the command is checked against the other commands to see if it is targeting the same cache line. This information is used to insure that the PAAM window constraints are met (See *PAAM Window* on page 74).
6. The command in a SnoopSlot reflects its command to the other processors at an appropriate time according to priority with other commands, the Snoop Window, and the PAAM Window. Also, before reflecting, the command waits in a SnoopSlot until its resources (Request Queues, Write Data Buffer, etc.) are available (see *Table 3-8* on page 73). Any resources that have a potential to be needed are allocated.
7. After reflecting its command, the SnoopSlot waits a fixed amount of time (SNOOPLAT) before receiving the command’s accumulated Snoop Status.
8. The command proceeds depending on the accumulated snoop response value.
   – If the status is non-**retry,** then a request is sent to the appropriate Target’s Request Queue for a non-memory request or the MemReqQ for a memory request.

   - If it is a **retry**, then an “Abort” is sent to the Agent’s Pending Queue to backup and retry all operations. Any of the Agent’s operations in the Snoop Slots are forced to retry and any of the Agent’s Operations waiting to reflect their command are aborted.

9. Any resources that were allocated before reflecting the command and are not required after the Snoop Response are de-allocated (see *Table 3-8* on page 73).
10. If the command was not retried then the command proceeds to access its target or memory just like a processor command.

3.5.2.2 Resources

The PI resources are the queues and buffers that a command or request needs before it can be processed. These resources might be needed at the input of the PI block or might be needed before a command can be reflected and processed.

3.5.2.3 Resource Descriptions

Command Queue

The Command Queue (sometimes called Command Slots) is the holding place for Commands from the processors. A Command Queue entry must be available before a command is accepted from the PI Bus. If an entry is not available, the command is responded with a Transfer Handshake Retry.

Some of the Command Queue entries are reserved or guaranteed based on a per processor basis. The size of the Command Queue is also configurable (See *Section 12.9.1 API Proc Command Slot Configuration Register (APIProcCmd)* on page 409).

Write Data Buffer

The Write Data Buffer (WDB) holds write data waiting to be written to memory or sent to an I/O Write Data queue. This data is from a processor write command or an intervention response to a read command. The Write Data buffer holds 32 entries of 128 bytes.

For write commands from a processor, a WDB entry must be available before the write command is accepted by the Command Slots. If an entry is not available, the PI responds with a Transfer Handshake Retry. If an entry is available, the address of the entry is remembered and kept with the command as it is processed. The WDB entry is released if the write command gets a Retry status or when the data is sent to memory or one of the I/O write data queues.

For processor read commands, a WDB entry needs to be allocated before the read command is reflected. If an entry is not available the command waits until one is available. A WDB entry is required because the read might receive a modified intervention response. The Intervention data needs to be sent to memory in addition to being sent to the requesting processor. In that case, the read command turns into a write command and uses the WDB allocated. For RWITM commands, a WDB entry is not required because the possible Intervention data does not need to be sent to memory, because the requesting processor becomes the owner of the data. For reads, the WDB entry is released if a status that is not Modified Intervention is received or when the Intervention data is written to memory.

The WDB entries are reserved based upon on a per processor basis and the type of data, Write Data, or Intervention Data. The size of the WDB is also configurable (See *Section 12.9.7 API Write Data Buffer (WDB) Configuration Register (APIWdbCfg)* on page 417).

Read Data Buffer

The Read Data Buffer (RDB) holds read data that is received from Memory before it is sent to the requesting processor. There is a dedicated RDB for each PI Port (0,1). Each buffer holds 16 entries of 128 bytes. For a read command from a processor, a RDB entry must be available before the read command is accepted. If an entry is not available, the command is responded with a Transfer Handshake Retry. If an entry is available the address of the entry is remembered and is kept with the command as it is processed. Read commands to memory proceed to the Snoop Pipe to be processed while being “bypassed” to send an early request to memory. For a RDB to be released both the normal “snoop” path and the “bypass” path must be finished with the buffer. (See *Bypass Queue* on page 72.)

The RDB entries are reserved based upon on a per processor basis. The size of the RDB is also configurable (See *Section 12.9.10 API Memory Read Configuration Register (APIMemRdCfg)* on page 421).

Intervention Buffer

The Intervention Buffer holds Intervention packets as they are sent from one processor to another. Intervention packets are Read Data Response packets. Each packet contains an 8 byte header and 128 bytes of data. There are dedicated Intervention Buffers for each PI Port (0,1).

For read and RWITM commands from a processor, an Intervention Buffer entry needs to be allocated before the command is reflected. If an entry is not available the command waits until one is available. An intervention buffer entry is required because the read might get an intervention response. If an intervention response is not received, the intervention buffer is released. If an intervention response is received, the buffer is released when the intervention packet is completely sent to the requesting processor.

The intervention buffer entries are reserved based upon on a per processor basis. The size of the intervention buffer per port is also configurable. (See *Section 12.9.8 API Intervention Buffer Configuration Register (APIIntCfg)* on page 418.)

I/O Input Queues

The I/O input queues hold I/O requests while they are waiting for arbitration to be translated in the DART. There are 4 I/O input queues (see PI unit diagram), 1 each for PCIe reads (PeRRqQ), PCIe writes (PeQRqQ), HT reads (HtRRqQ), and HT writes (HtWRqQ). A request is removed from the queue only when it is able to complete its DART translation and be sent to its I/O pending queue. When an entry is removed, an Acknowledge (credit) is sent to the appropriate I/O interface indicating to the requestor that another entry is available. It is up to the requestor to keep track of the number of queue entries available. The sizes of the I/O input queues are configurable and are the responsibility of the requestor. (See *Section 12.12.13 HT1/PI Interface Control Register* on page 633 and *Section 12.11.3.1 CORE_X: PI Core Interface Parameters Register* on page 586.)

I/O Pending Queues

The I/O pending queues hold I/O requests while they are waiting for arbitration and space in the Snoop Slots. There are 4 I/O Pending Queues (*Figure 3-11 CPC945 PI Unit* on page 62), 1 each for PCIe reads (PeRPQ), PCIe writes (PeWPQ), HT reads (HtRPQ), and HT writes (HtWPQ). Requests wait in the I/O input queues until there is space available in the I/O pending queues. Entries are fetched from the I/O pending queues when they are granted arbitration and there is room for their type in the Snoop Slots (*Snoop Slots* on page 71). The requests are not removed, though, until the request has received a nonretry snoop status. Whenever a retry status is received, the request is re-fetched from the queue and arbitrates again.

The sizes of the I/O Pending Queues are configurable. (See *Section 12.9.2 API I/O Pending Queue Configuration Register (APIIOPnd)* on page 411.)

Snoop Slots

The Snoop Slots are a holding place for commands while they are waiting to be reflected until their snoop status has come back and their PAAMWIN has expired. Commands sit in the command queue or the I/O pending queues and only arbitrate for access to the Snoop Slots if a Snoop Slot is available for that type. Once a command is in a Snoop Slot it waits to be reflected to the processors until the command becomes high enough priority and its downstream resources are available (see *Summary of Resources Needed to Reflect* on page 73). Once reflected the command stays active for a fixed number of cycles until its snoop response returns (SNOOPLAT) and it PAAM Window is exhausted (PAAMWIN).

The Snoop Slot entries are reserved based upon the requester and type of command (Processor, PCIe read, PCIe write, HT read, HT write). The number of the Snoop Slots is also configurable. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Sync and Target Request Queues

The Sync and Target Request Queues contain requests targeting PCIe, HT, or GCR. The Sync Queue is used to cross the asynchronous boundary from PI clk to DDR2 clk and is used for HT and GCR targets. There are 5 Target Queues, one each for reads and writes to GCR (GcrTRqQ), reads to HT (HtRTRqQ), writes to HT (HtWTRqQ), reads to PCIe (PeRTRqQ), and writes to PCIe (PeWTRqQ). For read operations the requests stay in the queues until the Target has space in its receiving queue and, for processor-initiated requests, there is space in the corresponding Read Data Queue. For write operations the requests stay in the queues until the Target has space in its receiving queue and, for processor initiated requests, there is data in the corresponding Write Data Queue.

The Sync and Target Queues are resources that are needed before a command can be reflected (See *Summary of Resources Needed to Reflect*). The PCIe and HT Target Queue entries are reserved based upon the requestor (I/O or Processor). The number of entries in the Sync and Target Request Queues is also configurable.)See *Section 12.9.4 API Target Request Queue Configuration Register (APITRqCfg)* on page 414.)

Space in a target's receiving queue is determined by how many requests have been sent on the Target Request Address Interface and have not been “acknowledged” (received a credit). This is compared to the number of entries in the receiving queue. There are 5 receiving queues for the 5 interfaces, PCIe Read, PCIe Write, HT Read, HT Writes, and GCR. The number of entries in the PCIe and HT receiving queues is configurable. (See *Section 12.9.4 API Target Request Queue Configuration Register (APITRqCfg)* on page 414.)

Target Response Queues

The Target Response Queues contain requests targeting PCIe, HT, or GCR that originate from a processor. There are 3 Target Response Queues, one each for reads to GCR (GcrRspQ), HT (HtRspQ), and PCIe (PeRspQ). The response queues contain information necessary to generate a Read Data Response header when the data comes back from the Target.

The Target Response Queues are resources that are needed before a Command can be reflected. (See *Summary of Resources Needed to Reflect* on page 73.) The number of entries in Target Response Queues is configurable. (See *Section 12.9.5 API Target Response Queue Configuration Register (APITRspCfg)* on page 415.)

Target Read and Write Data Queues

The Target Read and Write Data Queues contain the data for requests targeting PCIe, HT, or GCR that originated from a processor. There are 3 Target Read Data Queues, one each for reads to GCR (GcrRDtQ), HT (HtRDtQ), and PCIe (PeRDtQ). There are 3 Target Write Data Queues, one each for writes to GCR (GcrWDtQ), HT (HtWDtQ), and PCIe (PeWDtQ). For read requests there must be space in the corresponding read data queue before the read request is sent to the target. For Write requests the data must be fetched from the WDB and be in the Write Data Queue before the write request is sent to the target.

The Target Read and Write Data Queues are not required resources before a Command is reflected. The number of entries in Target Data Queues is configurable. (See *Section 12.9.6 API Target Data Queue Configuration Register (APIDtQCfg)* on page 416.)

Bypass Queue

The Bypass Queue contains processor read requests to memory. Read requests are sent to the Command Queue and the bypass path where they are written into the Bypass Queue if the conditions are not right for the requests to be sent to the Memory Controller (See *“Snoop Bypass” on page 75*). The size of the Bypass Queue is large enough so that it should never overflow. It can accommodate more than the number of Read Requests that are possible in the Snoop Slots and Command Queue. The size of the Bypass Queue is configurable (See *Section 12.9.9 API Memory Request Configuration Register (APIMemReqCfg)* on page 419), but it should not be changed from its default of 32 entries.

Memory Request Queue

The memory request queue (MemReqQ) contains requests from processors or I/O that are targeting memory. These requests are waiting to be sent to the memory controller and can originate from bypassed requests, snoop slot requests, or DART miss requests. Bypassed requests can be queued here as long as the bypass cutoff amount has not been reached or the bypass guarantee has not been satisfied (See *Snoop Bypass* on page 75). Otherwise they need to wait in the bypass queue.

Requests from the snoop slots need to allocate a MemReqQ entry before being reflected. After a request is reflected, the snoop status that is returned determines whether a memory request is sent to the MemReqQ. If a memory request is not needed and a MemReqQ entry was allocated, the entry is de-allocated when the snoop status is returned. Otherwise, the entry is deallocated when the request is fetched from the queue.

DART miss requests can only occupy one entry of the MemReqQ. This entry is guaranteed and is always available. Only when the request is removed from the queue can another DART request be sent. In addition to 1 reserved entry each for DART miss and bypass requests, the MemReqQ entries reserved for “snooped” requests are based upon the requestor (either processor or I/O). The size of the MemReqQ is also configurable. (See *Section 12.9.9 API Memory Request Configuration Register (APIMemReqCfg)* on page 419.)

Summary of Resources Needed to Reflect

Table 3-8 lists the resources needed before a command can be reflected to the processors and sent down the Snoop Pipe. Except where indicated these resources are allocated before the command is reflected. Resources indicated with an asterisk (*) are allocated when the command is accepted on the PI interface.

Table 3-8. Resources Required for Reflecting.

Requestor	Target	Command	Resources
Processor	Memory	Read	Memory Request Queue for Processor Port Number Intervention Buffer for Processor Number Write Data Buffer for Intervention and Processor Number *Read Data Buffer for Port Number
		RWITM	Memory Request Queue for Processor Port Number Intervention Buffer for Processor Number *Port Number Read Data Buffer
		Write	Memory Request Queue for Processor *Write Data Buffer for Write and Processor Number
		Other	None
	HT, PCIe	Read, RWITM	HT/PCIe Target Read Request Queue for Processor HT/PCIe Target Read Response Queue
		Write	HT/PCIe Target Write Request Queue for Processor *Write Data Buffer for Write and Processor Number
	GCR	Read, RWITM	GCR Target Request Queue GCR Target Response Queue
		Write	GCR Target Request Queue *Write Data Buffer for Write and Processor Number
HT, PCIe	Memory	Read, Write	Memory Request Queue for I/O
	HT, PCIe	Read	HT/PCIe Target Read Request Queue for I/O
		Write	HT/PCIe Target Write Request Queue for I/O
	GCR	Read, Write	GCR Target Request Queue

3.5.3 Ordering of Operations

3.5.3.1 Processor Commands

The CPC945 chip assumes that the processor insures correct ordering of operations affected by Sync and EIEIO instructions before the Accumulated Snoop Response time. After the Accumulated Snoop Response time, the operations to memory are sequentially ordered. Because of this, no special consideration is needed for Sync and EIEIO instructions to memory.

For PCIe and HT ordering, Sync and EIEIO instructions provide a barrier so that read or write combining does not occur across the barrier.

I/O Requests

The CPC945 core interface specification requires that all PCIe and HT requests stay in order with respect to their request buses (HT reads stay in order, PCIe writes stay in order, etc.). In addition to this in-order constraint, Reads have to wait until all writes from the same requestor that occurred earlier have been completely (reads push writes).

To accomplish this, any I/O request that is retried (with a snoop response of retry) must retry any I/O request of the same requestor and same command type that has also been reflected. In addition to this, since reads push writes, any write from an I/O requestor that is retried must also retry any reads from the same requestor that have been reflected.

PAAM Window

PAAMWIN specifies the number of beats (PI clks) that have to occur between reflecting successive commands that target the same 128-byte block address. Every time a command is reflected, a window (PAAM Window) is generated that is PAAMWIN cycles wide. There can be many PAAM Windows active at various stages of time. When a new command enters the Snoop Slots, a qualification is made whether this new command should check the PAAM Windows. If this command is from an HT or PCIe source and its destination is HT or PCIe, or if it is a power tuning command, then the check is not made. All other commands check the PAAM Windows. If the block address of the new command matches the address of the PAAM Window command, or the P Bit of the new command is 0 and the P bit of the PAAM Window command is 0, then the new command waits until the PAAM Window has completed.

In the implementation of the PAAMWIN check, a new command checks all the commands currently active in the SnoopSlots whether they have been reflected or not. If the check matches, then this new command must wait until any matching commands have been reflected and their PAAM window has timed out. This has the effect of ordering operations that have matching PAAM window constraints.

Ordering Deadlocks

The ordering requirements of I/O Operations in combination with the implementation of the PAAM window checking could lead to deadlocks in the snoop slots. For example a read from HT to PCIe could be waiting to be reflected and not proceed because a stream of reads is held up in front of it. A read from HT to memory location X could be held up behind it and not proceed because HT reads have to be processed in order. A Write from PCIe to memory location X, that entered the snoop slots after the HT read to memory, is also held up because of the PAAM window checking. This could lead to a deadlock because the PCIe write to memory is held up. In general, writes should not be held up throughout the system so that writes can pass reads preventing these types of deadlocks.

In this situation, the implementation of the PAAM window checking is artificially causing the PCIe write to memory to be held up. Since the HT read to memory is held up the PCIe write to memory could be reflected first without violating any ordering constraints. Although, this is not how the PAAM window checking works.

To break these types of deadlocks, a Snoop Watchdog Timer is implemented (see *“API Command Arbitration Register (APICmdArb)” on page 412*). This Timer is continually counting down except when a Processor request is granted arbitration. It is then reset to an initial value defined in the APICmdArb register. If the counter reaches zero, then the highest priority command waiting to be reflected is reflected. The reflected command’s Snoop Status is forced to “retry” and will be re issued. This allows any Commands behind it to be freed and allowed to proceed.

Snoop Bypass

Snoop bypass refers to the feature of reducing the processor-to-memory latency by allowing processor reads to get a request sent to the memory controller before the command has been “snooped” by the other processors. The read is eventually snooped to make sure that another processor does not have ownership of the line, in which case, the “bypassed” read is discarded.

In CPC945 all processor to memory reads are “bypassed”. This feature requires a method of checking to make sure that cache coherency is maintained, since the order that the memory controller sees the requests might be different than the order that requests are snooped. For example, suppose a write to a line has been reflected, but a memory request has not been made, and a read to the same line is bypassed. If the memory controller sees the bypassed read first, it will return the old and not the new data that is expected since the write was reflected first.

The mechanism necessary to support snoop bypass involves a method for checking for conflicts, a method for communicating between the snooped read and the bypassed read, and a method for discarding bypassed data that might be incorrect.

Snoop Bypass Conflict Detection

Insuring cache coherency between bypassed reads and other transactions requires comparing the bypassed address to the all the possible requests that are in front of it that could or have already reflected their addresses. This means that the bypassed address is checked against all reads and writes in the command queue, all the I/O pending queues, and all the snoop slots. If a match occurs, the bypassed memory read request is marked as having a “conflict”. This “conflict” indication is carried with the request until the point where it receives snoop status. This “conflict” might change its action on the return of the snoop status.

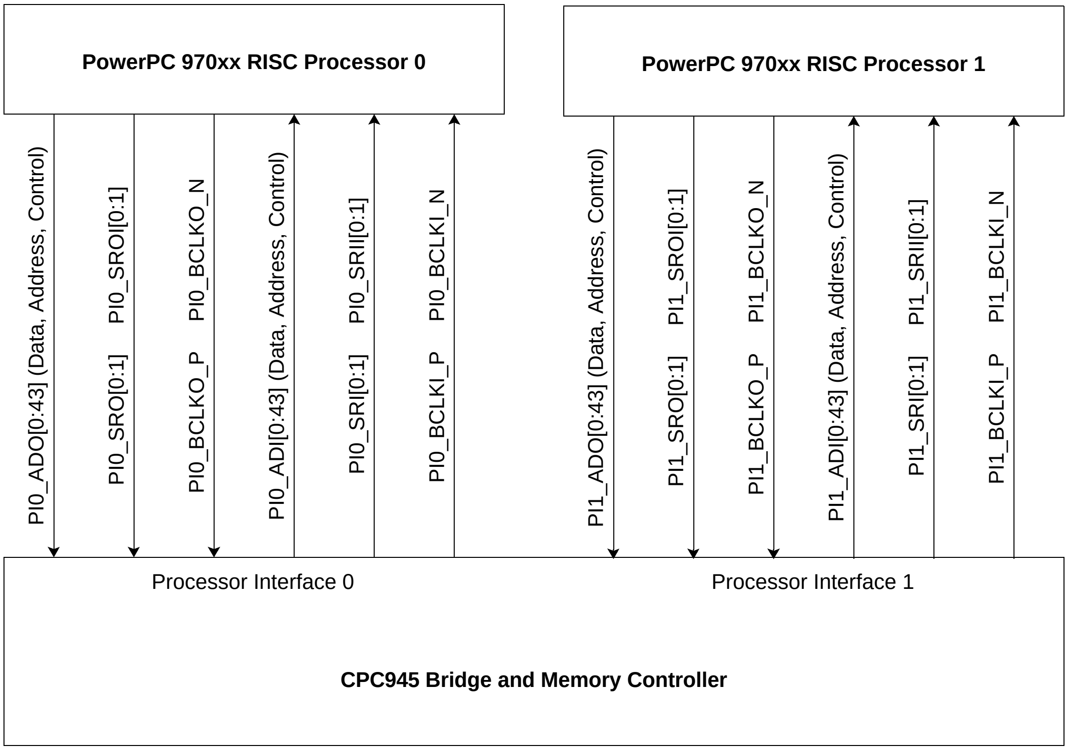
In addition to conflict detection, any I/O request that is fetched from a I/O pending queue checks to see if there are any matching read commands in the command queue that have not been marked with a conflict. If there are, then the I/O request is held off to allow the processor read command from the command queue to be sent to the snoop slots first. The processor read is then reflected first because of the PAAM window checking.

4. Processor Interconnect Bus

The CPC945 Bridge and Memory Controller connects the system microprocessors to memory and I/O devices through the processor interface bus, also called the Processor Interconnect. The interface consists of two parts: the slave, which is used by the processor to access memory and I/O, and the master, which is used to pass coherency information between the I/O devices and the processor. The IBM CPC945 Bridge and Memory Controller supports two complete processor interfaces. The two halves of each interface are unidirectional and point-to-point. The processor interface supports the 128-byte line size of the PowerPC 970xx processor.

As all connections between the processor and the CPC945 are point-to-point, any interaction between processors and other processors, memory, the peripheral component interconnect (PCIe), and so forth is routed through the CPC945 as shown in *Figure 4-1*. The signals are labeled from the perspective of the processor. The CPC945 buffers can hold outstanding requests and data.

Figure 4-1. Two Processors Connected to the CPC945


PowerPC 970xx RISC Processor 0
PowerPC 970xx RISC Processor 1
PI0_ADO[0:43] (Data, Address, Control)
PI0_SRO[0:1]
PI0_SROI[0:1]
PI0_BCLKO_P
PI0_BCLKO_N
PI0_ADI[0:43] (Data, Address, Control)
PI0_SRI[0:1]
PI0_SRII[0:1]
PI0_BCLKI_P
PI0_BCLKI_N
PI1_ADO[0:43] (Data, Address, Control)
PI1_SRO[0:1]
PI1_SROI[0:1]
PI1_BCLKO_P
PI1_BCLKO_N
PI1_ADI[0:43] (Data, Address, Control)
PI1_SRI[0:1]
PI1_SRII[0:1]
PI1_BCLKI_P
PI1_BCLKI_N
Processor Interface 0
Processor Interface 1
CPC945 Bridge and Memory Controller

4.1 Processor Interface Alignment Procedure

The processor interface is a high-speed, double data rate, source-synchronous interface. Its speed requires that all of the bits on the interface bus be precisely deskewed to maintain bit alignment around the clock edges that travel alongside the data bits. An initialization alignment procedure (IAP) state machine, which detects and deskews the flight time differences between the different bits of the interface, is implemented on the CPC945, as well as on the PowerPC 970xx processor.

It is assumed that all power and clocks have been correctly programmed and are stable. The processor interface bus clock from the PowerPC 970xx must be running and stable before releasing reset to the CPC945. There are two complete PI busses on the CPC945, PI0 and PI1. Even though the PowerPC 970MP is a dual core part it has a single PI interface. Both PI busses are only used in quad core systems with the 970MP.

1. Configure the APIPhy Configuration 0 Registers for PI0 and PI1 (address: 0xF802_2020 and 0xF802_3020)
   - target_time [61:63] - valid values are 0 - 4.
   - data_wind[20:23] - suggested value of 3 or 4.
2. Reset the Processor Interface enables
   - write all zeros to APIPhy Configuration 1 Registers for PI0 and PI1 (address: 0xF802_2030 and 0xF802_3030) except the APSyncRcv_en bit to 1 to enable external APSync.
3. Configure the CPU Power Management Register (address: 0xF800_0820)
   - Disable all power down related functions.
   - Ensure that the E [25] bit is set for external APSync mode.
4. Configure the APIPhy PMR IO Control Register (address: 0xF802_20F0 and 0xF802_30F0)
   - Suggested value for external APSync mode: 0x2400.0000.0000.0019
5. Program the IAP Pattern Mask (address: 0xF802_2000 - 0xF802_2010 and 0xF802_3000 - 0xF802_3010)
   - In case the default value is not applicable, set identical values for both APIPhy Driver IAP Pattern Mask and APIPhy Receiver IAP Pattern Mask registers.
6. Program the APIPhy IO Control Register (address: 0xF802_20E0 and 0xF802_30E0)
   - Some suggested values, depending on system requirements/configuration:
     0x0000.0000.073A.0306
     0x0000.0000.2F2A.0306
     0x0000.0000.2F3A.0304
     0x0000.0000.2712.0304
7. Prepare the CPU
   - The attached processors continue through their POR sequence. During IPL2 (MASTER CFG), the service processor scans the appropriate mode ring values into the PowerPC 970xx processor.
   - Once the mode ring has been loaded, the IAP on each of the processor interfaces can begin.
8. Assert the driver on the CPC945 (address: 0xF802_2030 and 0xF802_3030)
   - Set wiap bit [62] in APIPhy Configuration 1 Register
   - The CPC945 IAP signature will be continuously transmitted by the processor physical interface so as to allow the attached processor to properly align its receivers. The patterns should not be stopped until the receiving side (PowerPC 970xx processor) has completed the alignment.
9. CPU executes its IAP.
   - The processor continues through IPL3 (MASTER SYNC) and IPL4 (MASTER IOSYNC) sequence.
   - The CPC945 can now begin synchronization of its processor interface receivers.
10. Enable the receiver on the CPC945 RIAP (address: 0xF802_2030 and 0xF802_3030)

- Set riap bit [63] in APIPhy Configuration 1 Register.

11. Wait for IAP to complete on the CPC945
    - Wait time of approximately 80 ms.
12. Check the PI training status
    - Read the APIPhy Status 0 Register (address: 0xF802_2050 and 0xF802_3050).
    - riap_done bit [1] should be set - indicates completion.
    - rcv_status[48:63] should be all zero - indicates IAP completed without error.
13. Stop the training sequence
    - Deassert wiap and riap bits 62:63 in APIPhy Configuration 1 Register (address: 0xF802_2030 and 0xF802_3030).
    - Deassert the wiap and riap bits on the PowerPC 970xx.
    - Assuming that the IAP has completed without error on both the PowerPC 970xx and CPC945, the target cycles that do not result in an IAP error are noted. Note there is typically only 2 or 3 latencies, depending on the frequency of the PI bus clock and PCB trace delays.

4.1.1 Determining PI Bus Parameters

This section describes a typical procedure for determining valid values for the following PI bus parameters:

• PAAM Window	(APIPaamWin 0xF803_0100)
• Snoop Window	(APISnoopWin 0xF803_0110)
• I/O Snoop Window	(APIIOSnoopWin 0xF803_0120)
• Status Latency	(APIStatLat 0xF803_0130)
• Snoop Latency	(APISnoopLat 0xF803_0140)

In general, the values programmed into these registers depend on the system (length of PCB traces, clock speed of the bus, etc.) and are usually obtained empirically through a trial and error approach. This section also describes the relevant bus parameters on the CPU side.

All values described here are in decimal, unless otherwise stated.

The example in *Figure 4-2* (for illustrative purposes only) shows the relationship between the various bus parameters.

1. Determine working CPU StatLat.

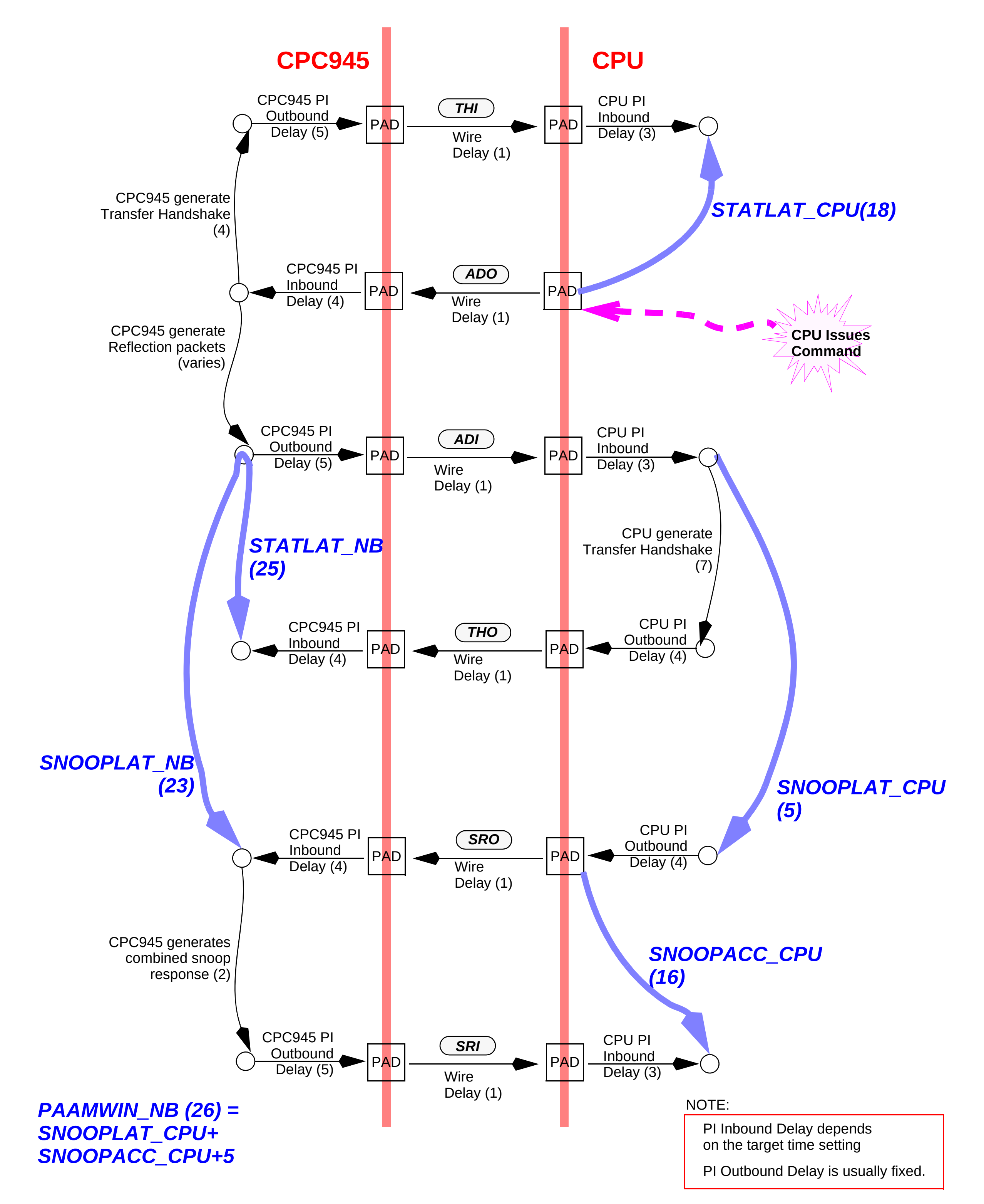
    a. Set APIStatLat Register to a value of 22. This is just a recommended starting point.
    b. Set the target cycles for the CPU and CPC945.
    c. Power-on reset (POR) the system and issue the first fetch.
    d. Start with a CPU StatLat of 8 and attempt one CPU StatLat value per POR attempt.
    e. Increase by one until the TH Null error is no longer present (scom 0xA0001, bit[49]).
    f. Note that there is only one valid value per target cycle setting.
    g. If the error still persists in step 1e, then select a different target time combination in step 1b and repeat the procedure.
    h. Record all working combinations of CPU target cycle, CPC945 target cycle and CPU StatLat values.

2. Determine working APIStatLat.

    a. Work with the values obtained in step 1h.
    b. POR the system and issue the first fetch.
    c. Starting with APIStatLat value of 16 and attempt one APIStatLat value per POR attempt.
    d. Before finishing POR, clear the Adi0Excp and Adi1Excp bits in the CPC945 APIExcp Register 0xF80300A0.

e. Increase APIStatLat by 1 until no error is recorded in CPC945 APIExcp Register 0xF8030060.
f. Note that there is only one valid value per target cycle setting, but the system will function properly whether APIStatLat is set correctly or not.
g. Record all working combinations for CPU target cycle, CPC945 target cycle, CPU StatLat and APIStatLat.

Figure 4-2. Processor and CPC945 Interface Timing Parameters


CPC945
CPU
CPC945 PI Outbound Delay (5)
PAD
THI
Wire Delay (1)
PAD
CPU PI Inbound Delay (3)
CPC945 generate Transfer Handshake (4)
STATLAT_CPU(18)
CPC945 PI Inbound Delay (4)
PAD
ADO
Wire Delay (1)
PAD
CPU Issues Command
CPC945 generate Reflection packets (varies)
CPC945 PI Outbound Delay (5)
PAD
ADI
Wire Delay (1)
PAD
CPU PI Inbound Delay (3)
STATLAT_NB (25)
CPU generate Transfer Handshake (7)
CPC945 PI Inbound Delay (4)
PAD
THO
Wire Delay (1)
PAD
CPU PI Outbound Delay (4)
SNOOPLAT_NB (23)
SNOOPLAT_CPU (5)
CPC945 PI Inbound Delay (4)
PAD
SRO
Wire Delay (1)
PAD
CPU PI Outbound Delay (4)
CPC945 generates combined snoop response (2)
SNOOPACC_CPU (16)
CPC945 PI Outbound Delay (5)
PAD
SRI
Wire Delay (1)
PAD
CPU PI Inbound Delay (3)
PAAMWIN_NB (26) = SNOOPLAT_CPU+ SNOOPACC_CPU+5
NOTE:
PI Inbound Delay depends on the target time setting
PI Outbound Delay is usually fixed.

3. Determine working APISnoopLat
   a. Work with the values obtained in step 2g.
   b. Fix CPU SnoopLat to minimum allowed value: PPC970FX = 5, PPC970MP = 6.
   c. POR the system and execute code such that a line is modified in the L2. This can be done by using a store or **dcbz** instruction.
   d. Start with APISnoopLat value of 16.
   e. Use the CPC945 System Command Registers (0xF803_0200 - 0xF803_0210) to drive in a read snoop from CPC945.
   f. Check snoop response received in CPC SysCmdStat Register 0xF803_0220.
   g. Increase APISnoopLat until a Retry snoop response is recorded.
   h. Note that there is only one valid value per target cycle setting.
   i. Record all the working combinations for CPU target cycle, CPC945 target cycle, CPU StatLat and APIStatLat, CPU SnoopLat and APISnoopLat.
4. Determine working CPU SnoopAcc
   a. Work with the values obtained in step 3i.
   b. POR the system and execute code such that a line is modified in the L2. This can be done by using a store or **dcbz** instruction.
   c. Start with CPU SnoopAcc value of 5.
   d. Access the modified line in cache with a cache-inhibited read. This is I-bit aliasing and causes a Retry snoop response.
   e. Increase CPU SnoopAcc until code completes, and the I = ‘1’ access gets the correct data and there are no L2 or BUS FIR bits set.
   f. Note that there is only one valid value per target cycle setting.
   g. Record all the working combinations for CPU target cycle, CPC945 target cycle, CPU StatLat and APIStatLat, APISnoopLat, CPU SnoopLat and CPU SnoopAcc.
5. Set APIPaamWin to a value of CPU SnoopLat + CPU SnoopAcc + 5 for combinations obtained in 4g.
6. Set APISnoopWin and APIIOSnoopWin to the same value as the CPU Compace, which is usually 4.
7. Determining stable values.
   a. Start with the values obtained in step 4g, 5, and 6.
   b. Run extended tests on different parts (fast, slow, normal) with some temperature variation.
   c. Record the stable combinations for CPU target cycle, CPC945 target cycle, CPU StatLat and APIStatLat, APISnoopLat, CPU SnoopAcc, CPU SnoopLat, APIPaamWin, APISnoopWin, APIIOSnoopWin and CPU Compace.
   d. Choose a combination from 7c that works for the desired frequency and temperature range and program these into the respective registers.
   e. Note that APIPaamWin and APIIOSnoopWin might need minor corrections to get better system performance.

4.1.2 Error Register Information

Register	Bit	Debug Description/Information
L2 FIR (SCOM 0x40000)	50	Unexpected data return 1. Phase of CPC945 might be off. 2. PID might be set incorrectly. 3. CPU StatLat might be 1 too large. 4. Logical CPU StatLat maximum has been hit.
	58	Illegal CRESP, SnoopAcc is off by 1.
	60	PAAM error, PaamWin has been calculated incorrectly.
BUS FIR (SCOM 0xA0001)	47	CPU detects parity error, CPU target cycle not stable at this frequency, bus ratio.
	48	CPC945 detects parity error, CPC945 target cycle not stable at this frequency, bus ratio.
	49	TH Null, adjust CPU StatLat until correct.
	51	Illegal CRESP, SnoopAcc is off by 1.
	52	CPU detects parity error, CPU target cycle not stable at this frequency, bus ratio.
	57	PAAM error, PaamWin has been calculated incorrectly.
	59:63	Tag of command associated with error.
CPC945 APIExcp Register (0xF80300A0)	1	StatLat error on PI port 0.
	2	StatLat error on PI port 1.
CPC945 APIPhySTAT0 (0xF8022050, 0xF8023050)	0	CPC945 detects parity error, CPC945 target cycle is not stable at this frequency, bus ratio.

4.1.3 Additional Debug Information

1. Coherency issues most likely indicate that CPU SnoopAcc is off by more than 2.
2. I/O, DMA errors indicate that CPC945 SnoopLat is incorrect.
3. Additional I-bit aliasing information (SnoopAcc test).

   a. CPC945 SnoopLat wrong
      - CPC945 sees Null, returns the wrong data.
      - CPC945 sees Intervention, CPC945 waits for intervention data and hangs.
      - CPC945 sees Retry, CPC945 aborts operation, does write, waits for new operations.

   b. CPU SnoopAcc / CPC945 SnoopLat (Null)
      - CPU sees Null, gets wrong data.
      - CPU sees Intervention, pulls L2 FIR[58].
      - CPU sees Retry, gets wrong data and pulls L2 FIR[50].

   c. CPU SnoopAcc / CPC945 SnoopLat (Retry)
      - CPU sees Null, hang.
      - CPU sees Intervention, pulls L2 FIR[58].
      - CPU sees Retry, works properly.

4.1.4 API Programming Procedure

The core interface registers in general configures the size of the various queues for each requester (CPU and I/O). The following procedure represent a typical programming sequence for the core interface registers.

1. APIProcCmd Register (0xF803_0000)
    a. ApiPortSel [30] should be left as it was.
    b. ApiEn [29] must be 0.
    c. MinGuarP0Cmd [5:6] should be between 1 and 3.
    d. MinGuarP1Cmd [9:10] should be between 1 and 3, where relevant.
    e. MinGuarP2Cmd [7:8] should be between 1 and 3, where relevant.
    f. MinGuarP3Cmd [11:12] should be between 1 and 3, where relevant.
    g. Calculate the sum of (MinGuarP0Cmd[5:6] + MinGuarP1Cmd[9:10] + MinGuarP2Cmd[7:8] + MinGuarP3Cmd[11:12] + 3).
    h. If the value in step g) is greater than 7, then NumProcCmd[0:4] should be between this value and the maximum value of 16. If the value is less than 7, NumProcCmd[0:4] should be 7.
    i. PE128WrKill [13] is 0 or 1 depending on requirements.
    j. CQNoFP [15] is 0 or 1 depending on requirements.
    k. DartDebugHalt[16] must be 0.
2. APIIOPnd Register (0xF803_0010)
    a. SizPcRPndQ[0:3] should be between values of 1 and 8 depending on requirements.
    b. SizPcWPndQ[4:7] should be between values of 1 and 8 depending on requirements.
    c. SizHtRPndQ[8:11] should be between values of 1 and 8 depending on requirements.
    d. SizHtWPndQ[12:15] should be between values of 1 and 8 depending on requirements.
3. APICmdArb Register (0xF803_0020)
    a. ArbWtProc [0:1] should be between values of 0 and 3 depending on requirements.
    b. ArbWtPcW [2:3] should be between values of 0 and 3 depending on requirements.
    c. ArbWtHtW [4:5] should be between values of 0 and 3 depending on requirements.
    d. DynArbWtEn [6] should be 1.
    e. DynArbWtCnt [7:10] should be 8.
    f. DynArbWtProc [11:12] should be 3.
    g. PcieWtrqEn [13] should be left at their default values.
    h. HtWtrqEn [14] should be left at their default values.
    i. DisArbProcTOLmt should be 1.
    j. ArbProcTOLmt [16:31] should have a value greater than 100.
4. APITRqGuar Register (0xF803_00D0)
    a. This register needs to be programmed before the APITRqCfg register gets programmed.
    b. MinGuarIOHtWTRqQ [0:1] should be between 1 and 3 depending on requirements. Then, MinGuarProcHtWTRqQ [2:3] should be between 1 and (4 - MinGuarIOHtWTRqQ).

   c. MinGuarIOHtRTRqQ [4:5] should be between 1 and 3 depending on requirements.Then, MinGuarProcHtRTRqQ [6:7] should be between 1 and (4 - MinGuarIOHtWRTRqQ).
   d. MinGuarIOPCWTRqQ [8:9] should be between 1 and 3 depending on requirements. Then, MinGuarProcPcWTRqQ [10:11] should be between 1 and (4 - MinGuarIOPCWRTRqQ).
   e. MinGuarIOPcRTRqQ [12:13] should be between 1 and 3 depending on requirements. Then, MinGuarProcPcRTRqQ [14:15] should be between 1 and (4 - MinGuarIOPcRTRqQ).
   f. NumHtRTaiTrgt [16:19], NumHtWTaiTrgt [20:23], NumPcRTaiTrgt [24:27] and NumPcWTaiTrgt [28:31] should be between 1 and 4 depending on requirements.

5. APISnpSltCfg Register (0xF803_00E0)
   a. NumSnpSlts [0:3] should be between 6 and 12 depending on requirements.
   b. MinGuarProcSnpSlt [4:7] should be between 2 and (NumSnpSlts - 4). It has to be 4 less because the other minimum guarantees are all 1.
   c. MinGuarHtWrSnpSlt [8:11] should be between 1 and (NumSnpSlts - MinGuarProcSnpSlt - 3).
   d. MinGuarHtRdSnpSlt [12:15] should be between 1 and (NumSnpSlts - MinGuarProcSnpSlt - MinGuarHtWrSnpSlt -2)
   e. MinGuarPcWrSnpSlt [16:19] should be between 1 and (NumSnpSlts - MinGuarProcSnpSlt - MinGuarHtWrSnpSlt - MinGuarHtRdSnpSlt - 1).
   f. MinGuarPcRdSnpSlt [20:23] should be between 1 and (NumSnpSlts - MinGuarProcSnpSlt - MinGuarHtWrSnpSlt - MinGuarHtRdSnpSlt - MinGuarPcWrSnpSlt).
   g. SafeQCntDisable [24] should be 1 if any of the “size”, “number” or “guarantee” values of the API queues and buffers change from their default value.
   h. SnpWtrqBypsDis [25] is 0 or 1 depending on requirements.

6. APITRqCfg Register (0xF803_0030)
   a. SizeSyncTRqQ [0:3] should be between 4 and 8.
   b. SizePcWTRqQ [4:6] should be between (MinGuarIOPCWTRqQ[8:9] + MinGuarProcPcWTRqQ[10:11]) in the *APITRqGuar Register (0xF803_00D0)* and 4.
   c. SizePcRTRqQ [7:9] should be between (MinGuarProcPcRTRqQ[14:15] + MinGuarIOPcRTRqQ[12:13]) in the *APITRqGuar Register (0xF803_00D0)* and 4.
   d. SizeHtWTRqQ [10:12] should be between (MinGuarIOHtWTRqQ[0:1] + MinGuarProcHtWTRqQ[2:3]) in the *APITRqGuar Register (0xF803_00D0)* and 4.
   e. SizeHtRTRqQ [13:15] should be between (MinGuarIOHtRTRqQ[4:5] + MinGuarProcHtRTRqQ[6:7]) in the *APITRqGuar Register (0xF803_00D0)* and 4.
   f. SizeGcrTRqQ[16:17] should be between 1 and 2.
   g. EorSDisable[18] is 0 or 1 depending on requirements.

7. APITRspCfg Register (0xF803_0040)
   a. SizePcRspQ [0:3] should be between 4 and 8 depending on requirements.
   b. SizeHtRspQ [4:7] should be between 4 and 8 depending on requirements.
   c. SizeGcrRspQ [8:10] should be between 2 and 4 depending on requirements.

8. APIDtQCfg Register (0xF803_0050)
   a. SizePcRdDtQ [0:4] should be between 8 and 16 depending on requirements.
   b. SizePcWtDtQ [5:9] should be between 8 and 15 depending on requirements.
   c. SizeHtRdDtQ [10:14] should be between 8 and 16 depending on requirements.
   d. SizeHtWtDtQ [15:19] should be between 8 and 15 depending on requirements.
   e. SizeGcrRdDtQ [20:11] should be between 1and 2 depending on requirements.
   f. SizeGcrWtDtQ [22:23] should be between 1and 2 depending on requirements.
9. APIWdbCfg Register (0xF803_0060)
   a. MinGuarWdbP0Wr [6:7] should be 2 or 3 depending on requirements.
   b. MinGuarWdbP0Int [8:9] should be between 1 and 3 depending on requirements.
   c. MinGuarWdbP1Wr [10:11] should be 2 or 3 depending on requirements.
   d. MinGuarWdbP1Int [12:13] should be between 1 and 3 depending on requirements.
   e. MinGuarWdbP2Wr [14:15] should be 2 or 3 depending on requirements.
   f. MinGuarWdbP2Int [16:17] should be between 1 and 3 depending on requirements.
   g. MinGuarWdbP3Wr[18:19] should be 2 or 3 depending on requirements.
   h. MinGuarWdbP3Int[20:21] should be between 1 and 3 depending on requirements.
   i. NumWDB[0:5] should be between (MinGuarWdbP0Wr[6:7] + MinGuarWdbP0Int[8:9] + MinGuarWdbP1Wr[10:11] + MinGuarWdbP1Int[12:13] + MinGuarWdbP2Wr [14:15] + MinGuarWdbP2Int[16:17] + MinGuarWdbP3Wr[18:19] + MinGuarWdbP3Int[20:21]) and 32.
10. APIIntCfg Register (0xF803_0070)
    a. MinGuarIntP0 [8:9] should be 2 or 3 depending on requirements.
    b. MinGuarIntP1 [10:11] should be 2 or 3 depending on requirements.
    c. NumIntBfr0 [0:3] should be between (MinGuarIntP0[8:9] + MinGuarIntP1[10:11]) and 15.
    d. MinGuarIntP2 [12:13] should be 2 or 3 depending on requirements.
    e. MinGuarIntP3 [14:15] should be 2 or 3 depending on requirements.
    f. NumIntBfr1[4:7] should be between (MinGuarIntP2[12:13] + MinGuarIntP3[14:15]) and 15.
11. APIMemReqCfg Register (0xF803_0080)
    a. BypsDis [0] is either 0 or 1 depending on requirements.
    b. SizBypsQ [2:7] should be 32.
    c. SizMemSyncQ [25:27] should be 4.
    d. SizMemReq [8:12] should be between 5 and 16 if SafeQCnt is disabled. If SafeQCnt is enabled, it should be between 6 and 16.
    e. MinGuarMemRqProc [13:16] should be between 1 and (SizMemReq[8:12] - 4) if SafeQCnt is disabled. If SafeQCnt is enabled, it should be between 1 and (SizMemReq[8:12] - 5).
    f. MinGuarMemRqIO [17:20] should be between 1 and (SizMemReq[8:12] - MinGuarMemRqProc[13:16] - 3) if SafeQCnt is disabled. If SafeQCnt is enabled, it should be between 1 and (SizMemReq[8:12] - MinGuarMemRqProc[13:16] - 4).

   g. NumBypsCutOff [0:3] should be between 1 and (SizMemReq[8:12] - MinGuarMemRqProc[13:16] - MinGuarMemRqIO[17:20] - 2).

   h. NumAvlInMemReqQ [0:3] should be between (MinGuarMemRqProc[13:16] + MinGuarMemRqIO[17:20] + 4) and SizMemReq[8:12]. If the value is greater than or equal to 16, set the value to 15 instead.

12. APIMemRdCfg Register (0xF803_0090)

   a. MinGuarMemRd0Proc [10:12] should be between 3 and 7 depending on requirements.

   b. MinGuarMemRd1Proc [13:15] should be between 3 and 7 depending on requirements.

   c. NumMemRdBfrA[0:4] should be between (MinGuarMemRd0Proc[10:12] + MinGuarMemRd1Proc[13:15]) and 16.

   d. MinGuarMemRd2Proc [16:18] should be between 3 and 7 depending on requirements.

   e. MinGuarMemRd3Proc [19:21] should be between 3 and 7 depending on requirements.

   f. NumMemRdBfrB[0:4] should be between (MinGuarMemRd2Proc[16:18] + MinGuarMemRd3Proc[19:21]) and 16.

   g. ApiMemDly [22:25] should match the value in the ApiRdTgDelay field in the DDR2 MemBusCnfg2 register.

   h. RdTgQSrchLmt [26:29] should be left with the default value.

   i. MemRdFastPathEn [30] should be 0 or 1 depending on requirements.

13. APIMask0 Register (0xF803_00B0)

   a. Individual exception enable bits are set depending on requirements.

14. APIMask1 Register (0xF803_00C0)

   a. Individual exception enable bits are set depending on requirements.

15. Enable the API interface.

   a. In the APIProcCmd Register (0xF803_0000), set ApiEn [29] bit to 1 and leave all other bits as programmed previously.

4.1.5 Configuring for Single PI Port Usage

Assuming that port0 is being used and port1 is unused, and it is configured for external APSync mode:

1. Allow the PI port1 registers (0xF802_3000 to 0xF802_3FFF) to power up to their default reset state.
2. Write zeros to all register bits except the APsyncRcv_en bit [22] (set to 1) in the port1 APIPhy Configuration 1 Register (0xF802_3030).
3. Write zeros to all register bits in the port1 APIPhy IO Control Register (0xF802_30E0).
4. In the port1 APIPhy PMR IO Control Register (0xF802_30F0), set APsync_ovr bit [0] to 1, clear APsyncDrvEnable to 0 and leave all the other bits in their default settings.
5. Clear the Adi1 Mask0 bit [2] to zero in the APIMask0 Register (0xF803.00B0) to prevent the assertion of the Check stop signal.
6. Clear the Adi1 Mask1 bit [2] to zero in the APIMask1 Register (0xF803.00C0) to prevent the assertion of the chip fault signal.

During IAP, do not initiate any training sequence for port1.

Since API_QREQ2 and API_QREQ3 are active low inputs, these need to be tied to 0 on the board, so that power tune operations can proceed with the other attached processor when it needs to.

In the API core, for the purpose of optimizing the buffer allocation:

1. In the Api Proc Command Slot Configuration register (0xF803_0000), set the “minimum guaranteed cmd proc2” and “minimum guaranteed cmd proc3” values to its minimum value of 1.
2. In the Api Write Data Buffer Configuration register (0xF803_00060), set the “minimum guaranteed for proc2 writes”, “minimum guaranteed for proc2 interventions”, “minimum guaranteed for proc3 writes” and “minimum guaranteed for proc3 interventions” to their minimum values.
3. In the Api Intervention Buffer Configuration register (0xF803_0070), set the “number Intervention buffers for Port1” to a value of 2 and set the “minimum guaranteed intervention Bfr Proc2” and “minimum guaranteed intervention Bfr Proc3” to their minimum values.
4. In the Api Memory Read Configuration register (0xF803_0090), set the “minimum guaranteed MemRead proc2” and “minimum guaranteed MemRead proc3” to their minimum. values.
5. In the Api Exception Mask 0 Register (0xF803_00B0) write the ADI1_Mask0 bit to zero.
6. In the Api Exception Mask 0 Register (0xF803_00C0) write the ADI1_Mask1 bit to zero.

4.1.6 Internal versus External APSync

The difference between the two modes for APsync in the northbridge (NB) (internal versus external) is which chip is generating the APsync for all of the chips in the system (processors and NB). In internal APsync mode, the NB generates APsync for itself and the processors in the system. In external APsync mode, the system clock chip that generates the reference clocks for the processors and the NB generates the APsync.

The circuit within NB that creates the internal timezeros for NB will sample a localized (NB)APsync (as opposed to the system wide APsync) that the NB power manager unit generates to create the timezeros. The NB power manger generates this localized (NB)APsync from a circuit but manages its phase relationship differently depending if the system is programmed to be in internal APsync mode versus external APsync mode.

If the system is programmed to be in internal APsync mode, the localized (NB)APsync is generated by the NB power manager and sent to the timezero generator within NB and driven as an output from NB through the APsync0 and APsync1 pins. These APsync0 and APsync1 pins must be connected to the APsync inputs of the respective processors in the system.

If the system is programmed to be in external APsync mode, the APsync0 pin becomes an input to NB. The APsync0 pin must be connected to the APsync output of the system clock chip. The NB power manager unit still generates a localized (NB)APsync for internal timezeros in NB but this time it samples the APsync input to NB and locks the localized (NB)APsync phase to the sampled APSync that is driven by the system clock chip.

APsync is required to be seen by all chips in the system at the same time so that all chips generate their respective timezeros at the same time.

In internal APsync mode, NB power manager generates (NB)APsync on one sysclock edge and the NB internal timezero generator creates the timzeros for NB the next sysclock edge. To adhere to the requirement that APsync must be seen by all chips in the system at the same time, this means that there is at most 1 sysclock delay allowed to send the APsync out of NB, through the APsync board traces, and into the processors for sampling.

As systems start running at higher frequencies, timing analysis shows that there will not be enough time for APsync to leave NB and reach the processors within 1 sysclock. We would run into a case, where the NB sampled the APsync 1 cycle it was generated to generate the timezeros while the processors sampled APsync more than 1 cycle after it was generated to generate its timezeros. This is a scenario where all chips on the system were seeing different timezeros. For this reason, external APsync mode was created.

If an external clock chip generated APsync and the APsync traces between the clock chip and the other chips in the system (NB, processors) were matched, then this timing requirement of 1 sysclock delay would not be needed. The only requirement would be that the traces are matched and could be as long as reasonably required and it becomes a more traditional clock insertion delay issue like all other reference clock routes. If the traces were such that they were say 5 sysclocks long, given that the APsync traces are matched, all chips would see the first rising edge of the APsync at the same 5 sysclocks after it was launched from the clock chip.

Although the APSync could be launched from NB ‘early’ by a programmable amount to allow for operation where the APSync propagation delay was longer than 1 sysclock cycle, this could potentially introduce complexities when the propagation delay is approximately on a sysclock cycle boundary.

Note: Only the external APSync mode is recommeded for CPC945 systems.

4.2 Processor Interface Endian Order

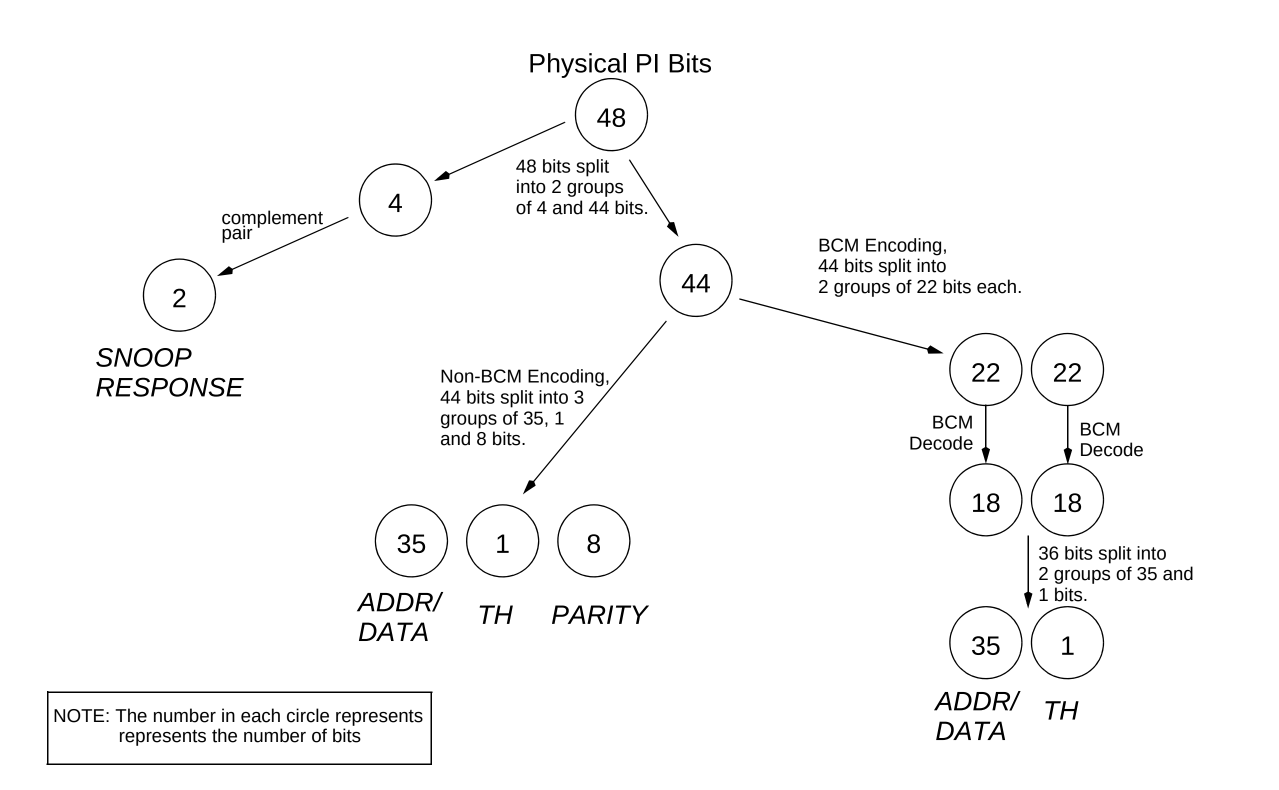
The processor interface is defined with big-endian byte ordering while the PCIe and HyperTransport buses are defined with little-endian byte ordering. The CPC945 performs the appropriate byte swapping and address mapping to support the differences. Byte addressing is similarly numbered in conflicting order on either side of the bridge. *Table 4-1* on page 89 describes the mapping of the processor interface to the PCIe bus addresses in big-endian mode for both memory and I/O accesses.

Table 4-1. Big-Endian Processor Interface to Little-Endian PCIe Bus Address Mapping.

Processor Interface Address Bit	PCIe Address Bit	Comment
32	31	
33	30	
. . .	. . .	
60	3	
61	2	
62	1 = ‘0’	Linear incrementing mode. These 2 bits are encoded into byte enables on the PCIe bus to select which bytes are involved in the transaction.
63	0 = ‘0’	

4.3 Processor Interface Balanced Encoding

Figure 4-3. Physical PI Bits to Logical PI Bits Mapping


Physical PI Bits
48
48 bits split into 2 groups of 4 and 44 bits.
4
complement pair
2
SNOOP RESPONSE
44
BCM Encoding, 44 bits split into 2 groups of 22 bits each.
Non-BCM Encoding, 44 bits split into 3 groups of 35, 1 and 8 bits.
35
1
8
ADDR/ DATA
TH
PARITY
22
22
BCM Decode
BCM Decode
18
18
36 bits split into 2 groups of 35 and 1 bits.
35
1
ADDR/ DATA
TH
NOTE: The number in each circle represents represents the number of bits

The processor interface provides a feature that reduces current-spike noise on the voltage supplies that power the processor bus. The balanced encoding mode of operation ensures that an equal number of zeros and ones are always present on the processor address and data buses for every transaction. This reduces the switching current variations in the bus drivers resulting in a more stable power supply and better signal quality. The feature is controlled by bit [31] of *Section 12.9.1 API Proc Command Slot Configuration Register (APIProcCmd)* on page 409. When this bit is set to ‘0’, the outgoing encoders and incoming decoders are turned on and the 36 bits of data and address and a generated parity bit are encoded into an equal number of high and low bits on the 44 address and data lines as illustrated in *Figure 4-3*. The receiving agent, which must also have its encoding enabled, converts the 44 signals back into the original 36 data bits. When this bit is set to ‘1’, the encoders and decoders are disabled, and the transfer-handshake and parity bits are sent alongside the data. The balanced coding feature allows the processor interface signals to have a higher signal-to-noise ratio with less chance of edge-alignment interference from spike-induced noise on the power supply line.

4.4 Bus Snoops and Coherency

Memory cache coherency is maintained through a global snooping mechanism. The CPC945 processor interface reflects inbound command packets back to the second processor to allow it to snoop the address. The CPC945 does not provide any coherency mechanism for memory located on the PCIe or HyperTransport buses. Any need to cache memory located on those buses requires a software coherency mechanism to ensure proper operation.

4.5 Processor Interface Slave Transactions

As a processor interface slave, the CPC945 acknowledges all transactions initiated on the processor interface. The CPC945 is also the source of the transfer-handshake response signal. The processor interface implements a 35-bit address and data (AD) bus segment, a one-bit transfer-handshake bus segment, and a 2-bit snoop-response bus segment in each direction per connected processor. A beat in the processor interface in the CPC945 is equal to 32 bits of data. Because data is transferred on both edges of the clock on the processor interface, two beats will take a full bus clock cycle. The CPC945 is capable of completing the following data transfer transactions:

- 1 to 8-byte, 2-beat read
- 1 to 8-byte, 2-beat write
- 16-byte, 4-beat read (aligned)
- 16-byte, 4-beat write (aligned)
- 32-byte, 8-beat read (critical word first)
- 32-byte, 8-beat write (critical word first)
- 64-byte, 16-beat read (critical word first)
- 64-byte, 16-beat write (critical word first)
- 128-byte, 32-beat read (critical word first)
- 128-byte, 32-beat write (critical word first)

The CPC945 can also forward address-only cycles by reflecting the address (no data is transferred in an address-only cycle).

When the CPC945 receives a synchronization cycle from the processor, it stalls new memory accesses until store operations issued before the synchronization cycle are complete. When the CPC945 receives an enforce in-order execution of I/O (EIEIO) cycle from a processor, it accepts and propagates the transaction toward memory-mapped I/O storage. It does not allow any cache-inhibited storage accesses to bypass the transaction. The CPC945 conducts the cache-to-cache data intervention cycles that are defined for the processor interface protocol. *Table 4-2* lists the transfer types defined for the processor interface and indicates how the CPC945 responds to each of the given types. If the CPC945 receives a transaction with an unsupported transfer type and transfer size combination, an error is returned during the data phase.

Table 4-2. CPC945 Responses to PI Transfer Types. (Sheet 1 of 2)

Address Modifiers (WIMGRP)	Transfer Type (TT[0:4])	Bus Operation	Class of Operation	CPC945 Response
XXMXRP	00000	Clean	Normal	Address only transaction
WIMXRP	00010	Write with Flush	Normal	Write indicated to address
XXMXRP	00100	Flush	Normal	Address only transaction
WXM0RP	00110	Write with Kill	Normal	Write indicated to address
WXM1RP	00110	Write with Clean	Normal	Write indicated to address
XXMXRP	01000	SYNC	Normal	Address only synchronization function applied

Note:
W = write through, M = memory coherent, N = intervention, A = atomic, R = rerunning, I = cache inhibited, S = noncaching coherent read, P = pipelined snoops, G = guarded read, X = drive ‘0’ when driving signal and don’t care when receiving the signal.

Table 4-2. CPC945 Responses to PI Transfer Types. (Sheet 2 of 2)

Address Modifiers (WIMGRP)	Transfer Type (TT[0:4])	Bus Operation	Class of Operation	CPC945 Response
NSMGRP	A1010	Read	Normal	Read of indicated address
XXMXRP	01100	Data Kill (DKill)	Normal	Address only transaction
NXMXRP	A1110	Read with Intent to Modify (RWITM)	Normal	Read of indicated address
XXMX0P	10000	EIEIO	Normal	Address only EIEIO function applied
XXMXXX	11100	Reserved	Error	Address only transaction
XXMX0P	11000	Translation Lookaside Buffer Invalidate Entry (TLBIE)	Normal	Address only transaction
XXMXXX	10100	Reserved	Error	Address only transaction
XXMXRP	00001	LARX-Reserve	Normal	Address only transaction
XXMXRP	A0011	Data Line Claim Transaction (DClaim)	Normal	Address only transaction
XXMXXX	001X1	Reserved	Error	Address only transaction
XXMXRP	01001	Translation Lookaside Buffer Synchronization Instruction (TLBSYNC)	Normal	Address only transaction
XXMXXX	01X11	Reserved	Error	Address only transaction
XXMX0P	01101	Instruction Kill (IKill)	Normal	Address only transaction
XXMXXX	10001	Reserved	Error	Address only transaction
XXMXXX	10010	Reserved	Error	Address only transaction
XXMXRP	10101	Deallocate Directory Tag	Normal	Address only transaction
XXMXXX	1011X	Reserved for customers	Error	Address only transaction
XXMXXX	110X1	Reserved	Error	Address only transaction
XXMX0P	11101	Rerun	Normal	Address only transaction
XXMX0P	11111	Null	Normal	Null

Note:
W = write through, M = memory coherent, N = intervention, A = atomic, R = rerunning, I = cache inhibited, S = noncaching coherent read, P = pipelined snoops, G = guarded read, X = drive ‘0’ when driving signal and don’t care when receiving the signal.

4.6 Processor Interface Master Transactions

All transactions on the processor interface bus are packet based. As a processor interface master, the CPC945 reflects all transactions to the processors to maintain memory coherence when there is an access to system memory. Accesses to memory or I/O address space that come from the PCI Express or from HyperTransport devices are reflected on the processor interface. Inter-HyperTransport accesses are not reflected onto the processor interface and remain on the HyperTransport bus. See *Section 3.4* on page 53 for a description of the DMA re-mapping logic.

4.7 TEA, DERR, Checkstop

4.7.1 Transfer Error Acknowledge

A master abort or a target abort on a PCIe bus would normally result in single transfer error acknowledge (TEA) that is passed back to the processor. The processor interface bus on the CPC945 does not have a TEA signal. Transfer errors are returned to the processor as a DERR in bit 34 of the first data beat on the processor interface. The DERR indicates an off-bus data error has occurred, and the full data transfer is not valid. An internal bus TEA that is propagated to the processor interface DERR causes either an exception or a checkstop. See *Section 4.7.2 Data Error Signal and Checkstop* on page 93 for details.

A TEA is only generated in response to a read request. Write data that encounters an abort is simply discarded.

Any TEA generated by the PCIe master that is produced during a transfer results in signaling an error for the entire transfer. As a PCIe slave, the correct number of TEAs is accumulated to complete the transaction, and a target abort is returned on the PCIe bus.

Any TEAs generated by the HyperTransport receiver result in an error for the entire transfer. For a HyperTransport transmitter, any TEAs detected result in a HyperTransport response of nonexistent address (NXA).

4.7.2 Data Error Signal and Checkstop

During the data portion of a data packet, the DERR within the processor interface is used to validate the current data beat pair on the processor interface bus. When DERR is received, the CPC945 can be programmed to issue a checkstop, PI_CSTP, which can generate either an exception or a full stop of the processor.

Figure 4-4. Checkstop Connections

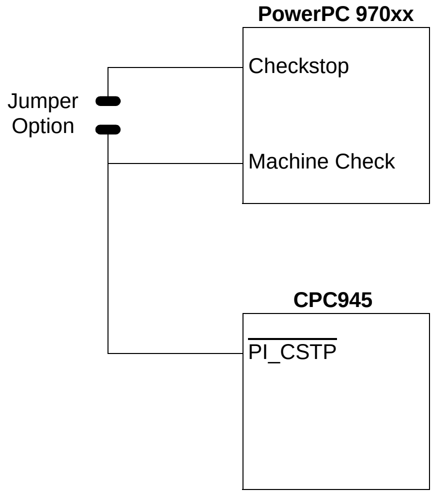


Figure 4-4 shows one possible way to implement the interconnections for the PowerPC 970xx pins: checkstop and machine check. The CPC945 checkstop pin, PI_CSTP, is connected to the processor's machine check pin. Depending on how the PowerPC 970xx processor is configured, an assertion of PI_CSTP generates an exception or issues a checkstop to halt the processor's operation. In *Figure 4-4*, two resistor pads are

placed on the printed circuit board to accept an optional 0-Ω jumper that can short two circuit paths. The resistor is not populated during manufacturing, but can be added for diagnostic purposes if the processor must be halted during a checkstop event or checkstopped for debug purposes. In the unpopulated state, a machine check is generated in the event of a checkstop assertion. If the jumper is added, then a checkstop assertion halts the processor, allowing the processor state to be examined through the I^2C or JTAG interfaces.

If the processor interface encounters an exception, the error is unrecoverable and the $\overline{\text{PI\_CSTP}}$ checkstop pin is asserted until the CPC945 is reset. The CPC945's *API Exception Register (APIExcp)* on page 423 can be read through the I^2C bus or from the processor to determine if the processor interface generated the exception. If the processor interface did generate the exception, the (APIExcp Register might only be readable through the I^2C bus (see *Section 12.9.11 API Exception Register (APIExcp)* on page 423). The PowerPC 970xx processor's machine check pin is an input-only signal, and the CPC945's checkstop pin should only be connected to the processor's machine check pin.

4.7.3 Additional System Exceptions

ECC memory error conditions are also logged in the PI exception register. With appropriate mask register settings, detected error conditions can be made to cause processor check stop and/or to assert a special chip fault pin ($\overline{\text{CHP\_FAULT\_N}}$). This external signal ($\overline{\text{CHP\_FAULT\_N}}$) can be used to signal an external interrupt to the processor or to external logic.

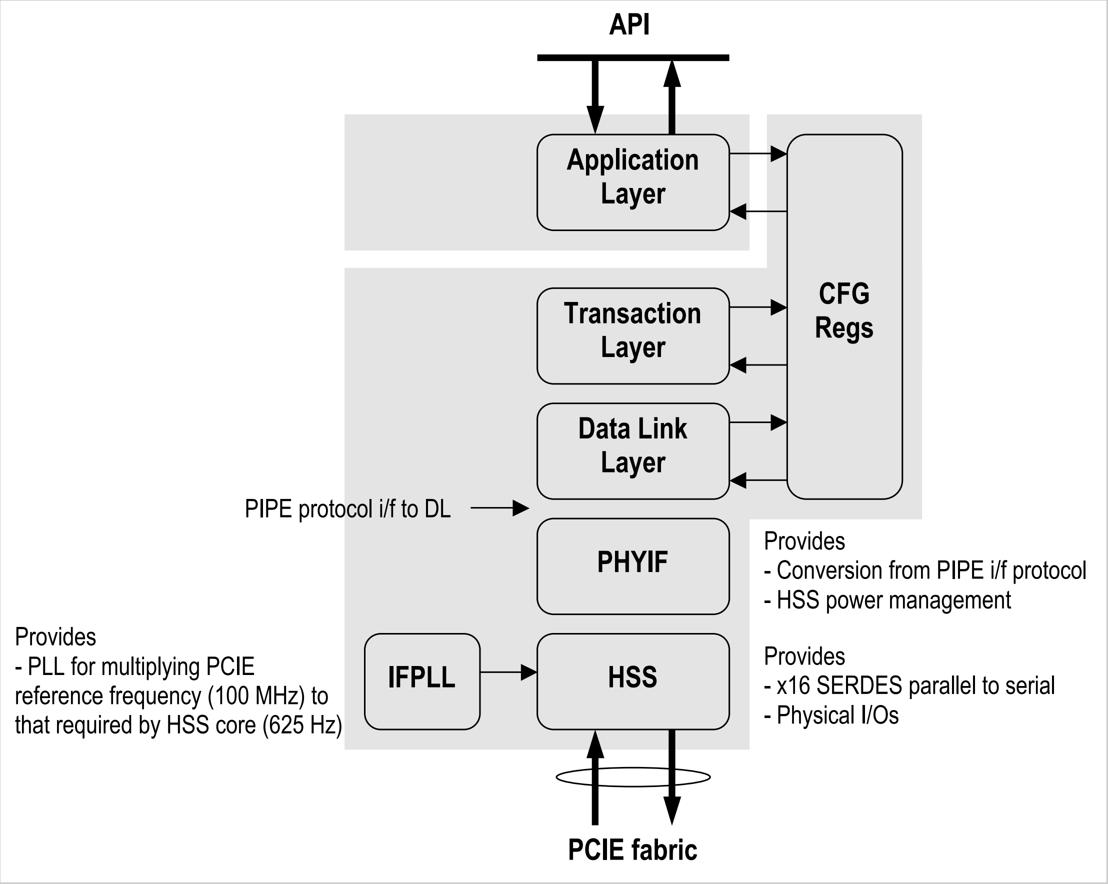
Settings in the PI Exception Mask Register (APIMASK) and the PI Exception Mask 1 Register specify what action are to be taken upon ECC detection. For more information, see *Section 12.9.11 API Exception Register (APIExcp)* on page 423, and *Section 12.9.13 API Exception Mask 1 Register (APIMask1)* on page 426.

5. PCI Express

5.1 Introduction

The CPC945 incorporates a single ×16 Link PCI Express Root Complex for interfacing into external components. This x16 interface is also referred to as a Root Port[1]. The overall PCI Express stack has 3 primary layers, referred to as the Transaction, Data Link, and Physical layers. The Base specification is very explicit as to the intended functionality contained in each of these layers. In addition to the stack layers depicted above, the CPC945 introduces an Application Layer (AL) to interface the PCIe stack into its backbone. The PCIe Root Complex as implemented in the CPC945 is depicted in *Figure 5-1*.

Figure 5-1. CPC945 PCI Root Complex High Level Block Diagram



The CFG Regs module provides all of the configuration register space that is required by PCIe. It interfaces into the Application, Transport, and Data Link layers. The PHYIF and HSS modules collectively provide the PCIe Physical layer functionality.

[1] The PCI Express specification allows for configurations with more than one Root Port in the Root Complex. Each Root port can be of any valid Link width and the entire set of Root ports is collectively referred to as a Root Complex. In the CPC945 implementation, there is a single Root Port that can be configured as either a x16, a x8, x4 or x1 link. Bifurcation is not supported.

The CPC945 Root Complex supports the following features.

- Compatible with the *PCI Express Base Specification 1.0a*.
- X16 Link operating as x16, x8, x4, or x1.
- Supports up to 8 outstanding outbound reads to the PCIe endpoint (for example, graphics) in 64-byte request mode, or 4 outstanding reads in 128-byte request mode.
- Outbound Posted writes limited only by Endpoint.
- Supports up to 16 incoming reads and 16 incoming (Posted) writes from the endpoint at max request size.
- 128 byte maximum request size.
  Note: The Max read_request field is not programmable and is set for 128 bytes. See *Section 12.11.1.3 PCI Express Capability Structure* on page 524 for details.
- 128 maximum payload size (that is, for reads or writes - matches read request size).
- Supports PCIe Active State Power Management (ASPM) L1 requests from endpoint.
- Supports PCIe Advanced Error Reporting as described in *“Advanced Error Reporting Extended Capability Structure” on page 538*.
- Master mode loopback for Link error debugging and system analysis.
- All reads to the memory controller, coherent and non-coherent, use Tags to uniquely identify (specific to the requester) the request. Completions for requests will be returned out of order and therefore the Tag is used to associate the Completion with the Request. The width of the Tag field is 6 bits.
- All writes to the memory controller, coherent and non-coherent, have their data written directly into an 8 cache line deep queue that resides in the memory controller (a cache line in the CPC945 is 128 bytes). The Requester has complete control of this write data interface. The write request for this transaction will only be issued to the PI interface (for coherent requests) or the memory controller interface (for non-coherent requests) concurrent with or after the last beat in the data transfer. Therefore there can be at most eight outstanding write transaction to the memory controller from a Requester (for example, PCIe) at any time.
- All coherent requests are issued synchronous to the PI clock domain. All data transfers (regardless of source or destination) over the PI, as well as non-coherent requests to the memory controller interface, are synchronous to the DDR2 clock domain.
- Both coherent and non-coherent write data transfers to the memory controller share the same write data bus.
- Both coherent and non-coherent read completion data transfers from the memory controller share the same completion data bus.

5.1.1 PCIe Registers

PCI Express (PCIe) register space consists of three separate groupings of registers: the PCIe Configuration Registers, the XBus Configuration Registers and the PCIe interface logic General Control Registers. For more information see, *“PCIe Configuration Registers” on page 505*. *“Configuration Register Access” on page 109* also discusses the six different methods of accessing the PCIe configuration registers. This section discusses DART, HT to PCIe transactions, and address mapping.

5.1.2 Addressing

The CPC945 responds to addresses on the Processor Interconnect and the PCIe Bus as a function of the PCIe Address Mask Register that exists in the PCIe XBus Configuration register space. See *Section 12.11.2.4 PCI Express 0 Address Mask Register* on page 556. This register identifies the address ranges that belong to the PCIe Bus. The PCIe Address Mask Register is also used by the PI core interface to determine which transactions are destined for the PCIe Bus, rather than the Processor Interconnect memory sub-system.

5.1.3 DART

Coherent PCIe DMA accesses are remapped using a DMA address relocation table (DART) so that I/O devices can transfer data directly to any memory location in the 36-bit extended memory map. The OS establishes entries in the DART when a buffer area is requested by a driver. The “physical” addresses the system passes back to the driver are then mapped by the DART when the device accesses memory via these addresses. The OS removes DART entries when the driver indicates the buffer area is no longer required. More details of the DART implementation in CPC945 are available in *Section 3.4 DMA Address Relocation Table (DART)* on page 53.

5.1.4 PCIe Bus I/O Space

The CPC945 uses two separate buses for expansion in a 36-bit extended address map: PCIe and HyperTransport (HT). These two expansion buses exist in the PCIe and HT bridge spaces within the memory map. These ranges are further allocated to allow all possible PCIe/HT Bus cycles from the processor.

5.1.5 CPC945 PCIe Bus Address Decoding

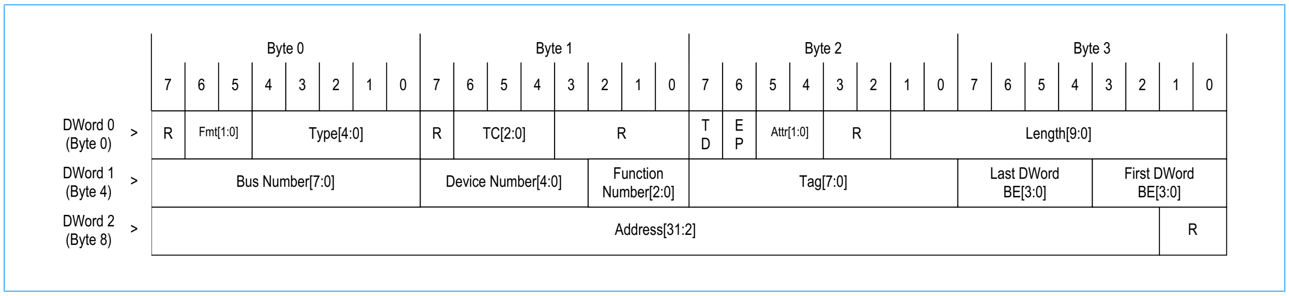
Processor Interconnect read or write accesses by the CPU to the PCIe bus memory space defined in *Table 12-1* on page 327 are serviced by the CPC945 and forwarded onto the PCIe Bus as memory read/write commands. The CPC945 passes the Processor Interconnect address to the PCIe bus address unmodified. The range of addresses responded to by the CPC945 from the PI is defined in *“PCI Express 0 Address Mask Register” on page 556*. This register defines fifteen coarse regions (256 MB) and sixteen fine regions (16 MB) of the 32-bit address space that CPC945 responds-to/ignores-from Processor Interconnect and PCIe Bus. The CPC945 implements a decode/antidecode scheme whereby, if a given address is decoded to be passed from the PI to the PCIe Bus, the same address is not allowed to pass from the PCIe Bus to the PI.

5.2 PCI Express Concepts

5.2.1 Transmit Layer Packet

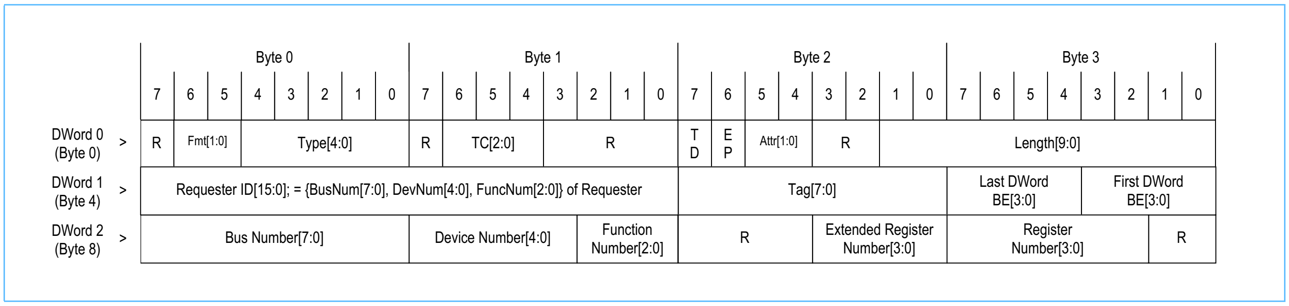
The basic construct of the PCIe Express packet is referred to as the *Transmit Layer Packet*, or TLP. The components of the Header within the TLP vary depending on whether the TLP is a nonposted request, a Posted request, or a Completion packet. The basic organization of the TLP Header for (Memory) Requests is shown in *Figure 5-2*, for Configuration Requests in *Figure 5-3*, and for Completions in *Figure 5-4*. Note that *Figure 5-2* details a 3 DWord Header, used for 32 bit addressing. For those packets requiring an address (for example, Posted), PCIe supports 32 bit and 64 bit addressing. From a TLP perspective, the difference is in an additional DWord in the Header for the upper 32 bits when 64 bit addressing is required. Only 32 bit addressing is allowed for outbound requests and therefore all outbound requests will use a 3 DWord Header, with the exception of the Message TLP which always uses a 4 DWord Header. For inbound requests (such as, from an Endpoint), either 3 or 4 DWord Headers can be received since any required page translation to map 32 bit virtual addressing into larger physical address space is performed by the Endpoint and not by the Root Complex. Note that all completions (outbound and inbound) are constrained to only use a 3 DWord Header. A description of the meaning and use of all indicated fields in the TLPs shown are presented in *Table 5-1*.

Figure 5-2. TLP Header for Requests with 32 Bit Addressing (3 DWord Header)



<table>
<tr><th></th><th></th><th colspan="8">Byte 0</th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th></tr>
<tr><td></td><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>DWord 0
(Byte 0)</td><td>></td><td>R</td><td colspan="2">Fmt[1:0]</td><td colspan="5">Type[4:0]</td><td>R</td><td colspan="3">TC[2:0]</td><td colspan="4">R</td><td>T
D</td><td>E
P</td><td colspan="2">Attr[1:0]</td><td colspan="2">R</td><td colspan="10">Length[9:0]</td></tr>
<tr><td>DWord 1
(Byte 4)</td><td>></td><td colspan="8">Bus Number[7:0]</td><td colspan="5">Device Number[4:0]</td><td colspan="3">Function Number[2:0]</td><td colspan="8">Tag[7:0]</td><td colspan="4">Last DWord BE[3:0]</td><td colspan="4">First DWord BE[3:0]</td></tr>
<tr><td>DWord 2
(Byte 8)</td><td>></td><td colspan="30">Address[31:2]</td><td colspan="2">R</td></tr>
</table>

Figure 5-3. TLP Header for Configuration Requests (reads and writes)



<table>
<tr><th></th><th></th><th colspan="8">Byte 0</th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th></tr>
<tr><td></td><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>DWord 0
(Byte 0)</td><td>></td><td>R</td><td colspan="2">Fmt[1:0]</td><td colspan="5">Type[4:0]</td><td>R</td><td colspan="3">TC[2:0]</td><td colspan="4">R</td><td>T
D</td><td>E
P</td><td colspan="2">Attr[1:0]</td><td colspan="2">R</td><td colspan="10">Length[9:0]</td></tr>
<tr><td>DWord 1
(Byte 4)</td><td>></td><td colspan="16">Requester ID[15:0]; = {BusNum[7:0], DevNum[4:0], FuncNum[2:0]} of Requester</td><td colspan="8">Tag[7:0]</td><td colspan="4">Last DWord BE[3:0]</td><td colspan="4">First DWord BE[3:0]</td></tr>
<tr><td>DWord 2
(Byte 8)</td><td>></td><td colspan="8">Bus Number[7:0]</td><td colspan="5">Device Number[4:0]</td><td colspan="3">Function Number[2:0]</td><td colspan="4">R</td><td colspan="4">Extended Register Number[3:0]</td><td colspan="6">Register Number[3:0]</td><td colspan="2">R</td></tr>
</table>

Figure 5-4. TLP Header for Completions

<table>
<tr><td></td><td></td><td colspan="8">Byte 0</td><td colspan="8">Byte 1</td><td colspan="8">Byte 2</td><td colspan="8">Byte 3</td></tr>
<tr><td></td><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>DWord 0
(Byte 0)</td><td>></td><td>R</td><td colspan="2">Fmt[1:0]</td><td colspan="5">Type[4:0]</td><td>R</td><td colspan="3">TC[2:0]</td><td colspan="4">R</td><td>T
D</td><td>E
P</td><td colspan="2">Attr[1:0]</td><td colspan="2">R</td><td colspan="10">Length[9:0]</td></tr>
<tr><td>DWord 1
(Byte 4)</td><td>></td><td colspan="16">Completer ID[15:0]; = {BusNum[7:0], DevNum[4:0], FuncNum[2:0]} of Completer</td><td colspan="3">Completion Status[2:0]</td><td>B
C
M</td><td colspan="12">Byte Count[11:0]</td></tr>
<tr><td>DWord 2
(Byte 8)</td><td>></td><td colspan="16">Requester ID[15:0]; = {BusNum[7:0], DevNum[4:0], FuncNum[2:0]} of Requester</td><td colspan="8">Tag[7:0]</td><td>R</td><td colspan="7">Lower Address[6:0]</td></tr>
</table>

Figure 5-5. TLP Header for Messages

<table>
<tr><td></td><td></td><td colspan="8">Byte 0</td><td colspan="8">Byte 1</td><td colspan="8">Byte 2</td><td colspan="8">Byte 3</td></tr>
<tr><td></td><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>DWord 0
(Byte 0)</td><td>></td><td>R</td><td colspan="2">Fmt[1:0]</td><td colspan="5">Type[4:0]</td><td>R</td><td colspan="3">TC[2:0]</td><td colspan="4">R</td><td>T
D</td><td>E
P</td><td colspan="2">Attr[1:0]</td><td colspan="2">R</td><td colspan="10">Length[9:0]</td></tr>
<tr><td>DWord 1
(Byte 4)</td><td>></td><td colspan="16">Requester ID[15:0]; = {BusNum[7:0], DevNum[4:0], FuncNum[2:0]} of Requester</td><td colspan="8">Tag[7:0]</td><td colspan="8">Message Code[7:0]</td></tr>
<tr><td>DWord 2
(Byte 8)</td><td>></td><td colspan="32" rowspan="2">(Fields in Bytes 8 through 15 depend on type of message)</td></tr>
<tr><td>DWord 3
(Byte 12)</td><td>></td></tr>
</table>

Table 5-1. TLP Header Fields

TLP Field	Description
Fields Common to all TLPs (See *Section 2.2.1, page 45 of the PCI Express Base Specification 1.0a* for more information.)	
R	These bits are reserved and must be filled with all 0's.
Fmt[1:0]	Format of the TLP header, as follows. 00 3 DWord header without data 01 4 DWord header without data (will never be generated by the AL) 10 3 DWord header with data 11 4 DWord header with data (will never be generated by the AL)
Type[4:0]	Indicates the type of transaction contained in the TLP. See *Section 2.2.1, page 47 of the PCI Express Base Specification 1.0a* for a complete description of type filed encodings.
TC[2:0]	Indicates the traffic class of the TLP transaction. Traffic Classes are not supported in the CPC945 implementation and this field will therefore always equal 000.
TD	Indicates the presence of a TLP Digest in the transmitted packet. TLP Digests are not supported in the CPC945 implementation and this field will therefore always equal 0.
EP	Indicates that the transmitted (or received) TLP contains "poisoned" (for example, bad) data; 0 Data payload is good 1 Data payload is bad.
Note: 1. I/O requests are not supported in the AL for inbound requests and will be returned as an unsupported request.	

Table 5-1. TLP Header Fields (Continued)

TLP Field	Description
Attr[1:0]	Indicates the Attributes of the TLP, as follows. Attr[1] = Relaxed Ordering indication for TLP. For the CPC945 implementation this bit will always = 0 (and will be forced to 0 if set to 1 on a received packet). 0 TLP must be compliant with PCI strong order rules. 1 TLP can use relaxed ordering rules as outlined in the PCI-X® specification. Attr[0] = No Snoop indication for TLP. Will always equal ‘1’ for outbound TLPs, but can equal ‘0’ or ‘1’ for inbound TLPs. 0 Hardware enforced cache coherency expected. 1 Hardware enforced cache coherency not expected. Attr[1:0] will always equal 00 for message TLPs.
Length[9:0]	Length of the data payload in DWords. For TLPs containing no data this field is undefined and must equal 0x000.
Fields Specific to Memory Requests[1] (See *Section 2.2.7, page 59 of the PCI Express Base Specification 1.0a* for more information.)	
Bus Number[7:0]	The primary bs number assigned to the root port in the PCI Express Type1 configuration space header (offset 0x18). This field, along with the device number and function number, is used to construct the requester ID for the root port. For the CPC945 implementation of the AL, the bus number will most probably be 0x00.
Device Number[4:0]	The device number used for Type 0 configuration accesses to the root port. This value can be extracted from bits 15:11 of the CONFIG_ADDRESS register used for software generation of configuration transactions (as performed in the CPC945) when the bus number in bits 23:16 matches the bus number at offset 0x18 in the AL’s configuration space header. This field, along with the Bus Number and Function Number, is used to construct the Requester ID for the root port. For the CPC945 implementation of the AL, the device number is fixed to 0x00.
Function Number[2:0]	The function number used for type 0 configuration accesses to the root port. This value can be extracted from bits 10:8 of the CONFIG_ADDRESS register used for software generation of configuration transactions (as performed in CPC945) when the bus number in bits 23:16 matches the bus number at offset 0x18 in the AL’s configuration space header. This field, along with the bus number and device number, is used to construct the requester ID for the root port. For the CPC945, the AL is a single function device and therefore the device number is fixed to 0x0.
Tag[7:0]	The Tag field of the packet header is an 8-bit value assigned to each nonposted requests issued by a requestor. (Note that posted requests never require a transaction Tag.) This value must be unique for all outstanding requests. When the requestor ID and tag are combined (in that order) they form the Transaction ID of the packet.
Last DWord BE[3:0]	The last double word byte enable field of the packet header contains the byte enables for the last double word referenced by a request. If the length field indicates a length of only one Double Word, this field must be filled with all 0’s. Byte enables are active high, and can be noncontiguous. BE[0] corresponds to byte 0 of the last double word, BE[1] corresponds to byte 1, etc.
First DWord BE[3:0]	The first double word byte enable field of the Packet Header contains the byte enables for the first (or only) Double Word referenced by a request. Byte enables are active high, and can be noncontiguous. BE[0] corresponds to byte 0 of the first double word, BE[1] corresponds to byte 1, etc.
Address[31:2]	The upper 30 bits of a 32-bit address targeting a 4 GB address space. (Note that the lower 2 bits of the address are not required, as all PCIe accesses are DWord oriented.)
Fields Specific to Configuration Requests (read and write) (See Section 2.2.7, page 59 of the PCI Express Base Specification 1.0a for more information.)	
Requester ID[15:0]	This 16-bit field is the concatenation of the BusNumber[7:0], DeviceNumber[4:0], and FunctionNumber[2:0] fields of the original requester of the transaction. For the CPC945, this field will always equal 0x0000.
Note: 1. I/O requests are not supported in the AL for inbound requests and will be returned as an unsupported request.	

Table 5-1. TLP Header Fields (Continued)

TLP Field	Description
Tag[7:0]	The Tag field of the packet deader is an 8-bit value assigned to each nonposted requests issued by a Requestor. (Note that posted requests never require a transaction Tag.) This value must be unique for all outstanding requests. When the requestor ID and tag are combined (in that order) they form the transaction ID of the packet.
Last DWord BE[3:0]	The last double word byte enable field of the packet header contains the byte enables for the last double word referenced by a request. Note that since all configuration requests are only ever one DWord in length, this field must always = 0x0 for configuration requests.
First DWord BE[3:0]	The first double word byte enable field of the packet header contains the byte enables for the first (or only) double word referenced by a request. Byte enables are active high, and can be noncontiguous. BE[0] corresponds to byte 0 of the first double word, BE[1] corresponds to byte 1, and so on.
Bus Number[7:0]	The bus number used for type 0 configuration accesses to the endpoint or type 1 accesses through a downstream bridge component. This field, along with the device number and function number, is used to direct the configuration access to its ultimate destination.
Device Number[4:0]	The device number used for type 0 configuration accesses to the endpoint or type 1 accesses through a downstream bridge component. This field, along with the bus number and function number, is used to direct the configuration access to its ultimate destination.
Function Number[2:0]	The function number used for type 0 configuration accesses to the endpoint or type 1 accesses through a downstream bridge component. This field, along with the bus number and device number, is used to direct the configuration access to its ultimate destination.
Extended Register Number[3:0]	The extended register number is used as the upper 4 bits of a 10-bit address to select one of 1024 dword registers in the PCIe configuration space. The lower 6 bits of this address are defined in the Register Number[5:0] field.
Register Number[5:0]	The Register Number is used as the lower 6 bits of a 10-bit address to select one of 1024 dword registers in the PCIe configuration space. The upper 4 bits of this address are defined in the Extended Register Number[5:0] field. Note that this field performs precisely the same function as the Register Number[5:0] field in the PCI configuration space and addresses the PCI compatible configuration space in PCIe is Extended Register Number[3:0] = 0000.
Fields Specific to Completions (See *Section 2.2.9, page 75 of the PCI Express Base Specification 1.0a* for more information.)	
Completer ID[15:0]	This 16-bit field is the concatenation of the BusNumber[7:0], DeviceNumber[4:0], and FunctionNumber[2:0] fields of the endpoint completing the transaction.
Completion Status[2:0]	The Completion Status filed indicates whether or not the target of the transaction was able to complete it successfully, as follows: 000 Successful Completion (SC) 001 Unsupported Request (UR) 010 Configuration Request Retry Status (CRS) 011 Reserved (should never occur) 100 Completer Abort (CA) 101 Reserved (should never occur) 11X Reserved (should never occur)
BCM	Byte count modified bit. This bit will never be set by PCI Express completers. It could only be set by PCI-X completers (that is, which would be on the secondary side of a PCIe to PCI-X bridge).
Byte Count[11:0]	The actual byte count remaining to complete the transaction. For completions that are completed within the current TLP this field will contain the total number of bytes contained within the TLP data payload. Since completions can be split across several TLPs (PCIe rules permitting), this field can be a greater value that the total number of returned bytes in the TLP data payload. Note that for the CPC945, all requests will never require more than 1 Completion TLP.
Requester ID[15:0]	This 16-bit field is the concatenation of the BusNumber[7:0], DeviceNumber[4:0], and FunctionNumber[2:0] fields of the original requester of the transaction. For the CPC945, this field will always = 0x0000.
Note: 1. I/O requests are not supported in the AL for inbound requests and will be returned as an unsupported request.	

Table 5-1. TLP Header Fields (Continued)

TLP Field	Description
Tag[7:0]	The Tag field of the packet deader is an 8-bit value assigned to each nonposted requests issued by a Requestor. For Completions, this filed allows the Completion to be paired with the original request at the Requester.
Lower Address[6:0]	For memory read completions, this field is equal to the lower 6 bits of the first returned data byte. For all other completions, this field must equal 0x00.
Fields Specific to Messages (See *Section 2.2.8, page 62 of the PCI Express Base Specification 1.0a* for more information.)	
Requester ID[15:0]	This 16-bit field is the concatenation of the BusNumber[7:0], DeviceNumber[4:0], and FunctionNumber[2:0] fields of the original requester of the transaction. For the CPC945, this field will always equal 0x0000.
Tag[7:0]	The Tag field of the packet header is an 8-bit value assigned to each nonposted requests issued by a requestor. For completions, this filed allows the completion to be paired with the original request at the requester.
Message Code[7:0]	Specifies the particular message embodied in the request.
(Bytes 8 -15) of TLP	Depend on type of message.
Note: 1. I/O requests are not supported in the AL for inbound requests and will be returned as an unsupported request.	

The flow of transmitted packets across the Link is controlled through the use of Flow Control Credits, or Credits for short. In PCI Express, there are separate Credits pools for NP, P, and Cpl TLP headers and data payloads, resulting in 6 discrete areas of credit management. The CPC945 will advertise the following Credits for P, NP, and Cpl packets.

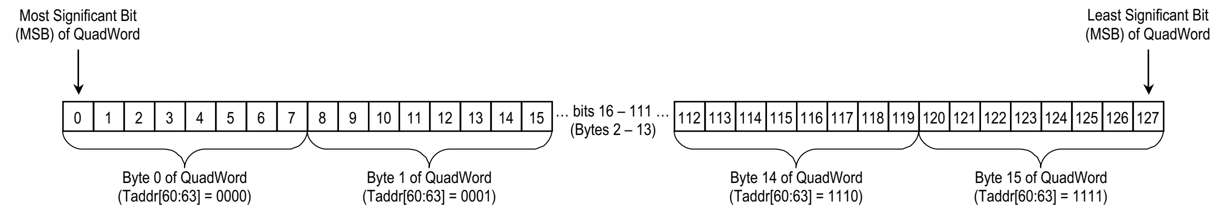
Table 5-2. CPC945 Advertised PCIe Credits

Credit Pool	Advertised Value	Meaning
Posted headers	16	Can accept 16 inbound posted headers (for example, write requests).
Posted data	0	Can always accept maximum payload size for each posted request. (Maximum data payload for the CPC945 will be 128 bytes ⇒ 2048 bytes total.)
Nonposted headers	16	Can accept 16 inbound nonposted headers (that is, read requests – the CPC945 will not accept any other inbound nonposted requests).
Nonposted data	0	The CPC945 never accepts any inbound nonposted write (that is, configuration or I/O). Therefore this “infinite” setting is really a degenerate case because any other value would indicate the ability to accept inbound nonposted requests with data payloads.
Completion header	0	The CPC945 always accepts a completion for a previously issued outbound nonposted request.
Completion data	0	Can always accept maximum payload size for each outbound nonposted read request issued by the CPC945 to an endpoint. Maximum data payload for the CPC945 will be 128 bytes ⇒ 512 bytes total as the CPC945 limits maximum issued outbound nonposted requests to four 128-byte requests. Alternatively, the CPC945 also supports eight 64-byte requests. The specific mode the CPC945 uses is configured at power-on-reset (POR) and can not be changed during the operational state.

5.2.2 PI Versus TL Data Formatting Differences

As data is transferred from system memory across the PI and ultimately to the TL interface for packeting and transmission over the PCIe, the formatting of the data must go through several transformations. This is predominantly due to the big-endian formatting of data in system memory and in the CPC945 versus little-endian formatting of data in the PCIe, but also due to bit ordering nomenclature differences as well. Bit ordering in system memory and on the PI is such that the leftmost, or most significant bit of data is defined as bit 0, whereas the rightmost, or least significant bit of data is defined as bit (n-1), where n is the width of the bus (for example, for a 64 bit bus, n = 64, and therefore the least significant bit is bit 63). This is shown in *Figure 5-6* for a 128-bit bus.

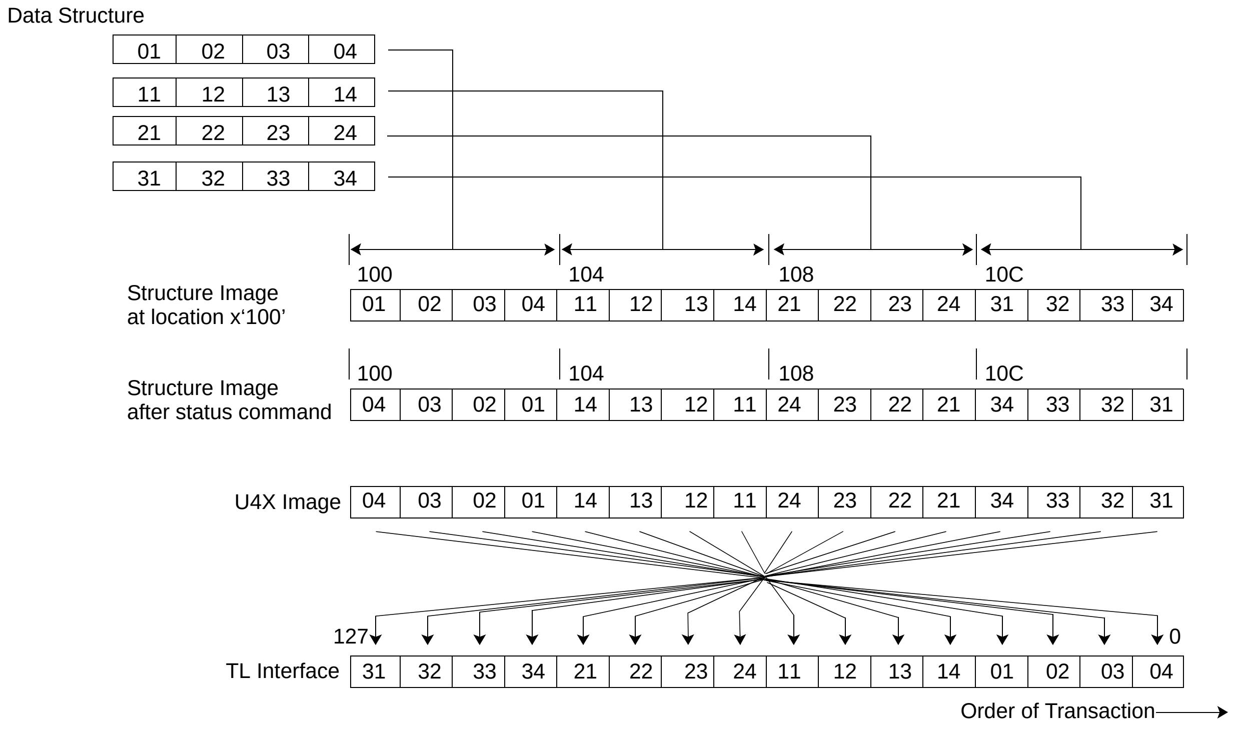
Figure 5-6. PI Data Bit and Byte Ordering



In PCIe, although the leftmost bit is still the most significant bit it is designated as bit (n-1), and the rightmost, least significant bit is bit 0. The process of converting the data format from system memory and the ordering of data bytes throughout this process is illustrated in *Figure 5-6*. For outbound data flow, the TL interface uses a single 128-bit interface that must be “packed”; that is, if a data payload follows a 3 Dword TLP Header then the first DWord of the Data Payload must be included in the first 128-bit transfer over the TL interface. This packing is not required with a 4 Dword header as the full 128-bit interface will be utilized by the TLP Header. Note that for this 128 bit/16 byte transfer over the TL interface, Byte 0 (0x04 as depicted below) is transmitted first over the PCIe, while Byte 15 (0x31 as depicted below) is transmitted last. (For a Link width greater than 1 Lane, note that there will be more than one byte transmitted over the Link during any given TL bus interface cycle. However, the general order of byte transmission will always be from lowest to highest.) Within each byte, the least significant (rightmost, bit 0) bit is transmitted first, with the most significant (left-most, bit 7) transmitted last[1].

[1] Note that the byte itself is not transmitted in its native form. Rather the byte is converted into a 10-bit “symbol” according to the rules outlined in section 4.2.1 (page 152) of the *PCI Express Base Specification 1.0a*. Within each symbol, bit 0 is transmitted first and bit 9 last.

Figure 5-7. System Memory to TL Interface Data Formatting Changes



The supported types of outbound PCIe requests in the CPC945 are shown in *Table 5-3*.

Table 5-3. Supported PCIe Outbound Request Types

PCIe Request Type	Posted/Nonposted	Supported in PCIe AL for Outbound Requests
Configuration reads	Nonposted	Yes
Configuration writes	Nonposted	Yes
I/O reads	Nonposted	Yes
I/O writes	Nonposted	Yes
Memory reads	Nonposted	Yes
Memory writes	Posted	Yes
Messages	Posted	No

All outbound requests are strictly ordered. Additionally, all outbound requests over the Link obey the PCI strict ordering rules, with the exception that all writes are strictly ordered (that is, there is no tracked distinction in the two differing types of writes). The consequence of this exception is that posted writes cannot pass nonposted writes and that therefore read completions cannot pass nonposted writes (since all writes are treated the same with respect to ordering). The outbound ordering rules for the PCIe AL are shown in *Table 5-4*.

Table 5-4. Ordering Rules for PCIe AL Outbound Requests

Request Type ↓→ Row pass Column?	Memory Write	Memory Read	Nonposted Write	Read Completion
Memory Write	No	Yes	No	Yes
Memory Read	No	No	No	Yes
Nonposted write	No	No	No	Yes
Read Completion	No	Yes	No/Yes – see note[1]	Yes[2]

Notes:

1. Since there is no distinction between a posted and nonposted write within the CPC945, it is not possible to allow read completions to pass nonposted writes. However, once the nonposted write is scheduled to be transmitted over the Link, meaning that it has been “popped” from the outbound write command buffer even though has not yet been committed to begin transmission over the Link, it is then possible to distinguish between a posted and nonposted write and therefore at this point a read completion is permitted to pass the pending outbound nonposted write and be issued over the Link.
2. From differing interfaces or from memory controller interface.

The maximum outbound write transfer size that the CPC945 will deal with is 64 bytes, as follows.

- For writes from the processor, PCIe is considered non-prefetchable memory space, and therefore non-cacheable. The maximum size transfer the processor can make to non-cacheable space is 64 bytes.
- For writes from HyperTransport, the maximum size transfer over HyperTransport is 64 bytes.

The AL stores up to 8 outbound writes internally which results in a 512-byte buffer, organized as thirty-two 128-bit-wide entries[1]. Whereas an outbound memory write transfer can be up to 64 bytes in length, a nonposted write should only have a data payload of one doubleword in a PCIe configuration. From the context of the system, the CPC945 looks like a PCI to PCI Bridge device, and therefore responds to “local” Type 0 configuration requests but passes on Type 1 configuration requests to the Link. Within the CPC945, Type 1 configuration requests are either translated into Type 0 configuration requests if targeting a bridge or an endpoint on its secondary bus, or left as Type 1 configuration requests if targeting a bridge or an endpoint on an attached bridge’s subordinate bus. The CPC945 does not support direct attachment to a switch. Type1 configuration transaction attempts to target a device with a nonzero device number on a secondary bus are considered invalid transactions and will not go out on the PCIe interface. The error will be indicated by bits 2 or 9 in the Invalid Transaction Register. Type1 configuration transaction attempts to target a device with a nonzero device number on a subordinate bus are considered valid transactions and will go out on the PCIe interface.

The CPC945 supports up to 64 local Dword registers (256 bytes) for the CPC945-specific configuration and control. All local registers are located in the PCIe 1 KB configuration space (as opposed to PCI’s 256 byte space). The CFG core has an extended configuration bus (XBUS) output from the CFG module. All local Type 0 configuration accesses that are not contained in the CFG register space are automatically passed to the XBUS. Therefore, access to these local registers is performed as normal Type 0 configuration access. In order to facilitate accesses to local registers, the CPC945 also incorporates a direct access mechanism.

[1] The specific organization of the Outbound Write Data Buffer is such that it will hold either eight 64-byte requests or four 128-byte requests (this is a POR configurable condition).

5.2.3 Error Checking

There are error checks that are performed within the PCI Express stack some of which are detected and dealt with in lower layers of the stack. According to the *PCI Express Base Specification 1.0a*, errors are categorized as Correctable or Uncorrectable errors. Uncorrectable errors are further categorized as Non-Fatal or Fatal errors (see *Section 6.2 of the PCI Express Base Specification 1.0a* for more information). The complete list is shown in *Table 5-5*. The errors that the AL must check for are indicated in the tables with a gray background.

Table 5-5. Physical Layer Errors

Error	Type	Description	Action	Logged in CFG
Receive	Correctable	Symbol or packet framing errors	Link is retrained, packet is re-transmitted	Yes, if enabled
Training	Uncorrectable (Fatal)	Non-coherent Memory	ERR_FATAL sent to Root Complex	Yes, if enabled

Table 5-6. Data Link Layer Errors

Error	Type	Description	Action	Logged in CFG
Bad TLP	Correctable	Calculated LCRC in incorrect	Packet is re-transmitted	Yes, if enabled
Bad DLLP	Correctable	DLLP has bad CRC	DLLP is discarded	Yes, if enabled
Replay timeout	Correctable	TLP re-transmission timeout	Data Link layer initiates reply	
Replay Num Rollover	Correctable	TLP re-transmitted too many times	Link is retrained, packet is re-transmitted	
Data Link Layer Protocol Error	Uncorrectable (Fatal)	DLLP is unrecognized	ERR_FATAL sent to Root Complex	

Table 5-7. Transaction Layer Errors

Error	Type	Description	Action	Logged in CFG
Poisoned TLP	Uncorrectable (Nonfatal)	Poisoned TLP bit set in Header	Packet is discarded or passed along; ERR_NONFATAL sent to Root Complex	Yes w/ TLP Header, if enabled
ECRC incorrect	Uncorrectable (Nonfatal)	Incorrect End-to-End CRC	Application dependent (but usually fatal); ERR_NONFATAL sent to Root Complex	Yes, w/ TLP Header
Unsupported Request (UR)	Uncorrectable (Nonfatal)	Received TLP indicates request that is not supported	Send Unsupported Request completion to requester; ERR_NONFATAL sent to Root Complex	Yes, w/ TLP Header
Completion Timeout	Uncorrectable (Nonfatal)	Timeout timer expired before receipt of completion	Application dependent (but usually fatal); ERR_NONFATAL sent to Root Complex	Yes
Completer Abort	Uncorrectable (Nonfatal)	Target aborted transaction	Application dependent (but usually fatal); ERR_NONFATAL sent to Root Complex	Yes, w/ TLP Header
Unexpected Completion	Uncorrectable (Nonfatal)	Completion TLP received when none expected	Application dependent (but usually fatal); ERR_NONFATAL sent to Root Complex	Yes, w/ TLP Header

Table 5-7. Transaction Layer Errors (Continued)

Error	Type	Description	Action	Logged in CFG
Receiver Overflow	Uncorrectable (Fatal)	Received request when no credits available	Fatal system error; ERR_FATAL sent to Root Complex	Yes, if enabled
Flow Control Protocol Error	Uncorrectable (Fatal)	Error in Flow Control processing and/or updating	Fatal system error; ERR_FATAL sent to Root Complex	Yes, if enabled
Malformed TLP	Uncorrectable (Fatal)	Any errors associated with the construction of received TLPs	Fatal system error; ERR_FATAL sent to Root Complex	Yes, w/ TLP Header

Malformed TLP errors have several sub-categories, not all of which are checked for by the AL (that is, some of them are detected by the TL). The complete list of errors that the AL will check for are listed in *Table 5-8*.

Table 5-8. PCI Express Stack Errors Addressed by Application Layer

Error	Subtype	Description	Action taken by AL
Bad TLP	N/A	Calculated LCRC in incorrect	Packet is ignored (it will be retransmitted)
Poisoned TLP	N/A	Poisoned TLP bit set in Header	Packet is passed along; Fatal system error (SERR# generated)
ECRC incorrect	N/A	Incorrect End-to-End CRC	Packet is passed along; Fatal system error (SERR# generated)
Unsupported Request (UR)	N/A	Received TLP indicates request that is not supported	Send Unsupported Request completion to requester
Completion Timeout	N/A	Timeout timer expired before receipt of completion	Fatal system error (SERR# generated)
Completer Abort	N/A	Target aborted transaction	Application dependent (but usually fatal); ERR_NONFATAL sent to Root Complex
Unexpected Completion	N/A	Completion TLP received when none expected	Fatal system error (SERR# generated)
Malformed TLP	Byte Enable Error	BE<3..0> and/or BE<7..4> have invalid values	Fatal system error (SERR# generated)
Malformed TLP	Address Boundary Error	Transaction crosses 4 KByte naturally occurring address boundary	Fatal system error (SERR# generated)
Malformed TLP	RCB Error	Completion does not conform to permissible Read Completion Boundary rules	Fatal system error (SERR# generated)
Malformed TLP	Configuration Retry Error	Illegal (unexpected) receipt of a Configuration Retry Request	Fatal system error (SERR# generated)

If the received TLP passes these basic integrity checks as listed above, it is passed up to the higher-level functions for processing. There are basically five types of legitimate TLPs that can be received as detailed in *Table 5-9*.

Table 5-9. Allowed Inbound TLPs Types and Destinations

TLP Type	Targeted Destination
Posted Write	Coherent Memory
	Non-coherent Memory
	HyperTransport
	Register File
Memory Read Request (Nonposted)	Coherent Memory
	Non-coherent Memory
	HyperTransport
	Register File
Completions	Outbound side of AL (for completion indication of outbound nonposted write)
	Processor Interface
	Hypertransport
Messages	Depends on type of message. Messages that are INTX assertions are decoded as issued “out of band” inside the CPC945. Other messages (for example, PME_To_Ack) are decoded and routed as required. Note that Vendor Specific Messages are not supported in the CPC945 and will be ignored.
Message Signaled Interrupts (MSIs)	MSIs are issued as posted writes and will be forwarded to the PI interface.

5.2.4 Message Decode

The Message Decode function is responsible for handling inbound Message TLPs transmitted from the Endpoint. the CPC945 will accept only the following inbound messages as shown in (see the *PCI Express Base Specification 1.0a*, *section 2.2.8, page 62* for a complete list of all PCI Express Messages.

Table 5-10. Message Decode

Message	Description
INTx Interrupt Signaling	Used to assert and de-assert a virtual INTx wire. If this message is received an output from the CPC945 will reflect the state indicated in the message. (This output would be attached to the CPC945 interrupt controller, if INTx interrupts are support.)
Power Management	Used in the inbound direction to either indicate an Endpoint has a PME event (not supported in the CPC945) or that an Endpoint is responding to a request sent to it from the CPC945 to enter the L2/3 Ready state (see the *PCI Express Base Specification 1.0a, section 5, page 221* for more information on power management in PCI Express devices). Upon receiving this Message, the CPC945 will indicate to the CPC945 Power Manager that the PCIe Endpoint has entered a low power state and is prepared to have its power removed.
Error Signaling	Used to indicate that some form of error has occurred in a PCI Express device. See the *“PCI Express Base Specification, Version 1.0a”* for more information on PCI Express errors.

5.3 Configuration Register Access

Accessing the configuration registers can be done one of three ways. The first, known as traditional configuration access, employs two access registers to initiate a configuration transaction. The second, known as direct configuration access, allows all local (on-chip registers) configuration registers to be accessed just like normal memory, but should be used only for testing purposes. The third, known as limited direct configuration access, allows a single cycle method of performing both type 0 and type 1 configuration transactions, so long as the target register lies within the first 256 bytes of the target device's configuration space. *Figure 12-1 PCIe XBus Configuration Space* shows all of the XBus configuration registers with the address offset for each of the three access methods described above. The upper nibble of the offset is the extended register number. The lower eight bits is the byte address. Since each register is 32 bits, the register number is the top six bits of the byte address.

On top of this there are three more ways supported by CPC945: SBus, I2C, and PI. The description for these three methods can be found in *"SBus Direct Configuration Access Method" on page 113*, *"I2C Direct Configuration Access Method" on page 113*, and *"PI System Command Registers" on page 436*.

5.3.1 Indirect Method

Traditional configuration accesses are performed via two access registers: the Configuration Address Register and the Virtual Configuration register. First a write is performed to the Configuration Address register identifying the target of the configuration transaction. This is followed by a read or a write to the Virtual Configuration Data register, which causes the transaction to be issued. A read to the Virtual Configuration Data register causes a read to the target identified in the Configuration Address register, while a write to the Virtual Configuration Data register causes the write data to be written to the target identified in the Configuration Address register.

Table 5-11. Configuration Access Registers

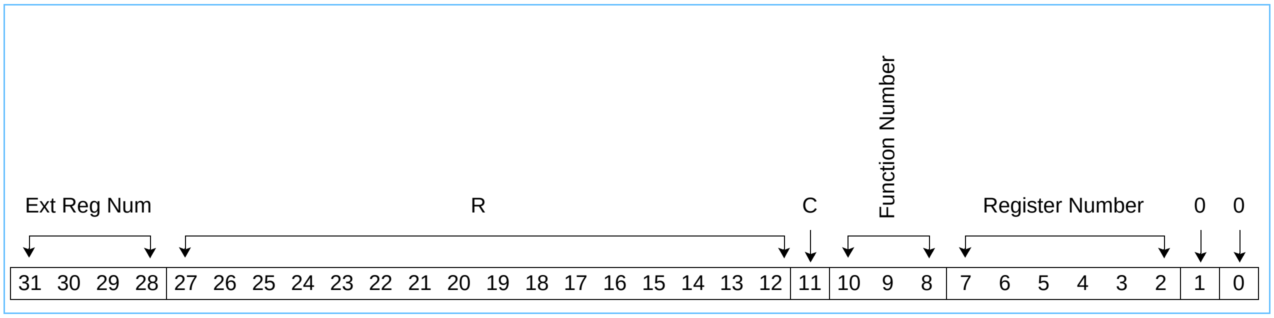
Address	Register Name	Type	Initial Value
0xF0800000- 0xF0BFFFFC	32-bit Configuration Address (direct configuration access mode disabled) The bottom two address bits determine if the access is a type 0 (0b00) or type 1 (0b01) configuration access.	R/W	N/A
0xF0800000- 0xF0BFEFFC	32-bit Configuration Address (direct configuration access mode enabled)	R/W	N/A
0xF0C00000- 0xF0FFFFFC	32-bit Virtual Configuration Data	R/W	N/A

The Configuration Address Register

The Configuration Address Register is a 32-bit register that exists in the range of addresses from 0xF0800000 to 0xF0BFFFFC (0xF0BFEFFC when direct access mode is enabled). Writing to this register is the first step in generating a traditional configuration transaction.

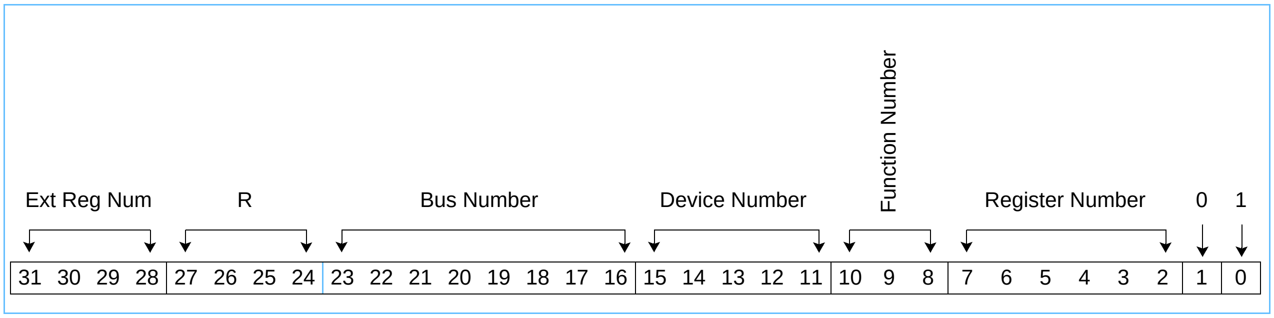
The format of the Configuration Address Register varies depending on what type of configuration transaction is being performed. The type of transaction is determined by the lowest two bits of the register. A value of 0b00 indicates the register is formatted for a type 0 configuration transaction, while a value of 0b01 indicates the register is formatted for a type 1 configuration transaction. Any other values are invalid.

Figure 5-8. Configuration Address Register Formatted For A Type 0 Configuration Transaction



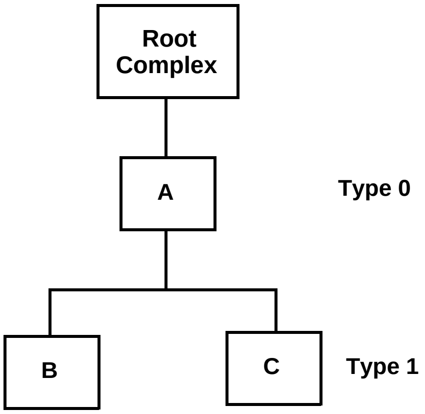
Type 0 configuration transactions (*Figure 5-8*) target local configuration memory space (addresses from 0xF0BFF000 – 0xF0BFFFFC). In order to successfully target this memory space, all reserved bits (R) must be 0, while the cycle bit (C) must be 1. For those familiar with the PCI specification, the cycle bit is actually the IDSelect (address bit 11) for the device. Also, because this is not a multifunction device, the Function Number must be 0. If these conditions are not met, any attempt to perform a traditional configuration transaction results in a master abort. Note that the Extended Register Number field is composed of the most significant four bits of the Configuration Address Register, bits that are traditionally reserved according to the PCI specification.

Figure 5-9. Configuration Address Register Formatted For A Type 1 Configuration Transaction



Unlike type 0 configuration transactions, type 1 configuration transactions target PCI Express devices located downstream from the PCI Express Application Layer. However, type 1 transactions must sometimes be converted to type 0 transactions before transmission. Whether this conversion is performed is based on what downstream device is targeted by the transaction. If the transaction is targeting the next downstream device (A), it is transmitted as a type 0 configuration transaction. If the transaction is targeting a device further downstream (B or C), it is transmitted as a type 1 transaction. Note that the Extended Register Number field is placed in the most significant four bits of the Configuration Address Register, bits that are traditionally reserved according to the PCI specification.

Figure 5-10. Transmission Type By Target



In order to determine which device is being targeted by a given type 1 configuration transaction, it is necessary to examine the Device Number field and to compare the Bus Number field to both the Secondary Bus Number and the Subordinate Bus Number. Both of these values are located in configuration space at location 0xF0BFF018 (bits 15:8 and 23:16, respectively). If the Device Number is 0 and the Bus Number is equal to the Secondary Bus Number, the transaction is transmitted as a type 0 configuration transaction. If instead the Bus Number is greater than the Secondary Bus Number and less than or equal to the Subordinate Bus Number, the transaction is transmitted as a type 1 configuration transaction. If neither of these conditions are true, a master abort occurs.

The Virtual Configuration Data Register

The Virtual Configuration Data Register is, as its name suggests, a virtual register that exists in the range of addresses from 0xF0C00000 to 0xF0FFFFFC. Though reads and writes can target this virtual register, they are actually being performed on whatever register is indicated in the Configuration Address Register. If the address in the Configuration Address Register is invalid, the transaction results in a master abort.

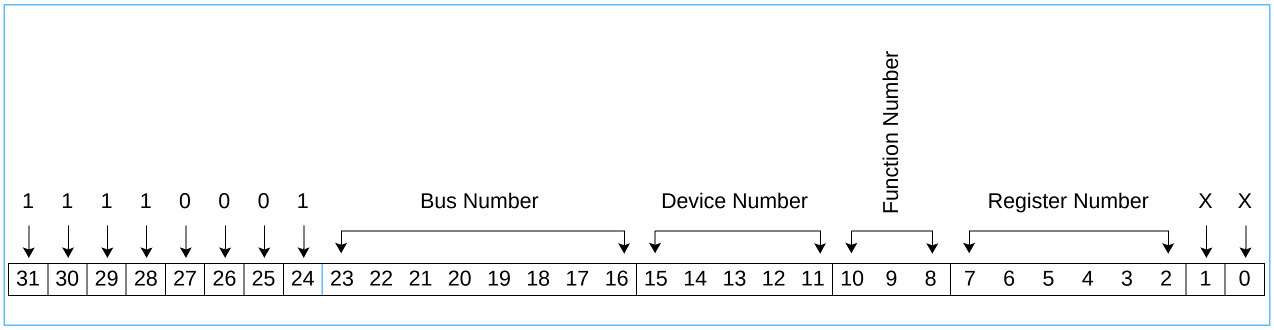
5.3.2 Limited Direct Access Method

The limited direct configuration access mode was developed in order to provide system software with a single cycle method of performing both type 0 and type 1 configuration transactions. This method was based on the existing HyperTransport method of performing single cycle configuration transactions.

In the HyperTransport method, two 16 MB (24 bit) memory apertures exist in the system memory map. These apertures allow address bit patterns needed to form a configuration access to be forwarded directly (for example, a read or a write to one of these locations is interpreted as a configuration transaction). One aperture produces type 0 transactions, while the other produces type 1.

For the CPC945 PCI Express port, only one memory region is available to direct access transactions: the memory space 0xF1000000 through 0xF1FFFFFF. This means that reads from or writes to any address beginning with 0xF1 will be interpreted as a configuration transaction.

Figure 5-11. Address Field For Limited Direct Configuration Transactions



In order to determine the characteristics of the configuration transaction, it is necessary to examine the lowest 24 bits of the target address as shown in *Figure 5-11* (the upper 8 being fixed as 0xF1). Bits 23:16 make up the 8 bit bus number, bits 15:11 make up the 5 bit device number, bits 10:8 make up the 2-bit function number, and bits 7:2 make up the 6 bit register number. The lowest 2 bits of the address are “don’t cares”. Note that because there are only 24 bits of each address available for defining the specifics of a configuration transaction, there is no way to access extended configuration space in limited direct mode.

Instead of relying on which memory region is used for a given transaction in order to distinguish between type 0 and type 1 (as is done in HyperTransport), the CPC945 relies on examining the bus number to make the distinction. If the bus number (bits 23:16 of the address field) matches the primary bus number, the transaction is type 0. If not, the transaction is type 1. A type 1 transaction is forwarded over the link as a type 0 if the bus number matches the secondary bus number. Otherwise, it is forwarded as a type 1. It is necessary to note that, at the very least, the primary bus number must be set via the classic two-step configuration mode before limited direct transactions can be attempted.

To access the CPC945 XBus Configuration registers, use the same address offset as shown in *Figure 12-1* on page 330, but add 0xF1BFF as the upper bits instead of 0xF0BFF as shown in *Table 12-23* on page 551. For example, to access the 32-bit Legacy Interrupt Control Register through a Limited Direct Configuration Access use the address 0xF1BFF0F0 instead of the address 0xF0BFF0F0 used for a traditional configuration access.

5.3.3 Direct Access Method

Direct configuration accesses, which are simply standard reads and writes directly targeting configuration registers, are disabled by default. In order to enable direct configuration accesses, simply write 0x1 to the Direct Address Mode Register (detailed in *Section 12.11.2.22 on page 571)* using a traditional configuration access. Note that even when direct configuration access mode is enabled, traditional configuration accesses can be made, though the supported address range of the Configuration Address register is reduced. The direct access mode is only meant for accessing the CPC945's configuration registers and therefore all accesses are Type 0. Direct accesses are always in the address range: 0xF0C00000 to 0xF0FFFFFC.

5.3.4 SBus Direct Configuration Access Method

The CPC945 also allows the processor another method of direct access for any PCIe registers located off of the SBUS. The address range 0xF8090000 - 0xF809FFFF is reserved for PCI Express Configuration Registers. Note that SBUS only decodes the upper four nibbles (0xF809) and passes the lower portion of the address to the PCIe configuration registers. The PCIe configuration registers only decode the lower fourteen bits, so the top two bits of the address passed to the PCIe configuration registers are don't cares. *Figure 12-1* on page 330 only shows the register offsets for the standard method of accessing the PCIe configuration registers. For accessing the configuration registers through the SBus or I^{2}C, the offsets must be shifted left by two bits. This means that the Legacy Interrupt Control register uses the offset 0x03C0 (0xF80903CD) instead of 0x0F0 when accessed by the SBus or I^{2}C

5.3.5 I^{2}C Direct Configuration Access Method

The CPC945 also allows a method of direct access for any PCIe registers located off of the SBUS using I^{2}C. Details are in *"*CPC945 *Address Specification"* on page 173 and use the same addresses as given in "SBus Direct Configuration Access Method", above.

6. HyperTransport

6.1 Overview

The CPC945 HyperTransport interface responds to addresses forwarded from the Processor Interconnect (PI) interface that fall into the range of “valid addresses” as specified by the HT Address Mask Register. Addresses on the external HyperTransport bus are qualified in the same manner.

6.2 Initializing HyperTransport Core in the CPC945

The initialization sequence of the HyperTransport unit is dependent on the detection of the ‘sync’ state. *Figure 6-1* illustrates a HyperTransport initialization sequence, followed by an HT_LDTSTOP_L.

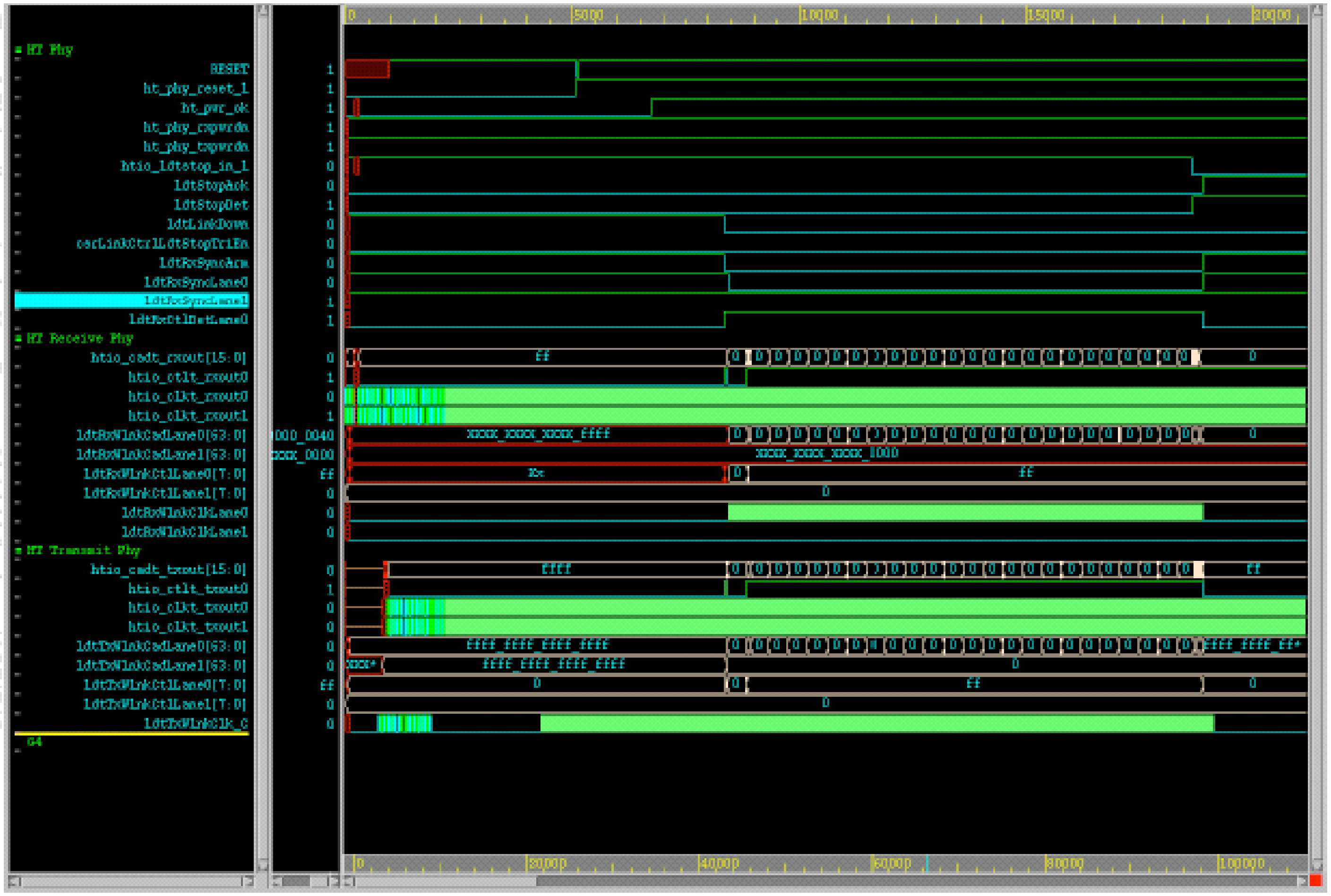
The first half of the link initialization is automatically attempt by the core when the link powers up. The core begins this sequence by asserting the HT_RESET_L. In the initialization sequence, the link width is based on the maximum width of the smallest transmitter or receiver, but limited to 8. The default clock frequency is 200 MHz for all devices. *Figure 6-1* demonstrates that first half of the initialization.

While HT_RESET_L is asserted during a cold reset, each device drives its CTL signal to a logical 0, and drives its output CAD signals to a value that is based on the width of its receiver. At the deasserting edge of HT_RESET_L, each device samples its input CAD signals and uses this sampled value to determine its transmitter and receiver widths then updates the LinkWidthIn and LinkWidthOut Registers. After the deassertion of HT_RESET_L, each device asserts its CTL signal across a rising CLK edge, initiating a sync sequence. The assertion of the CTL signal serves to indicate to the device at the other side of the link that this device is ready to complete initialization of the link.

At this time, the HyperTransport is active.

However, the second stage of the initialization is driven by the system firmware. This stage is used to operate at the maximum clock frequency and link width.The firmware reads the LinkWidthIn and LinkWidthOut Registers to determine the link’s maximum width. The firmware also determines the links frequency cababilites. It then updates the corresponding control registers for link width and frequency in both upstream and downstream traffic. Lastly, the firmware asserts the HT_LDTSTOP_L to force the updated values to take effect.

Figure 6-1. Initialization Sequence



6.2.1 Programming the HyperTransport core

The HyperTransport Core synchronizes its transmit and receive clocks through a hardware initialization at power up (see *Section 6.2 Initializing HyperTransport Core in the CPC945* on page 115). However the link width and frequency can be optimized with programming the HypterTransport core. This section discusses those registers and others that are important basic configurations.

6.2.1.1 HT1 Address Mask Register (0xF8070200)

The HT1 Address Mask Register defines the range of addresses that the HyperTransport will forward read or write from the CPU/PCIe Master to the HT Bus Memory Space. The Address Mask Register defines fifteen coarse regions (256 MB each) and sixteen fine regions (16 MB each) of the lower 32-bit address space that CPC945's HyperTransport responds to or ignores from the PI and PCIe Busses.

6.2.1.2 Link Config/Control Register (0xF8070110)

The Link Config Register has four register fields that pertain to the HyperTransport link width. LinkWidthOut(Out) and LinkWidthIn(In) are the utilized link widths. With a cold link reset, they are reset to zero, then are set by hardware during the Link Width Initialization.

The MaxLinkWidthOut(MaxOut) and MaxLinkWidthIn(MaxIn) reflect the physical width present on the device.

The low order half [16:31] of this register pertains to the link control register fields. These fields control various behaviors such as how to respond to CRC errors.

6.2.1.3 LinkFreqCap/LinkError/Link Freq/RevisionID Register (0xF8070120)

This register has register fields that pertain to the frequency of the link.The LinkFreqCap is a read-only register that defines what frequencies the CPC945 HyperTransport can support. The LinkFreq register field represents the link frequency.

6.2.1.4 Error Handling/Enumeration Scratchpad Register (0xF807140)

The ErrCtrl/Enum Register contains routing enables from the various error log bits to the various error reporting mechanisms, as well as the Chain Fail and Response Error status bits.

6.2.1.5 Example of Programming Sequence

1. Set the HT Address Mask Register (0xF8070200).
2. Set unique unitIds to all HT devices. The CPC945 hypertransport's unitId is hard wired, however the HT devices need to be unique on the link.
3. Set optimal speed frequency - LinkFreqCap/Link Error/Link Freq/ Revision ID Register (0xF8070120).
4. Set optimal link width- Link Config Register (0xF8070110).

The HT link must then be reset with either a cold reset (asserting $\overline{\text{HT\_RESET\_L}}$) or assert $\overline{\text{HT\_LDTSTOP\_L}}$. The HT devices then begin an initialization sequence.

5. Reassign unique unitIds to all HT if applicable. The asserting of $\overline{\text{HT\_RESET\_L}}$ clears all unitIds.
6. Set up any error handling features.

6.3 DART

Coherent HyperTransport DMA accesses are remapped via a DMA Address Relocation Table (DART) so that I/O devices can transfer data directly to any memory location in the 36-bit Extended memory map. The operating system (OS) establishes entries in the DART when a buffer area is requested by a driver. The “physical” addresses the system passes back to the driver are then mapped by the DART when the device accesses memory via these addresses. The OS removes DART entries when the driver indicates the buffer area is no longer required. More details of the DART implementation in CPC945 are available in *Section 3.4 DMA Address Relocation Table (DART)* on page 53.

Software must insure that the least significant nibble of the Configuration Data register address on the PI is set to the appropriate value to cause the proper offset for CPC945 Configuration Accesses.

6.4 HyperTransport Read Size Restriction

On the CPC945 HyperTransport interface, read operations to downstream devices are always 64 bytes in length, regardless of actual data size requested. Some processor initiated read transactions could be in excess of 64 bytes, so programmers should be aware of the need for software checks to ensure that this does not cause problems. When HyperTransport must respond to a read command sent from a HT device, 64 bytes is always requested from memory. 64 bytes is both the largest transfer size available on the HT bus and the smallest transfer size from DRAM. The HyperTransport interface extracts the actual bytes requested from the returning data and returns the extracted bytes to the requesting HT device.

Table 6-1.HyperTransport Portion of the Memory Map.

Start Address	End Address	Name	Comments	Page
0x0F2000000	0x0F2FFFFFF	HT Type 0 Configuration Space	HyperTransport interface is used to connect to I/O devices outside of CPC945 (no other bridges in CPC945).	120
0x0F3000000	0x0F3FFFFFF	HT Type 1 Configuration Space	HyperTransport interface is used to connect to I/O devices outside of CPC945 (no other bridges in CPC945).	120
0x0F4000000	0x0F43FFFFF	HT I/O Access	HyperTransport I/O Access space for CPC945 HT interface.	120
0x0F4400000	0x0F47FFFFF	HT EOI	HyperTransport End-of-Interrupt space for CPC945 HT interface.	120
0x0F4800000	0x0F4FFFFFF	Reserved	Reserved for HyperTransport.	120

6.5 HyperTransport Address Space

CPC945 uses two separate buses for expansion, PCI Express and a 16-bit wide HyperTransport Bus. CPC945 reserves the address range 0x0F2000000 through 0x0F4FFFFFF for HyperTransport. HyperTransport uses this memory range as shown in *Table 6-1*.

6.6 HyperTransport Bus Address Decoding

Read or write requests originating from either the CPU or a PCIe master to the HT Bus Memory space defined in Table 12-1 on page 327 are serviced by the CPC945 and forwarded onto the HT Bus as Memory read/write commands. CPC945 passes the PI Address to the HT Bus address un-modified. The range of addresses responded to by HyperTransport from the PI is defined by the *Section 12.12.12 HT Address Mask Register* on page 632. This register defines fifteen coarse regions (256 MB each) and sixteen fine regions (16 MB each in 0xF8000000-0xF8FFFFFF) of the lower 32-bit address space that CPC945's HyperTransport responds to/ignores from the PI, and PCIe Buses. CPC945 implements a decode/anti-decode scheme whereby, if a given address is decoded to be passed from the PI to the HT Bus, the same address is not allowed to pass from the HT Bus to the PI Bus. The fine grain decodes are defined in *Table 6-1*.

HyperTransport is only one agent requiring address space within CPC945. PCI Express, DRAM, CPC945 Registers, and ROM all require address space as well. When setting address spaces to HyperTransport, ensure other consumers of address space are not acquiring the same address space. For example, it is easy to give the PCI Express 0 Address Mask Register and HyperTransport (*Section 12.12.12 HT Address Mask Register*) the same address range. In the event that an address space is accidentally assigned to both HyperTransport and PCI Express, the requests will always be routed to HyperTransport space. (That is, HyperTransport always "wins" in case of duplicate assignments.)

6.7 HyperTransport Address Mapping

Determination of which transactions pass to and from the HyperTransport interface is controlled by the CPC945 address map. See *Table 12-1*. HyperTransport's portion of the address map (*Table 6-1*) is determined by both fixed and programmable means. Requests generated by either a processor or a PCIe master, and destined for the HyperTransport interface, are referred to as downstream requests. Requests that originate from the HyperTransport chain destined for Main Memory, PCI Express, or CPC945 register space are referred to as upstream requests. Mapping of upstream and downstream requests are handled differently and described separately. Any access to system memory from HyperTransport is always coherent. Access to memory residing on the PCI Express or HyperTransport Busses is always non-coherent.

6.7.1 Downstream Requests

For downstream requests the maximum transfer size allowed is 64 bytes. For requests originating from PCI Express this limit is ensured via hardware. The processor interface can transfer up to 128 bytes in a single transfer, so this limitation must be enforced in software for requests originating on the PI interface.

The following address regions are fixed in the CPC945 memory map. A read or write to these spaces results in a transaction being sent to the HyperTransport chain. See the HyperTransport I/O Link Specification (Chapter 9, Address Map) for more information on the individual HyperTransport address regions. The actual implementation of this decode is achieved by forcing the appropriate Fine Address Select bit in the HT Address Mask Register (*Section 12.12.12 HT Address Mask Register*).

0xF2000000 - 0xF2FFFFFF

Requests in this address range are mapped into the “Type 0” Configuration region of the HyperTransport address map. This is accomplished by converting the upper byte (F2) of the original 32 bit address into the upper two bytes (FDFE) of the 40 bit HyperTransport Type 0 configuration cycle. The lower 24 bits of the original address pass unchanged. As with accesses to all system and configuration registers, the size of transfers to this region must not exceed 4 bytes in length and not cross aligned four byte boundaries. For more information on HyperTransport configuration cycles see chapter 7 (Configuration Accesses) of the HyperTransport I/O Link Specification.

Note: The actual implementation of this decode is achieved by forcing the appropriate “Fine Address Select” bit in the HT1 Address Mask Register (see *Section 12.12.12 HT Address Mask Register* on page 632).

0xF3000000 - 0xF3FFFFFF

Requests in this address range are mapped into the “Type 1” Configuration region of the HyperTransport address map. This is accomplished by converting the upper byte (F3) of the original 32 bit address into the upper two bytes (FDFF) of the 40 bit HyperTransport Type 1 configuration cycle. The lower 24 bits of the original address pass unchanged. As with accesses to all system and configuration registers, the size of transfers to this region must not exceed 4 bytes in length and not cross the aligned four byte boundaries. For more information on HyperTransport configuration cycles see the HyperTransport I/O Link Specification (Chapter 7, Configuration Accesses). The implementation of this decode is achieved by forcing the appropriate Fine Address Select bit in the HT1 Address Mask Register (see *Section 12.12.12 HT Address Mask Register* on page 632).

0xF4000000 - 0xF43FFFFF

Requests in this address range are mapped into the I/O region of the HyperTransport address map. This is accomplished by converting the upper ten bits (1111_1000_00) of the original 32 bit address into the upper 18 bits (1111_1101_1111_1100_00) of the 40 bit HyperTransport I/O cycle. The lower 22 bits of the original address pass unchanged. As with accesses to all system and configuration registers, the size of transfers to this region must not exceed 4 bytes in length and not cross aligned four byte boundaries. The implementation of this decode is achieved by forcing the appropriate Fine Address Select bit in the HT1 Address Mask Register (see *Section 12.12.12 HT Address Mask Register* on page 632).

0xF4400000 - 0xF47FFFFF

Requests in this address range are mapped into the Interrupt/EOI region of the HyperTransport address map. This is accomplished by converting the upper ten bits (1111_1000_01) of the original 32 bit address into the upper 18 bits (1111_1101_0000_0000_00) of the 40 bit HyperTransport I/O cycle. The lower 22 bits of the original address pass unchanged. The transfer sizes must not exceed 4 bytes in length and must not cross aligned four byte boundaries. The implementation of this decode is achieved by forcing the appropriate Fine Address Select bit in the HT Address Mask Register (see *Section 12.12.12 HT Address Mask Register* on page 632).

0xF4800000 - 0xF4FFFFFF

Reserved.

0xFF000000 - 0xFFFFFFFF Requests in this address range pass unchanged to HyperTransport (bits 39:32 are padded with zeroes). This address range is dedicated to system ROM and serviced by the HyperTransport device connected to ROM. The implementation of this decode is achieved by forcing the appropriate Fine Address Select bit in the HT Address Mask Register (*Section 12.12.12 HT Address Mask Register*).

Programmable regions: The HT Address Mask register (address 0xF8070200, *Section 12.12.12 HT Address Mask Register*) is used to map other portions of the system memory map into HyperTransport Memory-Mapped I/O space. In CPC945 based systems, the lower 2 GB of the address range (0x00000000-0x7FFFFFFF) is mapped to DRAM. Addresses from 0x80000000 to 0xEFFFFFFF of the memory map can be mapped to the HyperTransport using the Address Mask Register, Coarse Address Space Select. Addresses from 0xF0000000 to 0xFFFFFFFF of the memory map can be mapped to the HyperTransport using the Address Mask Register, Fine Address Space Select. As mentioned above, the only transfer size limitation to this region is that the request must not exceed 64 bytes in length.

6.7.2 Upstream Requests

The following address regions are fixed in the CPC945 memory map. A read or write to these regions results in transaction acceptance by the HyperTransport chain. All addresses that are reserved or fall outside the regions listed above are reflected back down the HyperTransport chain (peer-to-peer). If no HyperTransport device accepts the address, then it reaches the end of the chain and causes a master abort. Note that HyperTransport uses 40 bit addresses, but CPC945 only accepts 32 bit addresses except for DMA accesses to System memory routed through the DART or in the case of interrupts.

0x0000000000- 0x007FFFFFFF Requests in this address range are mapped to main system memory via the DART. See *Section 6.3 DART* on page 118. All possible sizes of read and write operations available on HyperTransport can be accepted by CPC945. All sizes of reads to this region results in a 64 byte read request being issued to the Memory Controller. This is allowable, as system memory is considered prefetchable, meaning the memory has no read side effects. Writes to this part of the memory map are mapped into the smallest power of two sized transfer that contains all of the write data. For example, an unaligned 32 byte write results in a 64 byte write being issued to the Memory Controller with the required 32 byte enable bits asserted.

0x00F8000000 - 0x00F8FFFFFF Requests in this address range are mapped to the CPC945 Control Registers. All requests to this address range must not exceed 4 bytes in length. Non-posted requests of greater than 4 bytes return an error response. Posted requests of greater than 4 bytes are silently dropped.

0x00F9000000 - 0x00FFFFFFFF Access through this address range is achieved by forcing the appropriate Fine Address Select bit in the PCI Express Address Mask Register. See *Section 12.12.12 HT Address Mask Register* on page 632.

0x0100000000 - 0xFCFFFFFFFF Reserved - CPC945 only supports 32-bit addressing from HyperTransport.

0xFD00000000 - 0xFDF8FFFFFF Write requests in this address range are mapped to the MPIC Interrupt controller in CPC945 (See MPIC registers in *Section 12.11.3*). Address bits 23:16 are right shifted to select a register within the MPIC interrupt controller. This implementation accepts only write requests to a subset of the interrupt address range described in the HyperTransport Link Specification. Read requests to this range are not supported and can result in unexpected behavior.

0xFDF9000000 - 0xFFFFFFFFFF Reserved.

Programmable Regions: The PCI Express Address Mask register is used to map other portions of the system memory map into CPC945 memory space. (See *Section 12.11.2.4 PCI Express 0 Address Mask Register* on page 556.) The HyperTransport address decode logic uses the PCIe address mask register to determine if a transaction should be accepted and forwarded to the PCI Express interface. Since this portion of the address range is not considered prefetchable, read requests from HyperTransport must be presented to the CPC945 interface in such a way that the byte enable information for the request can be accurately communicated to the PCI Express master interface logic. This can result in the original request being divided into smaller requests (up to 6 in the worst case). The resulting sequence of read requests transfer at least one byte of data for every QW (8 bytes) of read data requested. The worst case example of one HyperTransport read resulting in 6 CPC945 read requests being generated occurs when a 14 DW (56 bytes) read from HyperTransport has a starting address offset of 4 within an aligned 64 byte block. This results in the following sequence of read requests to CPC945:

```
4  byte read from offset 4
8  byte read from offset 8
16 byte read from offset 16
16 byte read from offset 32
8  byte read from offset 48
4  byte read from offset 56
```

Writes to this area of the memory map are mapped into the smallest power of two sized transfer that contains all of the write data. For example, an unaligned 32 byte write results in a 64 byte write issued to the Memory Controller (with the required 32 byte enable bits asserted).

6.8 Reset

The HyperTransport link/chain can be reset by software using the SecBusReset bit in the Bridge Control Register (*Section 12.12.17 Bridge Control Register (BrCtrl)* on page 636) through a cold link reset (cold means that PwrOk is deasserted during the sequence) or a warm link reset. The setting of cold vs. warm link reset is controlled by the WarmReset bit in the Command/Pointer/Capability ID Description Register (*Section 12.12.7 Command/Pointer/Capability ID Register (HTCapability00)* on page 625). Note that a software link reset can be achieved with back-to-back writes to the SecBusReset bit since no hold time is required. When CPC945 is reset through a cold or warm system reset, all HyperTransport logic is reset, resulting in a HyperTransport link cold reset. The HyperTransport specification defines both a cold and warm reset, but the CPC945 HyperTransport implementation does not provide for a means of generating a HyperTransport warm reset directly from a system warm reset.

A HyperTransport link warm reset affects the four register fields listed below. Each field description consists of the register name and the field name. In parentheses below is the field name used in the HyperTransport Specification and a brief description of what happens on reset.

- Link Control Register, Init (InitializationComplete) -Resets to 0, 1 when link initialization is complete

- Link Control Register, EOC (EndOfChain) - Device or software can set it afterwards
- Link Control Register, TXO (TransmitterOff) - Transmitter is turned on after reset
- Error Handling Register, ChainFail (ChainFail) - cleared on reset

A HyperTransport link cold reset affects the previous four register fields as well as:

- Link Control Register, In (LinkWidthIn) - Resets to zero, then set by hardware during link width initialization
- Link Control Register, Out (LinkWidthOut) - Resets to zero, then set by hardware during link width initialization

A Warm Link Reset of a HyperTransport Link causes the Reset Line to be low (asserted) and the PwrOk Line to be high (asserted). Warm link resets can be used to remove sync flooding or change the HyperTransport link width. A Cold Link Reset on a HyperTransport Link causes the Reset Line to be low (asserted) and the PwrOk Line to be low (deasserted). A cold reset (due to a CPC945 reset) resets the HyperTransport link to its initial boot state.

6.9 Exceptions to HT Specification 1.04

This section lists all exceptions to the HT Specification, Version 1.04 implemented in CPC945 HyperTransport. If an exception is not listed below, CPC945 HyperTransport adheres to the HT Specification. The exceptions are as follows:

- Use of SERR# and SYNC
- CPC945 does not stop the HyperTransport clocks to change HyperTransport’s PLL
- Bridge Control Register
- CPC945 has no primary HT interface (see *Section 6.9.1 Bridge Control Register*)
- CPC945 primary interface has only one write and one read pipe
- No CPC945 support for HT atomic operations
- CPC945-specific registers
- CPC945 performance counters

Note: CPC945’s HyperTransport registers do not follow the standard PCI header format, particularly as address decode is concerned, but this is a PCI issue more than a HyperTransport issue.

6.9.1 Bridge Control Register

One required deviation from the device header format is the addition of a *“Bridge Control Register (BrCtrl)” on page 636*. The Bridge Control Register is used to specify the error and reset behavior of the HyperTransport link connected to the host (primary interface). The Bridge Control Register takes the place of the host header and includes the SecStatus/DetSerr bit.

6.9.2 Updating the PLL

The HyperTransport Specification describes the HT PLL as capable of being changed without placing the PLL in reset. However, the CPC945 HT PLL can only be changed when the PLL is in reset or by using the ForcePLLReset. Use of the ForcePLLReset bit is not recommended as data present on the HyperTransport bus can be lost.

The HyperTransport Specification mandates the HT clocks continue running while the clocks are changed. However, the CPC945 power manager design requires stopping the clocks while the PLL is changed.

6.9.3 Ordering through CPC945's Primary Interface

HyperTransport performs ordering through three pipes: posted, non-posted, and response. The CPC945 bus is used for the HyperTransport Host bridge's primary interface. The CPC945 bus has two pipes: reads and writes. Both interfaces have ordering rules between their respective pipes. Since the primary interface has only two pipes compared to HyperTransport's three pipes, CPC945 does not adhere to all of the ordering rules of HyperTransport.

6.9.3.1 Downstream Requests

Downstream writes that become nonposted when entering HyperTransport cause a conflict in ordering rules. The only nonposted writes are configuration writes and HyperTransport I/O (see *Table 6-1*). Since CPC945 only performs posted writes, a configuration write is treated as posted within CPC945, but converted to a nonposted write when the write reaches HyperTransport. When the nonposted write returns a response packet to CPC945 over HyperTransport the response packet is ignored by CPC945. This means that configuration writes and HyperTransport I/O from CPC945 are posted writes. From an ordering point of view, CPC945 handles all writes as posted internally. Therefore, the completion of a nonposted write (tgtDone issuance) does not imply the write data reached its final destination.

The CPC945 issues only one nonposted write and waits until the target is done with the first nonposted write before another one is issued. One outstanding at a time is the rule. From the CPC945 perspective, all writes are strongly ordered, and all reads have to push all writes. This requires CPC945 posted writes to push nonposted writes, CPC945 nonposted writes to push nonposted writes, and reads to push nonposted writes. To create this effect on HyperTransport, there are two choices. Putting a nonzero sequence ID on all reads and nonposted writes takes care of the second and third items. Since HyperTransport does not have any way to make a posted write push a nonposted write (this would result in a deadlock-prone design), the only way to cover the first item is to block posted writes while nonposted writes are outstanding. Sequencing all reads has a potentially bad performance effect, and nonposted writes are believed to be rare. Therefore, it was decided to stall reads on nonposted writes as well, rather than sequence the entire nonposted channel.

To summarize, CPC945 keeps reads in order, writes in order, and reads push writes. This is supported in HyperTransport until a nonposted write is issued. If a nonposted write is issued, no other transaction can be issued from CPC945 to HyperTransport until the response packet from the nonposted write is returned. Upon receipt of the response packet, CPC945 allows transactions for HyperTransport to proceed normally, and the response packet is discarded.

6.9.3.2 Upstream Requests

The HyperTransport Specification contains upstream I/O ordering and host ordering requirements. However, CPC945 implements an alternative scheme because CPC945 doesn't support the proper primitives. CPC945 instead implements a simplified scheme which is a superset of the requirements in the *HyperTransport Specification,* Table 22.

The ordering rules for CPC945 are as follows: Compare 2 received packets according to the virtual channel/ unitId/seqId rules specified in *HyperTransport Specification*, Table 21. If the compare indicates a dependence, the ordering points are determined according to the target of the request (CPC945, including memory, registers, or PCI Express; or peer-to-peer), following the rules listed in *Table 6-2*.

Table 6-2.CPC945 Upstream (from CPC945 to HT device) Ordering Rules .

First Transaction	Second Transaction	Ordering
CPC945 Request	CPC945 Request	Requests are issued across CPC945 in order.
CPC945 Request	CPC945 Response	The second transaction is issued across CPC945 after the first transaction clears the snoop pipeline.
CPC945 Request	Peer2Peer	The second transaction is issued downstream after the first transaction clears the snoop pipeline.
CPC945 Response	*	No requirement (nothing is ever ordered behind responses).
Peer2Peer	CPC945	The second transaction is issued across CPC945 after the first transaction is queued for transmission downstream.
Peer2Peer	Peer2Peer	Reflected packets are issued downstream in order.

Once a packet is handed to the CPC945 bus logic it is the responsibility of the CPC945 bus logic, and the receiver, to keep writes strongly ordered and make reads push writes. Once a packet is queued for transmission downstream it is the responsibility of any intermediate tunnels and the destination device to keep packets ordered to their final destinations.

Example of an Upstream CPC945 read following a Peer2Peer write (5th entry in *Table 6-2*):

> The first transaction is a peer-to-peer write and the second transaction is a read to CPC945. Therefore, the second transaction is issued across CPC945, after the first transaction is queued for transmission downstream.
>
> The requirement is that the peer-to-peer request is “ordered” before the read is passed upstream across CPC945. In this case, “ordered” is defined as meaning that the write must be queued for transmission in such a way that any traffic issued from the CPC945 after the CPC945 receives the read (that could in some way depend upon the read) must not be able to pass the write.
>
> Specifically, in the case of the example listed here, the requirement is that the read response (coming from the CPC945) push the write out to HyperTransport. The read is prevented from being issued to the CPC945 until the write receives the handshake indicating it has been queued for transmission.

6.9.4 HyperTransport SERR#

HyperTransport within CPC945 has implemented DetSerr (Bridge Control Register, bit 0) and not implemented SerrEN (Bridge Control Register, bit 14). To clarify the reasoning behind this, here is a brief discussion of SERR/sync flood handling for CPC945. First the general use of SERR# in PCI is discussed, then the general use of syncs in HT. The last part of this section describes how sync is used in CPC945. Note for the discussion below that Cmd/SerrEn and SecStatus/DetSerr are not implemented in CPC945. It is suggested that this entire section be read, before looking for any register fields named in the discussion.

Note: X/Y denotes field Y of CSR X.

For PCI, SERR# is a separate wire that acts purely an error status indicator, which propagates upstream towards the host. Devices on a bus can assert SERR# to indicate a bus error or internal error, if Cmd/SerrEn is asserted. Bridges can drive SERR# due to an internal error on their primary interface, if Cmd/SerrEn is asserted. Bridges detect the assertion of SERR# on their secondary interface, and set SecStatus/DetSerr. If BrCtrl/SerrEn and Cmd/SerrEn are both set, they also assert SERR# on their primary interface when they detect SERR# on their secondary interface. Bridges never drive SERR# on their secondary interface, and have no mechanism for logging SERR# assertion by other parties on their primary interface. Both devices and bridges (on their primary interface) log when they assert SERR# using the Status/SigdSerr bit (0xF8070010, bit 1).

In the HT world, we have sync flooding. Sync flooding places sync transactions on the HT bus, not on a separate wire like SERR#. This serves two error-processing purposes:

- It signals the host that something has gone wrong on the link.
- It stops data flow on the link, to prevent potentially-corrupted packets from reaching their destinations.

As such, sync flooding in HT is a superset of SERR# signaling in PCI. Function #1 of sync flooding is considered to be equivalent to SERR#, for purposes of propagation and logging. Therefore, devices and bridges (on their primary interface) initiating sync flooding require Cmd/SerrEn to be set, and Status/SigdSerr is set when it happens. Bridges will set SecStatus/DetSerr when they observe sync flooding on their secondary interfaces, and will propagate it to their primary bus if Cmd/SerrEn and BrCtrl/SerrEn are both set. In addition, all devices and bridges propagate sync floods from their receivers to their transmitters on the same chain, and bridges will propagate sync floods from their primary to their secondary interfaces, without requiring any enables to be set, or performing any logging.

There are typically three enables associated with HT link errors – one each enabling the error to be reported via sync flooding, fatal error interrupt, or nonfatal error interrupt. The sync flood enables always gate sync flooding, regardless of whether the link is on the primary or secondary interface of a bridge. In general, it is expected that sync flooding will always be on for the errors that indicate something basic is wrong (CRC, protocol, buffer overflow) -- the enables are provided primarily to allow faulty error checking hardware to be turned off. Bridges can initiate sync floods on their secondary bus if these enables are set, without requiring Cmd/SerrEn to be set or setting a log bit. This will cause SecStatus/DetSerr to be set, and the SERR can then propagate to the primary bus as described above. In addition, the HT spec allows a detected SERR# on the secondary bus to be forwarded as an interrupt on the primary bus, based on the ErrCtrl/SerrFatalEn (0xF8070140, bit 8) and ErrCtrl/SerrNonFatalEn (0xF8070140, bit 8) CSR bits.

CPC945 is in some ways a hybrid between a device and a bridge. It has a device header, with some bridge functions tacked on in a register called the Bridge Control register. However, this register is only tangentially related to the Bridge Control register in a bridge header, since standard software won't be looking for it, it isn't in the correct location, and has a slightly non-standard layout. CPC945 does, however, implement a standard HT host capability block.

CPC945 doesn't implement a SerrEn bit, because it doesn't really have a primary interface to propagate SERRs to; it also does not implement the SigdSerr log bit, for the same reason. CPC945 does implement the DetSerr bit, which is located in the BrCtrl register. The DetSerr bit is set whenever the link is sync flooded by a downstream HT device. When CPC945 sync floods the link, the DetSerr bit is not changed.

Sync flooding of the link by CPC945 requires having the appropriate sync flood enable bit set for the particular error condition, but no global enable. The ErrCtrl/SerrFatalEn and SerrNonFatalEn are implemented, and can be used to map detected sync floods to interrupts. In general, one of them should always be set so the host can detect a link failure.

6.10 HyperTransport Registers

The registers listed in *Section 12.12 HyperTransport Registers (HT1)* on page 614 control the HyperTransport Host bridge logic found in the CPC945 design. The layout of HyperTransport registers is intended to comply as much as possible with the governing specifications (*PCI Local Bus Specification, Revision 2.2 and HyperTransport I/O Link Specification, Revision 1.04*).

The set of registers described here is composed of three sub-sets of registers, the HyperTransport Header, the HyperTransport Interface Capabilities Block, and implementation specific registers. The header portion of the register block is declared as a Device Header. One required deviation from the Device header format is the addition of the *“Bridge Control Register (BrCtrl)” on page 636* at address 0xF8070300.

The Bridge Control Register is used to specify the error and reset behavior of the HyperTransport link connected to the host (primary interface). The HyperTransport Interface Capabilities section of the register block is used for configuration and status of the HyperTransport specific portions of the interface. The final section of registers controls items that fall outside the scope of the PCI and HyperTransport specifications. The top view of the HyperTransport register block is:

0xF8070000 - 0xF80700F3h:	Device Header
0xF8070100 - 0xF80701F3h:	Capabilities Block(s)
0xF8070200 - 0xF80703F3h:	CPC945 Specific Registers
0xF8070500 - 0xF80705F3h:	Performance Registers

7. DDR2 Memory Controller

7.1 Feature Summary

Following are features of the CPC945 Memory Controller:

- DDR2, 533 MHz and 400 MHz.
- 144-bit data bus (ECC) or 128-bit data bus (no ECC).
- JESD79-2 JEDEC 256 Mb, 512 Mb, 1 Gb and 2 Gb chip sizes and ×4, ×8 and ×16 organizations supported.The ×8 and ×16 chips might be mixed. When using ×4 chips, all chips must be ×4.
- Support for 8 ranks. 8 DIMMs max (4 double-sided pairs).
- 64 GB max capacity using 4 pairs of 8 GB double-sided DIMMs. (Only 62 GB addressable due to 2 GB I/O hole.)
- ECC supports single symbol (4-bit) correction and double symbol error detection. Chip kill correction with ×4 chips only.
- ECC scrub supported.
- Programmable SDRAM page policies (leave open, leave close), address interleave modes, and chip select decodes (optional “striping” across ranks) allow SDRAM page accesses to be optimized as appropriate for system configuration (for example, 1-way versus 4-way MP).
- Memory accesses are reordered to keep SDRAM pages open as long as possible (for low latency and maximum bandwidth). A variety of controls on the internal reorder queues (queue size, queue lookahead depth, read versus write selection, read aging, write high watermarks) allow the system to tune for high probability of hitting an open page for commonly accessed pages while not starving/blocking accesses to infrequently accessed pages.
- Programmable SDRAM timing parameters (CL, tRAS, tRP, tRC, and so on).
- Programmable refresh rate. Deferred refreshes supported.
- Support for dynamic power management (CKEs to SDRAMs are disabled after a period of inactivity).
- Support for self refresh (clocks to SDRAMs can be slowed or stopped and SDRAMs will retain their data).
- Programmable delay of SDRAM control signals (1/2 CK increments).
- Vernier control of CK, DQS, and data control signals (approximately 25 ps increments).
- Calibration logic for measuring internal data load/unload timings.

7.2 Memory Controller Basics

The CPC945 DDR2 memory controller provides access to external DDR2 memories. It responds to memory write and read requests in the ranges 0x000000000 through 0x07FFFFFFF (0 to ‘2 GB-1’) and 0x100000000 through 0xFFFFFFFFF (4 GB to’ 64 GB-1’).

The memory controller maps requests to installed memory to account for the 2 GB I/O hole (from 2 GB to ‘4 GB-1’). For example, if 8 GB are installed, the controller will respond to requests in the 0 to ‘2 GB-1’ address range, making use of 2 GB of the installed memory, and requests in the 4 GB to ‘10 GB-1’ address range, making use of the remaining 6 GB of installed memory.

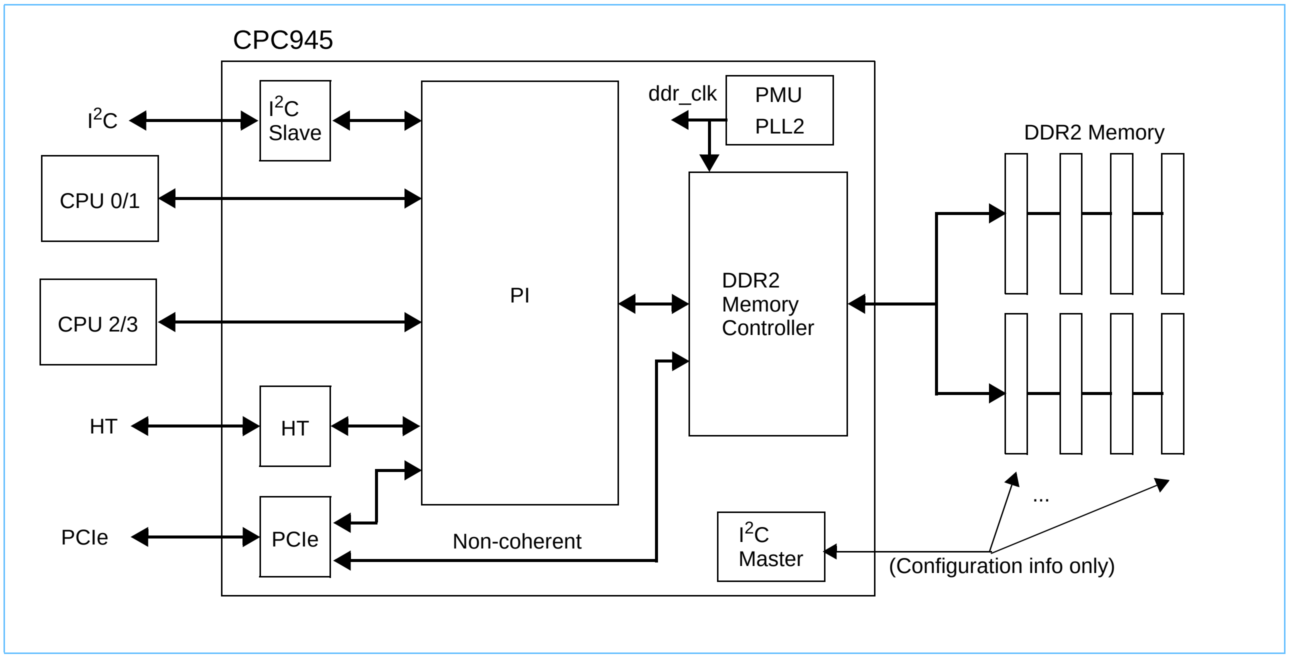
The memory controller programming registers occupy the address space 0x0F8002xxx.

As shown in *Figure 7-1*, the memory controller responds to memory requests from two sources:

- PI. These accesses are the aggregation of all coherent memory requests: the CPUs, HT, PCIe coherent, and debug (I^{2}C).
- PCIe non-coherent.

(Also, the memory controller responds to internally generated Refresh and Scrub requests.)

Figure 7-1. DDR2 Memory Controller



In general the memory controller contains all the CPC945 functions required for DDR2 support except:

1. Setting the clock frequency. This function is provided by PLL2 in the Power Manager, which generates the CPC945 “core” interconnect clock used by the memory controller and other units in the CPC945.
2. Reading Serial Presence Detect (SPD) data. The SPD ports on the memory DIMMs are I^{2}C slaves, and are typically accessed using a port of the CPC945 I^{2}C Master (or some other I^{2}C master).

7.3 Memory Configurations

The memory controller has a 144-bit data bus (128 for data, and 16 for check bits if ECC is used). There are 3 configurations for attaching memory to this bus:

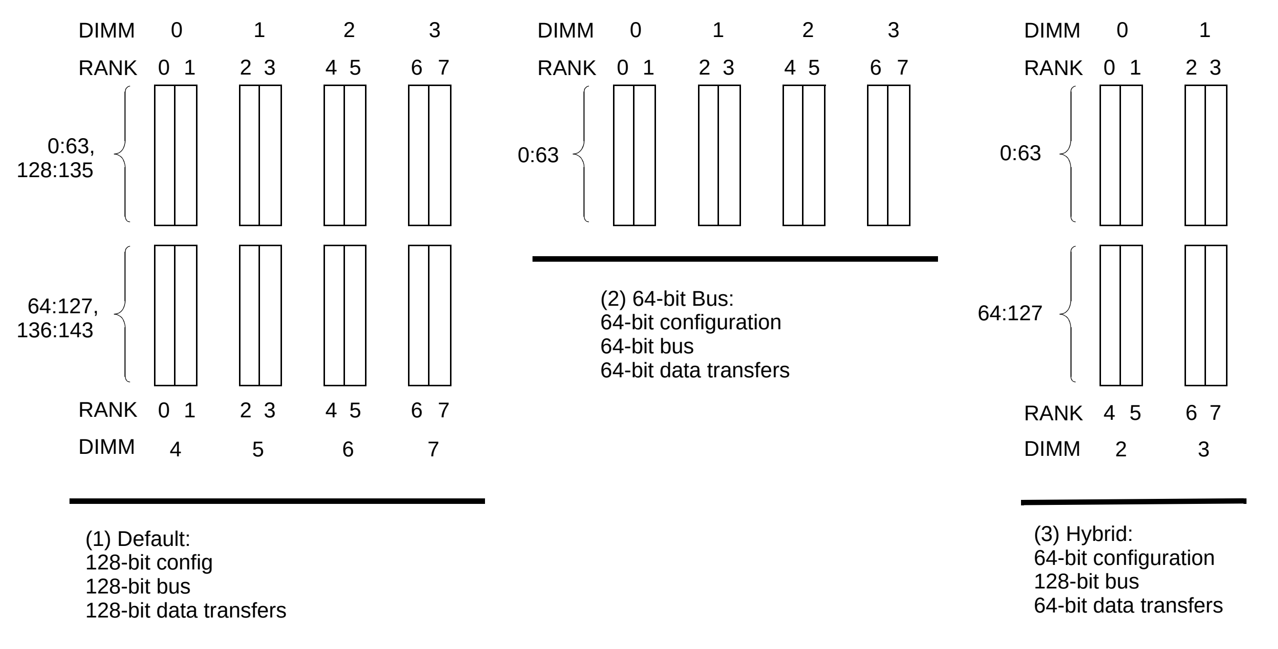
1. 128-bit configuration, 128-bit bus. This is the default. Pairs of DIMMs are required, either pairs of 64-bit DIMMs for non-ECC operation, or pairs of 72-bit DIMMs for ECC. Data is transferred 128-bits per beat.
2. 64-bit configuration, 64-bit bus. Single 64-bit DIMMs are used, connected to data bus bits 0:63. ECC is not supported. Data is transferred 64-bits per beat.
3. 64-bit configuration, 128-bit bus. Single 64-bit DIMMs are used. Half of the supported DIMMs attach to bits 0:63 and half attach to bits 64:127. ECC is not supported. Data is transferred 64 bits per beat.

The configuration is programmed using the 64BitCfg and 64BitBus bits in the MemBusConfig Register (0xF80022D0). Setting 64BigCfg=0 and 64BitBus=1 is an illegal combination.

In all 3 configurations, a maximum of 8 ranks is supported. Therefore the two 64-bit configurations have 1/2 the maximum amount of memory that can be attached, compared to the 128-bit configuration (as well as 1/2 the bandwidth).

To support the default 128-bit operation the memory controller data path and the internal buses that connect the memory controller to PI, HT and PCIe are all 128-bits wide. When configured for 64-bit operation, the memory controller assembles every two 64-bit external transfers into 128-bit internal transfers, and data is transferred on the internal buses every other clock.

Figure 7-2. External Memory Configurations



7.4 Supported Memories

7.4.1 Sizes

The CPC945 memory controller supports the ×4, ×8 and ×16 variations of the 256 Mb, 512 Mb, 1 Gb, and 2 Gb chips identified in the JESD79-2 DDR2 JEDEC Standard. 4 Gb chips are not supported.

With the 128-bit configuration, 8 ranks, and 2 Gb chips organized ×4 (32 chips/rank = 8 GB/rank) there is a theoretical maximum of 64 GB installed memory that can be supported. (But because of the 2 GB I/O hole, if 64 GB is installed only 62 GB can be accessed.)

Table 7-1. Supported Memory Sizes.

Chip Size	Organization	64-bit Configurations		128-bit Configuration	
		Single Rank DIMMs	Double Rank DIMMs	Single Rank DIMMs	Double Rank DIMMs
256 Mb	16 Mb x 16	128 MB	256 MB	256 MB	512 MB
256 Mb	32 Mb x 8	256 MB	512 MB	512 MB	1 GB
256 Mb	64 Mb x 4	512 MB	1 GB	1 GB	2 GB
512 Mb	32 Mb x 16	256 MB	512 MB	512 MB	1 GB
512 Mb	64 Mb x 8	512 MB	1 GB	1 GB	2 GB
512 Mb	128 Mb x 4	1 GB	2 GB	2 GB	4 GB
1 Gb	64 Mb x 16	512 MB	1 GB	1 GB	2 GB
1 Gb	128 Mb x 8	1 GB	2 GB	2 GB	4 GB
1 Gb	256 Mb x 4	2 GB	4 GB	4 GB	8 GB
2 Gb	128 Mb x 16	1 GB	2 GB	2 GB	4 GB
2 Gb	256 Mb x 8	2 GB	4 GB	4 GB	8 GB
2 Gb	512 Mb x 4	4 GB	8 GB	8 GB	16 GB

7.4.2 Speeds

The memory controller supports DDR2 devices with two data transfer speeds: 533 MT/s and 400 MT/s. These devices are designated by JEDEC as DDR2-533 and DDR2-400 for chips, and PC2-3200 and PC2-4300 for DIMMs. These speeds provide the peak bandwidths given in *Table 7-2*.

Table 7-2. Memory Bandwidths (Peak).

Speed	64-bit Configurations	128-bit Configurations
533 MT/s	4.264 GB/s	8.528 GB/s
400 MT/s	3.2 GB/s	6.4 GB/s

7.4.3 DDR2 Features

Following are DDR2 features:

- CL (CAS Latencies) of 3, 4, and 5 are supported.
- ODT (On Die Termination) is supported. See *Section 7.25.7 ODT - On Die Termination* on page 225.
- Differential DQS/$\overline{\text{DQS}}$ is supported for ×8 and ×16 memories. For ×4 memories only single-ended DQS is supported; the I/O circuits providing the differential DQS and $\overline{\text{DQS}}$ signals for every 8 DQ signals is configured (under program control) to provide two single-ended DQS signals, one each for every 4 DQ signals.
- Posted CAS operation is not supported. AL (Additive Latency) must be set to 0.
- For the 128-bit configuration the burst length is always 4. For the 64-bit configurations the burst length is always 8.
- DM (Data Mask) is *not* supported.

7.4.4 DIMMs

Following are the restrictions on DIMM characteristics:

- When ECC is enabled 72-bit wide DIMMs are required; otherwise 64-bit wide DIMMs are used.
- The memory controller supports both registered DIMMs and unbuffered (no register) DIMMs.
- Both single-sided and double-sided DIMMs are supported. These terms refer to DIMMs with one rank (one select) and two ranks (two selects).

  (The distinction is made because DIMMs using ×4 chips use both sides of the DIMM even when they only have one rank. DIMMs using ×4 chips with 2 ranks use both sides, with stacked chips.)

  (In general, DIMMs using ×8 and ×16 chips are physically single-sided when populated with one rank and double-sided when populated with two ranks.)
- The speeds of the DIMMs might be mixed. When this is the case, the PLL2 frequency must be set to match the speed of the slowest DIMM, and the memory controller must be programmed to use that slowest speed.
- CAS Latencies of the DIMMs might be mixed. When this is the case, the memory chips and the memory controller must be programmed to use the largest CAS Latency in the mix.
- The installed memory must be all registered DIMMs or all unbuffered DIMMs; they cannot be mixed.
- For the 128-bit configuration, the 2 DIMMs of a given pair must be the same:
  - Both DIMMs must have the same chip size and organization.
  - Both DIMMs must be single-sided or double-sided.
- Single-sided and double-sided DIMMs might be mixed with the 64-bit configurations. Single-sided pairs and double-sided pairs canbe mixed with the 128-bit configuration.
- When using ×4 chips all memories must be ×4, due to the requirement that ×4 chips require the IOs to be set to single-ended operation, one DQS per 4-bits = one DQS per chip for a given rank.
- ×8 and ×16 chips can be mixed, since ×8 chips have one DQS/$\overline{\text{DQS}}$ pair = one DQS/$\overline{\text{DQS}}$ pair per 8-bits, while ×16 chips have two DQS/$\overline{\text{DQS}}$ pairs = one DQS/$\overline{\text{DQS}}$ pair per 8-bits for a given rank.

7.5 Clocks

The memory controller uses 2 clocks: the ddr_clk from PLL2 (533 MHz), and mem_clk, where mem_clk = ddr_clk/2 (266 MHz).

The ddr_clk frequency generated by PLL2 is set in the Clock Control Register (0x0F8000800) in the Power Manager. Although 8 different clock frequencies can be generated only 533 MHz and slower are supported.

The half speed mem_clk is used to clock the circuitry generating the address and control signals to memory. It is also used to provide the clocks to memory (CK/$\overline{\text{CK}}$), and the data strobes (DQS/$\overline{\text{DQS}}$) for writes.

The full speed ddr_clk is used for the bulk of the data path. The double data rate transfers on the external memory bus are transferred to/from the data path pipeline running at the ddr_clk rate.

7.6 Data Transfers

Internal to the CPC945 the memory controller transfers data to/from PI, PCIe and HT on 128-bit data buses clocked at the ddr_clk speed (for example, 533 MHz).

(The PI, PCIe, and HT “cores” all run at different speeds but are required to speed match to the ddr_clk frequency for their internal buses that attach to the DDR2 memory controller core. For this reason the ddr_clk frequency set by PLL2 is also called the “core” clock frequency.)

PI, PCIe and HT can request reads and writes to memory with transfer sizes of:

- 1, 2, 3, 4, 5, 6, 7, or 8 bytes
- 1 quadword (16 bytes)
- 2 quadwords (32 bytes)
- 4 quadwords (64 bytes)
- 8 quadwords (128 bytes)

Note that a quadword is transferred using a single beat on both the internal and external 128-bit data buses.

For the 128-bit configuration the memory controller translates these requests to external DDR2 memory transfers as follows:

- An 8 quadword request is transferred as two 4-beat back to back bursts.
- A 4 quadword request is transferred as a single 4-beat burst.
- For reads, any request of less than 4 quadwords is handled as a 4-beat burst from memory (64 bytes). Bytes that were not requested are ignored by the requestor (PI, PCIe, HT).
- For writes of less than 64 bytes, the memory controller reads a 4-beat burst from memory, substitutes the write bytes into this 64-byte quantity, then writes the updated 64 bytes back to memory using a 4-beat burst.

For the 64-bit configurations 8-beat bursts are used instead of 4-beat bursts.

Maximum use of the DDR2 data bus is achieved with 64-byte and 128-byte transfers. Since CPU requests to memory are usually 128-byte cache line reads the bandwidth between the CPUs and memory comes closest to optimum.

If I/O requests are predominately less than 64-bytes then DDR2 bandwidth will be degraded. For example, if a graphics card performs a series of back-to-back 32-byte transfers, half of the bandwidth to DDR2 memory will be wasted.

7.7 Operational States

7.7.1 Power On Reset

After powering on and resetting the CPC945, the following conditions apply to the memory controller:

- The memory controller will be clocking at the default PLL2 speed, which is 400 MHz. To run the memories at a different speed the PLL2 speed needs to be changed, as described in *Section 12.5.7 PLL2 Control Register* on page 350.
- The CKE and ODT outputs are low (required by JEDEC).
- The CK clocks to the memories are disabled. They need to be enabled (*Memory Mode Control Register (MemModeCntl)* on page 483, CK_On bit).
- The registers are at their default values. In general, most of the registers will need to be programmed to match the external DIMM types and configurations, the memory speed, board delays, desired memory access and queue options, desired performance versus power-savings options, and so on.
- A JEDEC-specified initialization sequence needs to be supplied to the memories. This is done by programming and running the Memory Programming Control (*Section 7.9* on page 147).

  The first steps of the initialization sequence (after clocks are running) are supplied automatically by the memory controller: the CKEs are asserted, NOP commands are issued, and a delay of at least 400ns is applied. At that point the remainder of the initialization sequence is supplied by the Memory Initialization Registers. See *Section 7.10 Memory Device Initialization*.
- If ECC is desired, enable it now (*Section 7.19* on page 191).
- If desired, memory contents can be filled with a desired pattern and scrubbed of soft errors using the Scrub facility (*Section 7.20* on page 208).
- At this point the memory controller is ready to accept read and write requests. Although its arbiter will accept requests prior to this point, software is required to ensure that no unit (the CPUs, PCIe, HT, I^{2}C slave) attempts to access external SDRAM memory until the above initialization steps have been performed.

  By default once the CKEs go high they stay high (until or unless the memory controller issues Enter Self Refresh. See *Section 7.7.4 Self Refresh*). However, as discussed in *Section 7.7.3.2 Power Management During Normal Operation*, dynamic CKE can be enabled, which causes the CKEs to be deasserted by the memory controller for ranks that are not being accessed.

7.7.2 Memory Controller Bring-up Summary

The following sequence is an overview for initializion of the memory controller. The first 3 steps are usually done well in advance of memory controller bringup, when the chip is brought out of reset and all clock speeds are set to desired values.

1. Hold off memory accesses.
2. Read the SPD ports of the DIMMs; determine which banks are occupied, what the memory chip types are (size, organization), speeds, CAS latencies, and so on.
3. Set PLL2 frequency to the speed of the slowest DIMM.

Memory Controller bring up starts here.

4. Write the DIMM timing registers [RASTimer0, RASTimer1, CASTimer0, CASTimer1 (0xF8002030 - 0xF8002060)] with values based on chip types, the speed of the slowest DIMM, and the largest CAS latency value.
5. Write the Memory Refresh Control Register (0xF8002070) with values based on refresh policy and the largest chip size.
6. Write the DIMM configuration registers Dm0Cnfg, Dm1Cnfg, Dm2Cnfg, Dm3Cfg (0xF8002200 - 0xF8002230) and UsrCfg (0xF8002290) with values based on the DIMM bank occupancy, sizes and types.
7. Write the Memory Arbitration Weight Register (0xF8002280) with values based upon CPU versus I/O bandwidth requirements.
8. Write the operational control registers (queue sizes, aging values, arbitration algorithm, and so on.) MemRdQCnfg (0xF80022A0), MemWrQCnfg (0xF8002270), MemQArb (0xF80022B0), and MemRWArb (0xF80022C0) with values based on prior performance characterization data.
9. Write the memory bus control registers [MemBusConfig (0xF80022D0) and MemBusConfig2 (0xF80022E0) and the ODT Control Register (0xF80023A0)] with values based on clock speed, types of DIMMs, support components, prior Spice analysis and lab characterization data.
10. Write the IO Pad Control Register (0xF80029A0) strength control bits, termination value bits and termination control bits with values based on prior Spice analysis and lab characterization data. Set the MCSE_dqs (Mode control Single Ended) bit to 1 (single ended) if the DIMMs use ×4 chips, otherwise set to 0 for differential operation.
11. Write the MemPhyModeCntl Register (0xF8002880) with control and address clock adjust values based on prior Spice analysis and lab characterization data. Set the Half-bit Delay Override Enable bit to ‘0’ (no override).
12. Write the CK Control Registers (0xF8002890 and 0xF80028A0),
    Write Strobe Control Registers (0xF8002800 - 0xF8002830, 0xF8002900 - 0xF8002930, 0xF8002980, 0xF8002A00 - 0xF8002A30, 0xF8002B00 - 0xF8002B30, 0xF8002B80),
    Read Data Strobe Control Registers (0xF8002840 - 0xF8002870, 0xF8002940 - 0xF8002970, 0xF8002990, 0xF8002A40 - 0xF8002A70, 0xF8002B40 - 0xF8002B70, 0xF8002B90) and
    RstLdEnVerniersC0-C3 Registers (0xF890028D0, 0xF80029D0, 0xF8002AD0, 0xF8002BD0) with values based on prior lab characterization data.
13. ***Clocks on:*** (If PLL2 is not locked wait until it is locked.) Write the Memory Mode Control Register with CK_on set to ‘1’ to enable the CK clocks to the memories.

14. ***Clocks stable:*** Guarantee that 200 µs has passed since the CK_on bit was set to ‘1’ in the Memory Mode Control Register. A way to do that is to read the Memory Mode Control Register (which forces the write, if the write was pending), and then wait 200 µs.
15. If there are external CK buffer chips with PLLs, wait long enough such that the outputs of these chips have been stable for at least 200 µs to the inputs of the memory chips. Also if the registered DIMMs are used, wait long enough for their PLLs to have been stable for at least 200 µs.
16. ***Memory chip initialization:*** Write the Memory Initialization Registers (0xF8002100 - 0xF80021F0) with values that perform the JEDEC memory initialization sequence and set the EMRS(s) and MRS. For the MRS use the same CAS latency value that was used for programming the RAS timer and CAS timer registers. For the EMRS(1) use Rtt and a value of ‘00’ if ODT is disabled in the ODT Control Register; otherwise set to ‘01’ (75 ohm) or ‘10’ (150 ohm) based on prior Spice analysis and lab characterization data. Also for the EMRS(1) if ×4 memory chips are used set DQSEn_ to ‘1’ for single ended DQS operation; otherwise set to ‘0’ for differential operation. Finish with EMRS(2) values that set the OCDs to their default calibration.

    Write values to the MRSRegCntl Register (0xF80020C0) that replicates the bits of the last MRS in the initialization sequence. Write values to the EMRSRegCntl Register (0xF80020D0) that replicates the bits of the last EMRS(1) in the initialization sequence.
17. Write the Memory Programming Control Register (0xF80020B0) with the InitStart set to ‘1’ to kick off memory initialization sequence.
18. Poll (read) the Memory Programming Control Register until the InitCmplt bit has a value of ‘1’.

At this point the controller and the memories are initialized and the memories can be written and read.

The values in memory will be garbage at this point. If ECC operation is desired (and supported with 72-bit DIMMs), then ECC must be enabled, and the garbage values replaced with good data and check bits. The following steps are simplified, not taking into account the masking or unmasking of errors and so on:

19. (ECC only.) Continue to hold off memory accesses from the CPUs and I/O.
20. (ECC only.) Write the Memory Check Control Register (0xF8002440) with the ECC_EN bit set to ‘1’ to enable ECC.
21. (ECC only.) Write all of populated memory with data, using 64-byte or 128-byte writes so that only pure writes are performed (no Read-Modify-Writes). There are several ways to do this. One way is to:
    a. Write the memory start and end locations to the Memory Scrub Range Start and End Registers (0xF8002410 and 0xF8002420). Write any desired data pattern to the Memory Scrub Pattern Register (0xF8002430).
    b. Write the Memory Scrub Control Register (0xF8002400) with SCRUB_MOD set to ‘11’ to perform an Immediate Scrub with Fill.
    c. Poll (read) the Memory Scrub Control Register until the SCRUB_MOD bits read as ‘00’ (no scrub activity). At this point all of memory has its data and check bits written to values that do not produce ECC errors when read (assuming all good bits and timings).
22. (ECC only, optional.) Typically at this stage a diagnostic routine reads all of the memory to verify that in general reads can be performed without errors, and to log any errors that do occur. Based on these results, the diagnostic routine can do a number of things. For example, do a number of scrubs to determine if the failures are soft or hard, tell the operating system not to use certain address ranges, and so on.

23. (ECC only, optional.) If desired, the CPC945 memory controller can be programmed to do background scrubbing. If it is not already done, write the memory start and end addresses to the Memory Scrub Range Start and End Registers. Write the Memory Scrub Control Register with SCRUB_MOD set to '01' for background scrubs, and with the SI (Scrub Interval) bits set to the desired scrub interval timing.

If ECC is used, at this point the controller and the memories are initialized, the memories have good data and check bits, and the memories can be written and read.

7.7.3 Normal Operation

Once the memory controller has been initialized as described above, the memory controller performs writes and reads as requested (by the CPUs, PCIe (coherent and non-coherent), HT and the I^{2}C slave).

7.7.3.1 Page Access

DDR2 SDRAMs have the characteristic that each rank is organized into banks (either 4 or 8, depending on the chip size) and that a given bank is organized into rows and columns. There is an overhead for accessing a row. Once a row has been accessed (it has been opened), there is only the column access time (CAS latency) to access other columns in that row. If it is desired to access a different row the first row must first be closed, which has some overhead, and then the second row is opened, which again has some overhead.

The amount of data that can be accessed for a given row is often called a page.

The memory controller attempts to optimize memory latency and bandwidth by minimizing page openings and closings. It does this by placing requested reads and writes into buffers called, the read reorder queue and the write reorder queue.

- Entries in these queues that are directed to open pages are executed by the memory controller before those that require page closings and openings.
- Look-ahead is performed in the queues to enable opportunistic page openings (precharges) in otherwise unused memory clock cycles. This allows the overhead of opening a page in a new bank to be hidden (overlapped) while executing an access to a current bank.

Also, for lower latency, reads are favored over writes.

The options to control queue operation and the command arbiter that selects entries from the queues are described in *Section 7.17* and *Section 7.18*.

There are several options that control how the addresses of incoming requests are mapped to memory ranks and banks. These options can optimize page access depending on the addressing characteristics of the CPUs and I/O requests. (For example, bursts to the same address locale for uni-processor applications versus scattered accesses due to multi-processor access.) These options are described in *Section 7.14*.

7.7.3.2 Power Management During Normal Operation

The memory controller supports several options for reducing power during normal operation:

- SDRAM Power Management. When the Dynamic CKE option is enabled, the clock enables to memory ranks that have been idle for 20 cycles will be deasserted. This will put the SDRAMs into a JEDEC-defined power down mode, which draws less current than when CKE is asserted.

  The trade-off is that when a new command is directed at a powered down rank (including refresh), after asserting CKE the memory controller must wait until the memory devices in that rank have powered up, before issuing the command. This will have a negative impact on performance for systems with low latency requirements.

  See *Section 7.25.6 Dynamic CKE* for additional information on dynamic CKE. See the discussion on A[12] = Active Power Down Exit Time in *Section 7.10.1 MRS Settings* for information on the performance/power savings trade-off when Power Down Mode is exited.
- Data Bus I/O Driver Power Management. The I/O drivers for the DQ and DQS signals can be set to tri-state when there are no read or write transfers, which reduces the current draw of these I/O. The internal terminating resistance of the receivers are also turned off. The trade-off is that when new writes and reads are requested, the memory controller has to wait for the buses to come out of tri-state (for writes) and for the receivers to turn on their terminators (for reads), which has a negative impact on performance.

  See *Section 7.22.4 Bus Driving* for additional information on this option.

In addition, the I/O Pad Control register has options which control the strength of the off chip drivers and the internal terminations of the off chip receivers, which affect I/O current draw. Power savings can or cannot be achievable using these options, depending on the electrical analysis of the external wiring.

7.7.4 Self Refresh

DDR2 SDRAMs support a power-savings mode in which operations to memory are halted and the clocks to memory are stopped. Minimal power is dissipated by the memories yet they maintain their data contents. Later on the clocks can be resumed and memory operations can begin again, with the data contents preserved from when the clocks were stopped. This is called Self Refresh mode.

The CPC945 memory controller supports Self Refresh in two scenarios. In both scenarios the memory controller is instructed to enter Self Refresh. The difference in the two scenarios is the exit from Self Refresh:

- In one scenario the CPC945 is not powered down (and not reset). The memory controller retains its register settings. The internal clocks can be stopped, and restarted before exiting SDRAM Self Refresh.
- In the other scenario, the CPC945 is powered down. When it is powered back up again, the memory controller is in its Power On Reset condition as described above in *Section 7.7*, and all of the CPC945 registers, including those of the memory controller, will have to be programmed. The initialization steps outlined in *Section 7.7.1* will have to be re-run, with these exceptions:
  - Instead of programming the full power on memory initialization sequence, a dummy sequence is used to bring the SDRAMs out of Self Refresh.

  Note: The CKE signals to the SDRAMs must stay low during Self Refresh. If the CPC945 stays powered up the memory controller will hold the CKEs low. But if the CPC945 is powered off some other means must be used to hold the CKEs low (for example, pull-down resistors to ground).

7.7.4.1 Self Refresh Entry

Before entering Self Refresh, all requests to the memory controller (from the CPUs, PCIe, HT, and I^{2}C) must be halted and enough time allowed for the requests to be executed by the memory controller.

It is also required that the XSR Delay bit in the MemModeCntl register (*Section 12.10.24 Memory Mode Control Register (MemModeCntl)* on page 483) be set = 1.

It is the Power Management Unit which instructs the memory controller to perform the Self Refresh Entry:

- When the external signal Suspend_Request is asserted, the power manager will perform handshakes with all units on the CPC945 to perform a variety of power savings actions. Among these, the memory controller is sent a signal instructing it to enter sleep mode. See *Section 11 Power Management and Clocks* for more information on CPC945 sleep mode.
- The memory controller does the following:
  - The Enter Self Refresh sequence is sent to the SDRAMs.
  - The CKEs are brought low and stay low.
  - The CKEs are tri-stated if the CkeTsEn bit in the MemModeCntl register is set.
  - The CK_On bit in the MemModeCntl register is reset to 0. This causes the CK clocks to the SDRAMs to stop running.
  - Both the CK and $\overline{\text{CK}}$ outputs are driven low. This is a JEDEC convention which instructs external PLLs (for example, on registered DIMMs) to be disabled.
  - The memory controller sends a signal to the power manager to indicate that the SDRAMs are in Self Refresh. This signal can be read as the SA (Self Refresh Acknowledge) bit in the Power Manger Debug register.

7.7.4.2 Self Refresh Exit from Chip Sleep

Following are conditions on self-refresh-exit from chip sleep:

- If the CPC945 was put to sleep by asserting Suspend_Request, then de-asserting Suspend_Request will cause the Power Manger to handshake with the various units to bring them out of sleep, including the memory controller.
- The following occurs:
  - The memory controller completes the handshake with the Power Manager Unit.
  - The Power Manager Unit re-enables the CK drivers.

The programmer is required to perform the following to complete the exit from Self Refresh.

- The Ck_On bit in the MemModeCntl register must be set to 1.
- Enough time must elapse for the CK clocks to the SDRAMs to become stable. This includes waiting for any external PLLs (on the board or on the DIMMs) to lock and present a stable clock to the SDRAMs.
- The XSR Delay bit in the MemModeCntl register must be set to 0.
  - This triggers the memory controller to raise the CKEs and send the commands to SDRAMs to Exit Self Refresh.
- There must be a wait of at least 200 mem_clk cycles. When the SDRAMs enter Self Refresh they disable their DLLs. When they exit Self Refresh they re-enable their DLLs. It is a JEDEC requirement that 200 cycles pass after the DLL is enabled, before read commands can be issued.

At this point it is permissible for the various requestors (the CPUs, PCIe, HT, I^{2}C) to resume write and read requests to memory.

7.7.4.3 Self Refresh Exit from Chip Power On

The following steps are the same as those from *Section 7.7.1 Power On Reset*:

- The memory controller will be clocking at the default PLL2 speed, which is 400 MHz. To run the memories at a different speed the PLL2 speed needs to be changed, as described in *Section 12.5.7 PLL2 Control Register* on page 350.
- The CKE and ODT outputs are low (required by JEDEC).
- The CK clocks to the memories are disabled. They need to be enabled (*Memory Mode Control Register (MemModeCntl)* on page 483, CK_On bit).
- The registers are at their default values. The bulk of the registers must be re-programmed with the same values they had prior to entering Self Refresh.

This step deviates from the normal power on reset:

- The Memory Programing Unit should be used to initiate a programming sequence, but the only reason for doing so is to take advantage of the fact that when a sequence is initiated (by setting the InitStart bit in the MemProgCntl register), the CKEs are asserted and NOP commands are issued (see *Section 7.9.4 First Use*). This is the sequence required by JEDEC to exit the SDRAMs from Self Refresh.

  The set of commands normally found in the MemInitRegs are not needed. MemInitReg 0 should be set to all 0’s, so that bit 0 of this register, the Enable bit, indicates that no commands are to be issued. (See *Section 7.9.1 MemInitReg Execution*.)

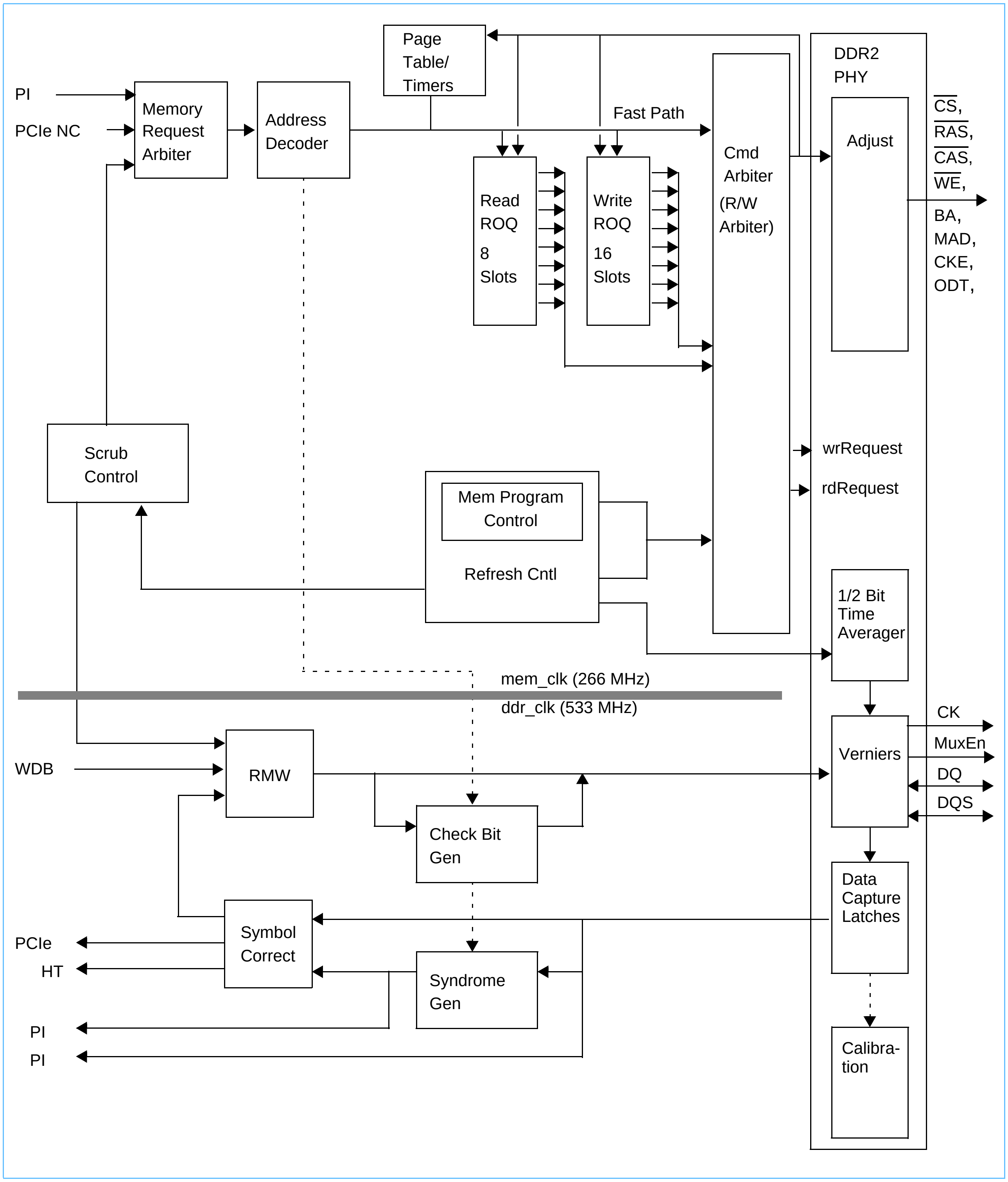
The next step is the same as exiting from chip sleep:

- There must be a wait of at least 200 mem_clk cycles. When the SDRAMs enter Self Refresh they disable their DLLs. When they exit Self Refresh they re-enable their DLLs. It is a JEDEC requirement that 200 cycles pass after the DLL is enabled, before read commands can be issued.

At this point it is permissible for the various requestors (the CPUs, PCIe, HT, I^{2}C) to resume write and read requests to memory.

7.8 Internal Operation Overview

Figure 7-3. Memory Controller Internals



A simplified diagram of the memory controller is given in *Figure 7-3*. The control logic occupies the top of the diagram, the datapath occupies the bottom, and the DDR2 PHY is shown on the right. (The DDR2 PHY, along with the I/O circuits, constitute the PHYsical interface to the external memories.)

7.8.1 Control

Write and read requests come into the memory controller from two sources:

- PI, which aggregates all coherent requests from the four CPUs, PCIe, HT, and I^{2}C debug
- Noncoherent requests from PCIe

In addition, the memory controller can internally generate Scrub requests.

The requests from these three sources are arbitrated by the ***Memory Request Arbiter***.

The winning request is sent to the ***Address Decoder***, which determines the rank, bank, row and column of the request. If the request is out of range the relevant information is latched in the Address Exception registers; otherwise the request proceeds to the Page Table.

The ***Page Table,*** also known as the ***Timers*** unit, keeps track of which pages in memory are open and closed. It also contains timers for each page which keep track of the number of clocks needed to wait before various SDRAM commands can be issued such as Activate, Read or Write, Precharge, and so on. The Page Table logic looks up the page (row) address of the incoming request and returns the required first SDRAM command and timer status for that page.

The read reorder queue holds read requests and the write reorder queue holds write requests until they are ready to be executed. They issue SDRAM command requests (precharge, activate, read, write) to the Command Arbiter.

The ***Command Arbiter*** (also known as the ***Read/Write Arbiter***) arbitrates among entries in the two reorder queues which are ready to execute, requests from the Refresh Controller. Also, there is a ***Fast Path*** into the Command Arbiter that allows Read requests to bypass the Read Reorder Queue.

The Command Arbiter unit issues the SDRAM commands (Activate, Read, Write, Precharge, Precharge All, Refresh, Self Refresh Entry/Exit, and so on.). These commands are sent to the DDR2 PHY where they can be adjusted (delayed) up to 3 ddr_clks before they are sent to the external memories.

The issued commands from the Command Arbiter unit are fed back to the Page Table and Reorder Queues so they can update their page open/close status and timer values.

Memory requests can have a size of 1 though 8 bytes and 1, 2, 4, and 8 quadwords (16, 32, 64, and 128 bytes). For incoming requests of size 64 bytes or less the memory Command Arbiter will always issue a single memory write or read command to memory, and 64 bytes will be transferred to/from the external memories. (A burst of 4 quadwords = 64 bytes for the 128-bit configuration, and a burst of 8 doublewords for the 64-bit configurations.) For requests of 128 bytes the Command Arbiter will issue two back-to-back writes or two back-to-back reads and 128 bytes will be transferred.

Entries taken from the Write Reorder Queue can be flagged as needing a Read-Modify-Write. The Command Arbiter does this as an atomic operation, issuing a Read command, allowing the data path to merge the partial write data, and then issuing the Write command.

The Command Arbiter unit is also responsible for generating CKE and ODT.

The ***Refresh Control*** unit shares control with the Command Arbiter of the memory controller:

- It receives control following chip reset.
- It contains the ***Memory Programming Control*** unit, used to issue the SDRAM initialization sequence and other sequences.
- Following the SDRAM initialization sequence, the Refresh Control unit generates periodic Refresh requests. If Scrub is enabled, it also kicks off a Scrub request to the Request Arbiter.
- The Refresh Control unit contains a state machine that mediates control for functions that are not normal reads and writes: entry from chip reset, enter and exit from Refresh, enter and exit from Self Refresh and running the Memory Programming Control Unit.

The ***Memory Programming Control*** unit is a general purpose unit which can send an arbitrary sequence of commands to memory:

- Following power on the external SDRAM memories must be issued a JEDEC-specified initialization sequence. The unit is used to generate this initialization sequence.
- The unit can also be used to generate arbitrary sequences of DDR2 commands (for example, for lab debug).
- The unit is also used in conjunction with the DDR2 PHY calibration logic, to generate the required SDRAM commands for calibration.

The ***Scrub Control*** unit is used to issue a special form of a Read-Modify-Write, and is used in conjunction with ECC. The Command Arbiter issues the Read, the datapath corrects the Read data (if the data has a correctable error), and the corrected data is written back when the Command Arbiter issues the Write.

The Scrub unit also has a fill mode in which just Writes are done, using write data supplied by a register in the Scrub unit.

7.8.2 Data

Write data from all sources - the CPUs, PCIe (coherent and non-coherent), HT, and I^{2}C debug - is aggregated in the Write Data Buffers (WDB) in the PI. When a memory write is performed the memory controller pulls the data out from the appropriate WDB. The write data timing is adjusted in the DDR2 PHY and then sent to memory on the DQ signals. The DDR2 PHY also generates the data strobes (DQS) and sends them to memory, with the rising and falling edges of DQS delayed by 1/2 bit time to center them within the DQ transitions.

If ECC is enabled the write data from the WDB is sent to the ***Check Bit Generator***, and the generated check bits are sent out to memory along with the data. The parity of the request address (from the Address Decode unit) can optionally be incorporated in the generation of the check bits.

When a memory read is performed, the incoming DQSs are delayed 1/2 cycle and the rising and falling edges are used to clock the incoming data into the ***Capture Latches***.

If ECC is enabled the captured data is sent to the ***Syndrome Generator***, which generates a non-zero vector (called the Syndrome) if an error is detected. Parity of the request address is also incorporated in the Syndrome, if this option is enabled.

For read data requested by PCIe or HT, the Syndrome is decoded by the ***Symbol Correction*** unit. If the error is correctable the data bits in error are flipped, otherwise the data is unchanged. The data is then sent to the requesting unit. The error status and the Syndrome, if non-zero, are latched in error registers along with the address of data containing the error.

To minimize latency from memory to the CPUs, the PI has its own Symbol Correction unit and read data going to the CPUs (or the I^{2}C slave) is corrected in the PI instead of the memory controller. This distinction is invisible to the programmer: the error status, error address, and syndrome are still latched in the memory controller.

7.8.2.1 Read-Modify-Writes

When a RWM (read-modify-write) is required the data is read from the memories as described above, corrected (if ECC is enabled), and then sent to the ***RMW*** unit to be merged with incoming write data. The merged data is sent to the memories. If ECC is enabled, new check bits are generated and sent along with the data.

It is possible that read data that is written back as part of a RMW will have an uncorrectable error detected during the read operation. In this case, a “Special UE” bit is incorporated in the generation of the check bits such that, when next read back, the generated Syndrome will indicate that this error was detected.

Memory requests of size 128 bytes and 64 bytes will never generate RMWs. Requests of size less than 64 bytes will always generate RMWs.

7.8.3 DDR2 PHY

The main purpose of the DDR2 PHY is to provide adjustable timings to the external interface.

As noted above in *Section 7.8.1*, the $\overline{\text{CS}}$, $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, $\overline{\text{WE}}$, BA, MAD, CKE, and ODT signals can be adjusted with a delay of 0 to 3 ddr_clks (0 to 7 clocks for ODT). These delays are generated by piping the signals through a set of latches clocked with ddr_clk and selecting the desired output.

A number of other signals have much finer tuning using delay ***Verniers***. These signals are generated by piping them through a chain of delay circuits and selecting the desired tap of the chain. The nominal granularity of the generated delay is approximately 20ps.

There are two miscellaneous verniers:

- mem_clk is provided on the external interface as 2 clocks, CK_A and CK_B. Each clock is individually adjustable.
- The timings of the MUX_EN outputs are individually adjustable for writes and reads.

The data path is divided into byte lanes. For each byte lane:

- For Write operations there is a vernier that controls the timing of the DQ out, OE (output enable of the DQ and DQS drivers), and DQS.
- For Write Operations the outgoing DQS and DQS OE has an additional vernier used to center the DQS within the DQ bit time window. When using ×8 and ×16 memories there is one of these verniers; when using ×4 chips there are two verniers, one for each DQS associated with a nibble.
- For Read Operations the incoming DQS has a vernier used to center the DQS within the DQ bit time window. As with the write DQS verniers, there is one vernier when using ×8 and ×16 memories and 2 verniers when using ×4 memories.

For Read operations there is another vernier:

- A vernier controls the time that clock logic for the ***Data Capture Latches*** is reset in anticipation of the DQS toggling from the external memory chips.

The verniers that shift DQS by 1/2 bit time use the output of the ***1/2 Bit Time Averager*** unit. This unit measures the time from one ddr_clk to another, which equals one bit time. Since the ddr_clk itself can be varying (from a spread spectrum reference clock), the delay is measured periodically (each time a Refresh is performed) and averaged. The averaged result is divided by 2 and sent to the verniers that shift DQS by 1/2 bit time. A programmable +/- offset can be applied to this control value.

The ***Calibration*** unit is used to measure margins involving read operations and the Data Capture Latches. Read strobe eye width, load time margin and unload time margin can be measured.

7.9 Memory Programming Control

Memory programming control consists of the block of MemInit registers (*Section 12.10.6 Memory Initialization Registers [0:15] (MemInitReg[0:15])* on page 462), which issue a sequence of SDRAM commands to the external memories, and the MemProgCntl register (*Section 12.10.3 Memory Programming Control Register (MemProgCntl)* on page 456) which initiates the sequence and controls other aspects of the sequencing.

7.9.1 MemInitReg Execution

There are 16 MemInit registers, all of them the same. When “executed”, bits 1:3 are sent to the $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WE}}$ output pins, bits 13:15 are sent to the BA[2:0] output pins, and bits 16:31 are sent to the A[15:0] (address) output pins (labeled as MAD[15:0] in the pin listing).

Memory programming is initiated by setting the InitStart bit in the MemProgCntl Register. When this is done, MemInit Register 0 is examined. If bit 0, the Enable bit is set, the contents of bits 1:3, 13:15, and 16:31 of that register will be sent out on the control and address outputs.

Chip selects will also be asserted, in sequence, every mem_clk (CK). If the InitRank bits in MemProgCntl = 0x00 then chip selects will only be generated for occupied ranks, as determined from the DmEn and SS bits in the DIMM configuration registers. (See *Section 7.14.3 Installed DIMMs*, *Section 7.14.4 Single-Sided/Double-Sided* and *Section 12.10.7 DIMM Configuration Registers*.) Otherwise the InitRank bits determine which chip selects are generated.

In parallel with sending the commands, a timer is started that will cause the controller will wait for a number of mem_clks, specified by the Delay field in the MemInit Register being used (bits 4:11). This delay is useful, for example, in generating the delays required by the JEDEC initialization sequence (tRP, tRFC, 400ns, 200 clocks). (Note that the actual delay is 2 more mem_clks than the programmed value. Also note that a portion of the delay is “hidden” during the time it takes to send out the chip selects.)

After the delay is met, MemInit register 1 is examined. If its En bit is set, it is used to send commands on the outputs, and its delay is processed. This process continues in sequence, until either the sequence hits a MemInit register with its En bit set to 0, or until all 16 MemInit Registers have been used.

The first MemInitReg with its En bit set to 0 determines the end of the sequence. For example, if MemInitRegs 0:9 have their En bits set to 1, MemInitReg 10 has its En bit set to 0, and MemInitRegs 11:15 have their En bits set to 1, the sequence will stop with MemInitReg 9 and MemInitRegs 11:15 will never execute.

7.9.2 Looping

An exception to the linear execution of MemInitRegs described above is the looping mechanism. If a MemInitReg is executed, and its Loop bit is set (bit 12), then the sequence will reset and the next MemInitReg to execute will be MemInitReg 0.

The memory programming control unit counts the number times a sequence loops. Looping will continue until one of the following occurs:

- A MemInitReg is encountered with its En bit set to 0. Meaning, a register write to that register was performed after looping started, to reset its En bit.
- No MemInitRegs in the sequence have their Loop bit set. Meaning, a register write was performed to any MemInitRegs that had their Loop bit set, to reset their Loop bit.
- The loop count is met. The loop count is specified in with the InitLoopCount bits in the MemProgCntl register.

Note: An InitLoopCount of 0 specifies an infinite loop. The only way to break the loop (short of resetting the CPC945) is by writing to the MemInitRegs to reset their En or Loop bits.

7.9.3 Termination

When a Memory Programming Control sequence is initiated, the InitCmplt bit in the MemProgCntl register is reset to 0. When the sequence terminates, the InitCmplt bit is set to 1, and the InitStart bit is reset to 0.

7.9.4 First Use

As noted in *Section 7.7.1 Power On Reset*, the CKE outputs to memory are low (deasserted) following reset, and the first use of the Memory Initialization Registers (the first time InitStart is set following reset) will immediately enable the assertions of the CKE outputs. NOP commands will be issued at that time, followed by a delay of 160 clocks before MemInitReg 0 is executed. The intent of these actions is to satisfy the JEDEC SDRAM initialization requirements of raising CKE, issuing NOP, and waiting 400ns before issuing the first operation command (Precharge All).

The CKEs and Chip Selects for the NOPs are staggered, the same as the execution of the MemInitRegs. The CKEs and Chip Selects that are generated are determined by the InitRank bits in MemProgCntl register and the DmEn and SS bits in the DIMM configuration registers, as described above in *Section 7.9.1*.

The delay of 160 clocks to guarantee a 400 ns delay is derived from the fastest possible DDR2 speed specified by JEDEC: 800 MHz (= 400 MHz mem_clk = 2.5ns tCK, times 160 = 400ns).

7.9.5 Auto Refresh

As discussed below in *Section 7.12 Refresh*, following reset the Refresh unit is inhibited from generating requests. If a MemInitReg sequence is initiated, at the termination of that sequence refresh requests will be enabled if the InitBlockAutoRef bit in the MemProgCntl register is not set. This is the default.

If a single MemInitReg sequence is used in memory bring up, to issue the DDR2 SDRAM initialization sequence, then the default setting of this bit should be used so that refreshes start after the SDRAMs are ready for normal operations. The default should also be used when the MemInitReg sequence is used to bring the SDRAMs out of Self Refresh, following a CPC945 power down.

If memory bring up involves two MemInitReg sequences, then InitBlockAutoRef should be set to 1 for the first sequence and to 0 for the second sequence. If this is done as just one combined sequence, then InitBlockAutoRef should be set to 0.

If the MemInitReg sequence is being used for lab bring up the user might or might not want to set InitBlockAutoRef, depending the debug operation of interest is being performed.

7.9.6 SDRAM Commands

The meaning of the SDRAM commands issued on $\overline{\text{RAS}}$, $\overline{\text{CAS}}$ and $\overline{\text{WE}}$ are specified by JEDEC and are reproduced in simplified form in *Table 7-3*. The bits are sent directly to the output pins without any inversion, so for the $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WE}}$ outputs the corresponding bits in the MemInitRegs should be set to 1 for the deasserted level (High) and they should be set to 0 for the asserted level (Low). For example, MemInitReg[1:3] = 100 means $\overline{\text{RAS}}$ = H, $\overline{\text{CAS}}$ = L, and $\overline{\text{WE}}$ = L, which is a SDRAM “Write” command.

Table 7-3. SDRAM Commands (Simplified).

Function	$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	BA[2:0]	A[15:11]	A[10]	A[9:0]
(E)MRS	L	L	L	BA	O P Code		
Refresh	L	L	H	-	-	-	-
Precharge	L	H	L	BA	-	L	-
Precharge All	L	H	L	-	-	H	-
Activate	L	H	H	BA	Row Address		
Write	H	L	L	BA	Column	L	Column
Write w/AP	H	L	L	BA	Column	H	Column
Read	H	L	H	BA	Column	L	Column
Read w/AP	H	L	H	BA	Column	H	Column
NOP	H	H	H	-	-	-	-

The entries in *Table 7-3* are only those SDRAM commands for which Chip Select is asserted, and CKE is high on the current cycle and in the previous cycle. Power Down Entry/Exit and Self Refresh Entry/Exit commands cannot be issued using the MemInitReg registers, since they require manipulation of CKE (the memory controller issues these commands where required).

7.10 Memory Device Initialization

The JEDEC Standard lists 12 steps to initialize DDR2 SDRAMs.

1. Apply power and maintain CKE and ODT low. During and following a CPC945 reset, CKE and ODT are held low. See the JEDEC Standard for VDD, VDDL, VDDQ, VTT, and Vref requirements.
2. Start the clock and maintain stable condition. As described in *Section 7.7.1 Power On Reset* the desired PLL2 frequency must be set, the clocks to the SDRAMs must be enabled with the CK_On bit of the MemModeCntl register must be set, and any external PLLs must be enabled.
3. Wait 200 µs after stable power and clock, then take CKE high and send a NOP command. As noted elsewhere, the first time that a MemInitReg sequence is initiated (the first time that InitStart in the MemProgCntl register is set), the memory controller takes CKE high and issues a NOP command to the SDRAMs.

   The programmer is required to guarantee the 200 µs time by waiting at least 200us from the time that the clocks are stable to the SDRAMs (including any time for PLL2 to settle, the board clocks to settle from the external clocks being enabled with CK_On in the MemModeCntl register and the PLLs on registered DIMMs to be enabled and settle) before writing to the MemProgCntl register to set the InitStart bit in the MemProgCntl register.
4. Wait 400 ns, then issue Precharge All. As described above in *Section 7.9.4 First Use*, following the raising of CKEs and the issuing of NOPs for Step 3, the memory controller will wait for 160 clocks, which satisfies the 400 ns requirement for any PLL2 frequency up to 800 MHz.

   The Memory Programming Unit will then execute the MemInitRegs. MemInitReg 0 should have a Precharge All command to finish this step, followed by the commands and delays for steps 5 through 12.

The following example shows how steps 4 through 12 of this memory initialization sequence can be generated with the MemInitRegs. This example assumes the following:

- DDR2-400 4-4-4 SDRAMs (400 MHz, CL = 4, tRCD = 4, tRP = 4)
- ×4 SDRAMs (single-ended DQS)
- 512 Mb SDRAMs (tRFC = 105 ns = 21 clocks)
- 128-bit Bus, 128-bit Cfg (Burst Length = 4)
- ODT is to be enabled, with a nominal termination of 75 ohms.

Note that the value entered in the Delay field = the desired delay - 2.

Table 7-4. Example DDR2 SDRAM Initialization Sequence.

JEDEC Step	Address	En_RCW_Delay_L_BA_Address	Comment
4	0xF8002100	1_010_00000010_0_000_0000010000000000	Precharge All, Wait tRP = 4 clocks
5	0xF8002110	1_000_00000000_0_010_0000000000000000	EMRS(2)
6	0xF8002120	1_000_00000000_0_011_0000000000000000	EMRS(3)
7	0xF8002130	1_000_00000000_0_001_0000010000000100	EMRS(1), $\overline{\text{DQS}}$ Disabled, Rtt = 75 ohms
8	0xF8002140	1_000_00000000_0_000_0001010**1**01001010	MRS: DLL Reset, tXARDS, WR = 3, CL = 4, Intrl, BL = 4
9	0xF8002150	1_010_00000010_0_000_0000010000000000	Precharge All, Wait tRP = 4 clocks
10	0xF8002160	1_001_00010011_0_000_0000000000000000	Refresh, Wait tRFC = 21 clocks
10	0xF8002170	1_001_00010011_0_000_0000000000000000	Refresh, Wait tRFC = 21 clocks

Table 7-4. Example DDR2 SDRAM Initialization Sequence.

JEDEC Step	Address	En_RCW_Delay_L_BA_Address	Comment
11	0xF8002180	1_000_11111111_0_000_0001010**0**01001010	MRS: Enable DLL, maintain settings, Wait 257 clocks
12	0xF8002190	1_000_00000000_0_001_0000011110000100	EMRS(1)
12	0xF80021A0	1_000_00000001_0_001_0000010000000100	EMRS(1)
	0xF80021B0	0_000_00000000_0_000_0000000000000000	Disable (end of sequence)
	0xF80021C0	0_000_00000000_0_000_0000000000000000	Don’t care
	0xF80021D0	0_000_00000000_0_000_0000000000000000	Don’t care
	0xF80021D0	0_000_00000000_0_000_0000000000000000	Don’t care
	0xF80021D0	0_000_00000000_0_000_0000000000000000	Don’t care

7.10.1 MRS Settings

When the MRS command is issued the contents of A[15:0] (MemInitReg[16:31]) are written into the Mode Register. BA[1:0] = 0 to specify MRS (versus EMRS).

- A[2:0] = Burst Length. For the 128-bit configuration BL = 4. For the two 64-bit configurations BL = 8. A[3] = Sequential/Interleaved access. A[3] must always be set to 1 = Interleaved.
- A[6:4] = CL (CAS Latency). In the example above CL = 4.

  Normally CL is the value supported by the memory device manufacturer (that is, the value in the SPD data on the DIMM. If the installed SDRAMs have a mixture of CL values, the *largest* CL value must be set.

  Note: There is a special case in which the memory controller cannot operate with CL=3: When commands are sent to the SDRAMs in a minimum number of clocks (unregistered DIMMs, no additional clocked delays (registers) on the board, no additional clocks added using the MemPhyModeCntl register), the write data to the DIMMs cannot be supplied in time if CL = 3. Therefore for this situation ***the minimum setting of CL is 4***.
- A[7] = Normal/Test Mode must always be set = 0 (Normal Mode).
- A[8] = DLL Reset = No/Yes. The MRS command is issued twice. The first time (step 8), A[8] = 1 to reset to DLL. The second time (step 11), all of bits of the MRS stay the same, but A[8] = 0 to enable the DLL.
- A[11:9] = Write Recovery for Precharge (WR). See the JEDEC Standard for setting this parameter. In general, WR = 2, 3, 4, 5 or 6 if the SDRAM speed is 266 MHz, 400 MHz, 533 MHz, 667 MHz or 800 MHz, respectively. So for this example WR = 3.
- A[12] = Active Power Down Exit Time (use tXARD or tXARDS). This parameter is significant for 2 cases:

  a. When Dynamic CKE is enabled, as explained in *Section 7.7.3.2* and *Section 7.25.6*, the SDRAMS of a given rank are put into Power Down mode when that rank has been idle for 20 cycles, and then brought out of Power Down mode when the rank is accessed (including refresh).

  b. When the SDRAMs exit from Self Refresh (*Section 7.7.4.2* and *Section 7.7.4.3*).

  In both cases, the delay to when a normal command can be issued is Fast (tXARD), with high transient power dissipation, or Slow (tXARDS) with lower transient power dissipation. Normally this trade-off is determined by the user, but there is a case when the setting is fixed: ***When both ODT operation and Dynamic CKE are enabled, A[12] must be set to 1 = tXARDS (Slow Exit).***

In this example, A[12] = 1 = Slow Exit because ODT is enabled (see EMRS, below).

- A[15:13] is required by JEDEC to = 0.

7.10.2 EMRS Settings

When the EMRS command is issued the contents of A[15:0] (MemInitReg[16:31]) are written into Extended Mode Register 3, 2 or 1, as determined by BA[1:0].

EMRS(3) and EMRS(2) are required by JEDEC to be set but to date these registers are not used, so A[15:0] = 0. EMRS(1) is new for DDR2.

- A[0] = DLL Disable/Enable. A[0] must be set = 0 = Enabled.
- A[1] = Output Driver Impedance Control = Normal/Weak. This setting is determined by electrical analysis (for example, Spice) of the board wiring, memory device model, and CPC945 receiver impedance (*Section 12.10.26 I/O Pad Control Register (IOPadCntl)* on page 487), and so on, for the DQ and DQS signals. This example sets A[1] = 0 = normal.
- A[6],A[2] = Rtt = ODT Disabled/75 ohm/150 ohm. If CPC945 ODT operation is disabled (*Section 7.25.7 ODT - On Die Termination* on page 225, *Section 12.10.16 ODT Control Register (ODTCntl)* on page 475), then A[6],A[2] must be set to 0 = ODT disabled on the SDRAM.

  If CPC945 ODT operation is enabled, then A[6],A[2] must be set to 01 = 75 ohms or 10 = 150 ohms. The setting is determined by electrical analysis of the DQ and DQS signals.
- A[5:4] = AL (additive latency). ***The CPC945 does not support any value of AL except 0, therefore A[5:4] must always be set = 0.***
- A[9:7] = OCD Program The user is required to set the SDRAM drivers to their default calibration. This is shown in step 12 of the example. First EMRS(1) is set with A[9:7] = 111 to enter OCD calibration and set the drivers to their default calibration. This is followed by an EMRS(1) with A[9:7] = 000 to exit OCD calibration.
- A[10] = $\overline{\text{DQS}}$ Enable/Disable. For systems with ×8 or ×16 chips set A[10] = 0 = $\overline{\text{DQS}}$ Enable. For systems with ×4 chips set A[10] = 1= $\overline{\text{DQS}}$ Disable.
- A[11] = RDQS Disable/Enable. Set A[11] = 0 = RDQS Disable.
- A[12] = Output Buffer Enabled/Disabled. Set A[12] = 0 = Output Buffer Enabled.
- A[15:13] is required by JEDEC to = 0.

In the example coded in *Table 7-4* EMRS(1) is issued 3 times. The values of A[9:7] = OCD Program should be set as shown in the example. This will cause the SDRAMs to use the OCD Calibration default.

The example shows the desired other values (Rtt, and so on) programmed on all three sets, but in actuality these values only need be set with the last EMRS(1).

7.11 MRS Register

With the use of the MemInitRegisters to set the MRSvalue in the SDRAM devices, there is not necessarily any indication in the memory controller of what this value is. (The sequence might differ from that in *Table 7-4* depending on varying scenarios, or might be destroyed by reprogramming the registers.) Therefore, the memory controller contains a register (*Section 12.10.4 Mode Register Set (MRS) Register (MRSRegCntl) and Extended Mode Register Set Register (EMRSRegCntl)* on page 457) that can be used to hold a copy of the value that was programmed into the SDRAMs.

The MRS register (0xF80020C0) is purely for the convenience of the programmer. The contents have no effect on memory controller operation. The MRS register is not loaded automatically; it is up to the programmer to write this register with the same value used in the initialization sequence (or any time the MemInitRegs are used to load the SDRAM MRS).

7.12 Refresh

Following power on reset, the Refresh unit is inhibited from generating requests. Requests are enabled when:

- An initialization sequence is kicked off by setting the InitStart bit in the MemProgCntl register, AND
- The initialization sequence finishes (the InitCmplt bit goes high), AND
- The InitBlockAutoRef bit in the MemProgCntl register is not set.

Once refresh requests start, they continue to run, independent of any subsequent initialization sequences, until the chip is reset.

Refresh is controlled by the MemRfshCntl register (*Section 12.10.2 Memory Refresh Control Register (MemRfshCntl)* on page 454).

The Refresh unit makes periodic requests to the Command Arbiter. When a refresh request is granted, the Refresh unit issues Precharge All commands to all ranks, to close all rows of all banks of all ranks. It then issues refresh commands to the ranks. Then, following a delay specified by the tRFC field in the MemRfshCntl register, it returns control to the Command Arbiter to allow reads and writes to continue. The tRFC field should be programmed, based on the frequency of mem_clk, with the number of clocks that satisfy the JEDEC Standard for tRFC.

To avoid excessive current surges, the Refresh unit staggers the Precharge All and refresh commands, one rank at a time. The commands are issued only to occupied ranks, as determined from the DmEn and SS bits in the DIMM configuration registers. (See *Section 7.14.3 Installed DIMMs*, *Section 7.14.4 Single-Sided/Double-Sided* and *Section 12.10.7 DIMM Configuration Registers*.)

The requests made by the Refresh unit to the Command Arbiter are based on a refresh period timer. This is a 13-bit counter which counts mem_clks, and indicates that a refresh period has elapsed when the counter equals the RefTime programmed in the MemRfshCntl register. (Only the high 9 bits of the refresh period are programmed.)

The Refresh Unit receives a “Queues Idle” signal from the Command Arbiter which indicates if the reorder queues have any commands that are ready to execute. When a refresh interval has elapsed, if this signal indicates that the queues are idle, the Refresh unit requests a refresh, which is immediately granted.

If the Queues Idle signal indicates that the queues are not idle, the action of the Refresh unit depends on the settings of the DeferRef bits in the MemRfshCntl register. If deferred refreshes are permitted, the Refresh unit will not request a refresh. Instead, it counts off another refresh interval. If the queues are idle this time the Refresh unit requests the refresh. If not, and another deferred refresh is permitted, then once again a refresh interval is counted off. This continues until either the queues are idle, or the maximum number of deferred refreshes has been hit, as set by the DeferRef count. When the maximum number of refreshes have been deferred, the Refresh unit requests a refresh regardless of the state of the Queues Idle signal. The Command Arbiter will honor this request as soon as it has finished any command in progress.

When a refresh request is made and granted, the Refresh unit will execute in series one refresh plus the number of deferred refreshes. If more than one refresh is performed, only one PrechargeAll command per rank is issued, followed by the refreshes.

The refresh period should be programmed to the smallest tREFI of all the memory devices (currently specified by JEDEC as 7.8 μs for all device types) based on the mem_clk frequency, even if deferred refreshes are programmed. JEDEC permits up to 8 refreshes to be deferred.

Each time a refresh period elapses, if Scrub is enabled, the Scrub unit requests a scrub operation, regardless if the refresh is deferred or not.

Each time a refresh is granted, the Delay Measurement Unit (1/2-bit time averager) updates its delay measurement value. Thus, if a refresh is deferred, the update is deferred. This is to avoid having the delay verniers in the DDR2 PHY change their delay values in the middle of a data transfer.

7.13 Memory Request Arbiter

The Memory Request Arbiter uses round-robin selection of the 3 memory requesters: PI, PCIe non-coherent, and Scrub. The arbiter is controlled with the Memory Arbiter Weight Register (MemArbWt 0xF8002280). This register contains weightings for the three sources.

If a requester’s weight is set to 0 it does not participate in the round-robin and only gets access when the other requesters are idle.

Requests from PI and PCIe non-coherent stay active until they are serviced. Scrub requests are different. If a Scrub request is made and not serviced by the time a second Scrub request is made, the first request is discarded.

7.14 Address Decode

The Address Decode unit is controlled by:

- the 64BitCfg and 64BitBus bits in the Memory Bus Configuration Register (MemBusCnfg 0xF80022D0)
- the DIMM Configuration Registers (Dm0Cnfg 0xF8002200, Dm1Cnfg 0xF8002210, Dm2Cnfg 0xF8002220, Dm3Cnfg 0xF8002230)
- the Memory User Configuration Register (UsrCnfg 0xF8002290)

7.14.1 64/128 Cfg/Bus

The 64BitCfg and 64BitBus bits in MemBusCnfg must be set to match the desired method of access, as given in *Table 7-5*.

Table 7-5. 64-Bit Configuration and 64-Bit Bus Settings.

64BitCfg	64BitBus	Memory Configuration
0	0	128-bit operation, 128/144-bit bus. DIMMs must be installed in pairs.
0	1	Illegal
1	0	64-bit operation, 128-bit bus. DIMMs 0,1 on bits 0:63, DIMMs 2,3 on bits 64:127.
1	1	64-bit operation, 64-bit bus. DIMMs 0:3 on bits 0:63.

7.14.2 DIMMs/DIMM pairs

The DIMM Configuration (DmxCnfg) and Memory User Configuration (UsrCnfg) registers control memory addressing on a DIMM basis for the two 64-bit configurations, and on a DIMM pair basis for the 128-bit configuration.

The discussion below will use “DIMM” in either case. For example, “DmEn is set to indicate a DIMM is installed” means that a *single* DIMM is installed using one of the 64-bit configurations, or a DIMM *pair* is installed for the 128-bit configuration.

7.14.3 Installed DIMMs

The DmEn (DIMM Enable) bit in the four DIMM Configuration Registers must be set to match the DIMMs that are installed/not installed.

- The DmEn bit is set to 1 if a DIMM is installed, and 0 if no DIMM is installed.
- If DmEn = 0 in a given DmCnfg register, then the remainder of bits in that register are don’t care.

7.14.4 Single-Sided/Double-Sided

The SS (Single-Sided) bit in the four DIMM Configuration Registers must be set to 1 if the DIMM is single-sided, and 0 if the DIMM is double-sided.

As discussed in *Section 7.4.4*, “single-sided” refers to DIMMs that have one rank (one Select) and “double-sided” refers to DIMMs that have 2 ranks (two Selects).

7.14.5 Chip Size and Organization

The MemMd (Memory Mode) bits in the four DIMM Configuration Registers must be set to match the types of memory chips used on the DIMM, as given in *Table 7-6*.

Table 7-6. Memory Modes.

Mem Mode [0:3]	Chip Size	Chip Width	Banks/ Chip	Bank Adrs Bits	Row Adrs Bits	Col Adrs Bits	Page Size/ Chip	64-bit config		128-bit config	
								Page Size/ Rank	Total Size/ Rank	Page Size/ Rank	Total Size/ Rank
0000	256 Mb	×16	4	2	13	9	1 KB	4 KB	128 MB	8 KB	256 MB
0001		×8		2	13	10	1 KB	8 KB	256 MB	16 KB	512 MB
0010		×4		2	13	11	1 KB	16 KB	512 MB	32 KB	1 GB
0100	512 Mb	×16	4	2	13	10	2 KB	8 KB	256 MB	16 KB	512 MB
0101		×8		2	14	10	1 KB	8 KB	512 MB	16 KB	1 GB
0110		×4		2	14	11	1 KB	16 KB	1 GB	32 KB	2 GB
1000	1 Gb	×16	8	3	13	10	2 KB	8 KB	512 MB	16 KB	1 GB
1001		×8		3	14	10	1 KB	8 KB	1 GB	16 KB	2 GB
1010		×4		3	14	11	1 KB	16 KB	2 GB	32 KB	4 GB
1100	2 Gb	×16	8	3	14	10	2 KB	8 KB	1 GB	16 KB	2 GB
1101		×8		3	15	10	1 KB	8 KB	2 GB	16 KB	4 GB
1110		×4		3	15	11	1 KB	16 KB	4 GB	16 KB	8 GB

7.14.6 Page Policy

The page policy, set by the PgPolicy bits in the UsrCnfg Register, is an option that can be set irrespective of the types of DIMMs installed. The page policy controls the setting of Column Address bit 10 (C10) when writes and reads are issued by the memory controller and is set by the PgPolicy bits in the UsrCnfg Register.

Although the Page Policy bits are found in the UsrCnfg Register, they are not involved in address decode. See *Section 7.17.4 Queue Page Policy* on page 182 for information on these bits.

7.14.7 Interleave Mode

The interleave iode, set by the IntrLvMd bit in the UsrCnfg Register, is an option that can be set irrespective of the types of DIMMs installed.

When set to 0 = SDRAM page, accesses to the banks of a given rank are interleaved on boundaries that are the size of the page (row) of the SDRAM being accessed. For example, the first row of *Table 7-23 Memory Mapping for Bank Interleaving at SDRAM Page Boundary (128-bit Configuration)* shows the address mapping of a 256Mb ×16 chip which uses a row size of 2**9 = 512 columns (addressed using 9 column bits). It can be seen that sequentially incrementing the low order system address bits will address an entire row (using system address bits 51:59) before a new bank is accessed (using system address bits 49:50).

When set to 1 = L2 Cache Line, accesses to a the banks of a given rank are interleaved on a processor L2 cache line size of 128 bytes. Given that 16 bytes are accessed at a time (with the 128-bit memory bus configuration), this means that accesses are interleaved every 128/16 = 8 columns. *Table 7-24 Memory Mapping for Bank Interleaving at Cache Line Boundary (128-bit Configuration)* shows that only 8 columns (using 3 column bits) are accessed before another bank is accessed.

7.14.8 Chip Select Mode

The CPC945 allows memory ranks to be “grouped” together such that sequential accesses are “striped” across the ranks within a group. Rank grouping is done on a per-DIMM basis, using 1 of 4 “Chip Select Modes.” The rank groupings are optional, but they can only be chosen if the installed memories support the desired grouping. The following combinations are supported:

- Any of the 4 DIMM pairs (128-bit configuration) or DIMMs (64-bit configuration), double-sided or single-sided, can be assigned CSMode = 0 = no ranks grouped. If double-sided, all of the front-side will be addressed before moving on to the back side.
- Any double-sided DIMM pair (128-bit configuration) or DIMM (64-bit configuration) can be assigned CSMode = 1 = two ranks are grouped. Accesses to the 2 sides will be interleaved.
- Two single-sided DIMM pairs (128-bit configuration) or DIMMs (64-bit configuration) can be assigned CSMode = 1 = two ranks grouped, if:
  - Both DIMM pairs/DIMMs are the same type (MemMd).
  - (128-bit config): The two DIMM pairs are DIMMs (0+4 and 1+5), or (2+6 and 3+7).
  - (64-bit config): The two DIMMs are DIMMs (0 and 1), or (2 and 3).

  Accesses to the 2 sides will be interleaved (striped across the 2 sides).
- Two double-sided DIMM pairs (128-bit configuration) or DIMMs (64-bit configuration) can be assigned CSMode = 2 = four ranks grouped, if:
  - Both DIMM pairs/DIMMs are the same type (MemMd).
  - (128-bit config): The two DIMM pairs are DIMMs (0+4 and 1+5), or (2+6 and 3+7).
  - (64-bit config): The two DIMMs are DIMMs (0 and 1), or (2 and 3).

  Accesses to the 4 sides will be interleaved (striped across the 4 sides).
- All 4 DIMM pairs (128-bit configuration) or DIMMs (64-bit configuration) can be assigned CSMode = 2 = four ranks grouped, if:
  - All DIMMs are the same type (MemMd).
  - All DIMMs are single-sided.

  Accesses to the 4 sides will be interleaved (striped across the 4 sides).
- All 4 DIMM pairs (128-bit configuration) or DIMMs (64-bit configuration) can be assigned CSMode = 3 = eight ranks grouped, if:
  - All DIMMs are the same type (MemMd).
  - All DIMMs are double-sided.

  Accesses to the 8 sides will be interleaved (striped across the 8 sides).
- CSMode = 3 is not supported for single-sided DIMMs.

The DmCsMd bits in the UsrCnfg register specify the CSModes. There are 4 sets of DmCsMd bits, one for each DIMM. When DIMMs are grouped, the DmCsMd bits for the first DIMM in the group are the controlling bits, and the other DmCsMd bits in that group are don’t care. For example, using a configuration in which all DIMMs are occupied, double-sided, and the same Mem Mode:

- If it is desired to not group any ranks,
  Dm0CsMd = 00, Dm1CsMd = 00, Dm2CsMd = 00, Dm3CsMd = 00
- If it is desired to group the 2 ranks of the first DIMM pair (128-bit config) or DIMM (64-bit config) and not group any other ranks,
  Dm0CsMd = 01, Dm1CsMd = 00, Dm2CsMd = 00, Dm3CsMd = 00
- If it is desired to group the 4 ranks of the first 2 DIMM pairs (128-bit config) or DIMMs (64-bit config) and not group any other ranks,
  Dm0CsMd = 10, Dm1CsMd = don’t care, Dm2CsMd = 00, Dm3CsMd = 00
- If it is desired to group all 8 ranks of the DIMMs,
  Dm0CsMd = 11, Dm1CsMd = don’t care, Dm2CsMd = don’t care, Dm3CsMd = don’t care
- If it is desired to group the 4 ranks of the 2nd and 3rd DIMM pairs (128-bit config) or DIMMs (64-bit config) and not group any other ranks,
  Dm0CsMd = 00, Dm1CsMd = 10, Dm2CsMd = don’t care, Dm3CsMd = 00
- If it is desired to group the 2 ranks of the 2nd DIMM pair (128-bit config) or DIMM (64-bit config) and to have a 2nd group of the 4 ranks of the 3rd and 4th DIMM pairs (128-bit config) or DIMMs (64-bit config), and not group any other ranks,
  Dm0CsMd = 00, Dm1CsMd = 01, Dm2CsMd = 10, Dm3CsMd = don’t care

For DIMMs that cannot be grouped, and DIMMs for which it is not desired to have any grouping, the DmCsMd bits must be set to 00.

For DIMMs that are grouped, it is a programming convention (but not a requirement) that the DmCsMd bits of all the DIMMs in a group are set to the DmCsMd bits of the first DIMM that defines that group. (That is, if you set DmCsMd = 10 or 11 for some DIMM, then the remaining “don’t care” bits are set to 10 or 11, respectively.)

Figure 7-4. Chip Select and Interleave Mode Addressing

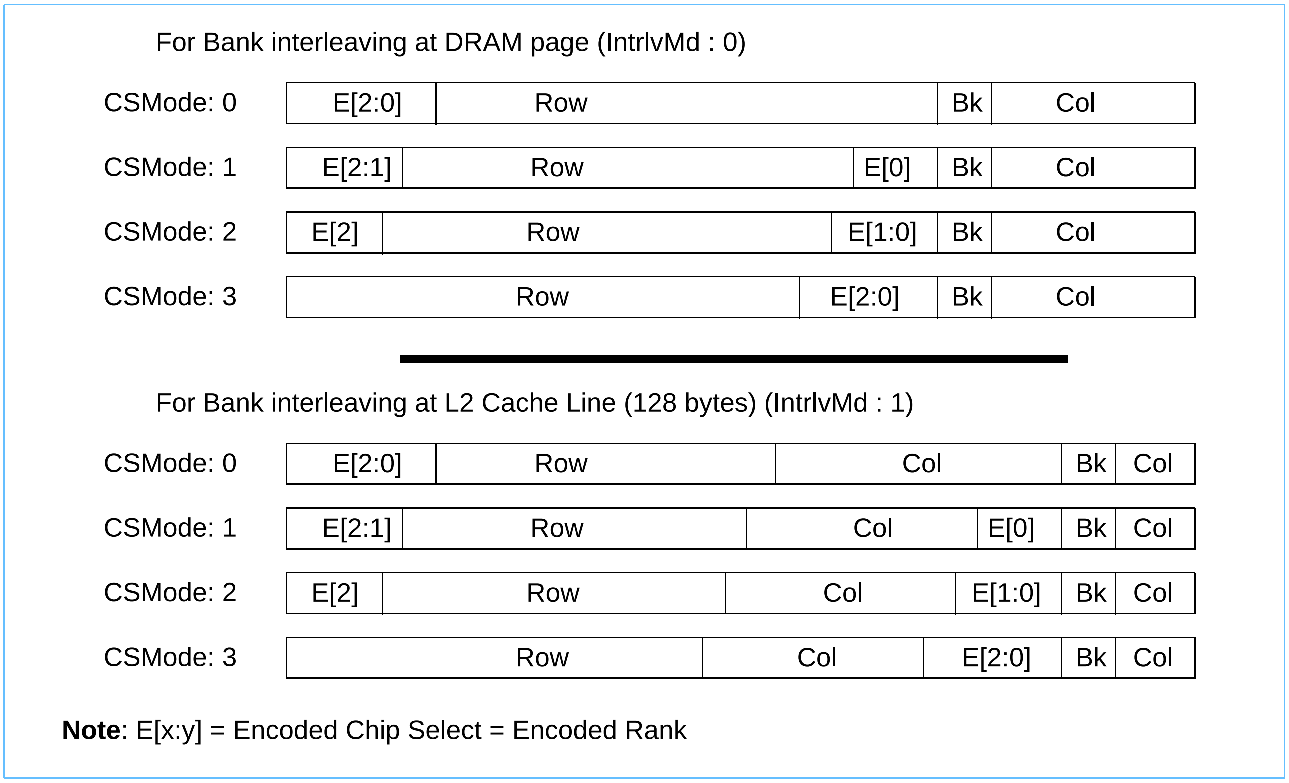

For Bank interleaving at DRAM page (IntrlvMd : 0)
CSMode: 0 | E[2:0] | Row | Bk | Col
CSMode: 1 | E[2:1] | Row | E[0] | Bk | Col
CSMode: 2 | E[2] | Row | E[1:0] | Bk | Col
CSMode: 3 | Row | E[2:0] | Bk | Col
For Bank interleaving at L2 Cache Line (128 bytes) (IntrlvMd : 1)
CSMode: 0 | E[2:0] | Row | Col | Bk | Col
CSMode: 1 | E[2:1] | Row | Col | E[0] | Bk | Col
CSMode: 2 | E[2] | Row | Col | E[1:0] | Bk | Col
CSMode: 3 | Row | Col | E[2:0] | Bk | Col
Note: E[x:y] = Encoded Chip Select = Encoded Rank

Figure 7-4 illustrates in a general way how the 4 Chip Select Modes and the 2 Interleave Modes affect the SDRAM address decoding.

Table 7-23 and *Table 7-24* show the detailed system address to SDRAM address mappings for 128-bit configuration, and *Table 7-25* and *Table 7-26* show the mappings for 64-bit configuration. For each chip type (MemMd) there are four system to SDRAM address mappings. The 4 encodings of the DmxCsMd bits select which of these four mappings is used. The difference involves which system address bits are used for chip selects, which are indicated by E[0:2] (Encoded Chip Select).

For DmCsMd = 0, the default, the high order system address bits are used for Chip Select. An entire rank of memory is accessed before moving on to the next rank.

For DmCsMd = 1, E[2] is moved towards the lower (least significant) end of the system address, adjacent to the bank address bits. This means, for example, that sequential accesses of increasing addresses will ping-pong between 2 ranks, accessing only 4 pages (for SDRAM page interleave) or 4 L2 cache lines (for L2 cache line interleave) before switching ranks.

For DmCsMd = 1, E[1:2] will cause accesses to sequence across 4 ranks, and for CsMd = 2, E[0:2] will cause accesses to sequence across all 8 ranks.

7.14.9 Start Address, Add 2G/Sub 2G

The StartAdrs field of each DmCnfg register must be programmed with the starting address of its corresponding DIMM. The Add2G and Sub2G fields must be programmed such that the address decode logic properly accounts for the 2 GB “I/O Hole.” The values of these fields are determined as follows:

- The programmer must consider the installed memory to be a set of one or more groups, as determined by the DmCsMd bits in the UsrCnfg register. The programmer must also know the size of the groups, in bytes. DIMMs with DmCsMd = 0 (ranks not grouped) are considered to be a group whose size is the number of bytes on the DIMM pair (128-bit config) or DIMM (64-bit config).
- Each group will consist of one or more DIMM pairs (128-bit config) or DIMMs (64-bit config). All of the DmCnfg registers for the DIMMs in a given group will be given the same StartAdrs, Add2G and Sub2G values (and MemMd, SS and DmEn).
- The memory controller uses 36-bits of the 64-bit address space, that is, address bits 28:63.
- The StartAdrs field for a group is programmed with the first address of that group. Only the first nine bits, [28:36], need to be specified. This gives a granularity of 128MB, which is the size of the smallest possible group (one rank of memory with MemMd = 0, in 64-bit data bus configuration).
- The Dm0Cnfg through Dm3Cnfg registers are programmed in sorted group size order, with the values for the largest group in Dm0Cnfg and the smallest in Dm3Cnfg (or the last DmCnfg Register used). Groups with the same size can be programmed in any order.

  The reason for the ordering rule is so that no group can partially overlap the 2 GB I/O hole.
- The groups must start at address 0 and be contiguous, except for the 2 GB I/O hole.
  – For example: four groups each of size 512 MB.
    Dm0Cnfg StartAdrs = 000000000 = 0 MB.
    Dm1Cnfg StartAdrs = 000000100 = 512 MB.
    Dm2Cnfg StartAdrs = 000001000 = 1024 MB = 1 GB.
    Dm3Cnfg StartAdrs = 000001100 = 1536 MB = 1.5 GB.
  – For example: four groups each of size 2 GB.
    Dm0Cnfg StartAdrs = 000000000 = 0 GB.
    Dm1Cnfg StartAdrs = 000100000 = 4 GB, I/O hole skipped.
    Dm2Cnfg StartAdrs = 000110000 = 6 GB.
    Dm3Cnfg StartAdrs = 001000000 = 8 GB.
  – For example: four groups each of size 4 GB.
    Dm0Cnfg StartAdrs = 000000000 = 0 GB, Add2G will take care of I/O hole overlap.
    Dm1Cnfg StartAdrs = 000100000 = 4 GB, Add2G, Sub2G will adjust for previous overlap.
    Dm2Cnfg StartAdrs = 001000000 = 8 GB, Add2G, Sub2G will adjust for previous overlap.
    Dm3Cnfg StartAdrs = 001100000 = 12 GB, Add2G, Sub2G will adjust for previous overlap.
- Add2G and Sub2G are used when the first group size is greater than 2 GB, which causes it to overlap the I/O hole. If the first group size is not larger than 2 GB, its Add2G and Sub2G fields should be set to 00001. Since the groups are programmed in decreasing size, the remaining groups, if any, will also be have their Add2G and Sub2G fields programmed to 00001. Their StartAdrs fields should be programmed according to the rule above that the groups are contiguous except for skipping the I/O hole.
- If a group size is larger than 2 GB and it is the first (largest) group, then it overlaps the 2 GB I/O hole. The Add2G field for that group must be programmed with the starting address of the 2 GB space following this group. Only 5 address bits, [28:32] are needed with a granularity of 2 GB.

For example, if a group starting at address 0 has 4 GB, the first 2 GB will map directly to the first 2 GB of system address space. To use the remaining 2 GB, the memory controller must map system addresses from 4 GB to 6 GB to that remaining 2 GB. With the Add2G field programmed to address bits 28:32 = 00010 = 4 GB, this indicates to the memory controller that it must add 2GB of memory, starting at address 4 GB to the range of memory addresses that will access the group.

- If the first group overlapped the I/O hole, then subsequent groups must be adjusted to compensate for the adjustment of that first group.
  - If the next group is greater than 2 GB, the process of “borrowing” the next 2 GB continues. That is, Add2G is set to the address following the end of the current group. Also, Sub2G is set to the Start Address of the current group, to subtract out the 2 GB that was “awarded” to the previous group using its Add2G field.
  - The Add2G/Sub2G process continues from group to group, until there are no more groups, or until a group is programmed whose size is not greater than 2 GB.
  - When a group whose size is not greater than 2 GB follows a group of size greater than 2 GB, its starting address is adjusted upward by 2 GB to compensate for the previous Add2G. The Add2G and Sub2G fields should be set to 00001. This is the address of the I/O hole, for which the memory controller never gets any requests. Subsequent groups, if any, continue setting their StartAdrs to start contiguously after the previous group, and Add2G and Sub2G should continue to be set to 1.
- There is a special case for a maxed out system. This is for the 128-bit data bus configuration when using a group of all 4 DIMM pairs, and the DIMM pair size is the maximum of 16 GB, for a total of 64 GB. Since there are only 36 bits of address = 64 GB, and there is a 2 GB hole, the last 2 GB cannot be accessed. There is no address that can be programmed in the Add2G field for this inaccessible memory. Therefore the Add2G field should be programmed to 00001.

7.14.10 DIMM Configuration Algorithm

The address decoding, or mapping, is complicated by 3 factors:

- As discussed above, there is a 2 GB “I/O hole.”
- As discussed above, the DIMMs can be “grouped” to provide striped access across the group.
- The address decode logic lies in a critical path with regard to low latency access.

The implemented decode logic, with the aid of the StartAdrs, Add2G, and Sub2G fields in the DmCnfg registers, provides for these factors. The values that must be programmed in these fields are not easy to explain: they are intended for minimal delay decoding that than the convenience of the programmer. However, an algorithm has been developed for the correct programming of these fields. This algorithm is presented below in pseudocode.

```
// DATA32 is an unsigned data type with a size of 32 bits.
// void RegWrite(DATA32 address, DATA32 data) is a function that writes a register

// Addresses work in granularities of 128MB, so 2GB = 16 × 128MB, and so on.
#define 2GB 16;
#define 4GB 32;
#define 64GB 512;

Struct Group {
    int StartDimm; // Starting DIMM in group (0,1,2,3)
    int NumDimm; // Number of DIMMs in group (1,2,3,4)
    DATA32 Size; // Size of group in 128MB chunks. 0 = no group
    unsigned MemMd; // For filling in reg field
    unsigned SS; // For filling in reg field
} GroupTable[4];

// Fill in GroupTable, with GroupTable[0] = largest group, GroupTable[1] next largest, and so on. (Not shown.)

DATA32 StartAdr = 0; DATA32 EndAdr; // only lower 9 bits used (actually 10 bits, if EndAdr goes to >64GB)
DATA32 Add2G = 1; DATA32 Sub2G = 1; // only 5 lower bits used
DATA32 RegAdr, RegData;

// Process each group, from largest to smallest
for (int i=0; i<4; i++) {
    EndAdr = StartAdr + GroupTable[i].Size;

    if (GroupTable[i].Size > 2GB) {
        // For groups larger than 2GB, if 1st group, set Add2G to adjust for IO hole
        // If subsequent group, set both Add2G and Sub2G to adjust for IO hole
        Add2G = (EndAdr & 0x1F0) >> 4; // extract starting address, on 2GB boundary, of next group
        if (EndAdr >= 64GB) {
            // Special case for maxed out 64GB system. Need to set Add2G to “nop” value
            Add2G = 1;
        }

        Sub2G = (StartAdr == 0) ? 1 : (StartAdr & 0x1F0) >> 4;
    } else {
        // If going from >2GB group to <= 2GB group, correct start and end addresses for previous Add2G
        // and reset Add2G and Sub2G to “nop”
        if (Add2G != 1) {
            StartAdr += 2GB;
            EndAdr += 2GB;
            Add2G = 1;
            Sub2G = 1;
        }
    }

    // For all DIMMs in group i, fill in corresponding DmCnfg registers with:
    // StartAdr (lower 9 bits), Add2G and Sub2G (lower 5 bits), Mode (4 bits) and SS (1 bit)
    // and DmEn = 1, if Size > 0, else just set DmEn = 0
    RegAdr = 0xF8002200 + (GroupTable[i].StartDimm * 0x10);
    RegData = 0;
    for (int j = 0; j < GroupTable[i].NumDimms; j++) {
        if (GroupTable[i].Size != 0) {
            RegData = 1;            // DmEn
            RegData |= GroupTable[i].SS << 1;
            RegData |= StartAdr << 3;
            RegData |= GroupTable[i].MemMd << 12;
            RegData |= Sub2G << 19;
            RegData |= Add2G << 27;
            }
        RegWrite(RegAdr, RegData); // <= Write the register
        RegAdr += 0x10;
    }

    // New start address = previous end address, skipping over IO hole if necessary
    if (EndAdr == 2GB)
        StartAdr = 4GB;
    else
        StartAdr = EndAdr;
}
```

7.14.11 DIMM Configuration Examples

7.14.11.1 Example 1

128-bit configuration, all 4 DIMM pairs occupied:

Table 7-7. DIMM Configuration Example 1 - DIMM Characteristics.

DIMM Pair	MemMd	Chip Size	Chip Width	Double/Single-sided	Total/Rank
Dm0 (DIMM slots 0 & 4)	1	256 Mb	×8	Double-Sided	512 MB
Dm1 (DIMM slots 1 & 5)	1	256 Mb	×8	Double-Sided	512 MB
Dm2 (DIMM slots 2 & 6)	8	1 Gb	×16	Double-Sided	1 GB
Dm3 (DIMM slots 3 & 7)	13	2 Gb	×8	Single-Sided	4 GB

In this DIMM configuration, Dms 0 and 1 can be grouped together to define a 4-rank group. The 2 ranks of Dm2 can be grouped into a 2-rank group. The single rank of Dm3 cannot be grouped.

The sorted list:

Table 7-8. DIMM Configuration Example 1 - Sorted Group List.

Group	Starting Dm	Num of DIMMs/Group	Group Size	CS Mode
Grp0 (largest)	3	1	4 GB	0 (1 rank)
Grp1	2	1	2 GB	1 (2 ranks)
Grp2 (smallest)	0	2	2 GB	2 (4 ranks)

Note: In this example Grp1 and Grp2's sort order does not matter because they are both 2 GB in size.

Using this sorted list with the DIMM configuration algorithm produces the following values:

Table 7-9. DIMM Configuration Example 1 - Register Fields.

Register	Add2G	Sub2G	MemMd	StartAdrs	SS	DmEn
Dm0Cnfg	0x1	0x1	0x1	0x40	0	1
Dm1Cnfg	0x1	0x1	0x1	0x40	0	1
Dm2Cnfg	0x1	0x1	0x8	0x30	0	1
Dm3Cnfg	0x2	0x1	0xD	0x0	1	1

Assuming, for example, IntrlvMd = 0 (DRAM page boundary) and Page Policy = 01 = usually closed:

Table 7-10. DIMM Configuration Example 1 - Register Values.

Register	Contents
Dm0Cnfg	0x08081201
Dm1Cnfg	0x08081201
Dm2Cnfg	0x08088181
Dm3Cnfg	0x1008D003
UsrCnfg	0xA4200000

7.14.11.2 Example 2

128-bit configuration, all 4 DIMM pairs occupied:

Table 7-11. DIMM Configuration Example 2 - DIMM Characteristics.

DIMM Pair	MemMd	Chip Size	Chip Width	Double/Single-sided	Total/Rank
Dm0 (DIMM slots 0 & 4)	0	256 Mb	×16	Single-Sided	256 MB
Dm1 (DIMM slots 1 & 5)	1	256 Mb	×8	Single-Sided	512 MB
Dm2 (DIMM slots 2 & 6)	1	256 Mb	×8	Single-Sided	512 MB
Dm3 (DIMM slots 3 & 7)	5	512 Mb	×8	Double-Sided	1 GB

Although Dm1 and Dm2 are identical they cannot be grouped together. Dm0 and Dm1 cannot be grouped because they are different MemMds. Same for Dm2 and Dm3. The only available possible group is the 2 ranks of Dm3.

The sorted list:

Table 7-12. DIMM Configuration Example 2 - Sorted Group List.

Group	Starting Dm	Num of DIMMs/Group	Group Size	CS Mode
Grp0 (largest)	3	1	2 GB	1 (2 ranks)
Grp1	2	1	512 MB	0 (1 rank)
Grp2	1	1	512 MB	0 (1 rank)
Grp3 (smallest)	0	1	256 MB	0 (1 rank)

Note: In this example Grp1 and Grp2’s sort order does not matter because they are both 512 MB in size.

Using this sorted list with the DIMM configuration algorithm produces the following values:

Table 7-13. DIMM Configuration Example 2 - Register Fields.

Register	Add2G	Sub2G	MemMd	StartAdrs	SS	DmEn
Dm0Cnfg	0x1	0x1	0x0	0x28	1	1
Dm1Cnfg	0x1	0x1	0x1	0x24	1	1
Dm2Cnfg	0x1	0x1	0x1	0x20	1	1
Dm3Cnfg	0x1	0x1	0x5	0x0	0	1

Assuming, for example, IntrlvMd = 0 (DRAM page boundary) and Page Policy = 01 = usually closed:

Table 7-14. DIMM Configuration Example 2 - Register Values.

Register	Contents
Dm0Cnfg	0x08080143
Dm1Cnfg	0x08081123
Dm2Cnfg	0x08081103
Dm3Cnfg	0x08085001
UsrCnfg	0x01400000

7.14.11.3 Example 3

128-bit configuration, all 4 DIMM pairs occupied:

Table 7-15. DIMM Configuration Example 3 - DIMM Characteristics.

DIMM Pair	MemMd	Chip Size	Chip Width	Double/Single-sided	Total/Rank
Dm0 (DIMM slots 0 & 4)	13	2 Gb	×8	Double-Sided	4 GB
Dm1 (DIMM slots 1 & 5)	13	2 Gb	×8	Double-Sided	4 GB
Dm2 (DIMM slots 2 & 6)	13	2 Gb	×8	Double-Sided	4 GB
Dm3 (DIMM slots 3 & 7)	13	2 Gb	×8	Double-Sided	4 GB

This DIMM configuration has all four DIMMs identical and double sided, which allows for 1 group of 8 ranks.

The sorted list:

Table 7-16. DIMM Configuration Example 3 - Sorted Group List.

Group	Starting Dm	Num of DIMMs/Group	Group Size	CS Mode
Grp0 (largest)	0	4	32 GB	4 (8 ranks)

Using this sorted list with the DIMM configuration algorithm produces the following values:

Table 7-17. DIMM Configuration Example 3 - Register Fields.

Register	Add2G	Sub2G	MemMd	StartAdrs	SS	DmEn
Dm0Cnfg	0x10	0x1	0xD	0x0	0	1
Dm1Cnfg	0x10	0x1	0xD	0x0	0	1
Dm2Cnfg	0x10	0x1	0xD	0x0	0	1
Dm3Cnfg	0x10	0x1	0xD	0x0	0	1

Assuming, for example, IntrlvMd = 0 (DRAM page boundary) and Page Policy = 00 = usually open:

Table 7-18. DIMM Configuration Example 3 - Register Values.

Register	Contents
Dm0Cnfg	0x1008D001
Dm1Cnfg	0x1008D001
Dm2Cnfg	0x1008D001
Dm3Cnfg	0x1008D001
UsrCnfg	0xFF000000

7.14.11.4 Example 4

128-bit configuration, all 4 DIMM pairs occupied:

Table 7-19. DIMM Configuration Example 4 - DIMM Characteristics.

DIMM Pair	MemMd	Chip Size	Chip Width	Double/Single-sided	Total/Rank
Dm0 (DIMM slots 0 & 4)	9	1 Gb	×8	Double-Sided	2 GB
Dm1 (DIMM slots 1 & 5)	12	2 Gb	×16	Double-Sided	2 GB
Dm2 (DIMM slots 2 & 6)	9	1 Gb	×8	Double-Sided	2 GB
Dm3 (DIMM slots 3 & 7)	13	2 Gb	×8	Single-Sided	4 GB

This DIMM configuration illustrates a more complicated I/O hole configuration; Add2G and Sub2G are “fully exercised.” Also, although the double-sided DIMMs could be given a CSMode = 1 to group their two ranks together, in this example we assume the user does not want to stripe the accesses across ranks, and therefore CSMode = 0 for all DIMMs.

The sorted list:

Table 7-20. DIMM Configuration Example 4 - Sorted Group List.

Group	Starting Dm	Num of DIMMs/Group	Group Size	CS Mode
Grp0 (largest)	3	1	4 GB	0 (1 rank)
Grp1	2	1	4 GB	0 (1 rank)
Grp2	1	1	4 GB	0 (1 rank)
Grp3 (smallest)	0	1	4 GB	0 (1 rank)

Note: This table could have been sorted in any order, since all groups are the same size.

Using this sorted list with the DIMM configuration algorithm produces the following values:

Table 7-21. DIMM Configuration Example 4 - Register Fields.

Register	Add2G	Sub2G	MemMd	StartAdrs	SS	DmEn
Dm0Cnfg	0x8	0x6	0x9	0x60	0	1
Dm1Cnfg	0x6	0x4	0xC	0x40	0	1
Dm2Cnfg	0x4	0x2	0x9	0x20	0	1
Dm3Cnfg	0x2	0x1	0xD	0x0	1	1

Assuming, for example, IntrlvMd = 0 (DRAM page boundary) and Page Policy = 01 = usually closed:

Table 7-22. DIMM Configuration Example 4 - Register Values.

Register	Contents
Dm0Cnfg	0x40308301
Dm1Cnfg	0x3020C201
Dm2Cnfg	0x20109101
Dm3Cnfg	0x11008D003
UsrCnfg	0x00200000

7.14.12 Address Mapping

The mapping of the incoming 36-bit system addresses to memory rank, bank, row and column is a function of the Memory Modes used (chip size and organization), Interleave mode, Chip Select Mode, and Data Bus Configuration (128-bit or 64-bit). The mappings for the 128-bit bus configuration are given in *Table 7-23* and *Table 7-24* and the mappings for the 64-bit configurations are given in *Table 7-25* and *Table 7-26*.

As indicated in previous discussion for a given bus width, 128-bit or 64-bit, the appropriate table is chosen depending on the IntrLvMd setting in the UsrCnfg register (SDRAM page boundary or cache line boundary). For each MemMd, one of 4 rows in the table (0, 1, 2, or 3) is selected depending on the setting of the DmCsMd bits in the UsrCnfg register.

E*x* indicates the (encoded) rank bit, B*x* indicates the Bank bit, R*x* indicates the Row bit and C*x* indicates the Column bit. Column bit C10 is not given in these tables because it is the AP (Auto Precharge) bit.

Note that the Encoded Chip Select bits are big-endian while the Bank, Row, and Column bits are little-endian.

System address bits 60:63 are used to select one of 16 bytes within the 128-bit quantity transferred (in the 128-bit bus configuration).

For the 128-bit bus configuration the burst size is 4, that is, four 128-bit quantities are transferred. The mappings in *Table 7-23* and *Table 7-24* indicate the first column that is accessed in the burst (“critical word first”). Accesses wrap on the burst boundary. For example, if the first access maps to C1:C0 = 10, then the next 3 C1:C0 accesses are 11, 00, 01.

The memory devices are required to be programmed to use interleaved access order (MRS bit A[3], see *Section 7.10.1 MRS Settings*). If the first access maps to C1:C0 = 01, then the next 3 C1:C0 accesses are 00, 11, 10. If the first access maps to C1:C0 = 11, then the next 3 C1:C0 accesses are 10, 01, 00.

For the two 64-bit bus configurations all entries in *Table 7-25* and *Table 7-26* are shifted right by 1 bit relative to *Table 7-23* and *Table 7-24*. System address bit 28 is unused and the maximum addressability is 32 GB. The burst length is 8. Internally the memory controller assembles the 64-bit quantities into 128-bit quantities for reads and disassembles 128-bit quantities into 64-bit quantities on writes. For each pair of accesses in a burst the first transfer of the pair becomes bits 0:63 (and 128:135 for ECC) and the second transfer of the pair becomes bits 64:127 (and 136:143 for ECC). Column bit C0 is always driven to 0.

7.14.13 Address Mapping Exceptions

Illegal accesses to memory are those that exceed the upper bound of installed memory.

- If the request is a read, a read from some legal location in memory will be performed, and the data from that location will be returned to the requestor.
- If the request is a write, the memory controller will generate all of the cycles associated with a write operation (including pulling data out of the WDB in the PI), but the chip select will be suppressed so that no write is actually performed.

Illegal accesses are trapped by the Memory Mapping Exception Registers (*Section 12.5* on page 338). The address is captured in the MemMapExcpAd Register (0xF80020E0) and information about the access is captured in the MemMapExcpCtl register (0xF80020F0). This information includes the Request Type (Coherent, Non-Coherent, Scrub), read or write type, Transaction Size, and Requestor ID. The Tag and Write All status are also captured; this information is for development engineers and might not be useful in the general case.

The first illegal access to memory traps the address and request information in the two exception registers and sets the MemExcpV bit in the MemMapExcpCtl Register. Subsequent illegal accesses are not trapped, until the MemExcpV bit is cleared by reading the MemMapExcpCtl Register.

When an illegal address is trapped, an exception is generated, if enabled, with the ExcpMask bit in the MemMapExcpCtl Register.

Table 7-23. Memory Mapping for Bank Interleaving at SDRAM Page Boundary (128-bit Configuration) .

Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
256Mb ×16	0						E0	E1	E2	R6	R5	R4	R2	R2	R1	R0	R12	R11	R10	R9	R8	R7	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1						E0	E1	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	E2	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2						E0	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E1	E2	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3						R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E0	E1	E2	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1	C0
256Mb ×8	0					E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1					E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2					E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3					R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E0	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
256Mb ×4	0				E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1				E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2				E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3				R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
512Mb ×16	0					E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1					E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2					E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3					R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E0	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
512Mb ×8	0				E0	E1	E2	R13	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1				E0	E1	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2				E0	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3				R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E0	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
512Mb ×4	0			E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1			E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2			E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3			R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
1Gb ×16	0				E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1				E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2				E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3				R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
1Gb ×8	0			E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1			E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2			E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3			R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
1Gb ×4	0		E0	E1	E2	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1		E0	E1	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2		E0	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3		R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	E0	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
2Gb ×16	0			E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1			E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2			E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3			R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
2Gb ×8	0		E0	E1	E2	R14	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1		E0	E1	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2		E0	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3		R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
2Gb ×4	0	E0	E1	E2	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	1	E0	E1	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	2	E0	R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
	3	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	E0	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

Table 7-24. Memory Mapping for Bank Interleaving at Cache Line Boundary (128-bit Configuration).

Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
256Mb ×16	0						E0	E1	E2	R6	R5	R4	R2	R2	R1	R0	R12	R11	R10	R9	R8	R7	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1	C0
	1						E0	E1	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1	C0
	2						E0	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1	C0
	3						R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1	C0
256Mb ×8	0					E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1	C0
	1					E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1	C0
	2					E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1	C0
	3					R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1	C0
256Mb ×4	0				E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C11	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1	C0
	1				E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1	C0
	2				E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1	C0
	3				R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1	C0
512Mb ×16	0					E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1	C0
	1					E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1	C0
	2					E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1	C0
	3					R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1	C0
512Mb ×8	0				E0	E1	E2	R13	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1	C0
	1				E0	E1	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1	C0
	2				E0	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1	C0
	3				R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1	C0
512Mb ×4	0			E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C11	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1	C0
	1			E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1	C0
	2			E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1	C0
	3			R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1	C0
1Gb ×16	0				E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1	C0
	1				E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1	C0
	2				E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1	C0
	3				R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1	C0
1Gb ×8	0			E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1	C0
	1			E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1	C0
	2			E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1	C0
	3			R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1	C0
1Gb ×4	0		E0	E1	E2	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1	C0
	1		E0	E1	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1	C0
	2		E0	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1	C0
	3		R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	C11	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1	C0
2Gb ×16	0			E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1	C0
	1			E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1	C0
	2			E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1	C0
	3			R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1	C0
2Gb ×8	0		E0	E1	E2	R14	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1	C0
	1		E0	E1	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1	C0
	2		E0	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1	C0
	3		R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1	C0
2Gb ×4	0	E0	E1	E2	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1	C0
	1	E0	E1	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1	C0
	2	E0	R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1	C0
	3	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	C11	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1	C0
Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

Table 7-25. Memory Mapping for Bank Interleaving at SDRAM Page Boundary (64-bit Configuration).

Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
256Mb ×16	0							E0	E1	E2	R6	R5	R4	R2	R2	R1	R0	R12	R11	R10	R9	R8	R7	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1
	1							E0	E1	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	E2	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1
	2							E0	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E1	E2	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1
	3							R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E0	E1	E2	B1	B0	C8	C7	C6	C5	C4	C3	C2	C1
256Mb ×8	0						E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1						E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2						E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3						R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E0	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
256Mb ×4	0					E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1					E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2					E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3					R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
512Mb ×16	0						E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1						E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2						E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3						R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E0	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
512Mb ×8	0					E0	E1	E2	R13	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1					E0	E1	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2					E0	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3					R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E0	E1	E2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
512Mb ×4	0				E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1				E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2				E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3				R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
1Gb ×16	0					E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1					E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2					E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3					R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
1Gb ×8	0				E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1				E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2				E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3				R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
1Gb ×4	0			E0	E1	E2	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1			E0	E1	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2			E0	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3			R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	E0	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
2Gb ×16	0				E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1				E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2				E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3				R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
2Gb ×8	0			E0	E1	E2	R14	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1			E0	E1	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2			E0	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3			R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E0	E1	E2	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1
2Gb ×4	0		E0	E1	E2	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	1		E0	E1	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	2		E0	R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
	3		R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	E0	E1	E2	B2	B1	B0	C11	C9	C8	C7	C6	C5	C4	C3	C2	C1
Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

Table 7-26. Memory Mapping for Bank Interleaving at Cache Line Boundary (64-bit Configuration).

Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
256Mb ×16	0							E0	E1	E2	R6	R5	R4	R2	R2	R1	R0	R12	R11	R10	R9	R8	R7	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1
	1							E0	E1	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1
	2							E0	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1
	3							R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1
256Mb ×8	0						E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1
	1						E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1
	2						E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1
	3						R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1
256Mb ×4	0					E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C11	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1
	1					E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1
	2					E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1
	3					R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1
512Mb ×16	0						E0	E1	E2	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1
	1						E0	E1	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1
	2						E0	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1
	3						R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1
512Mb ×8	0					E0	E1	E2	R13	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	R8	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1
	1					E0	E1	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1
	2					E0	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1
	3					R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1
512Mb ×4	0				E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C11	C9	C8	C7	C6	C5	C4	C3	B1	B0	C2	C1
	1				E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	E2	B1	B0	C2	C1
	2				E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E1	E2	B1	B0	C2	C1
	3				R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E0	E1	E2	B1	B0	C2	C1
1Gb ×16	0					E0	E1	E2	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1
	1					E0	E1	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1
	2					E0	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1
	3					R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1
1Gb ×8	0				E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1
	1				E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1
	2				E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1
	3				R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1
1Gb ×4	0			E0	E1	E2	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1
	1			E0	E1	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1
	2			E0	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1
	3			R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	C11	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1
2Gb ×16	0				E0	E1	E2	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1
	1				E0	E1	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1
	2				E0	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1
	3				R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1
2Gb ×8	0			E0	E1	E2	R14	R13	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	R9	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1
	1			E0	E1	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1
	2			E0	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1
	3			R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1
2Gb ×4	0		E0	E1	E2	R14	R13	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	R10	C11	C9	C3	C8	C7	C6	C5	C4	B2	B1	B0	C2	C1
	1		E0	E1	R14	R13	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	R11	C11	C9	C4	C3	C8	C7	C6	C5	E2	B2	B1	B0	C2	C1
	2		E0	R14	R13	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	R12	C11	C9	C5	C4	C3	C8	C7	C6	E1	E2	B2	B1	B0	C2	C1
	3		R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	C11	C9	C6	C5	C4	C3	C8	C7	E0	E1	E2	B2	B1	B0	C2	C1
Sys Addr		28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

7.15 Timing Parameters

Memory timings are programmed using the RAS Timer0, RAS Timer1, CAS Timer 0, and CAS Timer 1 registers (0xF8002030, 0xF8002040, 0xF8002050 and 0xF8002060). In general the RAS Timers are concerned with Activate (RAS) and Precharge timings and are largely governed by the JEDEC Standard, while the CAS timers are concerned with Read/Write (CAS) commands and are governed by the JEDEC Standard and the times required to turn around the external data bus.

The descriptions of the Timer registers in *Section 12.10.1 Memory Timing Parameter Registers* on page 449 give the calculations for each parameter. tRAS, tRTP, tWR, tRP, tRTP, tWR, tRRD, tRC, tRCD, tWTR are JEDEC specified values. AL is required to be 0. WL = RL - 1. RL = AL + CL, where CL (CAS Latency) is specified by the memory device manufacturer (generally = 3, 4 or 5) and must match the value programmed in the MRS (*Section 7.10.1 MRS Settings*). Since AL = 0 for the CPC945, WL = CL -1. BL (Burst Length) = 4 for the 128-bit bus configuration and 8 for the two 64-bit bus configurations. RRMux, WRMux, WWMux and RWmux are the times to allow the external data multiplexer (if used) to switch when switching access among different DIMMs. It might be required to program nonzero values for these parameters, even if external data multiplexers are not used, to allow enough time for the data bus to settle.

In general, for best performance, the minimum values that are legal should be programmed in the RAS and CAS Timer registers. However, if the installed DIMMs have mixed performance values, then the most conservative values among the DIMMs must be programmed. For example, if 400 MHz and 533 MHz DIMMs are mixed, the values for the 400 MHz DIMMs must be used. If DIMMs of mixed CAS Latency are used, CL must be programmed with the largest CAS Latency among the DIMMs.

7.15.1 Data Bus Delay Greater than tCK

As noted above some values in the CAS Timer registers are a function of the time required to turn around the external data bus. The most demanding requirement is for issuing a Read followed by a Write. After the memory controller issues the Read command, it must wait for the read data to come back and for the SDRAMs to tri-state the data bus, before sending the write data for the Write command. In the best case the Write command can be issued while the read data is on the bus. By the time the SDRAMs need the write data, the read data has cleared the bus and the memory controller has started writing the write data to the bus.

The design of the memory controller, the programming of the Timer registers, and the discussion above all assume that the delay of the DQ and DQS signals is less than one mem_clk (tCK). If the delay is greater than tCK, then the memory controller will not execute the read-followed-by-read sequence properly, because it will not allow enough time for the data bus to turn around.

To avoid this problem, it is required to add time to all programming values which control Read To Write timings. These are CAS Timer 1, TiRtWRk, TiRtWDm and TiRtWSy, to push the write out. For each additional tCK of data bus delay greater than tCK, “2” must be added to these values. (One mem_clk to account for the read data coming back delayed, one mem_clk to account for the write data getting to the SDRAMs delayed.)

Other values affected by excessive data bus delay are the TiRtWRMW field in the MemRWArb register (see *Section 12.10.13* on page 471), the RdMacDel, ResMuxDel, RdExtMuxDly, WrExtMuxDly, WdbRqDly, RdOEOffDly fields in the MemBusConfig register (see *Section 12.10.14* on page 472) and the PIRdTgDly and RdPipeDly fields in the MemBusConfig2 register (see *Section 12.10.15* on page 474).

Performance is degraded when these timings are increased.

7.15.2 Restrictions

The following restrictions must be observed. Usually it is no problem meeting these restrictions, using values that match or are close to JEDEC minimum timings.

- TiAtP (tRAS) must be > TiAtRW (tRCD). If this condition is violated the memory controller can lock up issuing an infinite series of Activate/Precharge/Activate/Precharge/...
- TiAtABk (tRC) must be >= TiAtARk (tRRD). If this condition is violated the memory controller can lock up issuing an infinite series of Activate/Precharge/Activate/Precharge/...
- TiAtRW + TiWAPtA must be >= TiAtP + TiPtA. If this condition is violated, sometimes a write to a given address can execute before a previous write to that same address.

7.15.3 Timing Parameter Examples

The RASTimer0, RASTimer1, CASTimer0 and CASTimer1 registers are a set of delay values specified as equations in units of mem_clk (CK). Minimum values are used for best performance. Higher values can be used if relaxed timings are needed for some reason. Notes:

- For DIMMs with a mixture of speeds, the slowest speed (*largest* tCK) must be used.
- All of these examples use a bus configuration width of 128, so BL (Burst Length) always = 4. For systems with a bus configuration of 64, set BL = 8.
- CL (CAS Latency) can be 3, 4 or 5, depending on the chips in the DIMMs (JESD79-2A. The 533MHz, CL=5 option was removed with JESD79-2B). For DIMMs with a mixture of CL, the *largest* CL must be used.
- For the CPC945, AL (Additive Latency) always = 0.
- JEDEC specifies WL (Write Latency) as RL (Read Latency) - 1. RL = AL + CL.
  Therefore WL = AL+CL - 1 = CL - 1 for CPC945.
- tRCD, tRP, tRC and tRAS are minimum JEDEC specifications (JESD79-2C, Table 40 on page 72) specified in nanoseconds. These must be converted to mem_clks by dividing by tCK and rounding up to the nearest integer, as shown in *Table 7-27*. These values depend on the speed bin (frequency and CL).

  Note that tRP is used for both precharge of a single bank and precharge of all banks (4 or 8), and that for precharge of all banks 1 additional clock must be added for 8 bank devices (MemMode 1xxx). This is taken care of in the calculation for RASTimer0, TiPAtA.

Table 7-27. tRCD, tRP and tRC.

Speed	DDR2-533B tCK = 3.75 ns		DDR2-533C tCK = 3.75 ns		DDR2-400B tCK = 5 ns		DDR2-400C tCK = 5 ns	
Bin	3 - 3 - 3		4 - 4 - 4		3 - 3 - 3		4 - 4 - 4	
Parameter	ns	mem_clks	ns	mem_clks	ns	mem_clks	ns	mem_clks
tRCD	11.25	**3**	15	**4**	15	**3**	20	**4**
tRP	11.25	**3**	15	**4**	15	**3**	20	**4**
tRC	56.25	**15**	60	**16**	60	**12**	65	**13**
tRAS	45	**12**	45	**12**	11	**40**	45	**12**

- tRTP, tWR, tRRD and tWTR are minimum JEDEC specifications (JESD79-2C, Tables 41 and 42 on pages 73 through 76) specified in ns. These must be converted to mem_clks by dividing by tCK and rounding up to the nearest integer, as shown in *Table 7-28*. Note that tRRD is larger for devices with 2K pages (MemMode = 0010, 0110, 1010, and 1110), therefor if any DIMM uses one of these chip types the *larger* tRRD must be used.

  JEDEC has a “Note 3” for tRAS and tRTP, indicating that these values are a minimum requirement, and that the minimum read to precharge timing is AL + BL/2 providing that tRPT and tRAS (minimum) have been satisfied. The memory controller takes care of that if RASTimer0 TiRtP is programmed correctly.

  JEDEC also has a “Note 4” for tRRD indicating that a minimum of 2 mem_clks (2 * tCK) are required regardless of mem_clk frequency. This is satisfied using *Table 7-28* for tCK = 5 ns and faster, but the user should note this requirement if using clocks slower than 5 ns.

Table 7-28. tRTP, tWR and tRRD.

Speed	DDR2-533, tCK = 3.75 ns		DDR2-400, tCK = 5 ns	
Parameter	ns	mem_clks	ns	mem_clks
tRTP	7.5	2	7.5	2
tWR	15	4	15	3
tRRD, 1K Page	7.5	2	7.5	2
tRRD, 2K Page	10	3	10	2
4 × tRRD, 1K Page	30	8	30	6
4 × tRRD, 2K Page	40	11	40	8
tWTR	7.5	2	10	2

- As shown above for tRRD, arithmetic should be performed on the times in ns before being converted to mem_clks, as shown for two more calculations in *Table 7-29*.

Table 7-29. tRTP+tRP, tWR + tRP.

Speed	DDR2-533B tCK = 3.75 ns		DDR2-533C tCK = 3.75 ns		DDR2-400B tCK = 5 ns		DDR2-400C tCK = 5 ns	
Bin	**3** - 3 - 3		**4** - 4 - 4		**3** - 3 - 3		**4** - 4 - 4	
Parameter	ns	mem_clks	ns	mem_clks	ns	mem_clks	ns	mem_clks
tRTP + tRP	18.75	5	22.5	6	22.5	5	27.5	6
tWR + tRP	26.25	7	30	8	30	6	35	7

- For the CASTimer0 and CASTimer1 register descriptions, CAS accesses to different DIMMs use the terms RRMux, WRMux, WRMux, and RWMux. These terms add time to allow the data bus to switch between DIMMs. If no external data multiplexers are used, a value of ‘1’ cycle must be used for these terms. If external data multiplexers are used, an additional cycle might be needed to account for the switching time of the multiplexers.

  These examples use a value of 1 cycle for RRMux, WRMux, WRMux and RWMux.

:

Table 7-30. Example Timing Parameters.

	Speed	DDR2-533B tCK = 3.75 ns	DDR2-533C tCK = 3.75 ns	DDR2-400B tCK = 5 ns	DDR2-400C tCK = 5 ns
	Bin	**3** - 3 - 3	**4** - 4 - 4	**3** - 3 - 3	**4** - 4 - 4
Parameter	Calculation				
TiAtP	tRAS - 2	10	10	7	7
TiRtP	(BL/2 - 2) + tRTP - 2	0	0	0	0
TiWtP	WL + BL/2 +tWR - 2	6	7	5	6
TiPtA	tRP - 2	1	2	1	2
TiPAtA, 4 bank	tRP - 2	1	2	1	2
TiPAtA, 8 bank	tRP - 1	2	3	2	3
TiRAPtA	(BL/2) - 2 + RND(tRTP + tRP) - 2	3	4	3	4
TiWAPtA	CL +(BL/2) - 1 + RND(tWR + tRP) - 2	9	11	8	10
TiAtARk 1K page	$tRRD_{1K\ page}$ - 2	0	0	0	0
TiAtARk 2K page	$tRRD_{2K\ page}$ - 2	1	1	0	0
TiAtABk	tRC - 2	13	14	10	11
TiAtRW	tRCD - 2	1	2	1	2
TiAtARkWin 1K page	4 * $tRRD_{1K\ page}$	8	8	6	6
TiAtARkWin 2K page	4 * $tRRD_{2K\ page}$	11	11	8	8
TiRtRRk	BL/2 - 2	0	0	0	0
TiRtRDm	BL/2 - 1	1	1	1	1
TiRtRSy*	BL/2 + RRMux - 2	1	1	1	1
TiWtRRk	(CL - 1) + BL/2 + tWTR - 2	4	5	4	5
TiWtRDm	BL/2 - 1	1	1	1	1
TiWtRSy*	BL/2 + WRMux - 2	1	1	1	1
TiWtWRk	BL/2 - 2	0	0	0	0
TiWtWDm	BL/2 - 1	1	1	1	1
TiWtWSy*	BL/2 + WWMux - 2	1	1	1	1
TiRtWRk	BL/2	2	2	2	2
TiRtWDm	BL/2	2	2	2	2
TiRtWSy*	BL/2 + RWMux - 1	2	2	2	2
(*) Use RRMux = WRMux = WWMux = RWMux = 1.					

Using the values in *Table 7-30*, the following register values are produced (*Table 7-31* and *Table 7-32*):

Table 7-31. RASTimer0/1 Register Values.

	RASTimer0 (0xF8002030)		RASTimer1 (0xF8002040)	
	Only 4 bank	1 or more 8 bank	No 2K page	1 or more 2K page
533 MHz 3 - 3 - 3	0x500C1080	0x500C1100	0x1A40D0A0	0x1A42D0AC
533 MHz 4 - 4 - 4	0x500E2100	0x500E2180	0x22C0E120	0x22C2E12C
400 MHz 3 - 3 - 3	0x380A1080	0x380A1100	0x1A00A098	0x1A00A0A0
400 MHz 4 - 4 - 4	0x380C2100	0x380C2180	0x2280B118	0x2280B120

Table 7-32. CASTimer0/1 Register Values.

	CASTimer0 (0xF8002050)	CASTimer1 (0xF8002060)
533 MHz 3 - 3 - 3	0x00424084	0x00422108
533 MHz 4 - 4 - 4	0x00425084	
400 MHz 3 - 3 - 3	0x00424084	
400 MHz 4 - 4 - 4	0x00425084	

Note: The register values given above are based on JESD79-2C. Strictly speaking, these values should be considered for illustration purposes. The DIMM SPD data contains the actual timing specifications determined by the manufacturers of the DIMMs. These data should be obtained using I2C and plugged into the calculations.

7.16 Page Table/Timers

The page table contains entries for all SDRAM banks that might be installed. With 8 ranks maximum, and 8 banks per rank maximum (using 1 Gb or 2 Gb chips) there are 8 × 8 = 64 entries.

Each entry contains various status:

- If a row is open in the bank.
- If a row is open, the number of that row (the row address).
- Various timing conditions: for an open row, how long until a CAS command or Precharge can be issued, for a closed row, how long until an Activate can be issued, and so on.

The Page Table updates its entries using the feedback from the Command Arbiter which indicates which SDRAM commands are being issued. It loads its timers using values from the RAS Timer 0, RAS Timer 1, CAS Timer 0 and CAS Timer 1 registers.

In previous designs (such as the CPC925), the Page Table logic was simply a lookup table that returned the page open/close status for a given access. For the CPC945 the logic is more complex, being dominated by the timers which are tracking the RAS and CAS activities for the 64 memory banks. Lookup operations return the first SDRAM command that should be executed (for example, Precharge, Activate, Read or Write), and a set of timer values which enable the reorder queues to properly time the requests of subsequent SDRAM commands. Given the nature of this logic, the Page Table unit is also referred to as the “Timers” unit.

7.17 Reorder Queues

Incoming reads that do not take the fast path are entered in the Read Reorder Queue. Incoming writes and scrubs are entered in the Write Reorder Queue.

7.17.1 Queue Sizes

The physical size of the Read Reorder Queue is 8 entries. The utilized size can be programmed using the SzRdQ field of the MemRdQCnfg register (*Section 12.10.10 Memory Read Request Queue Configuration Register (MemRdQCnfg)* on page 467), from 1 to 8 entries. The default is 8 entries. If SzRdQ > 8, the programmed size is 8. If SzRdQ = 0, it is the same as SzRdQ = 1.

The physical size of the Write Reorder Queue is 16 entries. The utilized size can be programmed using the SzWrQ field of the MemWrQCnfg register (*Section 12.10.11 Memory Write Request Queue Configuration Register (MemWrtQCnfg)* on page 468), from 1 to 16 entries. The default is 16 entries. If SzRdQ > 16, the programmed size is 16. If SzWrQ = 0, it is the same as SzWrQ = 1.

7.17.2 Queue Filling

The two queues are filled from bottom to top: requests enter a queue at the lowest unoccupied slot. Entries can be removed from any occupied slot. The queues operate in “pizza box” fashion: when an entry is removed, all the entries above that slot move down one slot.

If the Read Reorder Queue becomes full (number of occupied slots = size programmed by SzRdQ) the Memory Request Arbiter will stop taking read requests until one or more slots become available in the Read Reorder Queue. If the Write Reorder Queue becomes full (number of occupied slots = size programmed by SzWrQ) the Memory Request Arbiter will stop taking write and scrub requests until one or more slots become available in the Write Reorder Queue.

7.17.3 Queue Entries

Each queue entry consists of the rank, bank, row and column addresses of a requested access to memory, timers, and a state machine. (This is a simplified representation of the actual entry. For example, the source of the request, PI, PCIe, HT, and a RMW indicator, among other things, are also stored in each entry.)

The output of the state machine is a set of two indicators to the Command Arbiter, RV and CV, and associated information. (Again, simplified.)

- RV (RAS Valid) is a request to issue a “RAS” command. Along with the RV signal is an indicator of the command to be issued (Activate or Precharge) and the rank and bank address (and row address, for Activate).
- CV (CAS Valid) is a request to issue a “CAS” command. Along with the CV signal is the rank, bank and column address. (The Command Arbiter will infer the command, read or write, based on which queue is issuing the CV.)

When an entry is put into a queue it is initialized with the rank, bank, row and column information from the Address Decoder, and first SDRAM command and timer values from the Page Table. The state machine sequences to different states when the timers expire, and from feedback information from the Command Arbiter.

Simplified example: suppose a read request to a closed row was accepted by the Memory Request Arbiter.

- It will be entered in the Read Reorder Queue with the rank, bank, row and column address from the Address Decoder. The SDRAM command “Precharge” will be entered, from the Page Table. A timer in the queue entry will be loaded with a value from the Page Table. This loaded value is from a timer in the Page Table that has been counting off the time when it will be valid to issue a Precharge command to close the open row in that bank.
- The timer counts down. When it reaches 0, the state machine will activate RV and indicate that the valid command is Precharge. It sends the RV, Precharge command, rank and bank to the Command Arbiter.

  Note: The timer value from the Page Table could be 0, in which case RV will be activated immediately.
- If the Command Arbiter selects this entry, it will use the Command Arbiter to issue the Precharge command. This issued command is fed back to the Page Table, which updates it status to indicate that there are no open rows in that bank. The issued command is also fed back to the queue entry, which moves the state machine to a state which drops the RV.
- A timer in the queue entry is loaded with a value for the time that it will be valid to issue Activate (TiPtA from RAS Timer 0). When this timer counts down to 0, the state machine will issue RV, this time indicating

that the command is Activate. It sends the RV, Activate command, rank, bank and row to the Command Arbiter.

- If the Command Arbiter selects this entry, it will use the Command Arbiter to issue the Activate command. This issued command is fed back to the Page Table, which updates it status to indicate that this new row is now open. The issued command is also fed back to the queue entry, which moves the state machine to a state which drops the RV.
- A timer in the queue entry is loaded with a value for the time that it will be valid to issue read (TiAtRW from RAS Timer 1). When this timer counts down to 0, the state machine will issue CV. It sends the CV, rank, bank and column to the Command Arbiter.
- If the Command Arbiter selects this entry, it will use the Command Arbiter to issue the read command. The issued command is fed back to the queue entry, which moves the state machine to a state which drops the CV.
- The state machine will “retire” the entry. That is, the entry will remove itself from the queue, and entries above that slot, if any, will all drop down by one slot.

From the number of fields in the RAS and CAS Timer registers, it should be evident that the above example is highly idealized. There are many timing relationships that must be observed, which requires multiple timers in the queue entries, and a state machine that factors in multiple timing relationships. As one example among many, in the example above the state machine cannot simply issue CV some measured time after Activate if a previous read or write is still using the data bus. Therefore a second timer can be counting off a time that was initialized by TiRtRDm, if a read had just been performed to a different rank of the same DIMM, or TiWtRSy if a write had just been performed to a different DIMM, and so on.

Also, the state machine does not necessarily proceed linearly. In the example above, an RV is issued with a Precharge command and accepted by the Command Arbiter. But the Command Arbiter can follow that with an acceptance, from some other entry, of an Activate command to a different row. Now the example entry will have to start over again, counting down when it will be legal to request a precharge command again.

For the Write Reorder Queue and additional constraint on issuing both RV and CV is the availability of the data to be written. As discussed in *Section 7.8.2* on page 145 write data is supplied by the WDBs (Write Data Buffers) in the PI. The PI also signals to the memory controller if the Write data in the buffers is valid. In some cases (for example, intervention) it is possible that the write data is not in the WDB when the memory controller is ready to do the write. The memory controller will not issue either RV or CV until the PI signals that the write data is valid.

7.17.4 Queue Page Policy

For a given entry, when a queue indicates a CAS operation is valid with CV, it has the option of setting column bit 10 for that entry to 0 or 1. When the Command Arbiter selects that entry and sends it to the SDRAM, the SDRAM will keep its row open if C10=0, or it will close the row at the end of operation (auto pre charge) if C10=1.

If an operation is issued with C10=0 the queue will update the page table and timers to indicate that the row is still open; access to that same row is most likely to be governed by tCCD (CAS to CAS delay). If issued with C10=1, the queue will update the page table and timers to indicate that the row is closed; access to that same row will likely be dominated by tRP (Precharge to Activate).

The PgPolicy bits in the UsrCnfg register are a common control for the setting of C10 in the two queues:

- PgPolicy = 11 = Leave closed. C10 will always be set to 1; the page will always be closed.

- PgPolicy = 01 = Usually closed. The queue will look at all of its enabled entries and if any requests are still pending to that page then C10 will be set to 0 to leave the page open. If no entries in that queue are directed to that page, then C10 will be set to 1 to close the page.
- PgPolicy = 10 = Leave open. C10 will always be set to 0; the page will always be left open.
- PgPolicy = 00 = Usually open. The queue will look at all of its enabled entries and if no requests are still pending to that page and there are no other requests in that queue to a different page of the same rank and bank, then C10 will be set to 1 to close the page. Otherwise C10 will be set to 0 to leave the page open.

7.17.5 Queue Entry Aging

The “pizza box” operation of the queues, in which new entries always enter the top and move down as entries below them are removed, implies that the entries in each queue are sorted by age, with the oldest at the bottom and the newest at the top.

The Command Arbiter favors read and write commands to open pages over other commands. This implies that some entries in a queue can stay there for a very long time. For example, if an occasional request to a closed page is sprinkled within large streams of requests to open pages, the “closed page” requests will just drift down to the bottom of the queue and stay there while newer entries above them are serviced.

To place an upper bound on the amount of time an entry stays in the queue, the two queues each have an “Aging” counter monitoring the length of time an entry stays in the bottom-most slot. When the aging counter hits a programmed limit it signals the Command Arbiter. The Command Arbiter will then switch its priority, when servicing that queue, to servicing the bottom-most slot.

The aging limit can vary by the source of the request: PCI Express, Hypertransport, or PI. (For PCIe there is no distinction between coherent and non-coherent requests. For PI there is no distinction among the 4 CPUs and the I^{2}C access.)

For the Read Reorder Queue the aging limits are programmed with the PcieAgCnt, PcieAgMd, HtAgCnt, HtAgMd, ApiAgCnt and ApiAgMd fields of the MemRdQCnfg register (*Section 12.10.10 Memory Read Request Queue Configuration Register (MemRdQCnfg)* on page 467). For the Write Reorder Queue the aging limits are programmed with the PcieWrAgCnt, PcieWrAgMd, HtWrAgCnt, HtWrAgMd, ApiWrAgCnt and ApiWrAgMd fields of the MemWrQCnfg register (*Section 12.10.11 Memory Write Request Queue Configuration Register (MemWrtQCnfg)* on page 468).

If the AgCnt for one of the queues for a particular request source = 0xF, aging for that type of source in that queue is disabled. Otherwise, the aging count is = AgCnt * 1, 4, 16 or 64, depending on the value of the associated AgMd.

The Aging Count for the Read Reorder Queue is time based: the aging counter is loaded when the bottom-most slot receives a new entry and decrements towards 0 on every mem_clk.

The Aging Count for the Write Reorder Queue is event based: the aging counter is loaded when the bottom-most slot receives a new entry and decrements towards 0 each time an entry higher up in the queue is retired.

Note: There is only one timer per queue. Therefore, if a queue’s aging for a given request source is disabled, and a request from that source enters the bottom-most slot of that queue, then aging for that queue is effectively disabled until the entry from the bottom-most slot is eventually retired (if ever).

In general the longest time that an entry with disabled aging can stay in the bottom-most slot of the Read Reorder Queue is the time between 2 refreshes, plus the time to do a refresh sequence. This is because a refresh closes all open pages, and the Command Arbiter prioritizes requests of the same type from oldest to newest. Following a refresh sequence, all entries will be request an Activate, and the Command Arbiter will select the bottom-most (oldest) entry.

The same cannot be said of the Write Reorder Queue, because the Command Arbiter favors reads over writes. In the “worst” case, the Command Arbiter can continuously service the Read Reorder Queue and refresh requests while entries remain in the Write Reorder Queue forever. (But as described below, write conflicts and high water marks can switch priority to the Write Reorder Queue.)

7.17.6 Queue Write Conflicts

Because of the Read and Write Reorder Queues and the Command Arbiter policies, read and write requests presented to the memory controller can be executed out-of-order (hence the name, “Reorder Queue”). However, when read requests are presented with the same address as a preceding write request, the write will be guaranteed to be performed first (“reads push writes”).

Incoming read requests have their address compared with the addresses of all entries in the Write Reorder Queue. If there are one or more matches:

- The read request is put in the Read Reorder Queue and flagged as having a “Write Conflict.”
- The Read Reorder Queue entry also has a pointer set to latest write in the Write Reorder Queue having the conflict. (The pointer is updated as the write request drops down in the Write Reorder Queue.)
- All entries in the Write Reorder Queue having the matching address are flagged as having a Write Conflict.

Entries in the Read Reorder Queue that are flagged as having a Write Conflict are ignored by the Command Arbiter with regard to their CV requests. Hence, these entries cannot have their Read Command requests honored.

For entries flagged as having write conflicts, when the entry in the Write Reorder Queue pointed to by an entry in the Read Reorder Queue is retired, the write conflict flag is reset in the Read Reorder Queue entry. Now, that entry in the Read Reorder Queue can have its Read Command request honored and executed.

Reads normally have priority over writes, but entries in the Write Reorder Queue flagged as having write conflicts will cause the Command Arbiter to switch to servicing those entries, to prevent deadlock. See *Section 7.18.1 Queue to Queue Arbitration* on page 188 for more details.

7.17.7 Queue Write High Watermarks

As mentioned previously the Command Arbiter favors reads over writes. This can cause write requests to remain in the Write Reorder Queue much longer than read requests; this is the reason that the physical size of the Write Reorder Queue is twice the physical size of the Read Reorder Queue.

However, it would be bad to let the Write Reorder Queue fill up and stay that way for a long period of time, therefore the Write Reorder Queue has programmable high watermarks. There are three types of high watermark conditions:

- The number of all entries in the Write Reorder Queue has reached its specified limit.
- The number of all entries from HT requests in the Write Reorder Queue has reached its specified limit.

- The number of all entries from PCIe requests in the Write Reorder Queue has reached its specified limit.

These limits are specified by the WrQMk, HtWDBMk and PcWDBMk fields, in the MemRWArb Register (*Section 12.10.13 Memory R/W Arbitration Register (MemRWArb)* on page 471).

When a high watermark condition is met the Command Arbiter will switch its priority to the Write Reorder Queue. See *Section 7.18.1 Queue to Queue Arbitration* for more details.

The reason for having separate high watermarks for HT and PCIe, and the reason they are called “WDB” watermarks, is because of the fact that when these units request writes they put their write data into a Write Data Buffer (WDB) in the PI unit. There could be potential “lockup” scenarios in which a bus stalls because its WDB is filled, but no requests are made to the memory controller which would switch the Command Arbiter to servicing the write requests associated with this data. The HT and PCIe watermarks prevent these lockouts because the watermarks cannot be disabled. Their maximum size of 8 is equal to the maximum number of entries in the HT and PCIe WDBs, therefore if their WDB is filled up it is guaranteed that the associated watermark will be hit, the Command Arbiter will switch to the Write Reorder Queue, and eventually the write requests will be serviced.

Although the WrQMk was originally intended to be a generic high watermark, its value is constrained by how the PI WDB is programmed. The APIWdbCfg register (*Section 12.9.7 API Write Data Buffer (WDB) Configuration Register (APIWdbCfg)* on page 417) has fields that guarantee the number of processor write and intervention write entries, on a per-CPU basis. ***The WrQMk must be set less than or equal to the total number of “non-guaranteed” entries in the PI WDB, or a lockup condition can occur.***

If a write high watermark is programmed to a larger value than the programmed size of the Write Reorder Queue, its actual value is the same as the programmed size of the queue.

If the WrQMk is programmed as 0, the actual watermark value is = 1.

7.17.8 Queue Grant Mode

The QGrntMd bit in the MemRdQCnfg register, when set, prevents the Memory Request Arbiter from accepting requests when either queue is full (except for Reads that can be immediately executed using the Fast Path).

This bit is provided for diagnostic purposes (see *Section 7.18.7 Enforced Ordering*) and is normally set to 0.

7.18 Command Arbiter

The Command Arbiter services requests whenever the Memory Programming Control is not active (see *Section 7.9 Memory Programming Control*). The Command Arbiter services two basic requests, in the following order:

1. Refresh requests.
2. Queue requests.

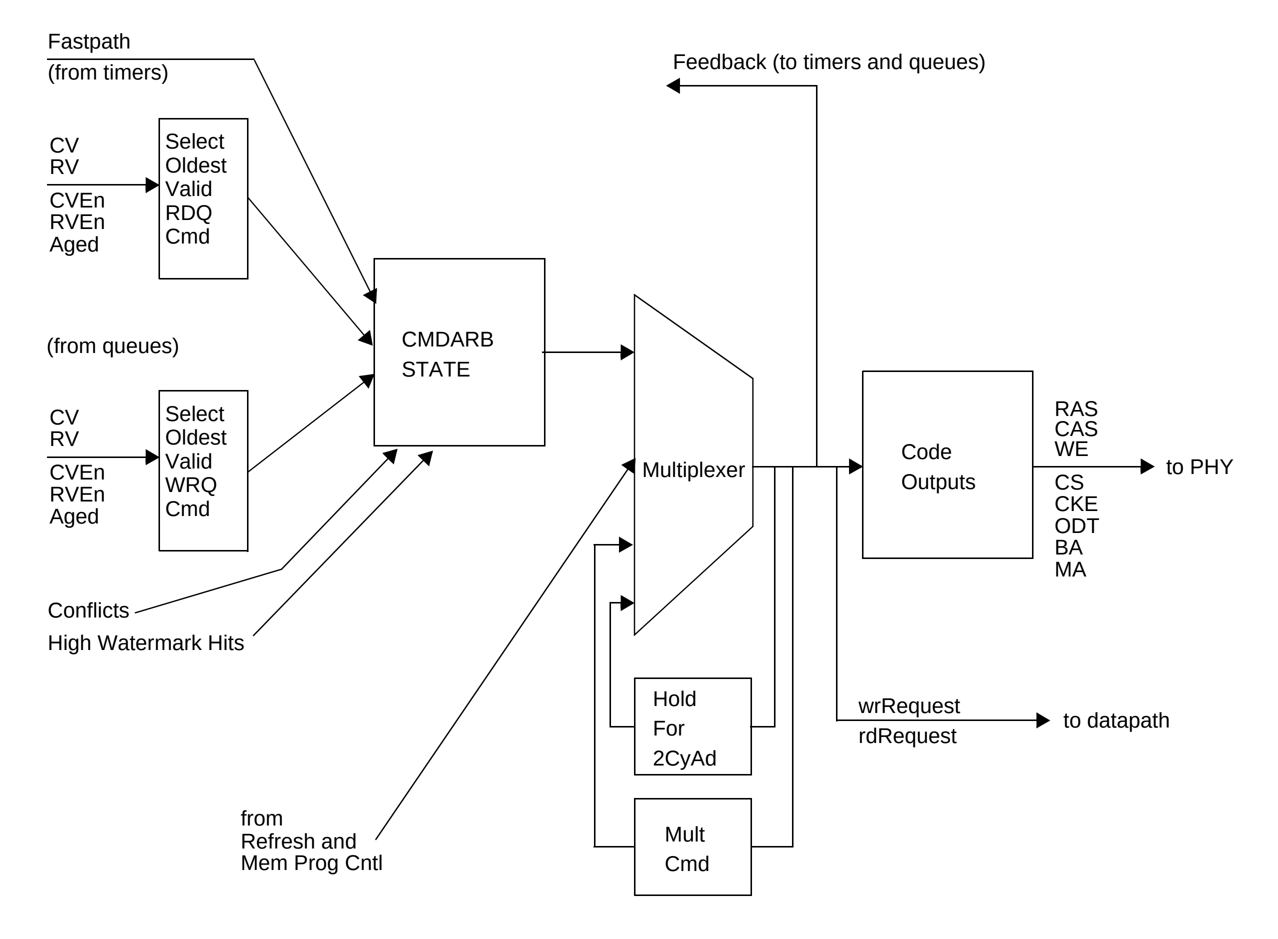
Figure 7-5 shows the basic flow of the Command Arbiter. The CMDARB state machine arbitrates among the two queues and the Fast Path (*Section 7.18.1*, *Section 7.18.2* and *Section 7.18.4*). Each queue has selection among its entries (*Section 7.18.3*). A multiplexer is used to select SDRAM commands generated by the Refresh unit (or the Memory Programming Control unit) or the arbitrated SDRAM command from the CMDARB state machine. The output of the multiplexer is sent to the Code Output unit which sends the signals to the SDRAMs. The Code Output unit also generates Chip Selects, CKEs and ODTs.

Information about the SDRAM command selected by the multiplexer is fed to the datapath controls so that Write data is sent at the proper time, Read data is captured at the proper time, and the data bus is enabled/tri-stated at the proper times.

As discussed in *Section 7.16* and *Section 7.17*, command information is fed back to the Page Table / Timers, and the two reorder queues, so they can modify their states and timing values based on the actual commands that are sent to the memories.

The outputs of the multiplexer can also feed back to the inputs to support two cycle addressing (*Section 7.18.5*) and “Multiple Commands” (RMWs and 128 byte transfers, *Section 7.18.6*).

Figure 7-5. Command Arbiter Flow



When the Memory Programming Control unit is active the Command Arbiter does no arbitration: it simply passes the commands from the Memory Programming Control unit through the multiplexer. (Using the Refresh port; the Memory Programming Control unit and the Refresh control unit share a common port into the Command Arbiter.)

Refresh requests are straight-forward: the Command Arbiter accepts the request, passes the commands through the multiplexer, and returns to arbitration.

The Queue request arbitration is more complex, taking into account the RV and CV signals from the fast path and the two queues, the aging indications, write conflict flags, write high watermarks, and controls in the MemRdQCnfg, MemQArb and MemRWArb registers.

7.18.1 Queue to Queue Arbitration

If one queue is empty and the other queue is not empty, the Command Arbiter will service the non-empty queue.

If neither queue is empty, reads have priority over writes, and in the general case the Command Arbiter will service the Read Reorder Queue.

When the Command Arbiter switches priority to a new queue, it resets a burst counter for the new queue, As entries in the new queue are serviced the burst counter counts the number of entries which are retired. When the burst count hits a programmed limit, it can cause the Command Arbiter to switch to the opposite queue.

The programmed burst limits are the WrBurst and RdBurst fields in the MemRWArb register (*Section 12.10.13 Memory R/W Arbitration Register (MemRWArb)* on page 471). If the programmed burst size is greater than the programmed size of a queue, it is the same as setting the burst size equal to the programmed size of that queue.

When a high water mark is hit, the Command Arbiter will switch priority to the Write Reorder Queue after the read burst count is met. Once the Write Reorder Queue receives priority, it retains it until the write burst count is met.

A write conflict flagged in the oldest (bottom-most) entry of the Read Reorder Queue will cause the Command Arbiter to switch priority to the Write Reorder Queue, regardless of the read burst count. As soon as the write(s) with the conflicts are retired (which resets the write conflict flag for that entry in the Read Reorder Queue), the Command Arbiter can switch back to the Read Reorder Queue, regardless of the write burst count.

If the Command Arbiter is servicing the Write Reorder Queue because the Read Reorder Queue was empty, and the Read Reorder Queue becomes not empty, the priority will switch back to the Read Reorder Queue, regardless of the write burst count.

7.18.2 Fast Path

If both queues are empty, a new request is a read, and the Fast Path is enabled, the first SDRAM command associated with that read (for example, Precharge, Activate or Read) will be accepted immediately by the Command Arbiter, bypassing the Read Reorder Queue. (This first command is determined by the results of the Page Table lookup.)

If the first SDRAM command was not a Read command, the request is entered in the Read Reorder queue with timing values and status appropriate for its next SDRAM command (for example, Activate, Read). The entry will be processed by the Command Arbiter as usual with the RV and CV signals.

If the first SDRAM command was itself a Read command, the Read Reorder Queue is bypassed entirely.

If Dynamic CKE is enabled (*Section 7.25.6 Dynamic CKE*) then the Fast Path is disabled. The Fast Path can also be disabled with the FastPathDis bit of the MemRWArb register.

7.18.3 Intra Queue Arbitration

Each queue presents a vector of RVs (RAS Command Valid) and a vector of CVs (CAS Command Valid). The Command Arbiter prioritizes the vectors by age of the queue that has priority. In general it selects the oldest enabled request.

The RdQCVEn, RdQRVEn, WrQCVEn and WrQRVEn bits in the MemQArb register (*“Memory Reorder Queue Arbitration Register (MemQArb)” on page 470*) specify the number of CVs and RVs that are examined by the Command Arbiter (counting from the bottom of the queue). Thus, the CV Enables can limit how deep the Command Arbiter examines a queue for data commands, and the RV Enables can limit how deep the Command Arbiter examines a queue for non-data commands. The deeper the examination, the more out of order the requests can be processed.

The default values of 0 enable examination of all entries in the queue. If an Enable for a queue is set to a higher value than the programmed size of that queue, it is the same as enabling examination of all entries in that queue.

As described above in *“Queue Entry Aging” on page 183*, if an aging counter hits its limit for a queue that has priority, the priority within that queue will be forced to the bottom-most entry, regardless of the CV and RV status of any of the entries. Within that queue, priority stays with the bottom-most entry until the bottom-most entry is retired.

7.18.4 RW Arbitration Mode

As described above in *Section 7.18.1 Queue to Queue Arbitration*, a given queue will have priority and only commands from that queue will be accepted by the Command Arbiter, until priority switches to the opposite queue. Using the RWArbMd bits in the MemRWArb register, it is possible to modify this restriction.

- When enabled with RWArbMd[1], if the Write Reorder Queue has priority but no CAS commands (CVs) in that queue are valid, and if the Read Reorder Queue has a valid CAS command, then the Command Arbiter will accept that request and perform the CAS (Read) Command.
- When enabled with RWArbMd[0], if the Read Reorder Queue has priority but no CAS commands in that queue are valid, and if the Write Reorder Queue has a valid CAS command, then the Command Arbiter will accept that request and perform the CAS (Write) Command.

  When RWArbMd[0] = 1 (enabled), it is required that all aging counts in the Read Reorder Queue be set to a non-zero value. Otherwise reads can be locked out from execution if many writes are streamed to open pages.

Priority stays with the first queue, until the usual reason for switching priority to the opposite queue is hit (burst count, write conflict, high watermark).

7.18.5 Two Cycle Addressing

Normally when the memory controller issues a command it asserts the appropriate Chip Select on the same CK cycle. But if two cycle address is set (*Section 12.10.14 Memory Bus Configuration Register (MemBus-Config)* on page 472, Ad2Cy bit), the command (and its address) is held an additional CK, and the Chip Select is asserted on the second clock. *Figure 7-5* shows that the Command Arbiter feeds back and reuses its issued commands to implement two cycle addressing.

The purpose of two cycle addressing is to allow enough time for the commands and addresses to propagate on the board if those signals are heavily loaded.

When two cycle addressing is not enabled, the Command Arbiter can achieve command and address bus optimizations by inserting opportunistic precharges and activates during otherwise dead cycles. But when two cycle addressing is enabled, this generally precludes these opportunistic cycles, and therefore performance is degraded.

7.18.6 Multiple Commands

There are two cases in which the Command Arbiter must execute multiple commands based on accepting a single command request from a queue:

- When the Write Reorder Queue indicates that a write requires a RMW (Read-Modify-Write).
- When the transfer size is 128 bytes.

Figure 7-5 shows that the Command Arbiter feeds back its command status to synthesize these multiple commands.

7.18.6.1 RMW

For RMWs, the Command Arbiter does not permit any other commands to execute between the Read and the Write.

RMWs are an “expensive” operation. Streams of back-to-back writes or back-to-back reads can keep the external data bus always transferring data. But a RMW requires that, after the Read is performed, that the memory controller wait for the memory devices to tri-state the data bus, then the memory controller enables its data bus drivers, and finally the write data can be written out.

The Command Arbiter uses the TiRtWRMW field in the MemRWArb register to determine how long to wait after issuing the Read command before issuing the Write command. The setting of this value is dependant on the external memory configuration (registered/unbuffered DIMMs, board delays, and so on). It is important to set this value to a high enough value to allow the data bus to turn around: if the value of TiRtWRMW is too small data corruption will occur. On the other hand, it is important to not make this value too large, because it will lengthen an already expensive operation.

See *Section 7.8.2.1 Read-Modify-Writes* on page 145 for more details on RMW.

7.18.6.2 128 Byte Transfers

With a burst size of 4 for the 128-bit bus configurations and a burst size of 8 for the 64-bit bus configurations, the CPC945 always transfers up to 4 * 16 or 8 * 8 = 64 bytes for a given Read or Write. In addition to requests of 64 bytes or less, the memory controller must also handle requests to transfer 128 bytes.

For 128 byte transfer sizes, when the Command Arbiter accepts a CAS command (Read or Write) it will feed that command back and re-issue it, thus generating two back-to-back Read commands or Write commands. The Command Arbiter can accept and issue nondata commands between the two back-to-back data commands.

7.18.7 Enforced Ordering

Ordering can be enforced (reads and writes are executed exactly in the order they are received by the memory controller), by setting the queue sizes to 1 (SzRdQ = 1 and SzWrQ = 1 in the MemRdQCnfg and MemWrQCnfg Registers), disabling the Fast Path (FastPathDis = 1 in the MemRWArb register) and not allowing requests to enter one queue when the other queue already has an entry (QGrntMd = 1 in the MemRdQCnfg register).

This is for diagnostic purposes. Enforced ordering will significantly degrade performance.

7.19 ECC

7.19.1 ECC Introduction

The CPC945 DDR2 memory controller implements an ECC (error checking and correction) code. When ECC is enabled:

- On writes, in addition to writing 128 bits of data, 16 additional check bits are written for a total of 144 bits.
- On reads, the memory controller is able to use the check bits to determine if an error has occurred (error checking). For some errors, the memory controller can replace the bad data from memory with good data before returning the data to the requestor (error correction).

Technically, the CPC945 implements a (144,128) SSC/DSD (single symbol correction/double symbol detection) ECC code, in which there are 4-bits per symbol. This is also referred to as a S4EC/D4ED code.

The 144 bits are divided into 4-bit nibbles (symbols) which lie on 4-bit boundaries:

- The ECC code will detect and correct up to 4 bits in error, if all 4 bits are from the same nibble.
- The ECC code can detect errors of more than 4 bits:
  - Up to 8 bits in error will be detected if the errors are confined to 2 nibbles.
  - Errors that span more than 2 nibbles might or might not be detected.

Note: When 4-bit wide memory chips are used, an entire memory chip can fail and the ECC code can still correct the 4 bits of data for that chip. This is known as “chip kill” support. Chip kill support does not have to be explicitly enabled; it is inherent for the CPC945 ECC code when 4-bit wide memory chips are used.

In addition to checking/correction of data, the ECC code also incorporates an address parity bit (AP) and a Special Uncorrectable Error bit (SPUE), which are discussed below.

ECC uses the MCCR (Memory Check Control Register at 0xF8002440), which contains the ECC_EN (ECC Enable) bit and other control bits, the MESR (Memory Error Syndrome Register at 0xF8002480) which contains status information when an error is detected, and MEAR0 and MEAR1 (Memory Error Address Register 0 and 1 at 0xF8002460 and 0xF8002470) which contains the address of the data which caused an error detection and error counts.

The CPC945 DDR2 memory bus bits are labeled [0:143]. Bus bits 0:127 are used for data bits 0:127 and bus bits 128:143 are used for check bits 0:15. DQS[16:17] are the strobes for the check bytes. (DQS [32:35] in single-ended mode.)

Because the code operates over 128 bits, only the 128-bit memory configuration is supported with ECC. ECC must not be enabled when using one of the 64-bit configurations.

By default ECC is disabled. In this condition, DQ[128:143] and DQS[16:17] (DQS [32:35] in single-ended mode) are always tri-stated. ECC errors are never generated. The DDR2 PHY Registers which control the timings of the check bits and the check bit DQS signals can be left at their default values.

When enabled with the ECC_EN bit in the MCCR, the check bit DQS signals operate the same as the rest of the DQS signals, check bits are generated on writes and checked on reads, and ECC errors can be generated. The check bits and their DQS signals must be adjusted using the DDR2 PHY Registers for correct timing, the same as for the data bits.

Table 7-33. Hamming Matrix.

	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	E17
	0 0 0 0 0 1 2 3	0 0 0 0 4 5 6 7	0 0 1 1 8 9 0 1	1 1 1 1 2 3 4 5	1 1 1 1 6 7 8 9	2 2 2 2 0 1 2 3	2 2 2 2 4 5 6 7	2 2 3 3 8 9 0 1	3 3 3 3 2 3 4 5	3 3 3 3 6 7 8 9	4 4 4 4 0 1 2 3	4 4 4 4 4 5 6 6	4 4 5 5 8 9 0 1	5 5 5 5 2 3 4 5	5 5 5 5 6 7 8 9	6 5 6 6 0 1 2 3	6 6 6 6 4 5 6 7	6 6 7 7 8 9 0 1
0	0 0 0 0	1 1 0 0	0 0 0 1	1 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	1 0 0 0	0 0 0 0	0 1 0 0	1 0 0 0	1 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 0 1 0	0 0 0 0	1 0 0 0
1	0 0 0 0	0 0 1 0	1 0 0 1	0 1 0 0	0 0 0 0	1 0 0 1	0 0 1 0	0 1 0 0	0 0 0 0	0 1 1 0	0 1 0 0	0 1 0 0	0 0 0 0	0 1 0 0	0 1 0 0	0 0 1 1	0 0 0 0	0 1 0 0
2	0 0 0 0	0 0 0 1	0 1 0 0	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 1	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 0	0 0 1 0	0 0 0 0	0 0 1 0	0 0 1 0	1 0 0 1	0 0 0 0	0 0 1 0
3	0 0 0 0	1 0 0 0	0 0 1 0	0 0 0 1	0 0 0 0	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 0	1 0 0 1	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 1	0 0 0 1	0 1 0 0	0 0 0 0	0 0 0 1
4	1 0 0 0	0 0 0 0	1 1 0 0	0 0 0 1	1 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	1 0 0 0	0 0 0 0	0 1 0 0	1 0 0 0	0 0 1 0	0 0 0 0	1 0 0 0	1 0 0 0	1 1 1 0	0 0 0 0
5	0 1 0 0	0 0 0 0	0 0 1 0	1 0 0 1	0 1 0 0	0 0 0 0	1 0 0 1	0 0 1 0	0 1 0 0	0 0 0 0	0 1 1 0	0 1 0 0	0 0 1 1	0 0 0 0	0 1 0 0	0 1 0 0	0 0 0 1	0 0 0 0
6	0 0 1 0	0 0 0 0	0 0 0 1	0 1 0 0	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 1	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 0	1 0 0 1	0 0 0 0	0 0 1 0	0 0 1 0	1 0 0 0	0 0 0 0
7	0 0 0 1	0 0 0 0	1 0 0 0	0 0 1 0	0 0 0 1	0 0 0 0	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 0	1 0 0 1	0 0 0 1	0 1 0 0	0 0 0 0	0 0 0 1	0 0 0 1	1 1 0 0	0 0 0 0
8	0 0 0 1	1 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0	1 0 0 0	0 0 0 0	0 1 0 0	1 0 0 0	0 0 1 0	0 0 0 0	1 0 0 0	0 0 0 1	1 1 1 0
9	1 0 0 1	0 1 0 0	0 0 0 0	0 0 1 0	0 0 1 0	0 1 0 0	0 0 0 0	1 0 0 1	0 1 0 0	0 1 0 0	0 0 0 0	0 1 1 0	0 1 0 0	0 0 1 1	0 0 0 0	0 1 0 0	1 0 0 1	0 0 0 1
10	0 1 0 0	0 0 1 0	0 0 0 0	0 0 0 1	0 0 0 1	0 0 1 0	0 0 0 0	0 1 0 0	0 0 1 0	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 0	1 0 0 1	0 0 0 0	0 0 1 0	0 1 0 0	1 0 0 0
11	0 0 1 0	0 0 0 1	0 0 0 0	1 0 0 0	1 0 0 0	0 0 0 1	0 0 0 0	0 0 1 0	0 0 0 1	0 0 0 1	0 0 0 0	1 0 0 1	0 0 0 1	0 1 0 0	0 0 0 0	0 0 0 1	0 0 1 0	1 1 0 0
12	1 1 0 0	0 0 0 1	1 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0	1 0 0 0	0 0 0 0	0 1 0 0	1 0 0 0	1 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	0 0 1 0	0 0 0 0	1 0 0 0	0 0 0 1
13	0 0 1 0	1 0 0 1	0 1 0 0	0 0 0 0	1 0 0 1	0 0 1 0	0 1 0 0	0 0 0 0	0 1 1 0	0 1 0 0	0 1 0 0	0 0 0 0	0 1 0 0	0 1 0 0	0 0 1 1	0 0 0 0	0 1 0 0	1 0 0 1
14	0 0 0 1	0 1 0 0	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 1	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 0	0 0 1 0	0 0 0 0	0 0 1 0	0 0 1 0	1 0 0 1	0 0 0 0	0 0 1 0	0 1 0 0
15	1 0 0 0	0 0 1 0	0 0 0 1	0 0 0 0	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 0	1 0 0 1	0 0 0 1	0 0 0 1	0 0 0 0	0 0 0 1	0 0 0 1	0 1 0 0	0 0 0 0	0 0 0 1	0 0 1 0

Table 7-34. Hamming Matrix Continued.

	E18	E19	E20	E21	E22	E23	E24	E25	E26	E27	E28	E29	E30	E31		E32	E33	E34	E35
	7 7 7 7 2 3 4 5	7 7 7 7 6 7 8 9	8 8 8 8 0 1 2 3	8 8 8 8 4 5 6 7	8 8 9 9 8 9 0 1	9 9 9 9 2 3 4 5	9 9 9 9 6 7 8 9	1 1 1 1 0 0 0 0 0 1 2 3	1 1 1 1 0 0 0 0 4 5 6 7	1 1 1 1 0 0 1 1 8 9 0 1	1 1 1 1 1 1 1 1 2 3 4 5	1 1 1 1 1 1 1 1 6 7 8 9	1 1 1 1 2 2 2 2 0 1 2 3	1 1 1 1 2 2 2 2 4 5 6 7	S A P P U	1 1 1 1 2 2 3 3 8 9 0 1	1 1 1 1 3 3 3 3 2 3 4 5	1 1 1 1 3 3 3 3 6 7 8 9	1 1 1 1 4 4 4 4 0 1 2 3
0	0 0 0 1	1 1 1 0	0 0 0 0	1 0 0 0	1 1 0 0	0 1 0 0	0 0 0 0	1 0 0 0	0 0 1 1	1 0 0 0	1 0 0 0	0 0 0 1	1 0 0 0	1 0 0 0	1 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	1 0 0 1	0 0 0 1	0 0 0 0	0 1 0 0	0 0 1 0	0 1 1 0	0 0 0 0	0 1 0 0	1 0 1 0	0 1 0 0	0 1 0 0	1 0 0 1	0 1 0 0	0 1 0 0	0 1	0 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0
2	0 1 0 0	1 0 0 0	0 0 0 0	0 0 1 0	0 0 0 1	0 0 1 1	0 0 0 0	0 0 1 0	1 1 0 1	0 0 1 0	0 0 1 0	0 1 0 0	0 0 1 0	0 0 1 0	0 0	0 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0
3	0 0 1 0	1 1 0 0	0 0 0 0	0 0 0 1	1 0 0 0	1 0 0 1	0 0 0 0	0 0 0 1	0 1 1 0	0 0 0 1	0 0 0 1	0 0 1 0	0 0 0 1	0 0 0 1	0 0	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0
4	1 0 0 0	0 0 0 1	0 1 0 0	0 0 0 0	1 0 0 0	1 1 0 0	1 0 0 0	0 0 0 0	1 0 0 0	0 0 1 1	1 0 0 0	1 0 0 0	0 0 0 1	1 0 0 0	1 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0
5	0 1 0 0	1 0 0 1	0 1 1 0	0 0 0 0	0 1 0 0	0 0 1 0	0 1 0 0	0 0 0 0	0 1 0 0	1 0 1 0	0 1 0 0	0 1 0 0	1 0 0 1	0 1 0 0	0 1	0 0 0 0	0 1 0 0	0 0 0 0	0 0 0 0
6	0 0 1 0	0 1 0 0	0 0 1 1	0 0 0 0	0 0 1 0	0 0 0 1	0 0 1 0	0 0 0 0	0 0 1 0	1 1 0 1	0 0 1 0	0 0 1 0	0 1 0 0	0 0 1 0	0 0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 0 0
7	0 0 0 1	0 0 1 0	1 0 0 1	0 0 0 0	0 0 0 1	1 0 0 0	0 0 0 1	0 0 0 0	0 0 0 1	0 1 1 0	0 0 0 1	0 0 0 1	0 0 1 0	0 0 0 1	0 0	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0
8	0 0 0 0	1 0 0 0	1 1 0 0	0 1 0 0	0 0 0 0	1 0 0 0	0 0 1 1	1 0 0 0	0 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0	0 0 0 1	0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0
9	0 0 0 0	0 1 0 0	0 0 1 0	0 1 1 0	0 0 0 0	0 1 0 0	1 0 1 0	0 1 0 0	0 0 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	1 0 0 1	0 0	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0
10	0 0 0 0	0 0 1 0	0 0 0 1	0 0 1 1	0 0 0 0	0 0 1 0	1 1 0 1	0 0 1 0	0 0 0 0	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	0 1 0 0	1 0	0 0 0 0	0 0 0 0	0 0 1 0	0 0 0 0
11	0 0 0 0	0 0 0 1	1 0 0 0	1 0 0 1	0 0 0 0	0 0 0 1	0 1 1 0	0 0 0 1	0 0 0 0	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 1 0	0 1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
12	1 1 1 0	0 0 0 0	1 0 0 0	1 1 0 0	0 1 0 0	0 0 0 0	1 0 0 0	0 0 1 1	1 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0	1 0 0 0	1 0 0 0	1 1	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0
13	0 0 0 1	0 0 0 0	0 1 0 0	0 0 1 0	0 1 1 0	0 0 0 0	0 1 0 0	1 0 1 0	0 1 0 0	0 0 0 0	1 0 0 1	0 1 0 0	0 1 0 0	0 1 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0
14	1 0 0 0	0 0 0 0	0 0 1 0	0 0 0 1	0 0 1 1	0 0 0 0	0 0 1 0	1 1 0 1	0 0 1 0	0 0 0 0	0 1 0 0	0 0 1 0	0 0 1 0	0 0 1 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
15	1 1 0 0	0 0 0 0	0 0 0 1	1 0 0 0	1 0 0 1	0 0 0 0	0 0 0 1	0 1 1 0	0 0 0 1	0 0 0 0	0 0 1 0	0 0 0 1	0 0 0 1	0 0 0 1	1 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1

7.19.2 Writes

ECC codes are implemented using a modified Hamming Matrix (H-Matrix). The CPC945 H-Matrix is given in *Table 7-33* and *Table 7-34*.

Bits 0 through 127 and AP are inputs to the check bit generator, and bits 128 through 143 are the generated check bits.

> AP is Address Parity. When enabled, address parity is calculated on a cache line size basis (odd parity is calculated for address bits 28:56), and injected into the check bit calculations. Address Parity is enabled by default; it can be disabled using bit 1, ECC_AP_DIS in the MCCR register. (When disabled, a “0” is injected into the check bit calculation.)
>
> SPUE is Special Uncorrectable Error. The H-Matrix has a column labeled SPUE but this is not a bit that is fed into the check bit calculations. SPUE is a condition that can be set for partial write requests that trigger read-modify-writes. SPUE is set if the read contains an uncorrectable error for the data to be written back on the write. The ECC unit forces the data bits to all 0’s on the write, and it forces the check bits to a special value. A subsequent read of that data will produce an uncorrectable error, and the syndrome will indicate that the error was from a previous SPUE (in the absence of other additional errors).

Check bits are parity bits: each check bit is generated by bit-wise Exclusive ORing (XORing) a specified set of data bits. Each row of the H-Matrix is an “equation” for generating a separate check bit.

For the inputs (data bits 0 through 127 and AP), the “1”s in the row indicate which bits are XORed. For the generated check bits (128 thorough 143), a “1” indicates which check bit is generated. For example, bits 4, 5, 11, 12, 23, 24, 25, 28, 37, 40, 44, 52, 56, 62, 68, 75, 76, 77, 78, 84, 88, 89, 93, 100, 106, 107, 108, 112, 119, 120, and “AP” are bit-wise XORed to form a check bit, and the “1” in the bit 128 indicates that bit 128 (check bit 0) is that check bit.

To avoid the case of having all 144 bits = 0, the generated check bits are inverted before they are sent off-chip. Therefore a write of all 0’s will generate all 1 check bits = 0xFFFF. (AP = 0.)

Check bits 2, 6, 10 and 14 cover an even number of bits, while the remaining check bits cover an odd number of bits. Writing all 1’s generates an internal check bit value of 0xDDDD, which because of the inversion becomes 0x2222 when written to memory. (AP = 0.)

When address parity is disabled, the AP bit = 0. When address parity is enabled, the AP bit is the inverted result of the bit-wise XOR of address bits 28:56, that is, AP = INV (A28 XOR A29 XOR ... A56). An address of 0x0_0000_0000 will generate AP = 1. An address of 0x0_0000_0080 will generate AP = 0.

The condition AP = 1 is XORed into check bits 0, 4, 10, 12 and 15 = 0x8829. A write of all 0’s with AP = 1 will write check bits = 0x77D6 to memory (0xFFFF XORed with 0x8829). A write of all 1’s with AP = 1 will write check bits = 0xAA0B to memory (0x2222 XORed with 0x8829).

For SPUE (*Section 7.19.4 Partial Writes*) the check bits generator is bypassed and special values are written to memory. These values are 0x4418 if Address Parity is disabled, 0x4418 if Address Parity is enabled and AP = 0, and 0xCC31 if Address Parity is enabled and AP = 1.

Table 7-35 summarizes the check bits that will be written to memory for the “interesting” cases of writing all zeros, writing all ones, and the write back of a quadword in a read-modify-write in which a UE was detected (SPUE).

Table 7-35. Check Bit Summary.

Data	AP = 0	AP = 1
All 0’s	0xFFFF	0x77D6
All 1’s	0x2222	0xAA0B
SPUE - data will be all 0’s	0x4418	0xCC31
AP = 0 when Address Parity is disabled or when Address[28:56] has an odd number of 1’s. AP = 1 when Address parity is enabled and when Address[28:56] has an even number of 1’s.		

7.19.3 Reads

On a read operation, the memory controller takes bits 0 through 127 and calculates check bits in the same manner as for writes. If there are no errors the check bits read in (bits 128 through 143) should be identical to the calculated check bits. The two sets of check bits, read value and calculated, are bit-wise XORed to compare the two vectors. The resultant 16-bit vector is called the Syndrome.

A syndrome of all 0’s indicates that the two vectors are indeed identical and that no error occurred. If any bit in the syndrome is a 1, then there is an error. The syndrome is further decoded to determine if the error is correctable or uncorrectable.

7.19.4 Partial Writes

Write requests of size 64 bytes will trigger a burst write operation of four 16-byte quadwords. New checkbits will be generated for each quadword. Write requests of 128 bytes will trigger 2 back-to-back write bursts of 4 quadwords each. Again, entirely new checkbits are written for each quadword.

Write requests of less than 64 bytes will trigger a read-modify-write operation in which a burst read of four quadwords is performed followed by a burst write of four quadwords. Error checking and correction will be performed on each of the four quadwords that are read. New checkbits will be generated for each of the four quadwords that are written.

- If a given quadword that is read in is entirely replaced with new write data, any error detected during the read of that quadword will be logged (if enabled), but otherwise the read data and check bits are discarded. Entirely new data and check bits for that quadword are written back.
- If a given quadword that is read is not to be modified with any new write data:
  - If no errors are detected, new checkbits are generated and written back. These checkbits will be identical to those read in.
  - If a correctable error is detected, the bad bits in the quadword are corrected. New checkbits are generated based on the corrected quadword, and the corrected quadword is written back with those checkbits.
  - If an uncorrectable error is detected, the 128 data bits that were read are replaced with all 0’s. New check bits are generated with the value 0x4418 or 0xCC31, depending on Address Parity. These check bits along with the 128 zeros are written back.
- If a given quadword that is read is to be partially modified with new write data:
  - If no errors are detected, new checkbits are generated and written back.
  - If a correctable error is detected, the bad bits in the quadword are corrected. New write bytes are merged with the corrected quadword. (This will cause corrected bits to be discarded if they are in

bytes that are replaced with new write data.) New checkbits are generated based on the corrected and merged quadword, and the corrected/merged quadword is written back with those checkbits.

– If an uncorrectable error is detected, the new write data is discarded. All 0's are written back for data, and 0x4418 or 0xCC31, depending on Address Parity, is written back for check bits.

7.19.5 Syndrome Decode

On a read, if the calculated check bits are bit-wise identical to the received check bits, the syndrome will be all 0's, indicating that no error was detected. If one or more bits is 1, then an error has been detected.

For non-zero syndromes, the syndrome is further decoded to determine the nature of the error. This is a two-step process, in which first the nibble in error is determined, and second the bit(s) in error are determined for that nibble.

7.19.5.1 Nibble in Error

The following boolean equations determine if there is a nibble in error. Conventions:

- S[0], S[1],... S[15] are the 16 syndrome bits calculated by the bitwise-XOR of the received 16 check bits with the generated 16 check bits.
- E[0], E[1], ... E[35] are the 36 possible nibbles in error.
  - If there is no error all nibbles in error will equal 0.
  - If there is an error confined to a nibble, one of the equations will be satisfied, indicating the nibble in error.
  - If there is more than one nibble in error, all nibbles in error should equal 0. This will be true if the bits in error are confined to 2 nibbles. If more than 2 nibbles have bits in error it is possible for a nibble in error equation to solve "true," falsely indicating a correctable error when in fact the error is uncorrectable. The probability of this happening increases with increasing number of bits in error.
- ! = logical NOT, & = logical AND, ^ = logical XOR, "==" indicates 2 bits are identical.

Note that check bits can be in error as well as data bits. That is, there are 32 nibble in error indicators for data and 4 nibble in error indicators for check bits.

In the H-Matrix shown in *Table 7-33* and *Table 7-34* the columns are grouped into nibbles and labeled E0, E1, ...E35. These labels are the 36 symbols used by the ECC code, and for error decoding the labels correspond with the E[0], E[1], ... E[35] nibble in error equations listed below.

The rows of the H-Matrix labeled 0, 1, ... 15 correspond to the 16 syndrome bits that are generated. It can be seen that the equations implement the Hamming Matrix. For example, in the H-Matrix nibble in error E[0] can only be generated if S[0], S[1], S[2] and S[3] are all 0's, AND S[7] is equal to S[8], AND S[4] XORed with S[7] is equal to S[9], AND... AND S[4] is equal to S[15].

```
E[0] = !S[0] & !S[1] & !S[2] & !S[3] &
         (S[ 8] ==  S[ 7]) &
         (S[ 9] == (S[ 4] ^ S[ 7])) &
         (S[10] ==  S[ 5]) &
         (S[11] ==  S[ 6]) &
         (S[12] == (S[ 4] ^ S[ 5])) &
         (S[13] ==  S[ 6]) &
         (S[14] ==  S[ 7]) &
```

```
          (S[15] ==  S[ 4]) ;

  E[ 1] = !S[4] & !S[5] & !S[6] & !S[7] &
          (S[ 0] == (S[ 8] ^ S[ 9])) &
          (S[ 1] ==  S[10]) &
          (S[ 2] ==  S[11]) &
          (S[ 3] ==  S[ 8]) &
          (S[12] ==  S[11]) &
          (S[13] == (S[ 8] ^ S[11])) &
          (S[14] ==  S[ 9]) &
          (S[15] ==  S[10]) ;

  E[ 2] = !S[8] & !S[9] & !S[10] & !S[11] &
          (S[ 0] ==  S[15]) &
          (S[ 1] == (S[12] ^ S[15])) &
          (S[ 2] ==  S[13]) &
          (S[ 3] ==  S[14]) &
          (S[ 4] == (S[12] ^ S[13])) &
          (S[ 5] ==  S[14]) &
          (S[ 6] ==  S[15]) &
          (S[ 7] ==  S[12]) ;

  E[ 3] = !S[12] & !S[13] & !S[14] & !S[15] &
          (S[ 4] ==  S[ 3]) &
          (S[ 5] == (S[ 0] ^ S[ 3])) &
          (S[ 6] ==  S[ 1]) &
          (S[ 7] ==  S[ 2]) &
          (S[ 8] == (S[ 0] ^ S[ 1])) &
          (S[ 9] ==  S[ 2]) &
          (S[10] ==  S[ 3]) &
          (S[11] ==  S[ 0]) ;

  E[ 4] = !S[0] & !S[1] & !S[2] & !S[3] &
          (S[ 8] == (S[ 4] ^ S[ 5])) &
          (S[ 9] ==  S[ 6]) &
          (S[10] ==  S[ 7]) &
          (S[11] ==  S[ 4]) &
          (S[12] ==  S[ 7]) &
          (S[13] == (S[ 4] ^ S[ 7])) &
          (S[14] ==  S[ 5]) &
          (S[15] ==  S[ 6]) ;

  E[ 5] = !S[4] & !S[5] & !S[6] & !S[7] &
          (S[ 0] ==  S[11]) &
          (S[ 1] == (S[ 8] ^ S[11])) &
          (S[ 2] ==  S[ 9]) &
          (S[ 3] ==  S[10]) &
          (S[12] == (S[ 8] ^ S[ 9])) &
          (S[13] ==  S[10]) &
          (S[14] ==  S[11]) &
          (S[15] ==  S[ 8]) ;

  E[ 6] = !S[8] & !S[9] & !S[10] & !S[11] &
          (S[ 0] == (S[12] ^ S[13])) &
          (S[ 1] ==  S[14]) &
          (S[ 2] ==  S[15]) &
          (S[ 3] ==  S[12]) &
          (S[ 4] ==  S[15]) &
          (S[ 5] == (S[12] ^ S[15])) &
          (S[ 6] ==  S[13]) &
          (S[ 7] ==  S[14]) ;

  E[ 7] = !S[12] & !S[13] & !S[14] & !S[15] &
          (S[ 4] == (S[ 0] ^ S[ 1])) &
          (S[ 5] ==  S[ 2]) &
          (S[ 6] ==  S[ 3]) &
          (S[ 7] ==  S[ 0]) &
          (S[ 8] ==  S[ 3]) &
          (S[ 9] == (S[ 0] ^ S[ 3])) &
          (S[10] ==  S[ 1]) &
          (S[11] ==  S[ 2]) ;

  E[ 8] = !S[0] & !S[1] & !S[2] & !S[3] &
          (S[ 8] ==  S[ 4]) &
          (S[ 9] ==  S[ 5]) &
          (S[10] ==  S[ 6]) &
          (S[11] ==  S[ 7]) &
          (S[12] ==  S[ 5]) &
          (S[13] == (S[ 5] ^ S[ 6])) &
          (S[14] == (S[ 6] ^ S[ 7])) &
          (S[15] == (S[ 4] ^ S[ 7])) ;

  E[ 9] = !S[4] & !S[5] & !S[6] & !S[7] &
          (S[ 0] ==  S[ 9]) &
          (S[ 1] == (S[ 9] ^ S[10])) &
          (S[ 2] == (S[10] ^ S[11])) &
          (S[ 3] == (S[ 8] ^ S[11])) &
          (S[12] ==  S[ 8]) &
          (S[13] ==  S[ 9]) &
          (S[14] ==  S[10]) &
          (S[15] ==  S[11]) ;

  E[10] = !S[8] & !S[9] & !S[10] & !S[11] &
          (S[ 4] ==  S[ 1]) &
          (S[ 5] == (S[ 1] ^ S[ 2])) &
          (S[ 6] == (S[ 2] ^ S[ 3])) &
          (S[ 7] == (S[ 0] ^ S[ 3])) &
          (S[12] ==  S[ 0]) &
          (S[13] ==  S[ 1]) &
          (S[14] ==  S[ 2]) &
          (S[15] ==  S[ 3]) ;

  E[11] = !S[12] & !S[13] & !S[14] & !S[15] &
          (S[ 4] ==  S[ 0]) &
          (S[ 5] ==  S[ 1]) &
          (S[ 6] ==  S[ 2]) &
          (S[ 7] ==  S[ 3]) &
          (S[ 8] ==  S[ 1]) &
          (S[ 9] == (S[ 1] ^ S[ 2])) &
          (S[10] == (S[ 2] ^ S[ 3])) &
          (S[11] == (S[ 0] ^ S[ 3])) ;

  E[12] = !S[0] & !S[1] & !S[2] & !S[3] &
          (S[ 4] ==  S[10]) &
          (S[ 5] == (S[10] ^ S[11])) &
          (S[ 6] == (S[ 8] ^ S[11])) &
          (S[ 7] ==  S[ 9]) &
          (S[12] ==  S[ 8]) &
          (S[13] ==  S[ 9]) &
          (S[14] ==  S[10]) &
          (S[15] ==  S[11]) ;

  E[13] = !S[4] & !S[5] & !S[6] & !S[7] &
          (S[ 8] ==  S[ 2]) &
          (S[ 9] == (S[ 2] ^ S[ 3])) &
          (S[10] == (S[ 0] ^ S[ 3])) &
          (S[11] ==  S[ 1]) &
          (S[12] ==  S[ 0]) &
          (S[13] ==  S[ 1]) &
          (S[14] ==  S[ 2]) &
          (S[15] ==  S[ 3]) ;

  E[14] = !S[8] & !S[9] & !S[10] & !S[11] &
          (S[ 4] ==  S[ 0]) &
          (S[ 5] ==  S[ 1]) &
          (S[ 6] ==  S[ 2]) &
          (S[ 7] ==  S[ 3]) &
          (S[12] ==  S[ 2]) &
          (S[13] == (S[ 2] ^ S[ 3])) &
          (S[14] == (S[ 0] ^ S[ 3])) &
          (S[15] ==  S[ 1]) ;

  E[15] = !S[12] & !S[13] & !S[14] & !S[15] &
          (S[ 0] ==  S[ 6]) &
          (S[ 1] == (S[ 6] ^ S[ 7])) &
          (S[ 2] == (S[ 4] ^ S[ 7])) &
          (S[ 3] ==  S[ 5]) &
          (S[ 8] ==  S[ 4]) &
          (S[ 9] ==  S[ 5]) &
          (S[10] ==  S[ 6]) &
          (S[11] ==  S[ 7]) ;

  E[16] = !S[0] & !S[1] & !S[2] & !S[3] &
          (S[ 4] == (S[12] ^ S[13] ^ S[14])) &
          (S[ 5] ==  S[15]) &
          (S[ 6] ==  S[12]) &
          (S[ 7] == (S[12] ^ S[13])) &
          (S[ 8] ==  S[15]) &
          (S[ 9] == (S[12] ^ S[15])) &
          (S[10] ==  S[13]) &
          (S[11] ==  S[14]) ;

  E[17] = !S[4] & !S[5] & !S[6] & !S[7] &
          (S[ 8] == (S[ 0] ^ S[ 1] ^ S[ 2])) &
          (S[ 9] ==  S[ 3]) &
          (S[10] ==  S[ 0]) &
          (S[11] == (S[ 0] ^ S[ 1])) &
          (S[12] ==  S[ 3]) &
          (S[13] == (S[ 0] ^ S[ 3])) &
          (S[14] ==  S[ 1]) &
          (S[15] ==  S[ 2]) ;

  E[18] = !S[8] & !S[9] & !S[10] & !S[11] &
          (S[ 0] ==  S[ 7]) &
          (S[ 1] == (S[ 4] ^ S[ 7])) &
          (S[ 2] ==  S[ 5]) &
          (S[ 3] ==  S[ 6]) &
          (S[12] == (S[ 4] ^ S[ 5] ^ S[ 6])) &
          (S[13] ==  S[ 7]) &
          (S[14] ==  S[ 4]) &
          (S[15] == (S[ 4] ^ S[ 5])) ;

  E[19] = !S[12] & !S[13] & !S[14] & !S[15] &
          (S[ 0] == (S[ 8] ^ S[ 9] ^ S[10])) &
          (S[ 1] ==  S[11]) &
          (S[ 2] ==  S[ 8]) &
          (S[ 3] == (S[ 8] ^ S[ 9])) &
          (S[ 4] ==  S[11]) &
          (S[ 5] == (S[ 8] ^ S[11])) &
          (S[ 6] ==  S[ 9]) &
          (S[ 7] ==  S[10]) ;

  E[20] = !S[0] & !S[1] & !S[2] & !S[3] &
          (S[ 4] ==  S[13]) &
          (S[ 5] == (S[13] ^ S[14])) &
          (S[ 6] == (S[14] ^ S[15])) &
          (S[ 7] == (S[12] ^ S[15])) &
          (S[ 8] == (S[12] ^ S[13])) &
          (S[ 9] ==  S[14]) &
          (S[10] ==  S[15]) &
```

```
         (S[11] ==  S[12]) ;

  E[21] = !S[4] & !S[5] & !S[6] & !S[7] &
         (S[ 8] ==  S[ 1]) &
         (S[ 9] == (S[ 1] ^ S[ 2])) &
         (S[10] == (S[ 2] ^ S[ 3])) &
         (S[11] == (S[ 0] ^ S[ 3])) &
         (S[12] == (S[ 0] ^ S[ 1])) &
         (S[13] ==  S[ 2]) &
         (S[14] ==  S[ 3]) &
         (S[15] ==  S[ 0]) ;

  E[22] = !S[8] & !S[9] & !S[10] & !S[11] &
         (S[ 0] == (S[ 4] ^ S[ 5])) &
         (S[ 1] ==  S[ 6]) &
         (S[ 2] ==  S[ 7]) &
         (S[ 3] ==  S[ 4]) &
         (S[12] ==  S[ 5]) &
         (S[13] == (S[ 5] ^ S[ 6])) &
         (S[14] == (S[ 6] ^ S[ 7])) &
         (S[15] == (S[ 4] ^ S[ 7])) ;

  E[23] = !S[12] & !S[13] & !S[14] & !S[15] &
         (S[ 0] ==  S[ 9]) &
         (S[ 1] == (S[ 9] ^ S[10])) &
         (S[ 2] == (S[10] ^ S[11])) &
         (S[ 3] == (S[ 8] ^ S[11])) &
         (S[ 4] == (S[ 8] ^ S[ 9])) &
         (S[ 5] ==  S[10]) &
         (S[ 6] ==  S[11]) &
         (S[ 7] ==  S[ 8]) ;

  E[24] = !S[0] & !S[1] & !S[2] & !S[3] &
         (S[ 8] == (S[ 6] ^ S[ 7])) &
         (S[ 9] == (S[ 4] ^ S[ 6])) &
         (S[10] == (S[ 4] ^ S[ 5] ^ S[ 7])) &
         (S[11] == (S[ 5] ^ S[ 6])) &
         (S[12] ==  S[ 4]) &
         (S[13] ==  S[ 5]) &
         (S[14] ==  S[ 6]) &
         (S[15] ==  S[ 7]) ;

  E[25] = !S[4] & !S[5] & !S[6] & !S[7] &
         (S[ 8] ==  S[ 0]) &
         (S[ 9] ==  S[ 1]) &
         (S[10] ==  S[ 2]) &
         (S[11] ==  S[ 3]) &
         (S[12] == (S[ 2] ^ S[ 3])) &
         (S[13] == (S[ 0] ^ S[ 2])) &
         (S[14] == (S[ 0] ^ S[ 1] ^ S[ 3])) &
```

```
          (S[15] == (S[ 1] ^ S[ 2])) ;

  E[26] = !S[8] & !S[9] & !S[10] & !S[11] &
          (S[ 0] == (S[ 6] ^ S[ 7])) &
          (S[ 1] == (S[ 4] ^ S[ 6])) &
          (S[ 2] == (S[ 4] ^ S[ 5] ^ S[ 7])) &
          (S[ 3] == (S[ 5] ^ S[ 6])) &
          (S[12] ==  S[ 4]) &
          (S[13] ==  S[ 5]) &
          (S[14] ==  S[ 6]) &
          (S[15] ==  S[ 7]) ;

  E[27] = !S[12] & !S[13] & !S[14] & !S[15] &
          (S[ 4] == (S[ 2] ^ S[ 3])) &
          (S[ 5] == (S[ 0] ^ S[ 2])) &
          (S[ 6] == (S[ 0] ^ S[ 1] ^ S[ 3])) &
          (S[ 7] == (S[ 1] ^ S[ 2])) &
          (S[ 8] ==  S[ 0]) &
          (S[ 9] ==  S[ 1]) &
          (S[10] ==  S[ 2]) &
          (S[11] ==  S[ 3]) ;

  E[28] = (S[ 4] ==  S[ 0]) &
          (S[ 5] ==  S[ 1]) &
          (S[ 6] ==  S[ 2]) &
          (S[ 7] ==  S[ 3]) &
          (S[ 8] ==  S[ 0]) &
          (S[ 9] ==  S[ 1]) &
          (S[10] ==  S[ 2]) &
          (S[11] ==  S[ 3]) &
          (S[12] ==  S[ 3]) &
          (S[13] == (S[ 0] ^ S[ 3])) &
          (S[14] ==  S[ 1]) &
          (S[15] ==  S[ 2]) ;

  E[29] = (S[ 0] ==  S[ 7]) &
          (S[ 1] == (S[ 4] ^ S[ 7])) &
          (S[ 2] ==  S[ 5]) &
          (S[ 3] ==  S[ 6]) &
          (S[ 8] ==  S[ 4]) &
          (S[ 9] ==  S[ 5]) &
          (S[10] ==  S[ 6]) &
          (S[11] ==  S[ 7]) &
          (S[12] ==  S[ 4]) &
          (S[13] ==  S[ 5]) &
          (S[14] ==  S[ 6]) &
          (S[15] ==  S[ 7]) ;

  E[30] = (S[ 4] ==  S[ 3]) &
          (S[ 5] == (S[ 0] ^ S[ 3])) &
          (S[ 6] ==  S[ 1]) &
          (S[ 7] ==  S[ 2]) &
          (S[ 8] ==  S[ 0]) &
          (S[ 9] ==  S[ 1]) &
          (S[10] ==  S[ 2]) &
          (S[11] ==  S[ 3]) &
          (S[12] ==  S[ 0]) &
          (S[13] ==  S[ 1]) &
          (S[14] ==  S[ 2]) &
          (S[15] ==  S[ 3]) ;

E[31] = (S[ 4] ==  S[ 0]) &
          (S[ 5] ==  S[ 1]) &
          (S[ 6] ==  S[ 2]) &
          (S[ 7] ==  S[ 3]) &
          (S[ 8] ==  S[ 3]) &
          (S[ 9] == (S[ 0] ^ S[ 3])) &
          (S[10] ==  S[ 1]) &
          (S[11] ==  S[ 2]) &
          (S[12] ==  S[ 0]) &
          (S[13] ==  S[ 1]) &
          (S[14] ==  S[ 2]) &
          (S[15] ==  S[ 3]) ;

E[32] = !S[ 4] & !S[ 5] & !S[ 6] & !S[ 7] &
          !S[ 8] & !S[ 9] & !S[10] & !S[11] &
          !S[12] & !S[13] & !S[14] & !S[15] ;

E[33] = !S[ 0] & !S[ 1] & !S[ 2] & !S[ 3] &
          !S[ 8] & !S[ 9] & !S[10] & !S[11] &
          !S[12] & !S[13] & !S[14] & !S[15] ;

E[34] = !S[ 0] & !S[ 1] & !S[ 2] & !S[ 3] &
          !S[ 4] & !S[ 5] & !S[ 6] & !S[ 7] &
          !S[12] & !S[13] & !S[14] & !S[15] ;

E[35] = !S[ 0] & !S[ 1] & !S[ 2] & !S[ 3] &
          !S[ 4] & !S[ 5] & !S[ 6] & !S[ 7] &
          !S[ 8] & !S[ 9] & !S[10] & !S[11] ;
```

7.19.5.2 Bits in Error

For each nibble in error, a group of 4 syndrome bits indicates which bit(s) are in error for that nibble. These groups are shaded in the H-Matrix. For example, if E[3] == 1 (nibble 3 is in error),

- if S[0] == 1 then bit 12 is in error
- if S[1] == 1 then bit 13 is in error
- if S[2] == 1 then bit 14 is in error
- if S[3] == 1 then bit 15 is in error

Since a bit can have only 2 values, 0 and 1, a bit in error is corrected by “flipping” its value (changing a 0 to a 1 or changing a 1 to a 0).

For diagnostics (software decode of the syndrome) the full 144 bit field is decoded to determine which bits are in error. For reads performed by the hardware, only data bits 0:127 are corrected. If check bits are in error there is no correction.

- For E[0]: S[4], S[5], S[6], S[7] indicate bits [0], [1], [2], [3] are in error
- For E[1]: S[8], S[9], S[10], S[11] indicate bits [4], [5], [6], [7] are in error
- For E[2]: S[12], S[13], S[14], S[15] indicate bits [8], [9], [10], [11] are in error
- For E[3]: S[0], S[1], S[2], S[3] indicate bits [12], [13], [14], [15] are in error
- For E[4]: S[4], S[5], S[6], S[7] indicate bits [16], [17], [18], [19] are in error
- For E[5]: S[8], S[9], S[10], S[11] indicate bits [20], [21], [22], [23] are in error
- For E[6]: S[12], S[13], S[14], S[15] indicate bits [24], [25], [26], [27] are in error
- For E[7]: S[0], S[1], S[2], S[3] indicate bits [28], [29], [30], [31] are in error
- For E[8]: S[4], S[5], S[6], S[7] indicate bits [32], [33], [34], [35] are in error
- For E[9]: S[8], S[9], S[10], S[11] indicate bits [36], [37], [38], [39] are in error
- For E[10]: S[12], S[13], S[14], S[15] indicate bits [40], [41], [42], [43] are in error
- For E[11]: S[4], S[5], S[6], S[7] indicate bits [44], [45], [46], [47] are in error
- For E[12]: S[8], S[9], S[10], S[11] indicate bits [48], [49], [50], [51] are in error
- For E[13]: S[12], S[13], S[14], S[15] indicate bits [52], [53], [54], [55] are in error
- For E[14]: S[4], S[5], S[6], S[7] indicate bits [56], [57], [58], [59] are in error
- For E[15]: S[8], S[9], S[10], S[11] indicate bits [60], [61], [62], [63] are in error
- For E[16]: S[12], S[13], S[14], S[15] indicate bits [64], [65], [66], [67] are in error
- For E[17]: S[0], S[1], S[2], S[3] indicate bits [68], [69], [70], [71] are in error
- For E[18]: S[4], S[5], S[6], S[7] indicate bits [72], [73], [74], [75] are in error
- For E[19]: S[8], S[9], S[10], S[11] indicate bits [76], [77], [78], [79] are in error
- For E[20]: S[12], S[13], S[14], S[15] indicate bits [80], [81], [82], [83] are in error
- For E[21]: S[0], S[1], S[2], S[3] indicate bits [84], [85], [86], [87] are in error
- For E[22]: S[4], S[5], S[6], S[7] indicate bits [88], [89], [90], [91] are in error
- For E[23]: S[8], S[9], S[10], S[11] indicate bits [92], [93], [94], [95] are in error
- For E[24]: S[12], S[13], S[14], S[15] indicate bits [96], [97], [98], [99] are in error
- For E[25]: S[0], S[1], S[2], S[3] indicate bits [100], [101], [102], [103] are in error
- For E[26]: S[4], S[5], S[6], S[7] indicate bits [104], [105], [106], [107] are in error
- For E[27]: S[8], S[9], S[10], S[11] indicate bits [108], [109], [110], [11] are in error
- For E[28]: S[0], S[1], S[2], S[3] indicate bits [112], [113], [114], [115] are in error
- For E[29]: S[4], S[5], S[6], S[7] indicate bits [116], [117], [118], [119] are in error
- For E[30]: S[8], S[9], S[10], S[11] indicate bits [120], [121], [122], [123] are in error
- For E[31]: S[12], S[13], S[14], S[15] indicate bits [124], [125], [126], [127] are in error
- For E[32]: S[0], S[1], S[2], S[3] indicate bits [128], [129], [130], [131] are in error
- For E[33]: S[4], S[5], S[6], S[7] indicate bits [132], [133], [134], [135] are in error
- For E[34]: S[8], S[9], S[10], S[11] indicate bits [136], [137], [138], [139] are in error
- For E[35]: S[12], S[13], S[14], S[15] indicate bits [140], [141], [142], [143] are in error

7.19.5.3 Single Bit Errors

A classic SEC/DED (Single Error Correct/Double Error Detect) Hamming Matrix has the property that for single bit errors, each column of the matrix gives the syndrome for that column being in error. The SSC/DSD H-Matrix used by the CPC945 also has that attribute. For example, if there is a single bit error and the bit in error is 00, then the syndrome will be 0b0000100001001001 = 0x0849.

7.19.5.4 Address Parity and Special Uncorrectable Errors

The Address Parity and Special Uncorrectable Errors are both uncorrectable.

The syndrome for an Address Parity error matches the AP column in the H-Matrix similar to that of single bit errors, as described above in *Section 7.19.5.3* . If there is an Address Parity Error, and no other error, the syndrome will be 0b1000100000101001 = 0x8829. An Address Parity Error can occur if, for example, a write intended to address A is instead written to address B, followed by a read intended for address B that is indeed read from address B.

If there is a Special Uncorrectable Error, and no other error, the syndrome will be 0xBBE7. This is the result of reading in all 0’s for data, calculating inverted check bits = 0xFFFF, and XORing that with the 0x4418 check bit value read in (AP = 0). A Special Uncorrectable Error will occur for a read of an address that had a previous uncorrectable error detected during a Read-Modify-Write cycle to that address.

If Address Parity is enabled, and AP =1, the internally (inverted) check bits calculated for all 0 data will be 0x77D6. The check bits read in 0xCC31. The syndrome will be 0x77D6 XOR 0xCC31 = 0xBBE7. Again, this is in the absence of other errors, that is, on the previous RMW the controller was able to write all 0’s back as data and 0x4418 or 0xCC31 as check bits, depending on AP, and that these same values are present when read back.

Summary: A syndrome of 0x8829 indicates an addressing error occurred. A syndrome of 0xBBE7 indicates a previous read had an uncorrectable error. These errors will create other syndromes whose values cannot be predicted, if there are additional errors.

7.19.5.5 Syndrome Decode Summary

For the purposes of determining and latching error status, the CPC945 hardware uses the following algorithm:

- If the generated syndrome = 0x00, then there is no error and no correction.
- Else if any of the nibble-in-error decodes = 1 (*Section 7.19.5.1 Nibble in Error*), then there is a Correctable Error (CE), and the appropriate bits are corrected using the bit-in-error decode (*Section 7.19.5.2 Bits in Error*).
- Else there is an uncorrectable error (UE) and there is no correction.

7.19.6 Error Logging

Errors are detected and logged on memory reads. These reads can be triggered by system read requests, the read portion of a read-modify-write cycle triggered by system partial write requests, or reads generated by scrub requests.

Errors are either uncorrectable or correctable.

Counts of each type of error are maintained in the MEAR1 as the UECnt and CECnt fields. Any read that has a detected error will increment the appropriate counter up to a value of 255 = 0xFF. Once a counter reaches a value of 255 it stays at that value. A read of the MEAR1 resets both counter values to 0.

In addition to counting the errors, additional information is latched and “frozen” upon the first detection of an error. This information is latched in MEAR0, MEAR1 and MESR simultaneously (as a logical set):

- The syndrome is latched in bits 16:31 of MESR.

- The address is latched in MEAR0. The address that is latched is the internally decoded rank, bank, row and column. (Software is required to reverse decode these values to a system address based on the settings of the DmCnfg[0:3] and UsrCnfg registers, and the address mappings given in *Table 7-23*, *Table 7-24*, *Table 7-25* and *Table 7-26*.)
- The burst count is latched in MEAR1. This value indicates which quadword contained the error (in a burst of 4 for 128-bit configuration and a burst of 8 for the 64-bit configurations).
- The type of error is recorded in the MESR. If the error was correctable the ECC_CE bit is set. If the error was uncorrectable the ECC_UE bit is set. In addition, if the error was uncorrectable and it was detected in the read portion of a read-modify-write cycle, the ECC_UEWT bit is set.

The ECC_UE, ECC_CE and ECC_UEWT bits are reset to 0 when the MESR is read.

Following a read of the MESR register (or a chip reset), the syndrome, address and burst count associated with an error are enabled to be captured. In the general case:

- Following a capture enable, if an error is detected the syndrome, address and burst count fields are updated and will not change until
  – The capture mechanism is re-enabled by reading the MESR (or there is a chip reset).
  – Following the re-enable, another error is detected.

However, uncorrectable error status has a higher priority than correctable error status. This means that, if the error information for a correctable error is captured, and some time later before the MESR is read an uncorrectable error is detected, then the information associated with the correctable error is discarded and replaced with the error information for the uncorrectable error.

The ECC_UE, ECC_CE and ECC_UEWT bits are independent of this capture mechanism. Once set, these bits stay set until the MESR are read. It is possible that both the ECC_UE and ECC_CE bits are “1” when the MESR is read. This means that since the last time the MESR was read, at least one UE and at least one CE error was detected. Because of the priority mechanism the syndrome, address and burst count will be associated with the first UE that was detected.

7.19.7 Error Reporting

Masked copies of the MESR ECC_UE and ECC_CE bits are forwarded to the PI Exception Register (0xF80300A0) in the Processor Interface Unit to become bits 9 and 10, EccUEExcp and EccCEExcp.

The ECC_UE and ECC_CE bits are forwarded to the PI Exception Register by default. They can be masked off using the ECC_UE_MASK and ECC_CE_MASK bits in the Memory Check Control Register (MCCR).

As described in *“API Exception Register (APIExcp)” on page 423*, the exception generated by the UE or CE can be used to activate the PI_CSTP pin (under control of APIMASK0 register or the CHP_FAULT_N pin (under control of the APIMASK1 register).

Bits in the PI Exception register are cleared when read. However, if the ECC_UE or ECC_CE bits in the MESR are still ‘1’ when the PI Exception register is read, the corresponding EccUEExcp and EccCEExcp bits will immediately be set to ‘1’ again, causing a new exception. When an exception in the PI unit is generated as the result of the ECC_UE or ECC_CE bits in the memory controller MESR, those signals should first be cleared (by reading the MESR or masking off the signals using ECC_UE_MASK and ECC_CE_MASK in the MCCR) before attempting to clear the exception caused by the PI Exception register.

Note that the ECC_UE and ECC_CE bits are writable, so diagnostic software can mimic the detection of an ECC error by setting either of these bits with a write to the MESR.

7.19.8 Error Injection

If the EI_EN bit in the MCCR is set, then the 16 EI_PAT (Error Injection Pattern) bits in the MESR will be used for check bits on write operations (and the write portion of a read-modify-write), instead of the normally calculated check bits.

This can be useful for diagnostic software. For example, software could calculate check bits for a given data quadword and use that value as the EI_PAT bits. The given data pattern, with selected bit(s) inverted, could be written to memory with EI_EN set. When read back, the ECC logic should detect an error, and when software decodes the captured syndrome it should calculate the same bits in error as were inverted in the write.

Note that the memory controller does writes in bursts of 4 or 8 quadwords. The same EI_PAT will be used for each quadword in the burst.This is the reason that Address Parity, when enabled, is only calculated on a cache line address basis. That is, the AP bit injected into the check bit calculation will be the same value for all quadwords of a burst of 8. This simplifies the diagnostic routine for when AP is enabled.

7.19.9 Byte Lane Substitution

If the ByteLaneECCSub bit in the MCCR is set, a special “calibration” ECC mode is enabled. In this mode:

- For writes, the least significant bit (LSB) of each of the 16 data bytes is replaced with the 16 check bits.
- For reads, the LSBs of each of the 16 data bytes is steered onto the 16 check bits internal to the DDR2 PHY.

The purpose of this calibration mode is to allow the DDR2 PHY Calibration logic to be used for tuning the check bit paths of systems that can use ECC, but that have nonECC DIMMs installed at the time calibration is run.

7.20 Scrub

7.20.1 General

DRAMs have the characteristic that their memory cells generally hold random values following power on. It is often desirable to write valid data patterns to all memory locations before using the memory. This is particularly true for systems with ECC. Following power on, the check bits will have garbage values, and reading uninitialized memory with ECC enabled will generate ECC errors. For the CPC945 ECC errors can be generated even if the first operations to memory are writes. These will occur if the write requests are for size less than 64 bytes, since this will trigger read-modify-write operations and the reads will have ECC errors.

DRAMs typically also have the characteristic that when operated for long periods of times, cells at random can be flipped or have other problems. Many of these failures are “soft,” that is, a bit might be flipped, but it continues to be writable as usual and if rewritten it will tend to retain its new value as usual. (There are a variety of causes, such as cosmic radiation, electrical noise, and so on.) Systems with ECC can “scrub out” these types of failures using an operation in which all memory locations are randomly read and written back with correctable errors corrected.

Although software can be used for both DRAM initialization and error scrubbing, the CPC945 memory controller has logic to perform both mechanisms automatically in hardware. This logic uses the Memory Scrub Control Register (MSCR), Memory Scrub Range Start Register (MSRSR), Memory Scrub Range End Register (MSRER), and the Memory Scrub Pattern Register (MSPR).

The SCRUB MOD (Scrub Mode) bits in the MSCR enable one of 3 modes of operation: Immediate With Fill, Background, and Immediate. When enabled, the Immediate With Fill mode generates Write requests. The Background and Immediate modes generate Read-Modify-Write requests.

7.20.2 Scrub Arbitration

As shown in *Figure 7-3* on page 142, requests from the Scrub logic are arbitrated with the usual write and read requests from the CPU and I/Os. The scrub arbitration weight is set in the Memory Arbiter Weight Register (0xF8002280). Arbitration for scrubs is slightly different than the other two requestors: if the scrub arbitration weight is set to 0 it will not participate at all in the round-robin arbitration. For this case a scrub can only be accepted by the arbiter if neither of the other two requestors has an active request. If the scrub arbitration weight is not 0 then it participates in the round arbitration the same as the other two requestors.

As described below, the scrub logic will make a series of scrub requests with incrementing addresses. If a given scrub request is not serviced by the arbiter before the next request is issued, the first request will be discarded.

7.20.3 Scrub Addresses

Requests accepted by the arbiter trigger the usual four quadword burst, that is, scrubs work on 64 bytes at a time.

A scrub sequence is triggered by setting the SCUB MOD bits to a non-zero value. A series of scrub requests will be issued, with incrementing addresses. The first address comes from the Memory Scrub Range Start Register, subsequent requests have incrementing addresses on 64 byte boundaries, and the final address is determined from the Memory Scrub Range End Register.

Since scrubs operate on 64-byte boundaries only 30 bits of the 36-bit system address are used to specify scrub addresses in the MSRSR and MSRER. The 30 bits are offset by 2 bits from the right, therefore an additional shift of 4 bits is required to convert the register value to the actual value used. For example, a register value of 0x000000EC in the MSRSR specifies that the actual scrub start address is 0x0000EC0.

The value of the MSRER specifies the first address of the last 64-byte scrub. For example, if MSRER = 0x000000EC then the four quadword addresses of the scrub will be 0x0000EC0, 0x0000ED0, 0x0000EE0 and 0x0000EF0. The last byte address is 0x0000EFF.

If the MSRER is equal to the MSRSR, the scrub range is a single 64-byte operation. If the MSRER is less than the MSRSR, no scrub will be requested.

A scrub address range that partially or fully overlaps the 2 GB “I/O Hole” will produce scrub requests with addresses within that 2 GB I/O space. For these requests, as soon as the request is accepted by the arbiter the memory controller will generate an internal signal indicating that the scrub operation has been performed, but no memory operation takes place. For background scrubs, the first scrub request to I/O space will generate an internal sequence in which the scrub address is incremented every mem_clk until the 4 GB boundary (MSRER > 4 GB) is reached. ***The MSRER must not be set to an address within the I/O 2 GB space. The memory controller can hang for this condition.***

If the scrub goes beyond the end of installed memory, an address map exception will be taken (*Section 7.14.13 Address Mapping Exceptions* on page 170). As with other out-of-range addresses, scrub writes will be suppressed. For the immediate and background modes with do Read-Modify-Writes, the reads will be performed, but the addresses of these reads cannot be predicted.

7.20.4 Immediate with Fill Mode

When the SCRUB_MOD bits are set to “11” a scrub sequence is initiated, using the scrub start and end addresses as described above. Each sequential scrub request is issued immediately following the completion of the prior scrub request. Each request is for a Write of 64-bytes. The write data will be the 4-byte contents of the Memory Scrub Pattern Register, replicated modulo-4 bytes for a total of 64 bytes.

Immediate with Fill Mode can be useful for both ECC and nonECC systems. For ECC systems, it is a useful way to rapidly fill memory with valid check bits in addition to the data bits. Because the data size is 64 bytes, there are no read-modify-writes; just writes.

After the last scrub operation of a single pass through memory has been performed the SCRUB_MOD bits are reset to ‘00’ and no more scrubs are requested.

For a scrub address range that spans the I/O hole, a burst of writes will be seen for the first 2 GB of memory, then a pause in which the scrub logic increments through 2 GB of I/O addresses, then a final burst of writes will be seen starting at 4 GB and ending at the scrub end address.

Due to the long burst of back-to-back requests generated by this scrub mode, it is recommended that no other accesses to DDR2 memory be attempted while Immediate with Fill Mode (or Immediate Mode) is in progress.

7.20.5 Background Mode

Background mode is intended for ECC systems; it performs the periodic scan of all memory locations, correcting soft errors as it goes.

When the SCRUB_MOD bits are set to “01” a scrub sequence is started. Requests start with the scrub start address and continue indefinitely, until the SCRUB_MOD bits are changed to something other than “01.” The scrub address increments, and each time the scrub end address is reached the scrub requests start over using the scrub start address.

Scrub requests occur at integral multiples of the refresh time. Intervals range from one request every refresh to one request every 256 refreshes. The interval multiple is taken from the SI (Scrub Interval) bits in the MSCR.

The scrub requests are Read-Modify-Writes (implemented internally as partial write requests with 0 bytes of write data). If ECC is enabled, as described in *Section 7.19.4* if there is a correctable error on the read, the write data written back will have the error fixed, and if there is an uncorrectable error the data is written back as is, but with new check bits that incorporate the SPUE bit.

Note that from an ECC viewpoint there is no difference between a scrub-initiated memory operation versus a CPU or I/O initiated operation; if exceptions are enabled for ECC errors, for example, then an ECC error detected during a scrub operation will cause an exception.

In the absence of other operations, when background scrub is enabled then periodic Read-Modify-Write operations will be seen. If the address range spans the I/O hole then pauses in this sequence will be seen as scrub logic increments scrub requests through the I/O address range.

Although this hardware-initiated scrubbing does not use much bandwidth on the average, for systems with heavy bursts of memory accesses disruptions from scrub requests happening at “the wrong time” might be undesirable. This is the reason that the arbiter does not include scrub requests in the round-robin arbitration for a scrub weight of 0. When the scrub weight is set to 0, very heavy usage of the memory controller by the CPUs or I/O can force scrub requests to be ignored for a very long time. There is a trade-off: as described above in *Section 7.20.2 Scrub Arbitration* if a given request is ignored long enough for a new scrub request to be generated the long-delayed request is discarded. Thus, the system has maximum access to memory, but scrubs of random addresses might be skipped from time to time. If this is not desired, the scrub arbiter weight can be set to non-zero. Now scrub requests should be honored such that none are discarded, at the potential expense of full access to memory (lowest latency) for the CPUs and I/O.

(This behavior of the arbiter for a scrub weight of 0 is another reason for recommending that no memory access be attempted while the Immediate Modes are executing.)

7.20.6 Immediate Mode

Immediate mode is a kind of hybrid of Immediate With Fill mode and Background mode. A single pass of memory is executed like Immediate with Fill mode, but the requests are Read-Modify-Writes instead of Writes. Thus, the entire range of memory from scrub start through scrub end is scrubbed of soft errors, but just once.

An Immediate mode scrub is initiated when the SCRUB_MOD bits are set to “10.” Like Immediate with Fill mode, the SCRUB_MOD bits are reset to “00” following the last operation of the sequence.

7.21 External Connections Overview

The memory controller interfaces to the external memories with the following types of signals:

- The memory clock CK. The clock is differential (CK_P, CK_N). This clock operates at mem_clk speed (for example, 200 MHz, 266 MHz).
- The clocked uni-directional control signals (Chip Select, Clock Enable, On Die Termination (optional)), command signals (RAS, CAS, WE), and address signal (Bank Address, Memory Address). The CK signal clocks this interface using the positive edge of CK (CK_P).
- The bi-directional data bus (DQ).
- The bi-directional data strobes (DQS). The DQS signals clock the data on the DQ bus. These strobes operate at mem_clk speed (for example, 200 MHz, 266 MHz) but data is clocked by both edges of DQS. Therefore the DQ interface operates at Double Data Rate speeds (400 MHz, 533 MHz).

  JEDEC specifies both differential and single-ended operation of the DQS. The CPC945 requires that systems using ×8 and ×16 chips use differential mode, and systems using ×4 chips use single-ended mode.
- Multiplexer controls for optional external FET-switch multiplexers on the DQ and DQS buses.

The I/O operate to the JEDEC specifications for a 1.8V Stub Series Terminated DDR2 interface (SSTL_18). Both the external memories and the CPC945 I/O circuits have “VREF” voltage reference inputs that are nominally set to 0.9V.

The electrical characteristics of the CPC945 DDR2 I/O are controlled by the IOPadCntl register (0xF80029A0). The bits in this register control the drive strength of the uni-directional drivers, the drive strength of the driver portion of the bi-directional driver/receivers, the termination values of the receiver portions of the bi-directional driver/receivers, and the operational mode of the receiver terminations (for example, termination enabled on reads only, or both writes and reads). A bit in this register also controls if the DQS driver/receivers are configured for differential or single-ended operation.

The DDR2 PHY is an intermediate layer of logic between the memory controller internal logic and the DDR2 external I/O.

The PHY has a control signal unit which in general passes the clocked control signal interface (CS, CKE, ODT, RAS, CAS, WE, BA, MAD) to the external I/O with no timing adjustment other than delays which operate on clock boundaries (for example, CS can be delayed by 0 to 3 clocks, CKE can be delayed by 0 to 3 clocks, and so on). The controls for these delays are in the MemPHYModeCntl register (0xF8002880).

Other registers which affect the operation of these signals are MemModeCntl (0xF8002500), ODTCntl (0xF80023A0), and the Ad2Cyc bit in MemBusConfig (0xF80022D0).

The PHY generates two copies of the differential CK: CK_AP/CK_AN, CK_BP/CK_BN. Each of the 2 clocks has its own vernier adjustment, controlled by the CK Control Registers (0xF8002890, 0xF80028A0). The clocks are enabled with the CK_on bit in MemModeCntl (0xF8002500).

The PHY has a data unit which drives DQ and DQS on writes and receives DQ and DQS on reads.

The internal control logic sends signals to the PHY data unit indicating that a write or read operation is to be performed. The MemBusConfig and MemBusConfig2 registers (0xF80022D0 and 0xF80022E0) provide coarse timing control (in terms of ddr_clk cycles) of when these signals are sent to the PHY data unit.

Fine control is provided by verniers controlled by the Write Strobe Control registers, Read Data Strobe Control registers, ResetLdEn Offset Delay registers, and, if optional external data multiplexers are used, the External Data Multiplexer Delay Registers.

The DQS signals must be delayed 1/2 bit time relative to DQ both for writes and reads. The PHY data unit has a “1/2 bit time averager” unit which generates the control value for this delay. The output of this unit is fed into the verniers produce the 1/2 bit time delay, and which further fine tune the DQS outgoing and incoming delays.

Finally, the PHY data unit has a “calibration” unit attached to the read data path. This unit essentially measures the timings of internal signals used to clock the read data. It can be used to assist in determining the nominal settings of the various timing controls involved in the capture of read data.

7.21.1 Bus Configurations

The CPC945 supports 1 through 8 ranks of memory, labeled ranks 0 through 7. Installed memory starts at rank 0 and grows upward through rank 7.

As discussed in *Section 7.3 Memory Configurations*, three basic configurations are supported with regard to data bus width and data transfer size. The system designer must wire up the DDR2 memories on the board to implement one of these 3 configurations, and the 64BitBus and 64BitCfg bits (bits 31 and 30) in the MemBusConfig register (0xF80022D0) must be set to match the implemented configuration. These configurations are summarized in *Table 7-36*.

Table 7-36. MemBusConfig Width Settings.

Bit 31 = 64BitBus	Bit 30 = 64BitCfg	Configuration	ECC
0	0	128/144-bit bus 128/144-bit transfers	Optional
1	1	64-bit bus 64-bit transfers	Not supported
0	1	128-bit bus 64-bit transfers	Not supported
1	0	Illegal	

7.21.2 External Data Multiplexers

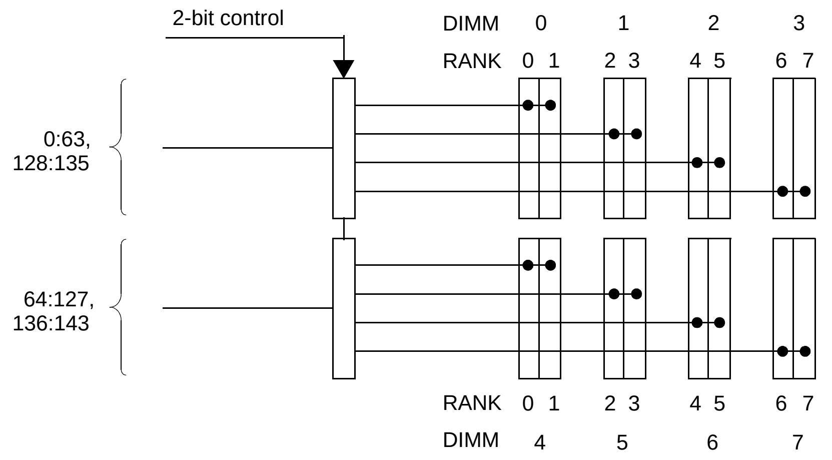
The CPC945 memory controller has provisions for controlling external multiplexers on the DQ and DQS signals. These switches connect the DQ and DQS signals on the CPC945 with one-of-four sets of DQ and DQS signals on the DIMM side. The switches are made of FET pass gate devices, such that the CPC945 only sees one DIMM load at a time. A 2-bit encoded value selects one of the four sets of signals to be passed through.

To distribute loading among a set of multiplexers, the CPC945 produces 4 identical sets of 2-bit multiplexer controls. DDR_MUXEN[0:1] = DDR_MUXEN[2:3] = DDR_MUXEN[4:5] = DDR_MUXEN[6:7].

Note: The CPC945 uses big-endian notation, whereas the multiplexer vendor notation is typically little-endian. For example, DDR_MUXEN[0:1] should be wired to vendor multiplexer pins [1:0].

The use of these external multiplexers depends on loading, board design, and timing tolerances. For example, a system with a maximum of 4 ranks running at 400 MHz might not need the data multiplexers, whereas a system with all 8 ranks populated running at 533 MHz might require the multiplexers.

Figure 7-6. External Multiplexers



7.21.3 Unused I/O

The DDR_ARB_ADDR and DDR_STOP signals are outputs which have no function. They must be left unconnected.

Systems which do not implement ECC should leave DQ[128:143] and DQS[16:17] unconnected.

As noted in *Section 7.19.1 ECC Introduction*, when ECC is disabled DQ[128:143] and DQS[16:17] are tri-stated (DQS[32:35] in single-ended mode).

7.22 DDR2 PHY

7.22.1 Byte Lanes

The CPC945 uses a common block called a “byte lane” which contains the data path registers and timing verniers for one byte of data (8 bits) and the DQS signal(s) associated with that byte. This unit is replicated 18 times to support 18 bytes of data.

When using ×8 or ×16 memory chips there is one DQS per byte lane. When using ×4 memory chips there are 2 DQS per byte lane. See *Section 7.23.1 I/O Pad Bit Settings*, *Figure 7-13 Byte Lane Write Verniers* and *Figure 7-16 Byte Lane Read Control* for more information.

The physical placement and wiring of the byte lanes within the chip is such as to minimize the skew among the 8 data bits and the DQS strobe(s). Since DQS, which is the data clock, has timing verniers to center the DQS edges within the bit-time window, this allows for reliable clocking of a given byte. Since the byte lanes each have their own DQS this permits some skew to be present on a byte to byte basis. (Or a nibble to nibble basis, for ×4 chips.)

7.22.2 Clusters

For chip placement reasons the byte lanes are organized at a higher level into four “clusters.” Cluster 0 has bytes lanes 0, 1, 2 and 3. Cluster 1 has byte lanes 4, 5, 6, 7 and 16. Cluster 2 has byte lanes 8, 9, 10 and 11. Cluster 3 has byte lanes 12, 13, 14, 15 and 17.

Cluster 0 also has the logic and verniers for the 2 clocks to memory (CK_A, CK_B), the logic for the command signals to memory (RAS, CAS, WE, BA, MAD), the logic and verniers for the external data multiplexer controls (MUXEN), and bit time calibration unit (1/2 Bit Time Averager).

Each cluster has its own set of registers for controlling its verniers. This explains why the register addressing for the verniers is clustered (for example, the Write Strobe Control Registers use 0xF80028xx for bytes 0, 1, 2, and 3; 0xF80029xx for bytes 4, 5, 6, 7, and 16; 0xF8002Axx for bytes 8, 9, 10, and 11; and 0xF8002Bxx for bytes 12, 13, 14, 15, and 17).

The byte-to-byte skew is more tightly controlled for bytes within a cluster than for bytes in different clusters.

7.22.3 Relationship to Board Wiring

It is common at the board level for some bit or byte swapping to be used to facilitate wiring and maintain small differences in wiring lengths (to maintain low skew).

The association of a DQS with its 8-bits of data (or 4 bits of data for ×4 chips) must be maintained by the board wiring to a given memory chip. For example, DQSP/N[0] and DQ[0:7] must be wired to the same memory chip, while DQSP/.N[1] and DQ[8:15] must be wired in common to a different memory chip, and so on.

It is permissible to swap chip to chip.

For example, a DIMM will have bits [0:7], [8:15], [16:23], [24:31], [32:39], [40:47], [48:55], and [56:63] wired to memory chips 0, 1, 2, 3, 4, 5, 6, and 7, respectively. A straight-forward wiring would have CPC945 bits [0:7] wired to DIMM bits [0:7] = chip 0, CPC945 bits [8:15] wired to DIMM bits [8:15] = chip 1, and so on. For this situation CPC945 DQSP/N[0] must be wired to DIMM DQSP/N[0] = chip 0 and CPC945 DQSP/N[1] must be wired to DIMM DQSP/N[1] = chip 1.

For this DIMM it is permissible to swap on a byte basis, for example, CPC945 bits [0:7] could go to DIMM bits [8:15] = chip 1, while CPC945 bits [8:15] could go to DIMM bits [0:7] = chip 0. The DQS must follow the data: CPC945 DQSP/N[0] must be wired to DIMM DQSP/N[1] = chip 1and CPC945 DQSP/N[1] must be wired to DIMM DQSP/N[0] = chip 0. (Note that for ×8 and ×16 chips the DQS is actually a differential pair.)

The same type of swapping on a chip basis applies to ×4 chips: 4 bits of data plus 1 DQS going to one chip can instead go to another chip. (For ×4 chips the CPC945 does not support differential operation, so there is only one wire per DQS. See *Table 7-37 DQS Bit Numbering* on page 220.)

For systems with ×16 chips, with regard to DQ/DQS swapping the ×16 memory chip should be treated as 2 ×8 chips.

As noted above in *Section 7.22.2* the tracking among bytes is best for those bytes whose byte lanes are in the same cluster. Therefore, while it is acceptable to swap, say, bytes 2 and 3 (because they belong to the same cluster), it is slightly less desirable to swap bytes 3 and 4 (because they belong to different clusters).

For systems with the 128-bit bus width, DQ and DQS for the DIMM with bits 0:63 should not be swapped with the DIMM for bits 64:127.

For the connections to a given memory chip, the bits to that chip can be swapped with each other. For example, CPC945 data bits can be wired to a ×8 chip as bits 0:7, respectively, or to bits 0, 1, 3, 2, 7, 6, 4, 5 respectively, or to bits 7, 6, 5, 4, 3, 2, 1, 0 respectively, and so on.

As described in *Section 7.31.1 Calibration Logic Overview*, the DDR2 PHY has logic which can be used to assist in discovering timing control values for read operations. This logic operates on a per byte basis. For systems with ×4 chips, the calibration logic only operates on even-numbered nibbles. Therefore it is recommended that for systems with ×4 chips, nibble pairs should have identical wiring lengths, so that when the calibration logic is used to find good timings for the even nibble these timings will be identically appropriate for the odd nibble.

7.22.4 Bus Driving

Following reset, the default is for the PHY to enable the DQ and DQS drivers. These drivers are always enabled except for read operations, in which the drivers are tri-stated at the beginning of a read, before the SDRAM I/Os come out of tri-state. The CPC945 drivers are re-enabled at the end of a read operation, after the SDRAMs have tri-stated their drivers.

When the CPC945 does not have DQ and DQS tri-stated for a read, and when it is not doing a write operation, it drives all DQ and DQS low. The intention is to have a quiet bus with any reflections being absorbed by the drivers/receivers being held low. In addition, switching to a power saving state should not drive any glitches onto the bus.

This behavior can be changed with the Idle_BusEn bit in the MemModeCntl register (0xF8002500). When Idle_BusEn is set, when the bus is idle there are no requests in the queues, then the DQ and DQS drivers are tri-stated. Also, the terminations on the receivers are disabled (if enabled - see *Section 7.23 I/O Pad Control*). The Idle_BusEn bit provides extra power savings for systems in which the DDR2 bus is often idle, by not drawing current through the receiver terminators and drivers.

7.22.5 Verniers

The PHY delay verniers used to control the timings of the external interface are based on chains of delay elements. The delay element circuit has a control which selects as the circuit output either the delayed input or the input with no delay. The delay values programmed into the vernier control registers are decoded and applied to successive control inputs in the chain such that the value in the register becomes the number of circuits in the chain with delay chosen, while the remaining elements in the chain have no delay chosen. For example, if “36” is programmed into the control register, then a serial daisy chain of 36 delay elements is created, with the remaining circuits in the chain short-circuited between their inputs and outputs. A total delay of 36 “units” is created, plus some small delay inherent in the delay selection logic, plus some constant fixed delay that is present in the surrounding support logic.

The unit of delay can vary considerably with PVT (process, voltage and temperature variations) and is therefore not specified. It varies, perhaps, from approximately 10 ps to 35 ps, with a nominal time of perhaps 20 ps. The fixed delay from the support logic has a nominal delay of perhaps 150ps.

The verniers use an 8-bit value to control a chain of 255 elements. These verniers are referred to as “one bit time” delays because the generated delay can span approximately one bit time when the delay elements are running at their fastest (10ps, say, * 255 = 2.55ns), for the slowest ddr_clk (400 MHz <=> 2.5 ns bit time).

Typically, as shown in *Figure 7-7*, a ddr_clk is fed into the input of the delay chain, and the output of the delay chain is used as a clock that latches the signal(s) to be delayed.

A variation of the full bit time vernier is to use two half bit time verniers back to back. The half bit time vernier uses 7 control bits. For the 8-bit control value from the register, the upper 7 bits go to each half bit time delay chain. In addition, one delay chain has the lowest control bit added to the upper 7 control bits. Thus, incrementing control values cause the two delay chains to increment their delays ping-ponging between the two delay chains. This tends to average out variations between the two delay chains to create a more linear function of delay versus control value.

7.22.6 1/2 Bit Time Averager

As noted above the delay verniers, for a fixed control value, will produce an amount of delay that can vary with process, voltage and temperature variations. This is unacceptable for producing the DQ to DQS offset of 1/2 bit time. The DQS edges must be tightly centered within the bit time window for reliable clocking of the DQ data.

Therefore, instead of providing a fixed control value to the 1/2 bit time verniers, an auto-adjusted value is fed to these verniers. This value is produced by the 1/2 Bit Time Averager, shown in *Figure 7-8*.

The 1/2 Bit Time Averager essentially works as follows. A bit is launched from a latch clocked by ddr_clk. The bit is propagated down a delay chain similar to those in the delay verniers. The output of the delay chain is captured by another latch clocked by ddr_clk. A state machine cycles different control values to the delay chain, and stops when it finds a value that just barely latches the bit at the end of the chain. This value, then, represents the control value needed to produce one ddr_clk delay = 1 bit time.

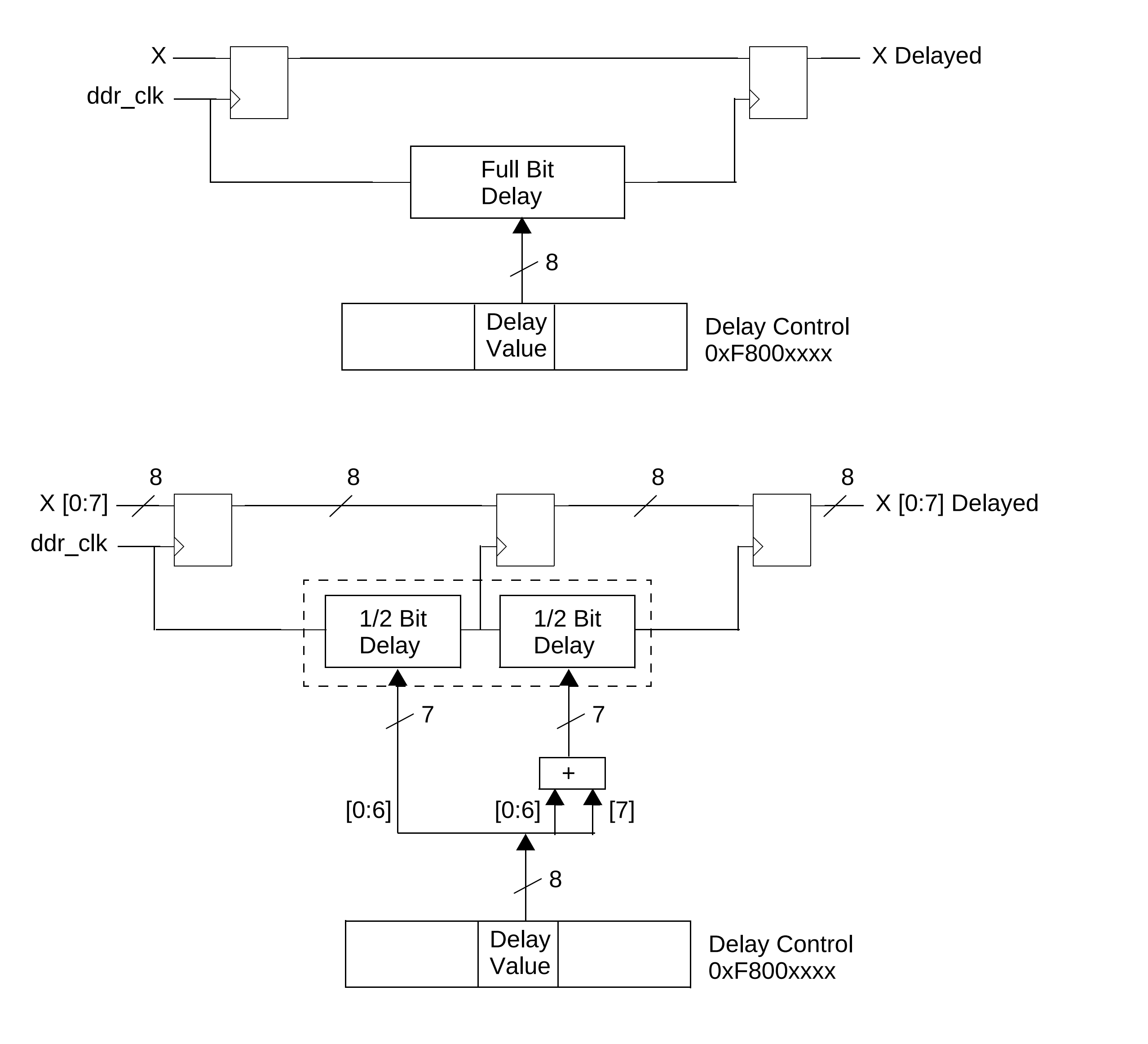
The process is repeated, this time measuring the time from one ddr_clk to two ddr_clk cycles later (2 bit times). The “1 bit time” result is subtracted from the “2 bit time” result to produce a “full bit time” value that is independent of clock-to-data delays and setup times of the latches used to launch and capture the timing signals.

The above measurement is triggered each time a refresh is executed.

This “raw” full bit time value produced can be read in bits 16:23 of the MeasStatusC0 register at 0xF80028F0. Three status bits from the measuring unit are available as bits 29:31 of CalC0 register at 0xF80028E0. Bit 29 indicates that a measurement is done; bit 30 indicates that an overflow occurred during the measurement (the clock is too slow to be measured by the delay chains); bit 31 indicates that an underflow occurred (the clock is too fast).

The full bit time value is fed into a running total. Every 64 updates (every 64 refreshes) the result is divided by 64 to provide an averaged full bit time. This is done to smooth out excessive differences that can occur due to PLL2 clock jitter, the source of ddr_clk, and other sources, particularly when the reference clock to PLL2 has Spread Spectrum variations applied.

Figure 7-7. Example Timing Verniers

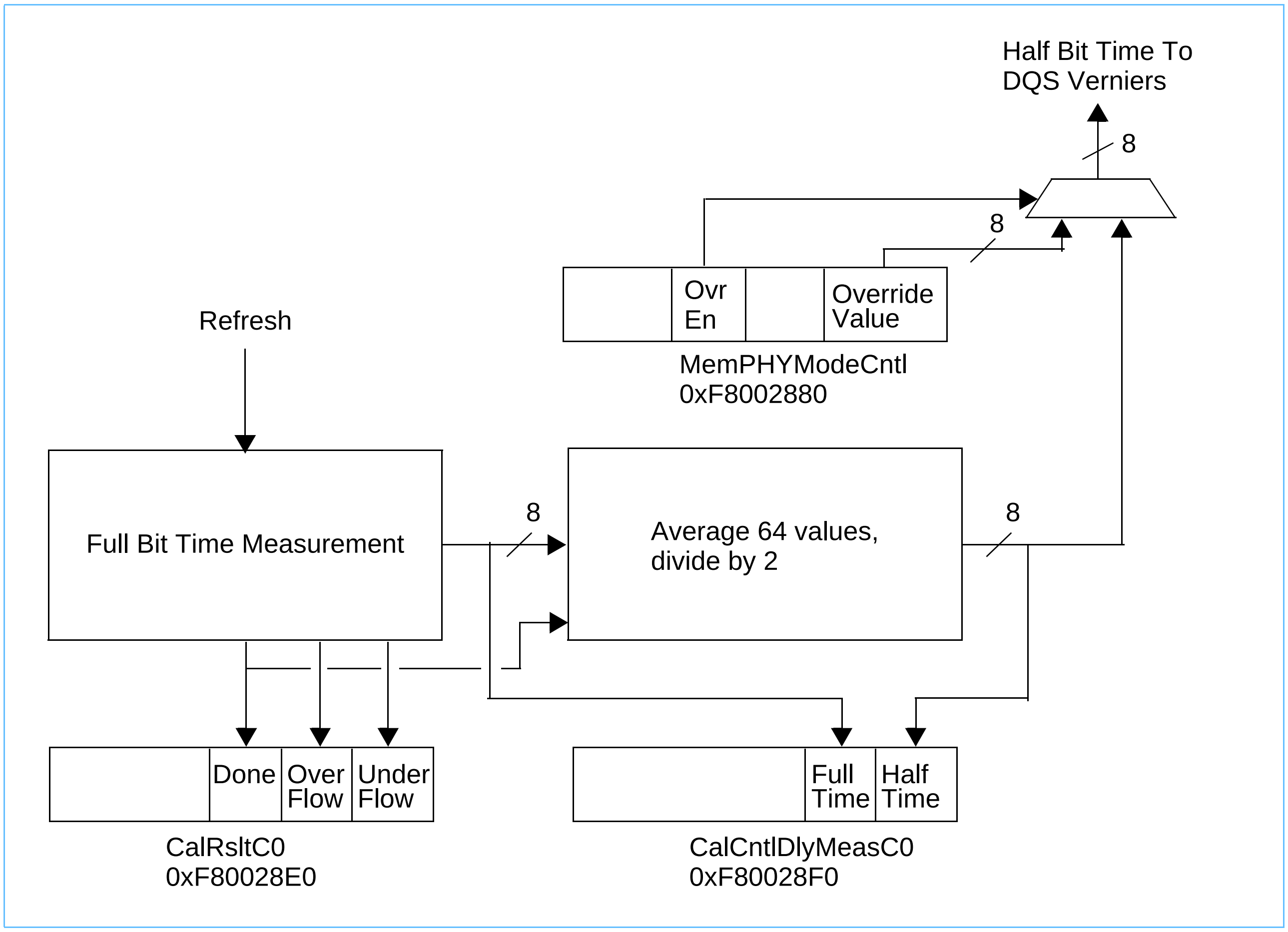


The averaged full bit time is divided by 2 to produce an averaged half bit time. This averaged half bit time value can be read in bits 24:31 of the MeasStatusC0 register at 0xF80028F0.

The value produced by the 1/2 Bit Time Averager should represent the control value required by the DQS verniers to produce 1/2 bit time delay of DQS relative to DQ. Therefore by default the averaged half bit time is sent to the DQS verniers. Note that this value is only updated during refresh. DQ and DQS are idle during refresh, so no jitter is seen on the bus due to the 1/2 bit time control values being updated.

If desired, the user can send any desired value to the DQS verniers by setting bit 16, the Half Bit Delay Override Enable, in the MemPHYModeCntl registers (0xF8002880), and setting the desired delay control value in bits 24:31 of MemPHYModeCntl. (Note that if the override is applied during a data transfer the DQS signals could jitter when the register update is applied.)

Figure 7-8. 1/2 Bit Time Averager



7.22.7 DQS 1/2 Bit Time Offset

The averaged half bit time value will be a function of the chip itself (it could be a “fast” or “slow” chip depending on process variations when the chip was manufactured), the chip voltage, and the chip junction temperature. Regardless, since the delay values used by the 1/2 Bit Time Averager and the delay lines used by the DQS verniers should track, the value produced by the 1/2 Bit Time Averager should cause a constant 1/2 bit time delay by the DQS verniers.

To compensate for any inaccuracy, and to allow somewhat for byte-to-byte (or nibble-to-nibble) variations, each DQS vernier has an offset circuit. This circuit, shown in *Figure 7-9*, adds a signed value to output of the 1/2 Bit Time Averager unit. The signed value, in Sign + Magnitude format (1-bit sign, 7-bit magnitude) comes from the Strobe Control registers. The output of the adder is clamped by a saturation circuit. If the signed control value is positive the output of the adder will be clamped at 0xFF; If the signed control value is negative the output of the adder will be clamped at 0x00.

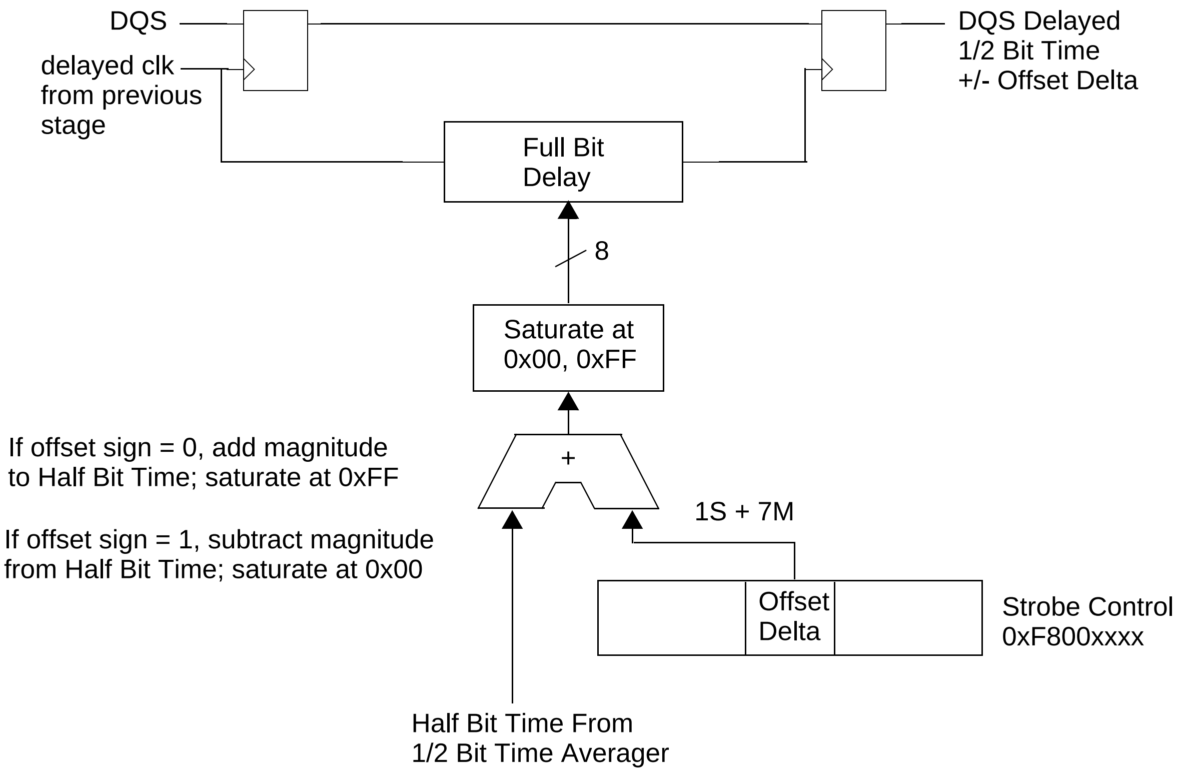
The circuit shown in *Figure 7-9* is replicated twice in each byte lane for the ***write*** DQS. With 18 byte lanes there are a total of 36 DQS delays, each with its own offset circuit and its own Offset Delta value in a Strobe Control register. For systems with ×4 chips, all 36 circuits are used to control the 36 single-ended DQS. (Assuming 4 DQS are used for ECC check bits.)

For systems with ×8 or ×16 chips, on writes half of these circuits are used for the 18 differential write DQS, and half are not used.

The circuit is replicated four times in each byte lane for the ***read*** DQS, for a total of 72 DQS delays, each with its own offset circuit and its own Offset Delta value in a Strobe Control register. Each DQS uses two circuits: one for the rising edge and one for the falling edge. For systems with ×4 chips, all 72 circuits are used to individually delay the rising and falling edges of the incoming 36 single-ended DQS.

For systems with ×8 or ×16 chips, on reads all circuits are used for the 18 differential read DQS. A single DQS is used to clock two 4-bit nibbles, each with its own rising edge and falling edge delays. Both rising edge offsets must be programmed (with equal values), and both falling edge offsets must be programmed (with equal values).

Figure 7-9. DQS Offset Delta



7.23 I/O Pad Control

7.23.1 I/O Pad Bit Settings

The I/O Pad Control register (0xF80029A0) controls the electrical characteristics of the CPC945 drivers and receivers for the DDR2 memory interface. This register must be programmed before any operations are attempted with the external memory.

Bit 11, the MCSE_dqs bit, controls the operating mode of the DQS driver/receivers. There are 18 I/O circuits for DQS. If MCSE_dqs = 0 each circuit operates as a single differential driver/receiver, providing 18 differential DQS connections. If MCSE_dqs = 1, these I/O operate as 2 separate single-ended driver/receivers, providing 36 single-ended DQS connections.

When using ×8 or ×16 memory chips MCSE_dqs must be set = 0, and when using ×4 memory chips MCSE_dqs must be set = 1.

The CPC945 memory controller and the external SDRAMs must match their DQS single-ended/differential mode. That is, the MCSE_dqs bit must match that of the A[10] bit programmed in the EMRS (see *Section 7.10.2 EMRS Settings* on page 153).

Table 7-37 gives the DQS pad numbering of differential mode vs. single-ended mode.

Table 7-37. DQS Bit Numbering.

Differential Mode MCSE_dqs = 0	Differential Mode Associated DQ	Single-Ended Mode MCSE_dqs = 1	Single-Ended Mode Associated DQ
DQS 0P	DQ[0:7]	DQS 0	DQ[0:3]
DQS 0N		DQS 1	DQ[4:7]
DQS 1P	DQ[8:15]	DQS 2	DQ[8:11]
DQS 1N		DQS 3	DQ[12:15]
DQS 2P	DQ[16:23]	DQS 4	DQ[16:19]
DQS 2N		DQS 5	DQ[20:23]
DQS 3P	DQ[24:31]	DQS 6	DQ[24:27]
DQS 3N		DQS 7	DQ[28:31]
DQS 4P	DQ[32:39]	DQS 8	DQ[32:35]
DQS 4N		DQS 9	DQ[36:39]
DQS 5P	DQ[40:47]	DQS 10	DQ[40:43]
DQS 5N		DQS 11	DQ[44:47]
DQS 6P	DQ[48:55]	DQS 12	DQ[48:51]
DQS 6N		DQS 13	DQ[52:55]
DQS 7P	DQ[56:63]	DQS 14	DQ[56:59]
DQS 7N		DQS 15	DQ[60:63]
DQS 8P	DQ[64:71]	DQS 16	DQ[64:67]
DQS 8N		DQS 17	DQ[68:71]
DQS 9P	DQ[72:79]	DQS 18	DQ[72:75]
DQS 9N		DQS 19	DQ[76:79]

Table 7-37. DQS Bit Numbering.

Differential Mode MCSE_dqs = 0	Differential Mode Associated DQ	Single-Ended Mode MCSE_dqs = 1	Single-Ended Mode Associated DQ
DQS 10P	DQ[80:87]	DQS 20	DQ[80:83]
DQS 10N		DQS 21	DQ[84:87]
DQS 11P	DQ[88:95]	DQS 22	DQ[88:91]
DQS 11N		DQS 23	DQ[92:95]
DQS 12P	DQ[96:103]	DQS 24	DQ[96:99]
DQS 12N		DQS 25	DQ[100:103]
DQS 13P	DQ[104:111]	DQS 26	DQ[104:107]
DQS 13N		DQS 27	DQ[108:111]
DQS 14P	DQ[112:119]	DQS 28	DQ[112:115]
DQS 14N		DQS 29	DQ[116:119]
DQS 15P	DQ[120:127]	DQS 30	DQ[120:123]
DQS 15N		DQS 31	DQ[124:127]
DQS 16P	DQ[128:135] ECC - goes to same DIMMs as DQ[0:63]	DQS 32	DQ[128:131] ECC - goes to same DIMMs as DQ[0:63]
DQS 16N		DQS 33	DQ[132:135] ECC - goes to same DIMMs as DQ[0:63]
DQS 17P	DQ[136:143] ECC - goes to same DIMMs as DQ[64:127]	DQS 34	DQ[136:139] ECC - goes to same DIMMs as DQ[64:127]
DQS 17N		DQS 35	DQ[140:143] ECC - goes to same DIMMs as DQ[64:127]

The remaining bits in the I/O Pad Control register affect the drive strength of the drivers (output impedance) and the termination values (input impedance) of the receivers. In general, the settings of these bits are determined by an electrical analysis (for example, Spice) of the board wiring and loads attached to the CPC945 I/O.

The output impedance of the drivers can be set to full SSTL_18 levels or half strength. Bit 0 controls the clock pads (CK_A/CK_AN, CK_B, CK_BN); bits 1, 5 and 7 control the control signal pads (CS, CKE, ODT); bit 2 controls the command/address pads (RAS, CAS, WE, BA, MAD); bits 3 controls the DQ pads, and bit 4 control the DQS pads, for driving during write operations.

All of the CPC945 DDR2 interface receivers are contained in bi-directional driver/receiver circuits.

The TT0 (termination type 0) bits control if and when the receivers are terminated. These are 2-bit values which have the following settings: 0x = termination always off (no termination), 10 = termination on for reads (when the driver is tri-stated), 11 = termination is always on.

Bits 17 and 18 specify the TT0 value for the DQ receivers.

As described above the DQS I/O have two pads (P and N) which are programmed by MCSE_dqs to be either the 2 pads of a differential pair, or 2 single-ended I/O. The TT0 values for the two pads are individually controlled, using bits 12 and 13 for the P pad and bits 14 and 15 for the N pad.

If and when the receiver terminations are enabled, the TT1 (termination type 1) bits set the termination value. 0 = 75 ohm and 1 = 150 ohm. Bit 19 specifies the TT1 value for the DQ receivers; bit 16 specifies TT1 value for the DQS receivers.

7.23.2 Relationship to Memory Chip Settings

As noted above it is important the both the CPC945 and the memory chips match with regard to the programming of the DQS I/O (differential vs. single-ended). It should also be noted that the memory chips have other settings that in general are determined in conjunction with the CPC945 I/O Pad Control Settings.

The SDRAM EMRS(1) register uses bit A1 to set the outputs to full or 1/2 drive strength.

The SDRAM EMRS(1) register uses bits A6 and A2 to specify ODT (On Die Termination - the SDRAM receiver termination) operation and termination values. The CPC945 generates control signals to tell the SDRAMs when to apply the receiver termination. This is discussed in *Section 7.25.7 ODT - On Die Termination*.

In summary, when considering the electrical operation of the CPC945 external connections to DDR2 memory, the I/O Pad Control register has to be programmed correctly, the DDR2 EMRS has to be programmed correctly, and the CPC945 ODT control has to be programmed correctly, based upon an analysis (for example, Spice) of the board wiring and the loads presented by the DDR2 memories.

7.24 Memory Clocks

The CPC945 has an internal clock, ddr_clk, which is driven by PLL2 in the power manager. PLL2 is programmed at the desired data transfer frequency, for example, 400 MHz or 533 MHz. A 1/2 frequency version of this clock, mem_clk, is used to clock the memory controller control logic, for example, 200 MHz or 266 MHz. This internal mem_clk is also driven off-chip to be used as the memory clock CK by the DDR2 memories, and by support components such as the latches on registered DIMMs or discrete registers on the board.

As required by the DDR2 memories the output clock is differential. There are two copies of the clock, A and B, for a total of 4 output pins: DDR_CK_A/DDR_CK_AN, DDR_CK_B/DDR_CK_BN.

The two copies are identical except that each copy has a unique vernier, so that the two copies can be individually offset in time relative to the internal mem_clk (and relative to the control signal outputs DDR_RAS, DDR_CAS, DDR_WE, DDR_BA[2:0], DDR_MAD[15:0], DDR_CS[0:15], DDR_CKE[0:7] and DDR_ODT[0:7]).

The vernier for a given clock output made up of 2 full bit time verniers in series. (Technically the first vernier is two 1/2 bit time verniers in series, in which the 1/2 of the control value is applied to the 2 verniers.)

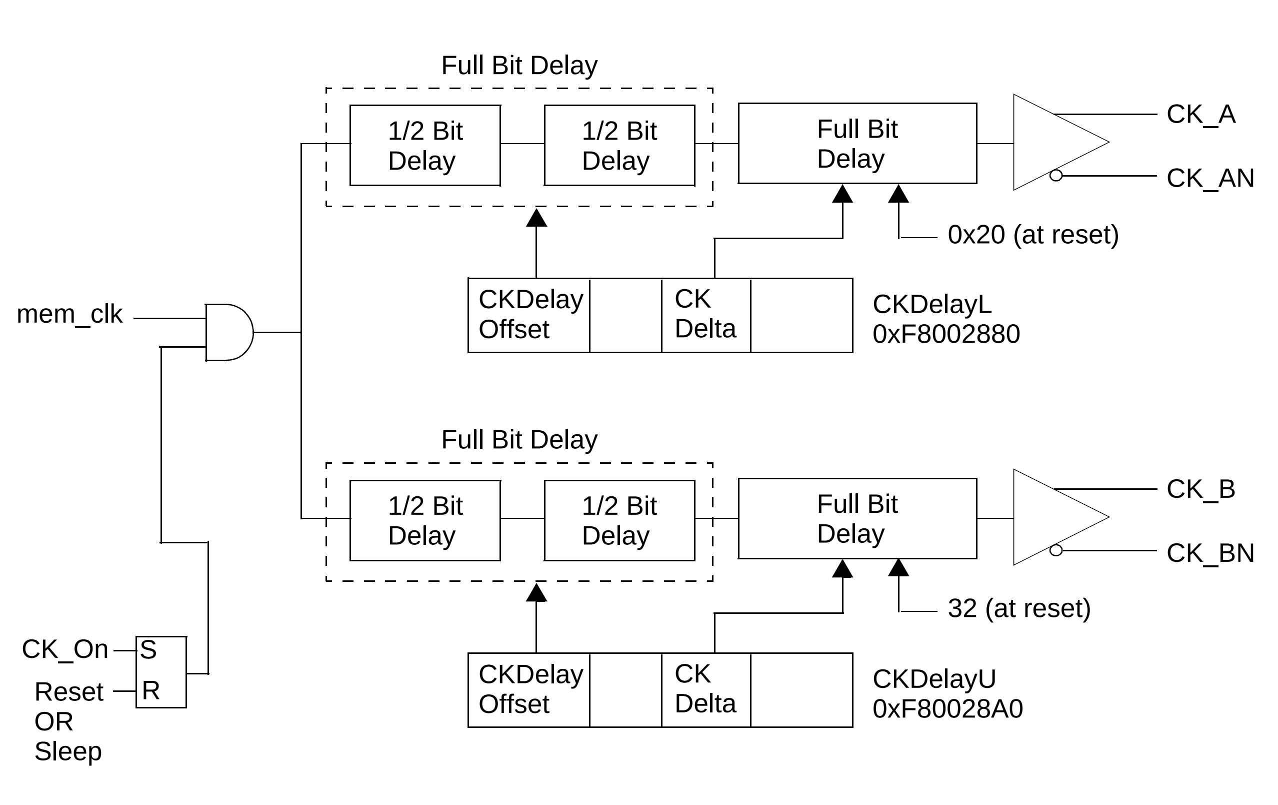
The verniers for CK_A are programmed using the CKDelayL register (0xF8002890). The verniers for CK_B are programmed using the CKDelayU register (0xF80028A0). Each register is identical and has to 8-bit fields, CKDelayOffset and CKDelta, to program the 2 full bit time verniers. Note that following reset, the 2nd vernier in the series will have a forced control value of 0x20 (decimal 32) instead of the CKDelta value from the register. The first write to the register will replace this forced control value with the user programmed value of CKDelta. (The intent of this mechanism is to have a memory interface that will be clocked with some sufficient setup and hold even if the user has not programmed the CKDelayL/U registers.)

At least one of these two clocks must be used to clock the DDR2 memories. Whether it is A or B, or both clocks that is distributed to the memories and support components is up to the system designer.

By default the external CK is disabled following reset. It must be explicitly enabled using bit 0, CK_On, in the MemModeCntl register (0xF8002500). Once enabled, the external CK cannot be disabled except by a chip reset, or when the SDRAMs enter Self Refresh. (See the sections on Self Refresh in *Section 7.7 Operational States*.)

Figure 7-10 gives a simplified diagram of the CK_A/CK_B vernier mechanism.

Figure 7-10. CK Timing Adjustment



7.25 Memory Control Signals

The outputs DDR_RAS, DDR_CAS, DDR_WE, DDR_BA[2:0], DDR_MAD[15:0], DDR_CS[0:15], DDR_CKE[0:7] and DDR_ODT[0:7] are signals which are driven off-chip to the memories. These signals are the only DDR2 I/O which are not controlled by verniers.

7.25.1 Adjustable Cycle Delay

Controls in the Mem PHY Mode Control Register (0xF8002880) enable the address and control outputs to the DIMMs to be delayed by an integer number of ddr_clks. Although these signals switch on mem_clk boundaries (for example, 200 MHz or 266 MHz), the delay control is in increments of ddr_clks (400 MHz or 533 MHz).

Control signals other than ODT can be delayed 0, 1, 2, or 3 ddr_clks:

- DDR_RAS, DDR_CAS, DDR_WE, DDR_BA[2:0] and DDR_MAD[15:0] are delayed as a group, controlled by MemPHYModeCntl bits 0:1.
- DDR_CKE[0:3] are delayed as a group, controlled by MemPHYModeCntl bits 4:5.
- DDR_CKE[4:7] are delayed as a group, controlled by MemPHYModeCntl bits 6:7.
- DDR_CS[0:15] are delayed as a group, controlled by MemPHYModeCntl bits 8:9.

ODT can be delayed 0 through 7 ddr_clks:

- DDR_ODT[0:7] are delayed as a group, controlled by MemPHYModeCntl bits 10:12.

7.25.2 Command

The DDR_RAS, DDR_CAS and DDR_WE are negative active signals. They are connected to the inputs $\overline{\text{RAS}}$, $\overline{\text{CAS}}$ and $\overline{\text{WE}}$, respectively, of all memory chips.

7.25.3 Address

The DDR_BA[2:0] and DDR_MAD[15:0] outputs are connected to the BA[2:0] and A[15:0] inputs of all memory chips. Little-endian numbering must be preserved, for example, DDR_MAD[0] must be connected to A[0], DDR_MA[1] must be connected to A[1], and so on. Unused high order bits can be left unconnected.

7.25.4 Chip Select

DDR_CS[0:7] are negative active outputs that are wired to the $\overline{\text{CS}}$ inputs of ranks 0 through 7, respectively. Outputs to unused ranks can be left unconnected.

DDR_CS[8:15] are identical copies of DDR_CS[0:7], respectively.

Traditionally for systems using the 128-bit bus with 128-bit transfers (DIMM pairs), CS[0:7] are wired to the DIMMs providing data bits 0:63, and CS[8:15] are wired to the DIMMs providing data bits 64:127, but this is just a convention.

7.25.5 CKE - Clock Enables

CKE[0:7] are positive active outputs that are wired to the CKE inputs of ranks 0 through 7, respectively. Outputs to unused ranks can be left unconnected.

For systems using the 128-bit bus with 128-bit transfers (DIMM pairs), a given CKE must be wired to both DIMMs of the pair (for example, CKE[0] must be wired to the CKE inputs for rank 0 for both DIMMs in the pair).

As noted in *Section 7.7.4 Self Refresh* on page 139, it is possible to put the DDR2 memories in self refresh, remove the power to the CPC945 (while leaving the DDR2 memories powered on), restore the CPC945 power, take the DDR2 memories out of self refresh, and continue to use the DDR2 memories with the data contents preserved. But if the power to the CPC945 is removed there must be some way to guarantee that the CKEs stay low, e.g, by using pull-down resistors on the CKE signals.

By default the CKEs stay low during self refresh. Bit 12, CkeTsEn in the MemModeCntl register (0xF8002500), when set causes the CKEs to be tri-stated during self refresh. When this option is chosen, external pull-down resistors should be used.

7.25.6 Dynamic CKE

Following the DRAM initialization sequence (*Section 7.10 Memory Device Initialization*), by default the CKE signals to the memories will be active, enabling the fastest access at the expense of power. As discussed in *Section 7.7.3.2 Power Management During Normal Operation* the CPC945 has a Dynamic CKE option, in which the CKEs are made inactive to memories that have not been accessed for 20 cycles. This reduces DRAM power to ranks with CKE made inactive, at the expense of additional latency to access a rank that has had CKE at the inactive level.

Dynamic CKE operation is enabled with bit 6 of the Memory Mode Control Register (0xF8002500). If dynamic CKE is enabled, the internal fast path around the queues is disabled (*Section 7.18.2 Fast Path* on page 188).

By default dynamic CKE operation, when enabled, operates in “DIMM Mode,” that is, the pair CKEs which control the two ranks of a DIMM are disabled/enabled together. For finer-grained control, if bit 5 of the MemModeCntl register is set then dynamic CKE operates in “Rank Mode,” in which CKE to each rank is individually disabled/enabled.

There are two restrictions when both ODT and Dynamic CKE are enabled:

- For this case it is required that dynamic CKE operation be set to DIMM mode (MemModeCntl bit 5 = 0).
- Also as noted in *Section 7.10.1 MRS Settings* on page 152, MRS bit A[12] must be set to 1 = tXARDS (Slow Exit).

7.25.7 ODT - On Die Termination

ODT[0:7] are positive outputs that are connected to ODT inputs on the memories.

ODT usage is optional. For systems that do not wire up ODT from the CPC945, the ODT inputs to the memories must be grounded.

7.25.7.1 ODT Operation

ODT (On Die Termination) operation is controlled with the ODT Control Register (0xF80023A0). When ODT is enabled the CPC945 memory controller activates the ODT signals to the memories to dynamically enable/disable the memory chip on-die termination of the DQ and DQS signals.

By default ODT operation is enabled. ODT Control Register bit 0 must be set to 1 to disable ODT signaling.

When ODT operation is enabled, bit 1 (ODT Resolution) must be set to 1. *Note the quirk in default values: following reset the value of bit 1 = 0 is illegal, because bit 0 = 0 which specifies ODT is enabled*.

The CPC945 memory controller operates ODT on a per-DIMM basis. For DIMMs which have 2 ranks, an ODT output is wired to the DIMM ODT input for one rank (for example, rank 0). The other DIMM ODT input must be disabled (for example, rank 1 ODT input is wired to ground).

The CPC945 DDR_ODT[0:3] outputs are wired to DIMMs 0 though 3 (ranks 0, 2, 4 and 6). DDR_ODT[4:7] are identical to DDR_ODT[0:3], respectively, and are wired to DIMMs 4 through 7 for the 128-bit bus configuration (ranks 0, 2, 4 and 6).

By default the CPC945 drives the ODT signals only on memory writes. If ODT Control Register bit 3, ODT RdEn is set then the ODT signals will be driven on both writes and reads.

By default the CPC945 drives the ODT signal of the DIMM being accessed. This is called Direct Mode. If ODT Control Register bit 2, ODT Assign is set, then ODT operates in Indirect Mode, in which ODT is driven to the DIMM adjacent to the accessed DIMM. (That is, if DIMM 0 is accessed then ODT[1] is driven, if DIMM 1 is accessed then ODT[0] is driven, if DIMM 2 is accessed than ODT[3] is driven, and if DIMM 3 is accessed then ODT[2] is driven.)

If general for systems with DQS and DQ connections that essentially have one drop (systems in which a multiplexer steers the DQS and DQ signals to one and only one DIMM) then Direct Mode must be used. For systems with multi-drops Indirect Mode might be preferable, although an electrical analysis should be performed to determine which Mode is best.

7.25.7.2 ODT Timing

The CPC945 generates ODT timing that is valid only for a CAS latency of 4. ODT for reads is generated one cycle later than for writes, to match the times when data is on the bus.

- If ODT operation is enabled, and if CAS latency is greater than 4, then the ODT signals must be delayed using the adjustable cycle delay described in *Section 7.25.1*. That is, MemPHYModeCntl bits 10:12 must be programmed to add delay to the ODT signals. For each increment in CAS latency, the ODT signals must be delayed by 2 ddr_clks. For example, if CAS latency = 5, then MemPHYModeCntl[10:12] should be increased by 2, if CAS latency = 6, then MemPHYModeCntl[10:12] should be increased by 4, and so on.
- For systems with ODT enabled, it does not make sense to use memories with a CAS latency of 3. Since ODT timings cannot be adjusted with negative values, the remaining signal timings must be delayed to match that of the ODT timings. This increases the CAS latency, using one of two methods:
  - The easiest way is to program the memory chips (MRS) and the memory controller (for example, RAS and CAS Timer registers) to use CL = 4.
  - A less elegant way is to leave the memory chips at CL = 3, and to delay all signals except ODT (for example, MemPHYModeCntl reg bits 0:1, 4:5, 6:7, and 8:9 to delay RAS, CAS, WE, BA, MAD, CKE and CS). A value of “2” should be used to delay these signals by one mem_clk.

7.25.7.3 Other ODT Considerations

As described in *Section 7.10 Memory Device Initialization* the memory chips have their MRS registers loaded in an initialization sequence. The MRS has 2 bits which specify if ODT is enabled, and if enabled, what impedance value should be used (for example, 75 ohm or 150 ohm).

As described in *Section 7.23 I/O Pad Control*, there are control bits in the I/O Pad Control Register which might be used in conjunction with ODT. Similar to how ODT controls the pad termination attributes of the memory chip DQS and DQ I/O, these bits control the pad termination attributes of the CPC945 DDR_DQS and DDR_DQ I/O. They are MCTT0_dqs and MCTT0N_dqs, which determine when the DQS I/O have their terminations applied, MCTT1_dqs, which determines the value of the DQS termination, MCTT0_dq, which determine when the DQ I/O have their terminations applied, and MCTT1_dq, which determines the value of the DQ termination.

If ODT is disabled in the memory chips MRS, then ODT should be disabled in the CPC945. If ODT is enabled in the memory chips, then ODT should be enabled in the CPC945.

A board level signal analysis (for example, Spice) should be performed to determine when the CPC945 DDR_DQS and DDR_DQ signals should have termination applied and what the termination value should be, if ODT should be enabled, what the memory chip termination value should be, if memory chip termination should be just for writes or for both reads and writes, and if the ODT mode should be Direct or Indirect.

When Dynamic CKE is enabled, note the restrictions discussed in *Section 7.25.6 Dynamic CKE* that CKE DIMM mode must be set if ODT is enabled, and MRS A[12] must be set to 1 = tXARDS (Slow Exit).

7.25.8 Control Signal Summary

Figure 7-11 summarizes the wiring of the CS, CKE, and ODT controls to the DIMMs for the 3 system memory configurations. Also shown are the ranks that are selected by the MUXEN[0:1] bits.

Figure 7-11. Control Signal Wiring Summary

a) 128-bit: 64BitBus = 0, 64BitCfg = 0

	DQ[0:63], [128:135]	DQ[64:127], [136:143]	
Rank0 Rank1	DIMM0 CS0 CKE0 ODT0 CS1 CKE1 GND	DIMM4 CS8 CKE0 ODT4 CS9 CKE1 GND	Mx = 00
Rank2 Rank3	DIMM1 CS2 CKE2 ODT1 CS3 CKE3 GND	DIMM5 CS10 CKE2 ODT5 CS11 CKE3 GND	Mx = 01
Rank4 Rank5	DIMM2 CS4 CKE4 ODT2 CS5 CKE5 GND	DIMM6 CS12 CKE4 ODT6 CS13 CKE5 GND	Mx = 10
Rank6 Rank7	DIMM3 CS6 CKE6 ODT3 CS7 CKE7 GND	DIMM7 CS14 CKE6 ODT7 CS15 CKE7 GND	Mx = 11

b) 64-bit: 64BitBus = 1, 64BitCfg = 1

	DQ[0:63]	
Rank0 Rank1	DIMM0 CS0 CKE0 ODT0 CS1 CKE1 GND	Mx = 00
Rank2 Rank3	DIMM1 CS2 CKE2 ODT1 CS3 CKE3 GND	Mx = 01
Rank4 Rank5	DIMM2 CS4 CKE4 ODT2 CS5 CKE5 GND	Mx = 10
Rank6 Rank7	DIMM3 CS6 CKE6 ODT3 CS7 CKE7 GND	Mx = 11

c) Hybrid: 64BitBus = 0, 64BitCfg = 1

	DQ[0:63]	DQ[64:127]		
Rank0 Rank1	DIMM0 CS0 CKE0 ODT0 CS1 CKE1 GND	DIMM2 CS4 CKE4 ODT2 CS5 CKE5 GND	Rank4 Rank5	Mx = x0
Rank2 Rank3	DIMM1 CS2 CKE2 ODT1 CS3 CKE3 GND	DIMM3 CS6 CKE6 ODT3 CS7 CKE7 GND	Rank6 Rank7	Mx = x1

7.26 Data Timing Coarse Controls

This section gives an overview of the timing relationship between the control signals (RAS, CAS, WE) and the data bus (write data timing, read data timing, data bus tri-state timing, external multiplexer timing).

As shown in *Figure 7-3 Memory Controller Internals* on page 142 and *Figure 7-5 Command Arbiter Flow* on page 187, the Command Arbiter selects a memory operation to perform. It generates 2 internal signals, wrRequest and rdRequest, which are sent to the datapath, and it generates the RAS, CAS, WE, BA, MAD, CS, CKE and ODT signals which are sent to the memories.

The internal wrRequest and rdRequest signals are sent through internal 16-deep pipelines. Taps in the pipeline are used to control when write data is sent on the data bus, when read data is captured from the data bus, when the external data multiplexers are switched, and when the data bus is tristated. The taps are selected using values in the Memory Bus Configuration register (0xF80022D0). These values work in units of ddr_clks (for example, 533 MHz).

For systems with very large delays on the data bus, the MemBusConfig values affecting read operations can be extended using the RdPipeDly field in the MemBusConfig2 Register (0xF80022E0).

As the signals are pulled off taps from the pipeline, they are sent to verniers in the DDR2 PHY to provide extra delay that can be adjusted in delay line increments of approximately 20 ps. In general, with zero values in the verniers, the coarse controls are adjusted to the point that one more ddr_clk cycle would delay the desired signal too late, then the vernier is used to provide additional sub-cycle delay.

7.27 Write Data Timing

7.27.1 Write Coarse Timing

Figure 7-12 shows the internal wrRequest signal generated by the Command Arbiter for a write. One mem_clk (2 ddr_clks) later, the CS, CAS, WE, etc., control signals are driven off the chip (assuming no delay specified in the MemPHYModeCntl Register).

Also one mem_clk (2 ddr_clks) later, the internal signal ddr_DdrWdbRdV is sent to the WDB (Write Data Buffer) to request write data. The write data from the buffer is available one mem_clk (2 ddr_clks), then it takes an additional 4 ddr_clks to cross the chip, select the correct source (for example merge RMW data), and generate ECC check bits. The cycle to select the correct source is always taken, regardless if the operation is a RMW or not. The cycle to generate ECC check bits is always taken, regardless if ECC is enabled or not.

In summary, from the time that the control signals such as CAS and WE are driven out, until the time that the data (DQ) is driven out, there is a minimum of 3 mem_clks (6 ddr_clks).

The internal signal ddr_DdrWdbRdV is moved to the right in ddr_clk increments using the WdbRqDly field (bits 0:3) in the MemBusConfig register. This in turn moves all write data pipeline stages to the right, with the result that the time from CAS, WE going valid to DQ going valid = WdbRqDly + 6 (in ddr_clks).

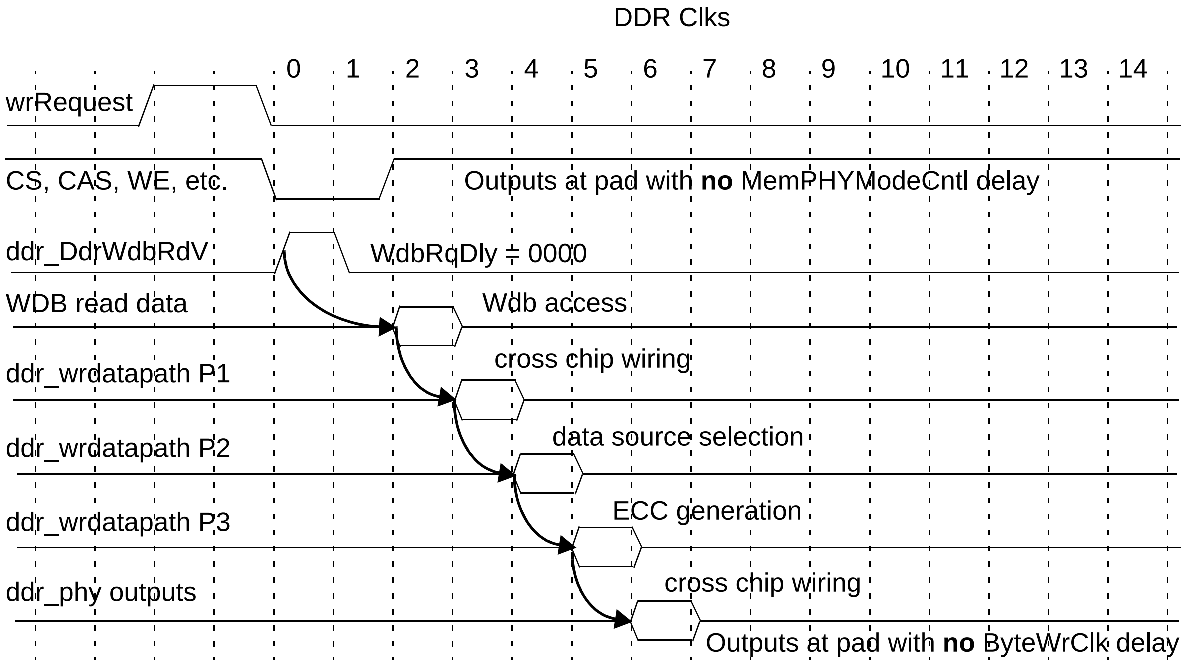
The DQS signals are timed using the same pipeline as the data. As discussed below and elsewhere, an extra 1/2 bit time of delay is added (using a vernier) to center the edges of DQS within the data.

The correct value for WdbRqDly is a function of several factors, including:

- Board wiring delays on the control signals.

- Additional delays on the control signals from the MemPHYModeCntl register.
- Delays in increments of mem_clk, from registers on the DIMMs or on the board.
- The CAS latency of the memories.
- Wiring delays on the DQ and DQS signals.
- Additional delays on the DQ and DQS signals from external multiplexers.

Figure 7-12. Write Data Timing Example



7.27.2 Write Vernier Timing

Figure 7-13 gives a simplified diagram of the write verniers for a byte lane.

For each byte lane, 8 internal DQ, 1 internal OE, and 1 internal DQS are latched by ddr_clk, on the left side of the figure. The outputs of these latches corresponds to the last line in *Figure 7-12* with the comment “Outputs at pad with no ByteWrClk delay.”

As can be seen in *Figure 7-17*, each byte lane has a Write Strobe Control Register, also known as the ByteWrClkDelay register. For example, byte lane 0 in cluster 0 has the ByteWrClkDelayC0B00 register.

Bits 0:7 of the ByteWrClkDelay register contain the 8-bit WrClkOffset field. As shown in *Figure 7-17* this field controls a full bit time vernier that delays equally the 8 DQ which are sent to the DQ pads, the OE which is connected to the DQ pad driver enables, and the DQS signal. (The full bit time vernier uses two half bit time verniers, as discussed in *Section 7.22.5* and shown in more detail in *Figure 7-7*.)

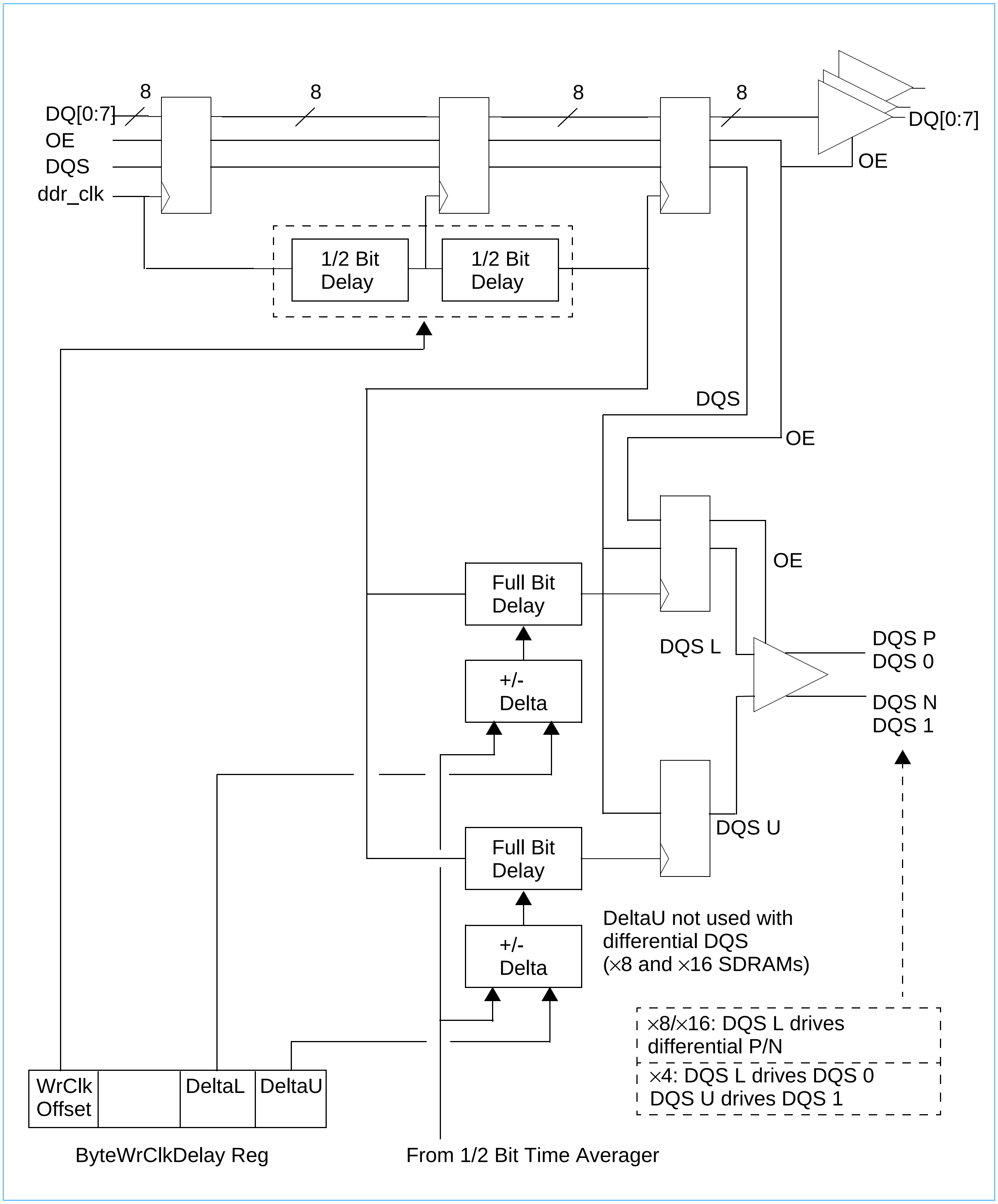
The DQS and OE are additionally delayed by a full bit time vernier which is controlled by a 1/2 Bit Time Offset circuit as discussed in *Section 7.22.7* and shown in more detail in *Figure 7-9*. The 1/2 Bit Time Offset circuit is fed by the output of the 1/2 Bit Time Averager to nominally add 1/2 bit time of delay to DQS and its OE, and register bits 16:23, WrClkOffsetDeltaL to adjust this delay +/-. For systems with ×8 or ×16 chips this DQS and OE are used by an I/O circuit configured as a single differential driver. For systems with ×4 chips the I/O circuit is configured to drive the “P” pad in single-ended mode.

The DQS delayed by WrClkOffset goes to a 2nd full bit time vernier controlled by a 2nd 1/2 Bit Time Offset circuit which is fed by register bits 24:31, WrClkOffsetDeltaU. For systems with ×8 or ×16 chips this 2nd vernier is not used. For ×4 systems it is used by the I/O circuit configured to drive the “N” pad in single-ended mode.

The OE delayed by WrClkOffsetDeltaL is used to tri-state the differential driver in differential mode, and both single-ended drivers in single-ended mode.

Summary: All DQ and DQS are adjusted together using WdbRqDly in the MemBusConfig register as a coarse control. (OE is discussed in *Section 7.29*.) DQ, OE and DQS are adjusted together, on a byte basis, using the WrClkOffset fields of the Write Strobe Control registers. DQS and its OE are additionally delayed by 1/2 bit time using the 1/2 Bit Time Averager output as a control value, which can be adjusted +/- using the 1 Sign +7 Magnitude value of WrClkOffsetDelta in the Write Strobe Control register. For ×8 and ×16 chips the WrClkOffsetDeltaL supplies this delay to a single differential DQS driver. For ×4 chips the WrClkOffsetDeltaL and WrClkOffsetDeltaU supply 2 separate delays to an I/O with 2 separate single-ended drivers.

Figure 7-13. Byte Lane Write Verniers



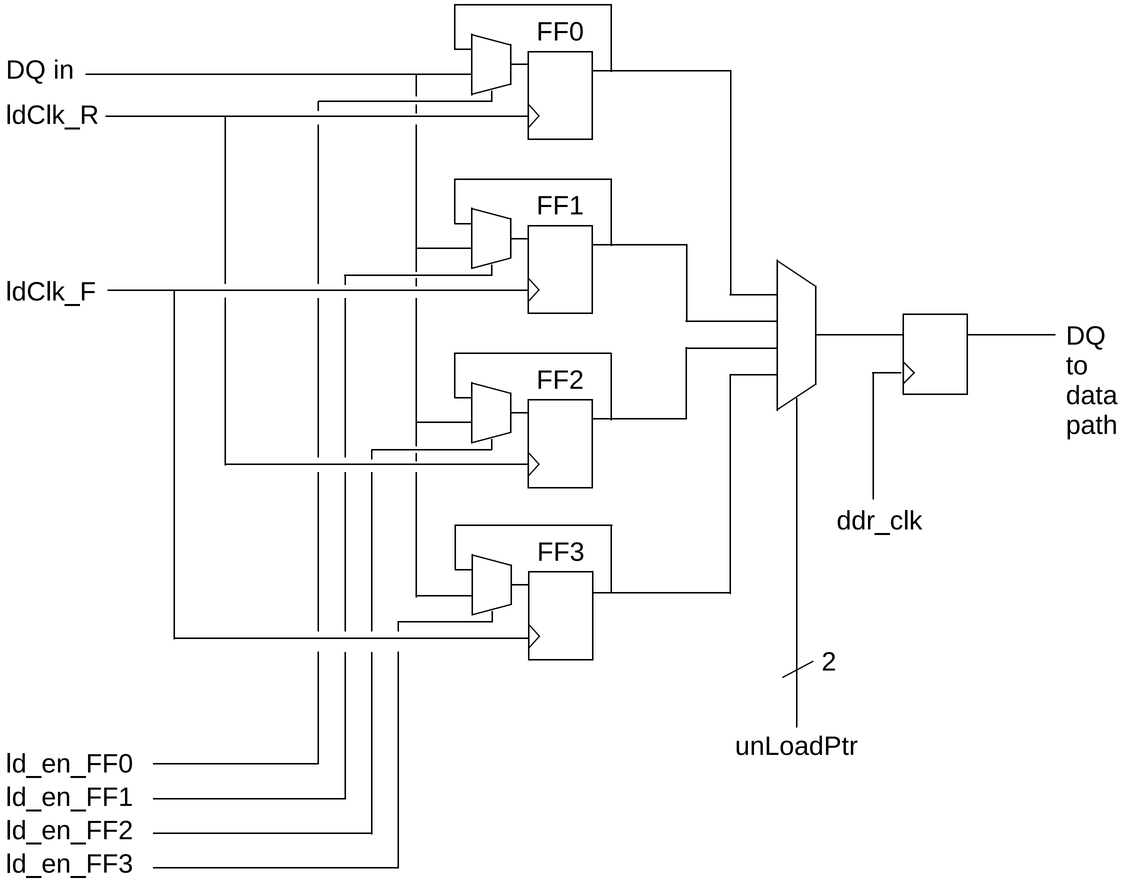
7.28 Read Data Timing

7.28.1 Read Timing Overview

For read operations the memory chips drive read data (DQ) and the read clocks (DQS) into the CPC945, with the DQ and DQS being clocked at the same time using the CK input to the memory chips. There are four beats of data and four DQS edges to clock that data: 2 rising edges and 2 falling edges.

Each bit of DQ is clocked successively into a set of 4 data capture latches, that is, a 1-bit wide, 4-deep FIFO. With the 4 latches labeled FF0, FF1, FF2, FF3, the rising edge of DQS is used to clock FF0 and FF2, and the falling edge of DQS is used to clock FF1 and FF3. *Figure 7-14* shows the FIFO for one bit of DQ. This circuit is replicated 8 times in each byte lane. With 18 byte lanes, there are 144 instances of this circuit.

Figure 7-14. Read Data Capture FIFO

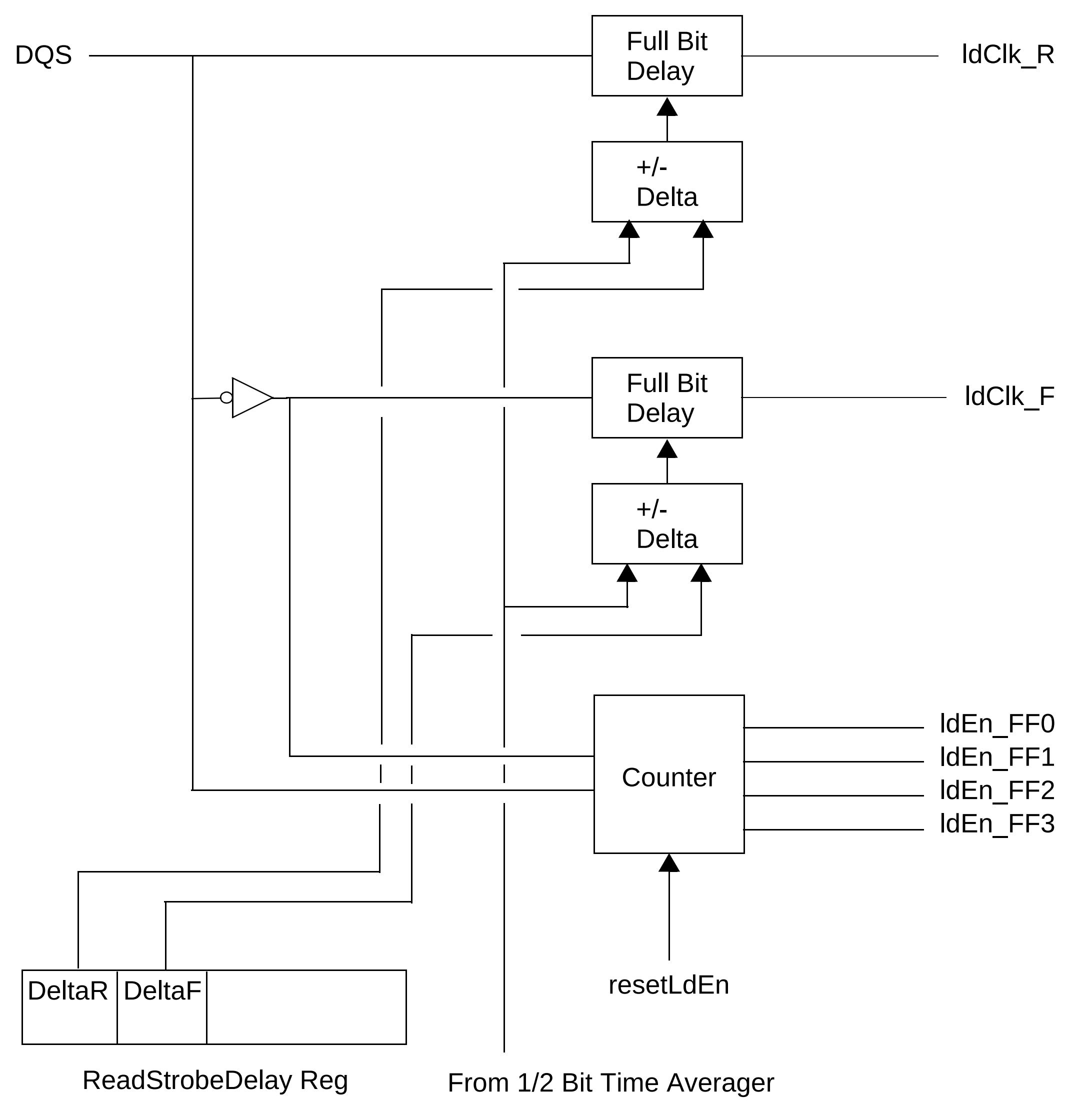


The clocks to each FF are gated with the signals ldEn_FF0, ldEn_FF1, ldEn_FF2, ldEn_FF3. A 2-bit counter driven off the rising and falling edges of DQS is used to successively generate these four load enable FF signals. At the beginning of a read an internal signal, resetLdEn, is used to reset this counter (*Figure 7-15*) and enable it for counting. The timing of this signal is critical: it is set with both a coarse control and a vernier.

To center the incoming DQS within the data window, the DQS signal is delayed by 1/2 bit time before being used as a clock to the FIFO bits. (The undelayed DQS is used to clock the 2-bit load enable counter.) The 1/2 bit time delay is generated using the same mechanism used to generate 1/2 bit time for the write DQS: the output of the 1/2 Bit Time Averager value is added to a Delta time that adjusts the 1/2 bit time + or -, and the result is used to control a full bit time delay unit.

A positive version of the incoming DQS is delayed with a 1/2 bit time unit and sent to FF0 and FF2 for clocking on the rising edges. A negative version of the incoming DQS is delayed with a 1/2 bit time unit and sent to FF1 and FF3 for clocking on the negative edges. The two 1/2 bit time units are independent; the Read Data Strobe registers have 2 fields, RdStrOffset_DeltaR and RdStrOffset_DeltaF to provide independent offsets to the two 1/2 bit time delay units. See *Figure 7-15*.

Figure 7-15. Read DQS



When a memory chip performs a read operation, initially DQS is in tri-state. The SDRAM drives DQS low: this is the preamble. For the burst length of 4 used by the CPC945, the SDRAM then drives DQS high, low, high, low. This last low level is the postamble. Finally, DQS is put back in tri-state.

The transitions from and to tri-state can have undershoot and overshoot such that, internal to the CPC945, it looks like the DQS has extra transitions. It is important that the CPC945 only latch data on the 4 valid edges. As mentioned previously, an internal “resetLdEn” signal is used to reset the load enable circuits and prepare them for use of the following DQS edges. After 4 edges are detected, a “glitch filter” prevents the transition from the low level postamble to tri-state from accidently being used to clock the data capture latches. This glitch filter can be disabled with bit 20, StartLdEn, in the MemBusConfig Register. However, it is recommended that the glitch filter always be enabled by setting StartLdEn to 0.

Once the data are latched in the FIFO, they must be transferred to the internal ddr_clk to be pipelined through the data path and sent to the requesting unit. A multiplexer (*Figure 7-14*) is used to select one of the four latches outputs to be clocked by ddr_clk. A counter drives a 2-bit unLoadPtr signal to successively transfer the four latch outputs to the ddr_clk domain. An internal signal resets this pointer during a read, roughly at the time the first beat of data has become stable in its FF. A coarse control sets the timing of this signal.

Each byte lane has a DQS driver/receiver. When ×8 or ×16 memory chips are used the receiver receives a single differential DQS. When ×4 memory chips are used the receiver receives two independent single-ended DQS.

Figure 7-16 shows the read clocking control for a byte lane. To accommodate the ×4 mode of operation for reads, a byte lane operates on two nibbles. The 8 FIFOs are clustered into 2 nibbles of 4 bits each. The two received DQS (using the ZPD and ZPN outputs of the receiver) are labeled internally DQS_L and DQS_U. DQS_L has positive and negative copies (*Figure 7-15*, *Figure 7-16*) that feed a load enable counter which in turn feeds 4 FIFOs. The positive and negative versions of DQS_L are each fed through a 1/2 bit time delay each (*Figure 7-15*, *Figure 7-16*); the two delayed strobe versions are used to feed/clock the 4 FIFOs. In a similar fashion, DQS_U drives a load enable counter and two 1/2 bit time delays to clock the 2nd nibble of 4 FIFOs. So altogether a byte lane has four 1/2 bit time delay units, and four offset controls. The four controls in the Read Data Strobe Register are RdStrOffset_DeltaLR and RdStrOffset_DeltaLF, for tweaking independently the rising edge and falling edge delays to one nibble, and RdStrOffset_DeltaUR and RdStrOffset_DeltaUF, for tweaking independently the rising edge and falling edge delays to the other nibble.

When a single differential DQS is received for ×8 or ×16 chips, the received DQS (using the ZDF output of the receiver) is steered through multiplexes to feed the 2 sets of clock and control for the 2 nibbles. ***Therefore all 4 values of RdStrOffset must be programmed***. (Usually the 2 nibbles are given the same delays.)

For the resetLdEn fine tuning, there are 8 verniers; 2 per cluster. A vernier controls resetLdEn for either 2 byte lanes or 3 byte lanes. (The vernier for data bytes 4 and 5 also controls check byte 16; the vernier for data bytes 12 and 13 also controls check byte 17; the remaining verniers control 2 bytes each.)

The verniers and the ResetLdEnVernier Registers are not shown. They are full bit time delay verniers made up of 2 half bit time delays, similar to dual-stack full time delay vernier controlled by WrClkDelay in the write controls.

The unLoadPtr control operates in common to all bits of all byte lanes.

7.28.2 Read ResetLdEn Timing

Figure 7-17 shows the relationship of the internal resetLdEn signal to the internal DQS signal. The internal signal ldClk_LR, is the positive version of one of the incoming DQS signals after it has been delayed 1/2 bit time and fine tuned with the Read Data Strobe Control registers. In this example the CAS latency = 4.

The timing of the internal resetLdEn is critical: the trailing edge of resetLdEn must fall within the (internal, delayed) DQS Read Preamble driven by the memory chips at the start of the read data transfer. Too late, and the first beats of the data transfer will not be clocked into the FIFO. Too early, the last beats of a previous read (back-to-back) will not be clocked into the FIFO.

Due to pipeline delays the internal resetLdEn signal arrives 3 cycles after the control signals are sent off-chip, plus the value of ResMuxDel in the MemBusConfig register. That is, CAS, WE going valid to internal resetLdEn going valid = ResMuxDel + 3 (in ddr_clks).

The example timings in *Figure 7-17* shows the internal resetLdEn using a ResMuxDel value of 4.

Figure 7-16. Byte Lane Read Control

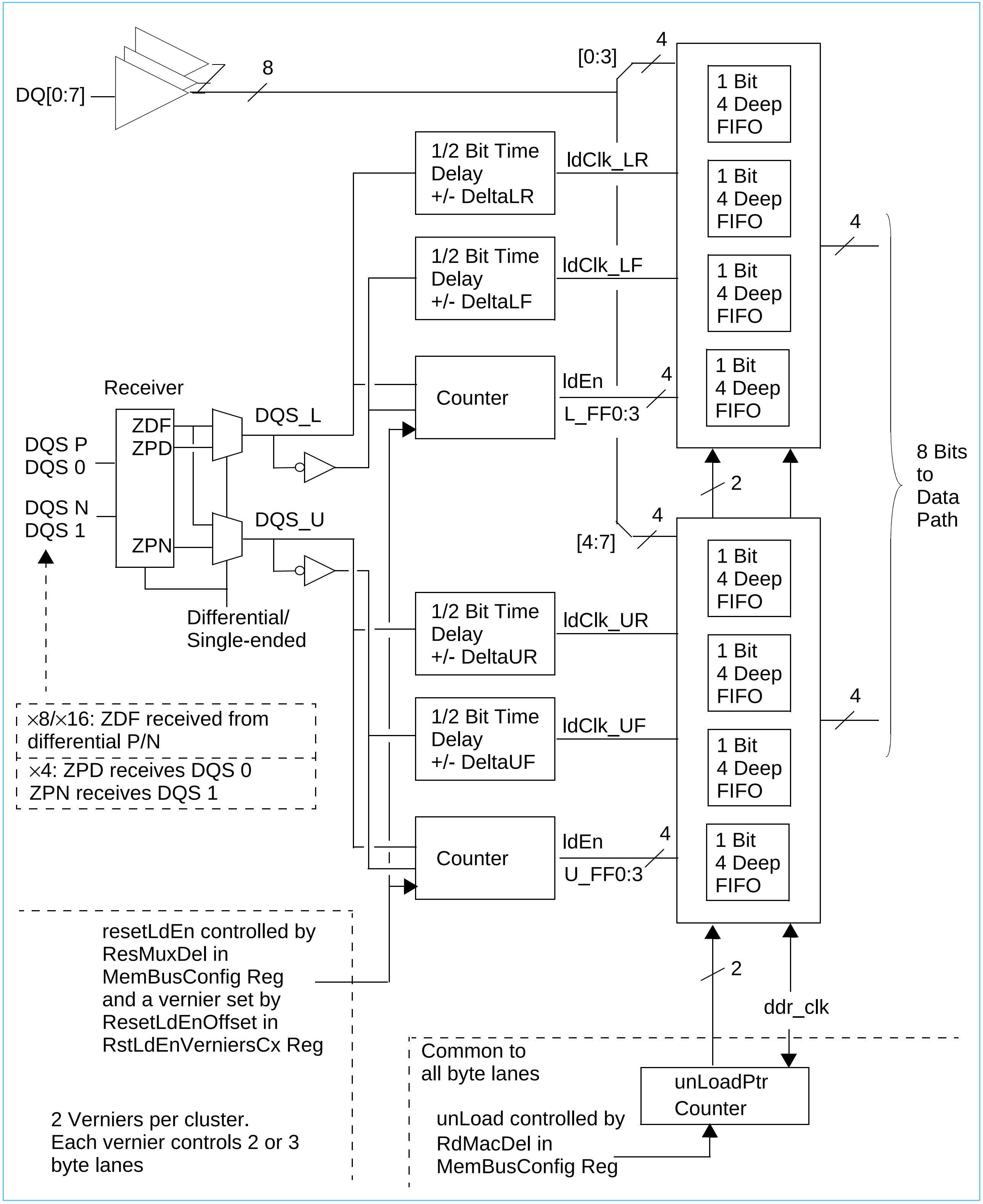
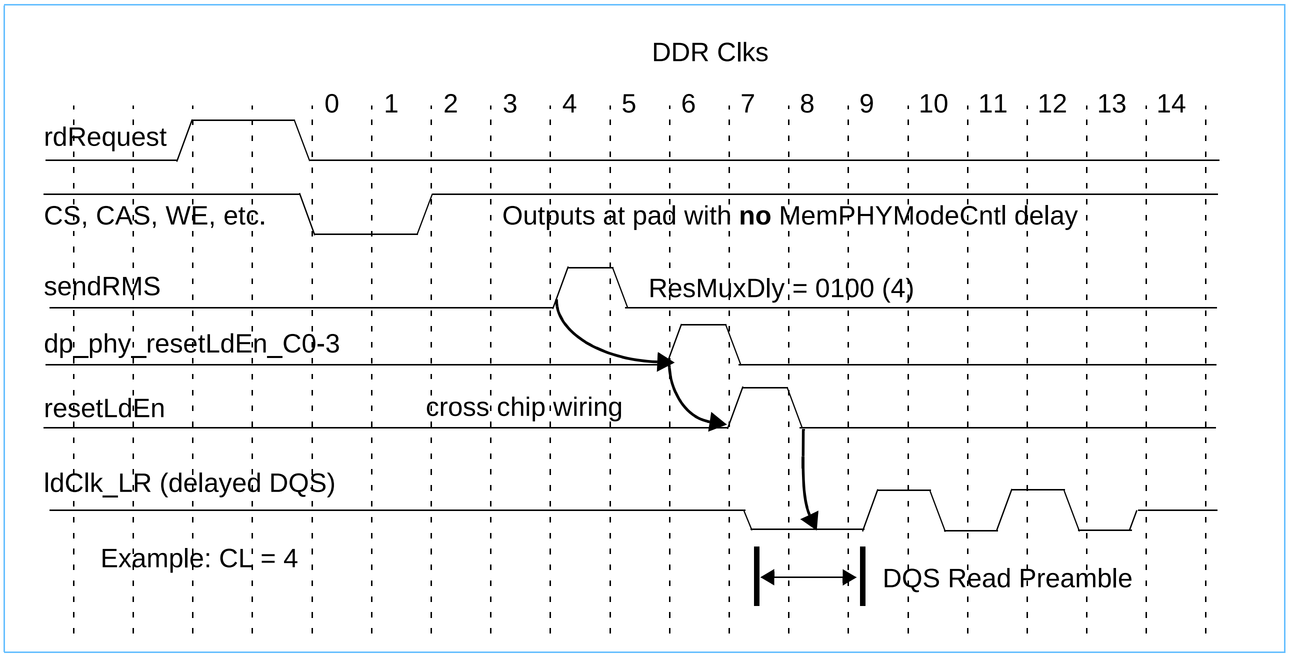

DQ[0:7]
8
[0:3]
4
1 Bit 4 Deep FIFO
1 Bit 4 Deep FIFO
1 Bit 4 Deep FIFO
1 Bit 4 Deep FIFO
1/2 Bit Time Delay +/- DeltaLR
ldClk_LR
1/2 Bit Time Delay +/- DeltaLF
ldClk_LF
4
Receiver
Counter
ldEn
4
L_FF0:3
ZDF
ZPD
ZPN
DQS_L
DQS P
DQS 0
DQS N
DQS 1
DQS_U
2
8 Bits to Data Path
[4:7]
4
Differential/ Single-ended
1 Bit 4 Deep FIFO
1 Bit 4 Deep FIFO
1 Bit 4 Deep FIFO
1 Bit 4 Deep FIFO
1/2 Bit Time Delay +/- DeltaUR
ldClk_UR
1/2 Bit Time Delay +/- DeltaUF
ldClk_UF
4
×8/×16: ZDF received from differential P/N
×4: ZPD receives DQS 0
ZPN receives DQS 1
Counter
ldEn
4
U_FF0:3
resetLdEn controlled by ResMuxDel in MemBusConfig Reg and a vernier set by ResetLdEnOffset in RstLdEnVerniersCx Reg
2
ddr_clk
Common to all byte lanes
unLoadPtr Counter
unLoad controlled by RdMacDel in MemBusConfig Reg
2 Verniers per cluster. Each vernier controls 2 or 3 byte lanes

Figure 7-17. Read Reset Ld En Timing Example



7.28.3 Read Unload Timing

Once data is loaded into the FIFO, it must be transferred to the internal data path that is clocked using the internal ddr_clk. An internal “unload pointer” is incremented to successively pull read data out of the FIFO. The timing of this unload operation is controlled by the RdMacDel (Read Macro Delay) field in the MemBusConfig register.

Figure 7-18 shows the unloading of the FIFO. Because of pipeline delays it takes 4 ddr_clks to start the unload operation, plus the value of RdMacDel. That is, CAS, WE going valid to internal the unload pointer being updated = RdMacDel + 4 (in ddr_clks).

The example timings in *Figure 7-18* shows the internal unloadPtr using CL = 4 and a RdMacDel value of 5.

In theory if all bytes of data are simultaneously loaded into the FIFO (internally all DQ and DQS are perfectly aligned), then read data will be valid in each FIFO location for approximately 3 ddr_clk cycles, which would allow for 3 values of RdMacDel which could be used to start the unload operation. In practice often only 1 or 2 values are seen to work. If multiple values are found to work, the smaller value should be used to minimize the read latency.

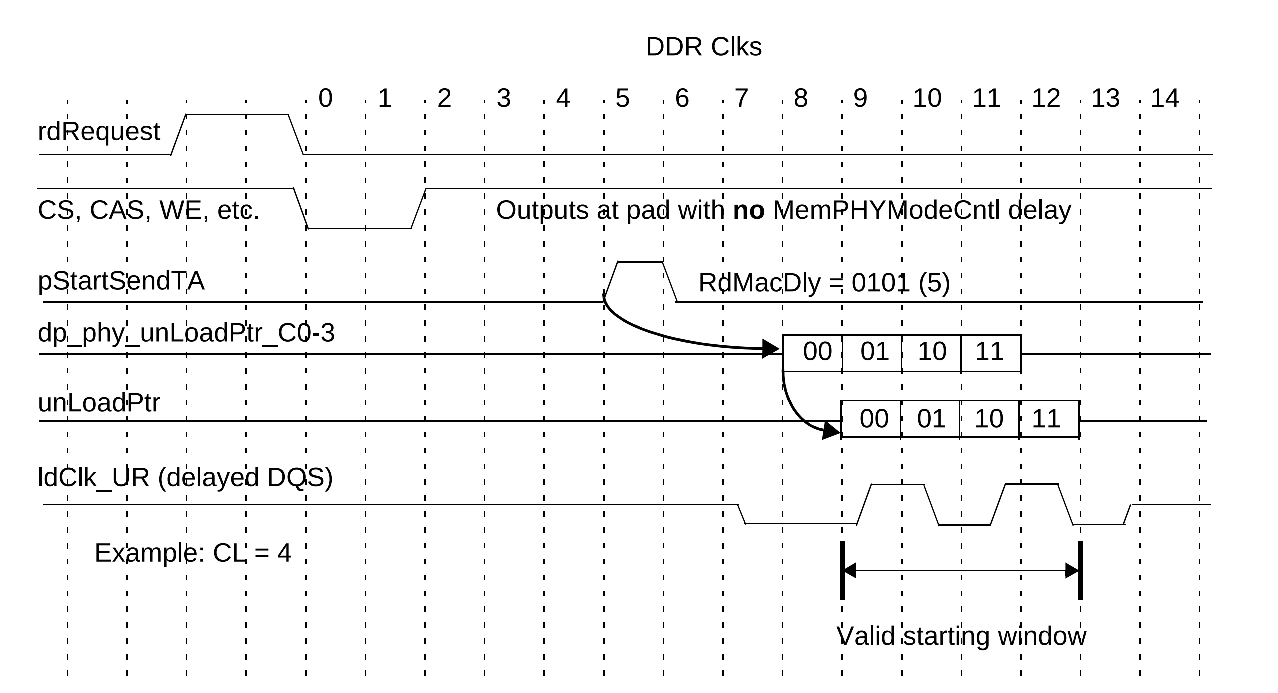
Since the timing of RdMacDel is closely related to the setting of ResMuxDel, the value of RdMacDel is often in the range of ResMuxDel+1 to ResMuxDel+4.

Summary: Read data from the memory chips arrives at an internal FIFO which is clocked on a byte or nibble basis by DQS from the memory chips. Internally, the individual DQS signals are delayed nominally by 1/2 bit time to center the DQS edges within the data. Due to board wiring variations and on-chip variations the various bytes or nibbles can arrive at the FIFO latches at slightly varying times. The rising and falling edges of the delayed DQS arriving at the FIFO clocking logic can be fine tuned +/- using the RdStrOffset_Delta fields of the Read Data Strobe registers.

The FIFO has a load pointer and an unload pointer. The load pointer must be reset during the DQS read preamble time; the coarse timing is set using the ResMuxDel field in the MemBusConfig register and byte pairs timings are set using the Reset LdEn Offset fields of the Reset Ld En Offset Delay registers. The unload timing is set using the RdMacDel field in the MemBusConfig register.

Although the DQS edges across the bytes or nibbles are adjusted to center the DQS edges within the DQ beats at the input to the internal FIFO, there cannot be too great a range from the fastest arriving data to the slowest arriving data. If this range is too great, it might be impossible to find a value of ResMuxDel that reliably resets the load pointer for all bytes or nibbles. To a lesser degree this might also make it impossible to find a value of RdMacDel that reliably unloads all bytes or nibbles. For this reason, at the board level the skew of the data bytes must be tightly controlled.

Figure 7-18. Read UnloadPtr Timing Example



7.29 Output Enable Timing

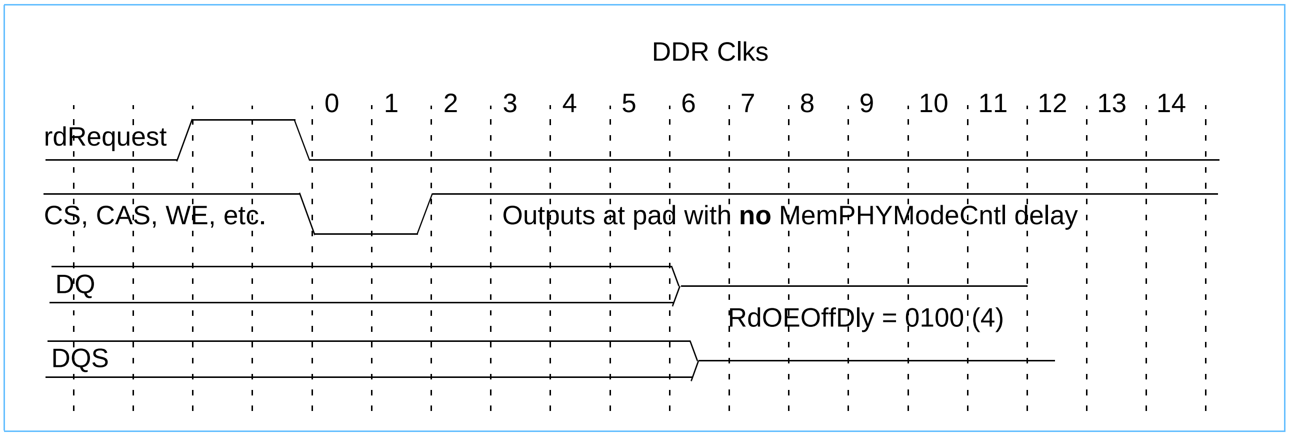
Normally the CPC945 enables the DQ and DQS I/O for driving to the memories, in anticipation of write operations occurring. When a read operation is performed these I/Os are tri-stated at the beginning of the read, and re-enabled for driving at the end of the read.

7.29.1 OE Coarse Timing

In the MemBusConfig register the RdOEOffDly field controls when the I/O are tri-stated, and the RdOEOnDly field controls when the I/O are re-enabled.

For read operations the time from CAS, WE going valid to DQ being tri-stated = RdOEOffDly + 2 (in ddr_clks). DQS is tri-stated approximately 1/2 bit time later. An example showing RdOEOffDly = 4 is shown in *Figure 7-19*.

Figure 7-19. RdOEOffDly Timing Example



Once tri-stated, the DQ and DQS I/O will be held tri-stated for a minimum of 3 + n × ddr_clks, where “n” = the number of data beats:

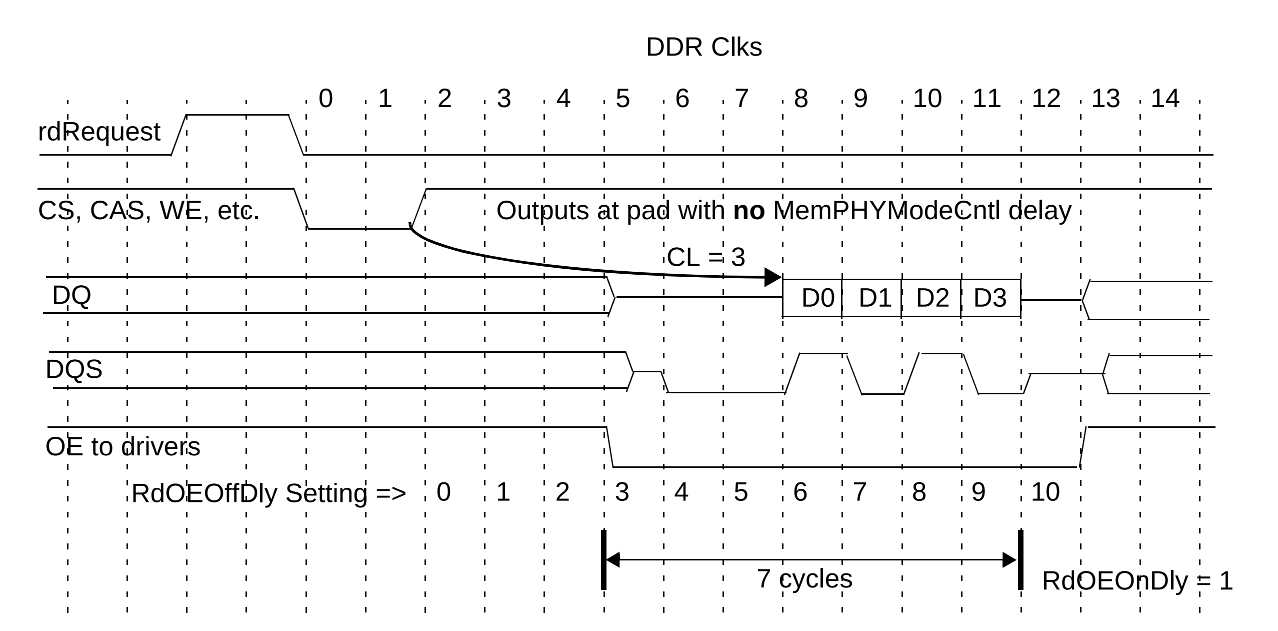
- n = 4 for 64 bytes or less with a 128-bit bus configuration
- n = 8 for 128 bytes with a 128-bit bus configuration, or 64 bytes or less with a 64-bit bus configuration
- n = 16 for 128 bytes with a 64-bit bus configuration

This minimum tri-state time can be extended using the RdOEOnDly value. Thus the time that the DQ and DQS I/O are held in tri-state = 3 + n + RdOEOnDly.

Note that the OE Off window can be widened on both ends by decrementing the RdOEOffDly value and incrementing the RdOEOnDly value. For example, the window can be widened by one ddr_clk at each end by decrementing RdOEOnDly by 1, and incrementing RdOEOffDly by 2.

*Figure 7-20*s shows an example for a 128-bit bus, 4-beat transfer, CL = 3 system in which RdOEOffDly is set to 3. RdOEOnDly is set to 1, to widen the window to 8 ddr_clks.

Figure 7-20. DQ and DQS Tri-state Timing Example



7.29.2 OE Vernier Timing

The coarse timing provided by the RdOEOffDelay and RdOEOnDelay values generates an internal Output Enable (OE) signal. This internal signal is further delayed using the same Write Data vernier control, WrClkOffset, discussed in *Section 7.27.2*.

In *Figure 7-20* the signal labeled “OE to drivers” is sent to all byte lanes. It is shown as the input “OE” in *Figure 7-13*. The same delay applied to DQ using WrClkOffset is also applied to the Output Enable for the DQ drivers. For the DQS drivers, the Output Enable to the drivers has the same 1/2 bit time delay as is applied to DQS.

7.30 External Multiplexer Timing

7.30.1 ExtMux Coarse Timing

The DDR_MUXEN[0:7] chip outputs control the optional external data multiplexers.

For write operations the multiplexer value switches once, at a time controlled by the WrExtMuxDly field in the MemBusConfig register.

For read operations the multiplexer value switches once, at a time controlled by the RdExtMuxDly field in the MemBusConfig register.

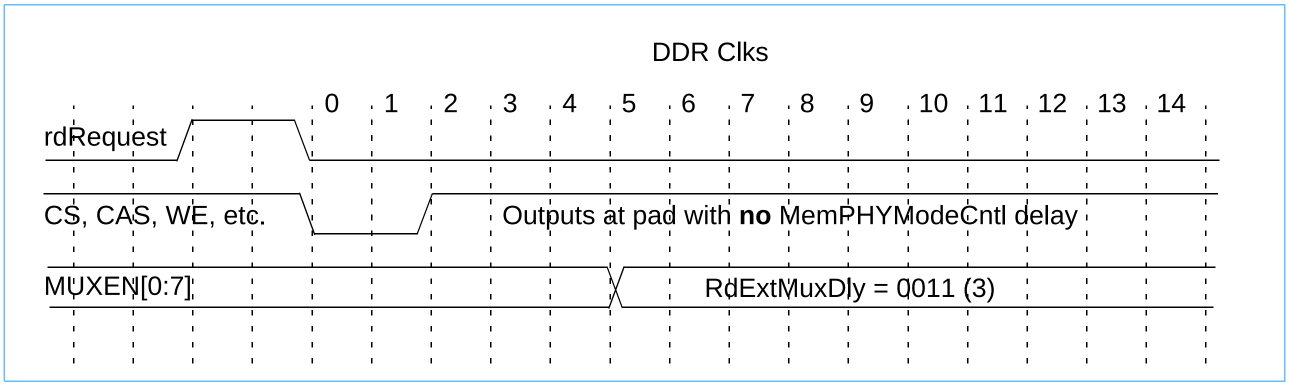
In essence there is one 2-bit control to select 1-of-4 multiplexer paths. There are 4 copies of these 2 bits: DDR_MUXEN[0:1], DDR_MUXEN[2:3], DDR_MUXEN[4:5] and DDR_MUXEN[6:7].

For write operations the time from CAS, WE going valid to MUXEN switching = WrExtMuxDly + 2 (in ddr_clks).

For read operations the time from CAS, WE going valid to MUXEN switching = RdExtMuxDly + 2 (in ddr_clks).

Figure 7-21 shows an example in which RdExtMuxDly = 3.

Figure 7-21. External multiplexer Basic Timing



The critical timing of the external data multiplexer switching usually occurs when the memory controller accesses two different DIMMs in quick succession. In general, when the CPC945 changes the DDR_MUXEN[0:7] outputs there must be one dead cycle (one mem_clk = 2 ddr_clks) to allow time for the multiplexers to switch.

There are 4 scenarios that must be analyzed: a read followed by a read, a write followed by a read, a write followed by a write, and a read followed by a write. The TiRtRSy, TiWtRSy, TiWtWSy and TiRtWSy fields, in the CASTimer0 and CASTimer1 Registers (0xF8002050 and 0xF8002060) must be programmed to allow enough time between the two operations to meet DDR2 chip specifications, plus an extra cycle for the external multiplexer switching.

The WrExtMuxDly/RdExtMuxDly fields in the MemBusConfig register and the ExtDataMuxSelWrOffset/ExtDataMuxSelRdOffset fields in the ExtMuxVernier registers are then used to dial in a switching time between the two operations such that the data transfer of both the first and the second operation are not clipped by the multiplexer switching.

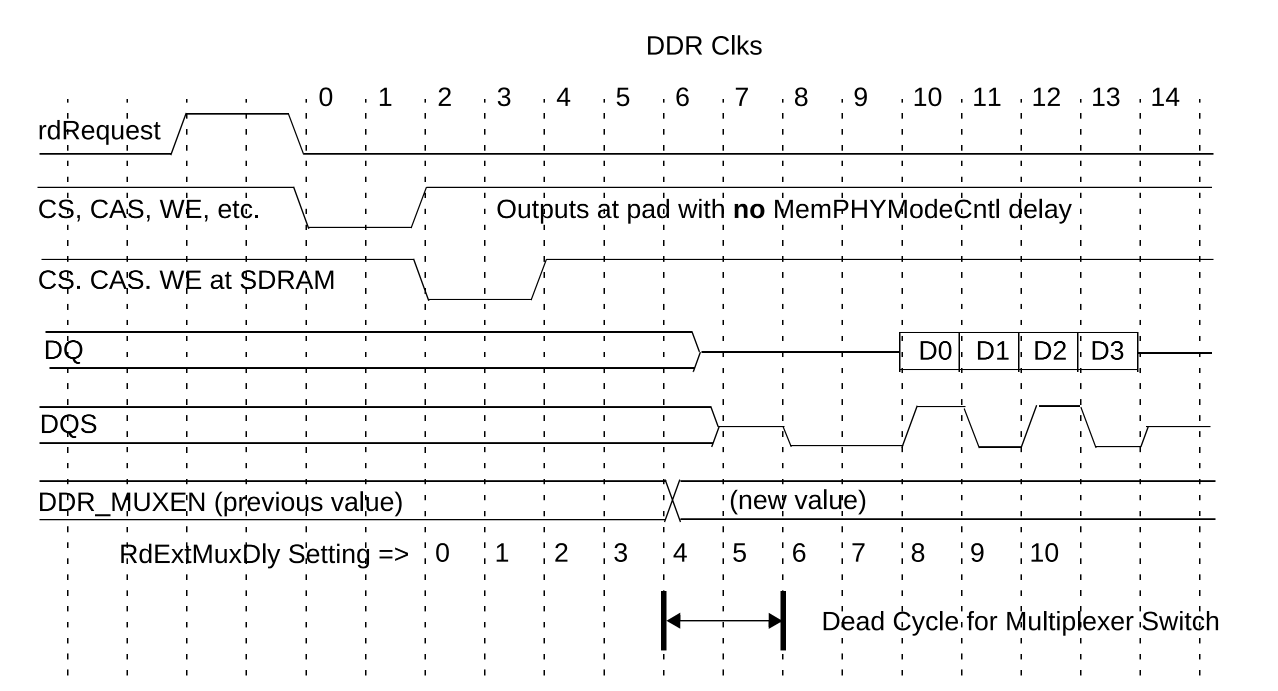
An initial setting of WrExtMuxDly and RdExtMuxDly values would place the multiplexer switching time and its dead cycle just before the activation of DQS. This cycle is a function of when CAS arrives at the memory chips. An example equation for RdExtMuxDly (assuming no significant board wiring delay) would be RdExtMuxDly = 2 * (CL + RegDIMM -2), where CL = CAS latency, RegDIMM = 1 if registered DIMMs are used, otherwise 0, and the “-2” represents backing off one cycle for the DQS preamble and one dead cycle for the multiplexer switching. The number of cycles is multiplied by 2 because RdExtMuxDly is programmed in ddr_clks.

For writes, DQS is activated one mem_clk cycle earlier than for reads, so WrExtMuxDly = RdExtMuxDly - 2 = 2 * (CL + RegDIMM -3).

(Properly speaking the above equations should have AL = Additive Latency as an additional term inside the parentheses, but the CPC945 requires that AL = 0 so this term has been left out.)

Figure 7-22 shows an example read operation in which CL = 3, registered DIMMs are used, and RdExtMuxDly = 4 based on the above equations.

Figure 7-22. External multiplexer Read Timing Example



7.30.2 ExtMux Vernier Timing

The coarse timing provided by the WrExtMuxDly and RdExtMuxDly values generates an internal 2-bit MUXEN signal. This 2-bit signal is sent to each of the four clusters. Each cluster has a vernier for adding additional delay for reads and another vernier for adding delay for writes.

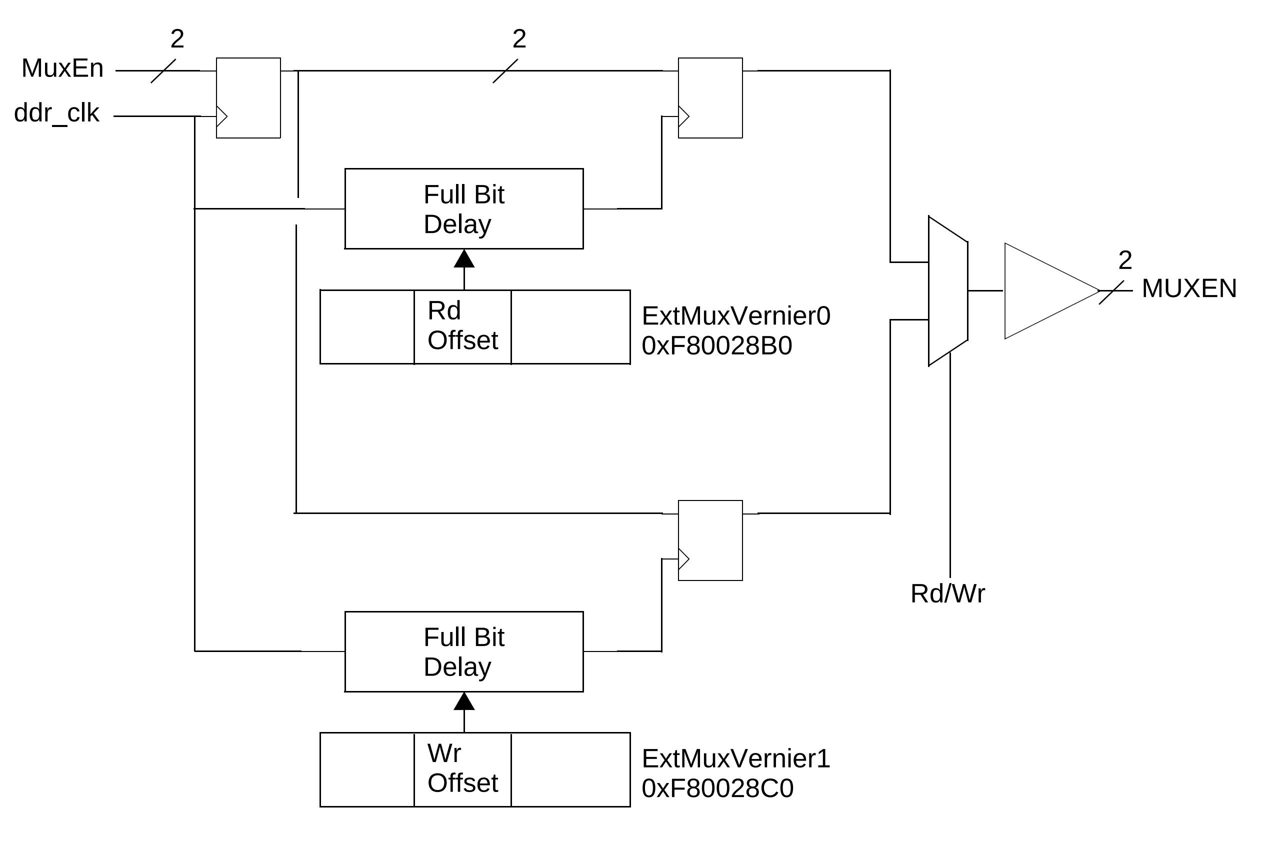
The 2 delayed values of the multiplexer enabled signal are selected by an internal multiplexer, read versus write, and driven off chip. Cluster 0 generates DDR_MUXEN[0:1], Cluster 1 generates DDR_MUXEN[2:3], Cluster 2 generates DDR_MUXEN[4:5], Cluster 3 generates DDR_MUXEN[6:7].

Figure 7-23 gives a simplified diagram of the external multiplexer enable verniers for a cluster.

The ExtMuxVernier0 register at 0xF80028B0 provides 4 RdOffset fields, one per cluster.

The ExtMuxVernier1 register at 0xF80028C0 provides 4 WrOffset fields, one per cluster.

Figure 7-23. Cluster MuxEn Verniers



7.31 DDR2 PHY Calibration Logic

7.31.1 Calibration Logic Overview

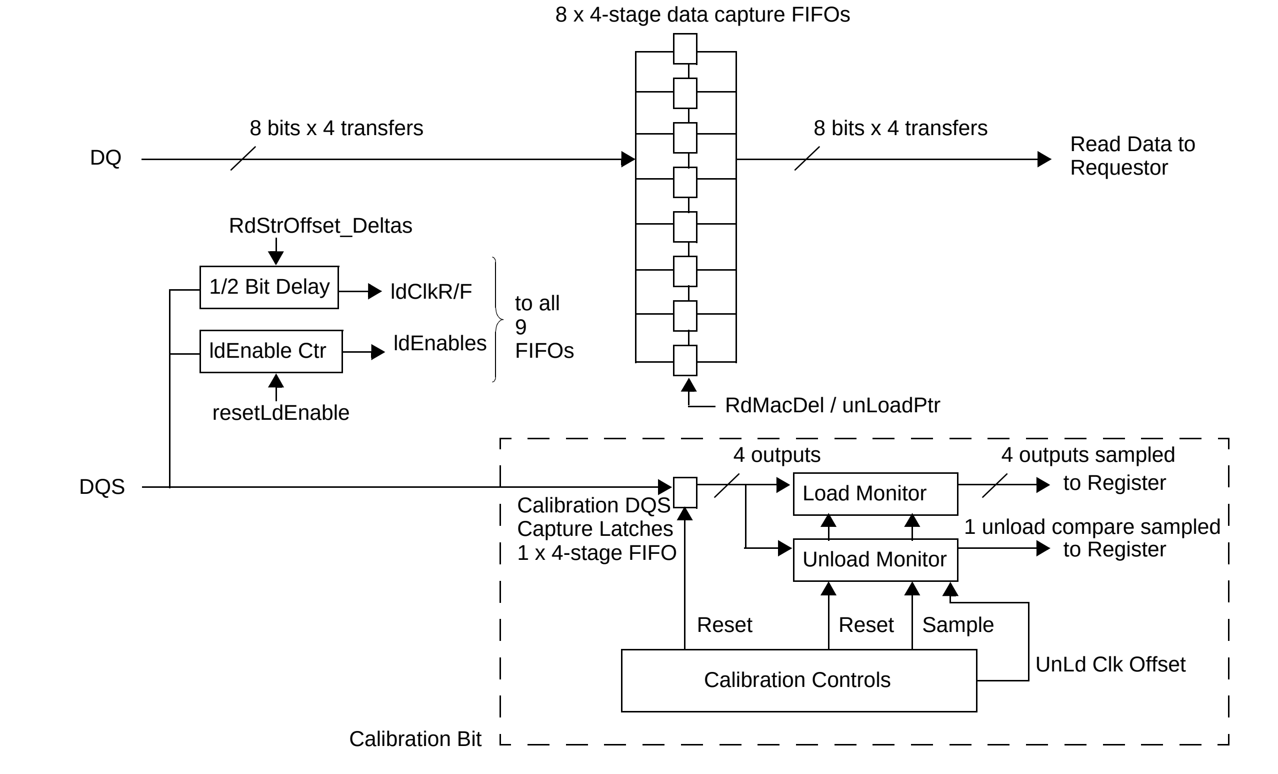
The DDR2 PHY has calibration logic that can be used to discover good timing control values for DDR2 read operations. ***The use of this logic is optional.*** It can be used as an aid to finding timing control values, but it is not a requirement. It is not used for normal functional operation of the DDR2 memory controller.

Timings for the clock and control signals to memory, and data and strobe signals on write operations, can be observed at the board or DIMM level using equipment such as an oscilloscope or logic analyzer. Read operations are different. Although the data and strobes coming out of the DIMMs can be observed, the desired point of observation is at the data capture latches in the CPC945.

The DDR2 PHY calibration logic adds some visibility into the internal control and clocking of the data capture latches.

For DDR2 reads the DDR2 DRAMs drive DQ and DQS into the CPC945. Internally the arriving DQS are delayed by 1/2-bit time and used to clock the DQ into the capture latches. Properly timed control signals enable the load of the capture latches and the unload to the internal ddr_clk.

Figure 7-24. Simplified Byte lane with Calibration Bit



Each bytelane has 8 sets of data capture latches. The calibration logic adds a ninth calibration bit. The calibration bit contains a set of DQS capture latches, a load monitor, and an unload monitor.

- The DQS capture latches are similar to the DQ capture latches except that the incoming, undelayed DQS are latched instead of the DQ. That is, the delayed strobe is used to clock in the undelayed strobe.

  Depending on values used to control the 1/2-bit time on the rising edge and falling edge copies of the delayed strobe, and the values used to control the reset of the load enable circuitry, the capture of the undelayed strobe by the delayed strobe will be either successful (pass) or unsuccessful (fail).
- The load monitor reports the value of the capture latches.

  The calibration logic has the concept of *streaming* a set of reads into the capture latches, and accumulating the result in the load monitor using “sticky” latches. At the beginning of the first read, the load monitor is reset to a “pass” result. The load monitor then stays at the “pass” state only if all reads in the stream generate a “pass” result. If any of the reads generates a “fail” result, the load monitor goes to the “fail” level and stays there.
- The unload monitor indicates the arrival timing of a delayed strobe edge at a capture latch relative to the internal ddr_clk. A pass/fail bit is generated which indicates if the latching of a capture latch by the ddr_clk matches the latching by a delayed version of ddr_clk. This delay is varied by the user.

  Similar to that of the load monitor, the unload monitor has a sticky latch to accumulate the result over a stream of reads.

The use of read streams and sticky result latches is to accommodate jitter. This is discussed more fully in *Section 7.31.6 Jitter Considerations*.

The calibration logic makes use of the MemInitRegs and the Memory Programming Control mechanism described in *Section 7.9 Memory Programming Control* on page 147. This mechanism allows the user to set up and issue and arbitrary set of memory commands, with user defined delays between commands, and looping controls to generate very long sequences of commands if desired. When using the DDR2 PHY calibration logic, the user is required to set up one or more read commands in the MemInitRegs.

For a stream of reads, the calibration logic is set up to run in pulsed or continuous mode.

- In pulsed mode the sticky latches in the load and unload monitors are opened once, for a short sampling time (one ddr_clk), per read. In pulsed mode the user is required to space the reads apart. (By using the Delay fields in the MemInitRegs.)
- In continuous mode, the sticky latches are opened for sampling in the first read and held open. Typically the user will program no delay between reads.

The discovery of good timing control values usually consists of executing a series of calibration operations. A calibration operation, or step, involves resetting the calibration capture latches and monitors, kicking off a stream of reads, observing that the reads have ended, and reading the monitor results. A given timing control is varied between calibration steps. The series of steps is terminated when the monitor results change from pass to fail.

The timing control that is varied is usually one of the following verniers: reset load enable, rising read strobe offset delta, falling read strobe offset delta, unload monitor clock offset. Also, for reset load enable timing, the ResMuxDelay field in the Memory Bus Configuration can be varied.

The calibration logic supports single step mode and auto calibration mode.

- Single step mode is manual. The programmer writes registers to reset the calibration logic, writes the Memory Programming Control Register to kick off the reads, polls the Memory Programming Control Register to determine when the reads are done, and reads the monitor results. Between steps, the programmer writes a new value to the timing control of interest.

  For a given step the monitor results for all byte lanes are available.
- Auto calibration makes use of a Finite State Machine (FSM) to automatically sequence through a number of calibration steps. The programmer writes a register to kick off the sequence and polls a status bit to determine when the sequence has terminated. The FSM performs a series of calibration steps, incrementing a timing control value between each step. For a given step, the FSM resets the capture latches and monitors, kicks off the reads, polls for the end of reads, reads the monitor results, and compares the monitor results against a programmed value. Based on the compared results, either another step is executed or the series is terminated.

  Prior to the auto calibration, the programmer identifies which timing control value is to be incremented between steps. Following the auto calibration, the programmer reads a register to find the incremented value which caused a calibration step to terminate the series. Only the verniers listed above (reset load enable, rising/falling read strobe delta offset, unload monitor clock offset) can be varied by the auto calibration FSM.

  There are actually 4 FSM machines, one per cluster. It is intended that all 4 FSMs are operated in parallel. Of the 4 or 5 bytes in a cluster a FSM can only work with a single byte at a time. Since there are 4 FSMs a given auto calibration sequence will find the results for 4 bytes at a time, one per cluster.

  For the 2 ECC bytes only 2 FSMs are run in parallel.

The calibration logic is configured with 2 registers which are common to all byte lanes.

- CalConfig0 (0xF80029B0) has some controls which must be set for any type of calibration. Some fields are only applicable for the unload monitor. Some fields are only for auto calibration.
- CalConfig1 (0xF80029C0) is only used with the unload monitor. It contains the control value for the vernier which delays the internal ddr_clk (unload clock offset).

There are 4 Control and Delay Measurement registers, one per cluster, located at 0xF80028F0, 0xF80029F0, 0xF8002AF0 and 0xF8002BF0.

In single step mode this Control register contains a bit used to reset the monitors and control logic.

In autocal mode this Control register contains a bit used to kick off an autocal sequence, the bit that is polled to determine when the sequence has completed, and the timing control value (that had been auto-incremented between steps) used for the final step (the “Delay Measurement”).

(The register at 0xF80028F0 also contains the measurement delay produced by the 1/2-bit time averager, which is not part of the calibration logic.)

There are 4 Results registers, one per cluster, located at 0xF80028E0, 0xF80028F0, 0xF8002AE0 and 0xF8002BE0. These registers contain the outputs of the load and unload monitors.

Clusters 0 and 2 contain 4 bytes. There are 4 load monitor result bits per byte, for a total of 16 bits: 0 through 15. Bit 0:3 contain the 4 result bits of the first byte, bits 4:7 contain the 4 result bits of the next byte, and so on.

When ECC is used clusters 1 and 3 contain 1 additional byte each for the check bits. The additional 4 load monitor result bits are contained in bits 16:19.

The unload monitor has a single result bit. For clusters 0 and 2 with 4 bytes each, the 4 unload monitor result bits are bits 24:27 in the Results register. For clusters 1 and 3 with 5 bytes each the 5 unload monitor result bits are bits 24:28.

(The register at 0xF80028E0 bits 29:31 contain the 3 status bits of the 1/2-bit time averager, which is not part of the calibration logic.)

The DDR2 PHY calibration logic has the following characteristics:

1. The calibration logic does not produce *optimum* timing values. Rather, the calibration logic finds the margins of good versus bad operation. Once these margins are discovered, the optimum timing is calculated or inferred.
2. There is a hierarchy of the direct relevance of what the calibration logic is measuring versus the timing controls of interest.
   a. The Reset Load Enable (resetLdEn) timing observed by the calibration logic is exactly the timing applied to the data capture latches. The resetLdEn timings affect the operation of the load enables controlling all of the capture latches in a byte lane: the 8 sets of DQ capture latches and the 9th set of capture latches for the calibration bit.
   b. The rising and falling Read Strobe Delta Offset timing observed by the calibration logic is not exactly the timing of interest at the data capture latches. The calibration logic is observing the latching of the undelayed strobe by the delayed strobe, whereas the timing of interest is the latching of the data (DQ) by the delayed strobe.

      For a given bytelane, there is an assumption that the timing of DQ and DQS is so closely matched that the results observed by latching the undelayed DQS will also apply to the latching of DQ. This is another reason that the lengths of the board wiring should closely match the DQ versus DQS.
   c. The unload monitor might have the least utility for the user, for 2 reasons.

      First, note that for the load operation the controls that are varied when making observations with the calibration logic (ResMuxDel, ResetLdEn Offset verniers, rising and falling Read Strobe Delta Offset verniers) are identically the controls affecting the load operation of the data. But for the unload operation the control of interest (RdMacDel) is not observed by the calibration logic. Instead, the unload monitor has a vernier which is only used by the unload monitor, but does not affect the unload operation of the data sent to the requestor.

      Second, while the unload monitor can give the user a relative feel for the relationship of the DQS edges versus the internal ddr_clk, the precise relationship can only be determined with detailed calculations involving the exact implementation of the calibration logic and knowledge of the logic delays (for example, clock splitter delays, overhead delays in the vernier delay chain, and so on.).

      Since this knowledge is not available to the user, it is difficult or impossible to determine the timing value of interest (RdMacDel) from the observations obtained with the calibration logic unload monitor. Indeed this User Manual does not provide such a calculation.

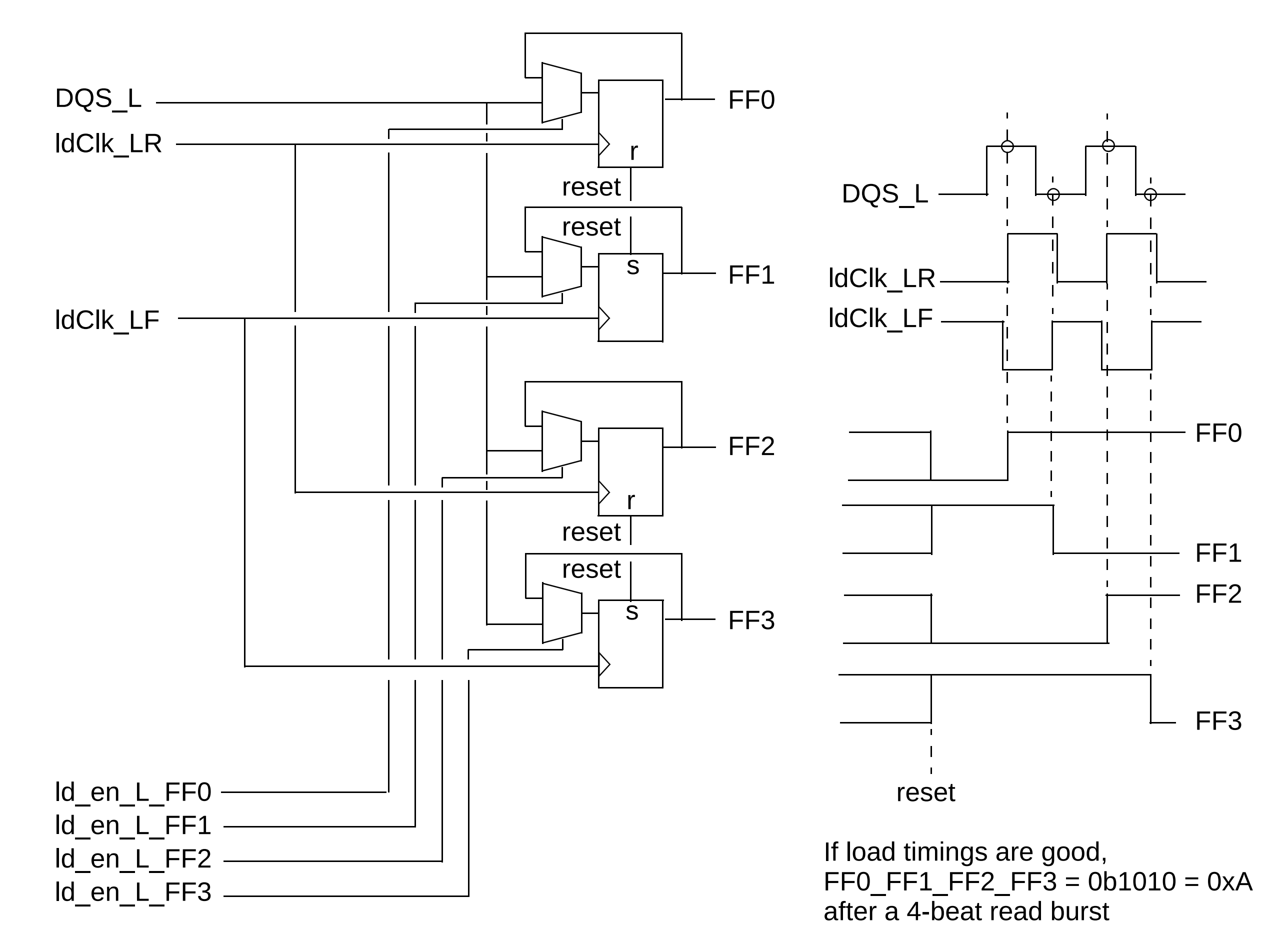
      The unload monitor operation is described for completeness. Also, it might be possible to develop an “ad hoc” use of the unload monitor based on laboratory correlation of the unload monitor results versus the general read results produced as RdMacDel is varied. Also, in addition to relating the strobes to the internal ddr_clk, the unload monitor can be used to observe the amount of jitter on DQS.

7.31.2 Calibration Bit DQS Capture Latches

Figure 7-25 shows the 4 capture latches in the calibration bit. This is essentially the same 4-stage FIFO as the normal read data capture FIFO (*Figure 7-14* on page 233), with the following differences:

- As already described the strobe (DQS), rather than a data bit (DQ) is the data that is clocked into the 4 flops.
- The FIFO does not have an output multiplexer/unload pointer. (The unload monitor, discussed later, has a multiplexer but the output is statically selected and is not controlled with RdMacDel.)
- The 4 flops have a “reset” input which forces their values to be 0b0101.

Figure 7-25. Calibration Bit DQS Capture FIFO



As shown in *Figure 7-25*, if a read operation is performed in which DQS goes high-low-high-low, the delayed strobe should clock in the values 0b1010 = 0xA. If the ldClk or ld_enable timings are not correct it is possible that one or more of the flops is not clocked at all. To guarantee that a successful clocking is detected (not data left over from a previous read), the 4 flops are first reset to the opposite values of a successful load, that is, 0b0101 = 0x5.

By default the reset of the flops is timed to occur at the same time as the ResetLdEnable. The timing can be adjusted by ddr_clks using the Measurement Reset Latency field (bits 16:19) in the CalConf0 register at 0xF80029B0. This is a sign/magnitude value so the reset can be adjusted to occur earlier (-) or later (+).

As noted in *Section 7.31.1 Calibration Logic Overview* there are two read streaming modes: pulsed and continuous. The control for this is bit 8, Read Stream Mode, of the CalConf0 Register at 0xF80029B0. (0 = pulsed; 1 = continuous.)

In pulsed mode the DQS capture latches are reset to 0x5 before every read. In continuous mode the latches are only reset before the first read.

The general principle for using the DQS capture latches is as follows:

- If reads are performed and the DQS sampled by the calibration bit yields a value of 0b1010 = 0xA, then it can be assumed that the DQS is being generated properly by the DRAM, DQS is being received properly, resetLdEnable is timed well enough that the 4 ldEnables are generated properly and enable sampling at the correct times, and that the separate delays for producing the delayed copies of DQS, ldClk_R and ldClk_F, produce rising edges and falling edges that are timed well enough to sample the four levels of DQS with sufficient setup and hold times. However, nothing is known about the “quality” of the timings. (If the timings are optimum, on the edge of failure, or somewhere in between.)
- If reads are performed and the DQS sampled by the calibration bit does *not* yield a value of 0b1010 = 0xA, then something is wrong. If the DRAM is sending DQS (which can be observed with an oscilloscope) and if the CPC945 is otherwise programmed correctly (for example, the I/Os are properly set to differential versus single-ended operation, as appropriate), then the bad result is probably produced by bad timing values for either resetLdEnable, the strobe 1/2-bit time delays, or both.
  - The resetLdEnable timing is controlled by the ResMuxDly value in the Memory Bus Configuration register, and the ResetLdEn values in the Reset Ld En Offset Delay registers.
  - The strobes are delayed nominally by 1/2 bit time and the delay of the rising and falling edges are individually adjusted plus or minus using RdStrOffset values in the Read Data Strobe registers.
- The user is required to somehow find a set of timing values that produce the 0xA pattern indicating good capture of DQS. Once this “seed” is found, a timing control is varied until the 0xA pattern changes to something else, indicating that the capture is no longer valid. The value(s) at which this occurs indicates the margin(s) of good timing values. From these margins, optimum timing control values can be calculated or inferred.
  - The 3 sets of load controls (resetLdEn, rising clock, and falling clock), are individually adjusted.
  - If the byte timings (board wiring lengths) from byte to byte are well matched, then a seed can probably be found in which all byte lanes produce 0xA values. If byte timings are not well matched, then it might happen that a seed is found in which a majority of byte lanes produce 0xA values, and that further experimentation (for example, adjusting the RdStrOffsets on a byte basis) might be required to find a seed that produces all 0xA values. Naturally the margins for these values will be tighter than for a system with well matched byte lanes.

As noted in *Section 7.31.1* for a byte lane the DQ capture latches and the calibration DQS capture latches share the same timing verniers and controls, so the tuning of the timings based on DQS sampling as described above should produce identically optimized timings and margins for the capture of the DQ data bits. Also in *Section 7.31.1* is the observation that this will only be true if the DQS and DQ board wiring lengths are closed matched for a given byte.

It should be noted that *Figure 7-24* is simplified to illustrate that 8 bits of data + 1 calibration bit are associated with the same DQS strobe. In actuality, as shown in *Figure 7-16 Byte Lane Read Control* on page 237, each byte lane has 2 strobes, one “lower” and one “upper,” and 2 sets of verniers for controlling the delta offsets of the 1/2 bit-time delay. For systems with ×8 and ×16 DRAMs this is of no consequence (from a calibration logic viewpoint) because the lower and upper strobes are simply 2 copies of the external DQS and the lower and upper verniers are required to have the same programming values.

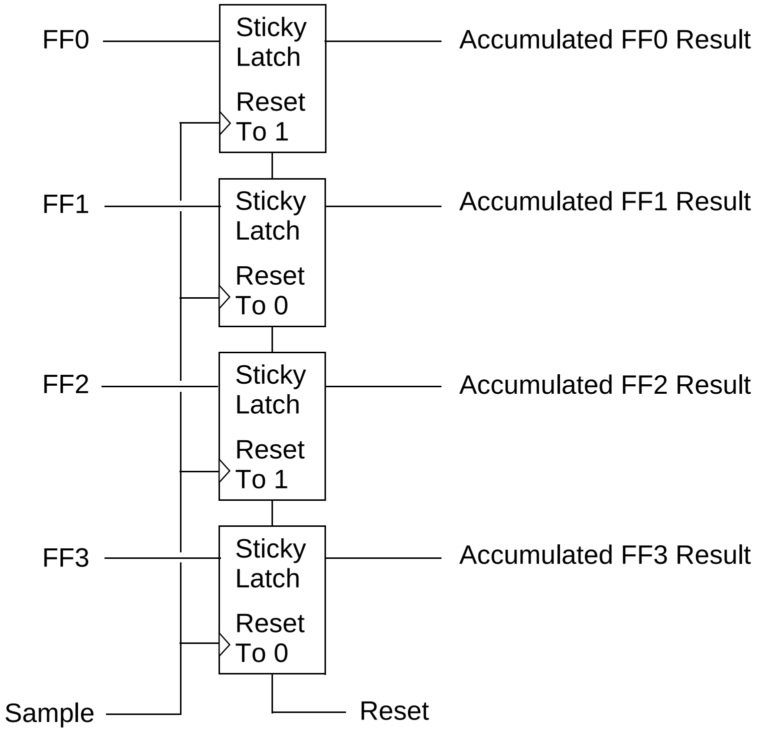
For systems with ×4 DRAMs, however, the “lower” and “upper” strobes correspond to 2 distinct external strobes which might have different times. The calibration bit is connected to the “lower” (even-numbered) strobe. The “upper” strobe does not have a calibration bit. This is a limitation of the DDR2 PHY Calibration Logic for systems with ×4 DRAMs: only the even-numbered strobes can be calibrated with this logic.

For this reason, as noted in *Section 7.22.3 Relationship to Board Wiring*, it is recommended that nibble pairs have identical even/odd wiring lengths.

7.31.3 Calibration Load Monitor

The load monitor consists of 4 sticky latches which sample the outputs of the 4 DQS capture latches.

Figure 7-26. Calibration Load Monitor



The sticky latches have the characteristic that first they are reset to a given level; then afterward whenever the sample control is active the latch will only change state if the input is the opposite of the reset value, and only for the first occurrence of this event. In other words, the two sticky latches reset to “1” will stay at a “1” only if the input is a “1” for every sample. If the input is a “0” during a sample time, the latch output will change to a “0”. It will stay at a “0” independent of the input value for subsequent samples, and will only change back to a “1” when reset.

Similarly, the two sticky latches reset to “0” will stay at “0” only if the sampled inputs are always “0”. If a “1” is sampled the output will change to a “1” and will not change back to a “0” until reset.

In single step mode the user is required to reset the sticky latches at the beginning of each calibration step. This is done by writing bit 0, Reset Calibration Registers, in the four calibration Control and Delay Measurement Registers at 0xF80028F0, 0xF80029F0, 0xF8002AF0, and 0xF8002BF0.

In auto calibration mode the FSM takes care of this reset.

The sample time of the load monitor sticky latches is relative to reset time of the DQS capture latches. It is adjusted in ddr_clk units using the Measurement Result Latency field (bits 20:23) in the CalConf0 register at 0xF80029B0. This is an unsigned value.

The user often sets the sample time relative to ResetLdEnable, for example 8 clocks later. Since the reset time of the DQS capture latches changes relative to ResetLdEnable based on the value of Measurement Reset Latency, this must be taken into account. For example, if it is desired to have the sample time 8 ddr_clks later than ResetLdEnable, and Measurement Reset Latency = 0xA = -2 (sign/magnitude), then the Measurement Result Latency value must be set = 0xA = 10 (unsigned).

In pulsed mode the sample control to the sticky latches is active for one ddr_clk during each read in a stream. As just explained, the pulse occurs after ResetLdEnable with the delay specified by Measurement Result Latency.

In continuous mode the sample control goes active on the first read, using this same delay, but then it stays active for the remaining reads in a stream. It goes inactive when the calibration is subsequently reset, either by the programmer setting the Reset Calibration bit (bit 0) in the calibration Control and Delay Measurement registers (single step mode) or automatically by the FSM (autocalibration mode).

7.31.3.1 Summary of Load Calibration in Pulsed Mode

- A stream of reads is performed.
- Before the start of this stream, the load monitor is reset to the “pass” condition of 0b1010 = 0xA. (In single step mode, bit 0 Reset Calibration Registers in the Calibration Control and Delay Measurement Registers at 0xF80028F0, 0xF80029F0, 0xF8002AF0, and 0xF8002BF0.)
- On every read, the DQS capture latches are first reset to the “fail” condition of 0b0101 = 0x5. If the incoming strobe is successfully captured for all four beats the capture latches will change to 0xA. If the capture is not successful for all four beats then the result will be something other than 0xA.
- After every read, the load monitor sticky latches sample the outputs of the DQS capture latches for one ddr_clk. If any one of the bits is the opposite of its reset value, it changes to the new value and stays at that level (it “sticks”).
- After the stream of reads has completed, the outputs of the load monitor sticky latches are read to determine the overall pass/fail results. (In single step mode, bits 0:15 of the Calibration Read Margin Results registers at 0xF80028E0, 0xF80029E0, 0xF8002AE0 and 0xF8002BE0 will read 0xAAAAxxxx if all bytes had passing results. If ECC is used 0xF80029E0 and 0xF8002BE0 will read 0xAAAAAAxx if all bytes had passing results.)

7.31.3.2 Summary of Load Calibration in Continuous Mode

- A stream of reads is performed.
- Before the start of this stream, the load monitor is reset to the “pass” condition of 0b1010 = 0xA.(In single step mode, bit 0 Reset Calibration Registers in the Calibration Control and Delay Measurement Registers at 0xF80028F0, 0xF80029F0, 0xF8002AF0, and 0xF8002BF0.)
- On the first read, the DQS capture latches are reset to the “fail” condition of 0b0101 = 0x5. If the incoming strobe is successfully captured for all four beats of all reads the capture latches will change to 0xA and stay at that state. If the capture is not successful for all four beats of any read then the result will be something other than 0xA for that read.

- After the first read, the load monitor sticky latches open up to sample the outputs of the DQS capture latches. At the end of the first read and continuously afterward if any one of the bits is the opposite of its reset value, it changes to the new value and stays at that level (it "sticks").
- After the stream of reads has completed, the outputs of the load monitor sticky latches are read to determine the overall pass/fail results. (In single step mode, bits 0:15 of the Calibration Read Margin Results registers at 0xF80028E0, 0xF80029E0, 0xF8002AE0 and 0xF8002BE0 will read 0xAAAAxxxx if all bytes had passing results. If ECC is used 0xF80029E0 and 0xF8002BE0 will read 0xAAAAAAxx if all bytes had passing results.)

7.31.4 ResetLdEnable

The Calibration unit is perhaps most useful for determining an optimum setting for ResetLdEnable: a setting can be determined that does not require knowledge of the internal delays of the chip.

ResetLdEnable has a coarse timing set by the ResMuxDly field in the MemBusConfig register, and a fine timing set by the ResetLdEn Offset value in the ResetLdEn Offset Delay registers.

Consider the case where a coarse setting produces a "pass" condition, and the ResetLdEn Offset has been used to find the point where "pass" becomes "fail." An "optimum" setting can be had by reducing the coarse setting, ResMuxDly, by 1, and leaving the fine setting, ResetLdEn Offset as is.

JEDEC specifies a read preamble of 1 mem_clk = 2 ddr_clks. Therefore backing off ResMuxDly by 1 (ddr_clk) moves the internal resetLdEn signal from a just failing position to exactly in the middle of the read preamble - the position with the most margin on either side.

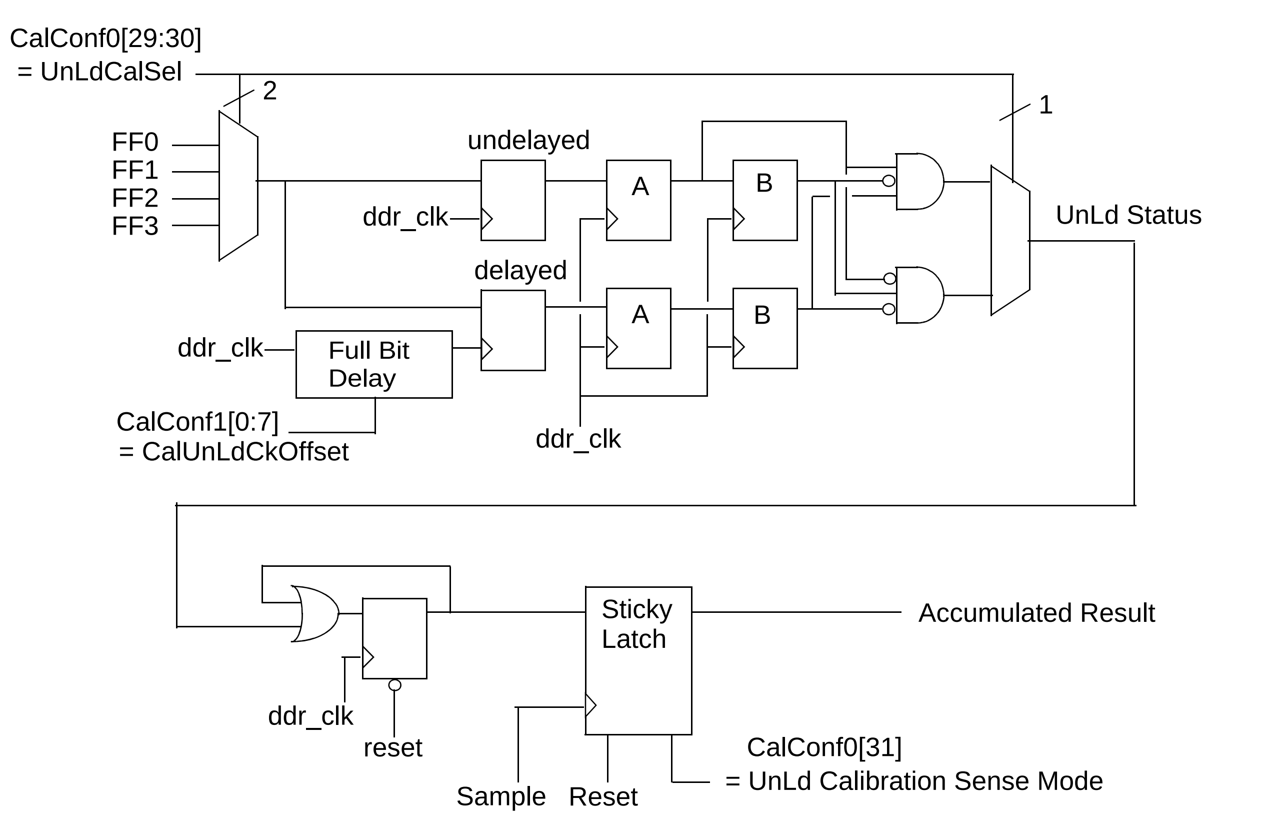
Because the read preamble is 2 ddr_clks long, often 2 different settings of ResMuxDly will be found to work (with 0 ResetLdEn Offset). For this condition, the larger (later) of ResMuxDly should be used when incrementing ResetLdEn Offset to find the fail position.

7.31.5 Calibration Unload Monitor

Use of the calibration unload monitor requires that the DQS capture latches are working properly (load results = 0xA). With this condition it is known that output of a given DQS will change sometime within a bit-time window (because it is forced to a fail level at the beginning of a read and it successfully changes to the pass level). The unload monitor finds *when* this transition happens relative to the internal ddr_clk.

The unload monitor works with a single bit of the DQS capture latch FIFO; bits 29:30, UnLd calibration select[0:1] in the CalConf0 register at 0xF80029B0 are used to select the capture latch of interest.

Figure 7-27. Calibration Unload Monitor



The output of the selected latch is sampled by two flops each using a different clock. One latch uses the internal ddr_clk. The other latch uses ddr_clk delayed by some programmable amount. The delay is generated with a full bit delay vernier (the same kind of vernier used elsewhere in the PHY) using the 8-bit control value CalUnLdClkOffset[0:7] in the CalConf1 register at 0xF80029C0.

The outputs of these 2 flops are labeled “undelayed” and “delayed.” Each output is fed to a chain of 2 flops, A and B, which are clocked by the internal ddr_clk.

The outputs of the A and B flops are connected to an AND circuit. Depending on the amount of delay for the delayed ddr_clk and the relationship of the DQS capture latch output changing relative to ddr_clk, the output of the AND either stays low or it pulses high for 1 ddr_clk. (A more detailed explanation is given below.)

There are actually 2 AND circuits: one to detect a capture latch going low-to-high (FF0 and FF2), the other to detect high-to-low (FF1 and FF3). A multiplexer selects the appropriate AND output based on UnLd calibration select[1].

Since the output of the selected AND is only 1 ddr_clk wide, it is latched by a flop that is wired to hold its value if it clocks in a “1”. This flop is reset at the beginning of every read in pulsed mode, and it is reset at the beginning of the first read in continuous mode.

Finally, the output of the hold flop is fed to a sticky latch which accumulates the unload monitor result. The sticky latch is reset once at the beginning of a calibration step, the same as the sticky latches in the load monitor. However, the “sense” operation of this sticky latch is programmable, under control of bit 31, UnLd Calibration Sense Mode in the CalConf0 register at 0xF80029B0. If Sense Mode = 0 then the latch is reset to a “1” at the beginning of a calibration step and if it samples a “0” value it changes to “0” and sticks at the level. If Sense Mode = 1 then the latch is reset to a “0” at the beginning of a calibration step and if it samples a “1” value it changes to “1” and sticks at the level.

See *Section 7.31.6 Jitter Considerations* for a description of how the Sense Mode can be used to measure jitter on the incoming DQS.

The Sample control to the sticky latch works the same as for the load monitor: the sample time is controlled by the Measurement Result Latency field (bits 20:23) in the CalConf0 register at 0xF80029B0. It samples for one ddr_clk in pulsed mode; in continuous mode after waiting for the Measurement Result Latency time during the first read it opens up the latch and stays open.

Figure 7-28. Unload Monitor Small Offset Timing

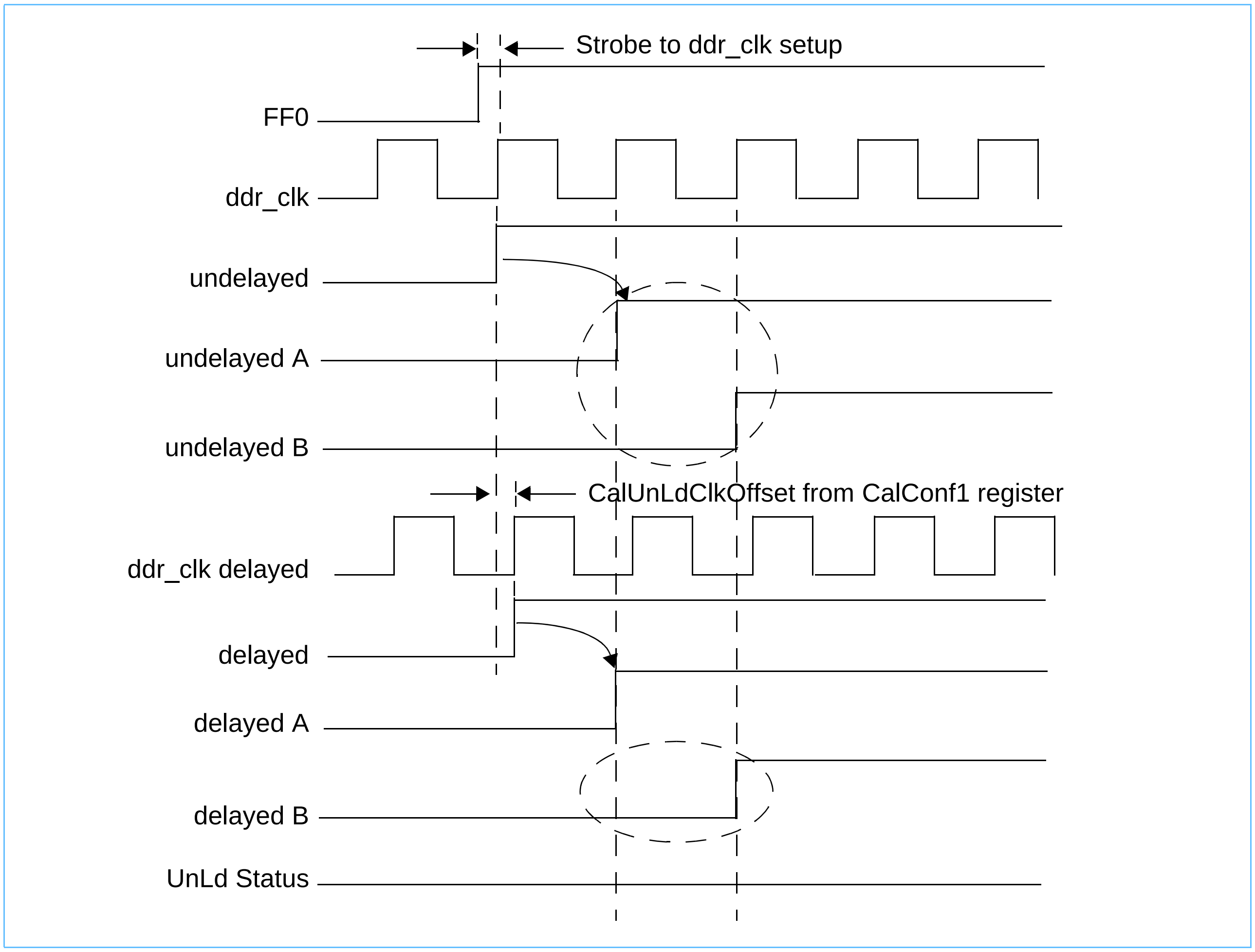


Figure 7-28 shows an example timing in which the AND condition *is not* satisfied. The output of DQS capture latch FF0 has been selected for examination. For the incoming DQS the first rising edge causes the output of FF0 (after some propagation delay) to go from “0” to “1”. In this example this occurs with a relatively short setup time to ddr_clk, perhaps 1/4-bit time.

After this approximately 1/4-bit time, there is a rising edge of the internal ddr_clk which causes the “1” level to be captured in the FF with the output labeled “undelayed.” One ddr_clk later the undelayed “A” FF goes high, followed a cycle later by the undelayed “B” FF going high.

Figure 7-28 shows the “delayed” FF output going high a little later than the “undelayed” FF: there is a relatively small delay programmed in the CalConf1 register. Although it doesn’t have as much setup time to the next ddr_clk, never-the-less the same clock edge which captured the undelayed FF in FF “A” also captures the delayed FF. That is, delayed A has the same timing as undelayed A and delayed B has the same timing as undelayed B.

The AND condition is not satisfied. When undelayed A is high and undelayed B is low, indicating a rising edge has been detected, undelayed B is still low. The UnLd Status signal stays low, the hold FF in *Figure 7-27* stays low, and the monitor result bit (the sticky latch) goes low and sticks low (assuming the UnLd Calibration Sense Mode bit = 0, which resets the sticky latch at the beginning of the calibration step to be high).

Figure 7-29. Unload Monitor Large Offset Timing

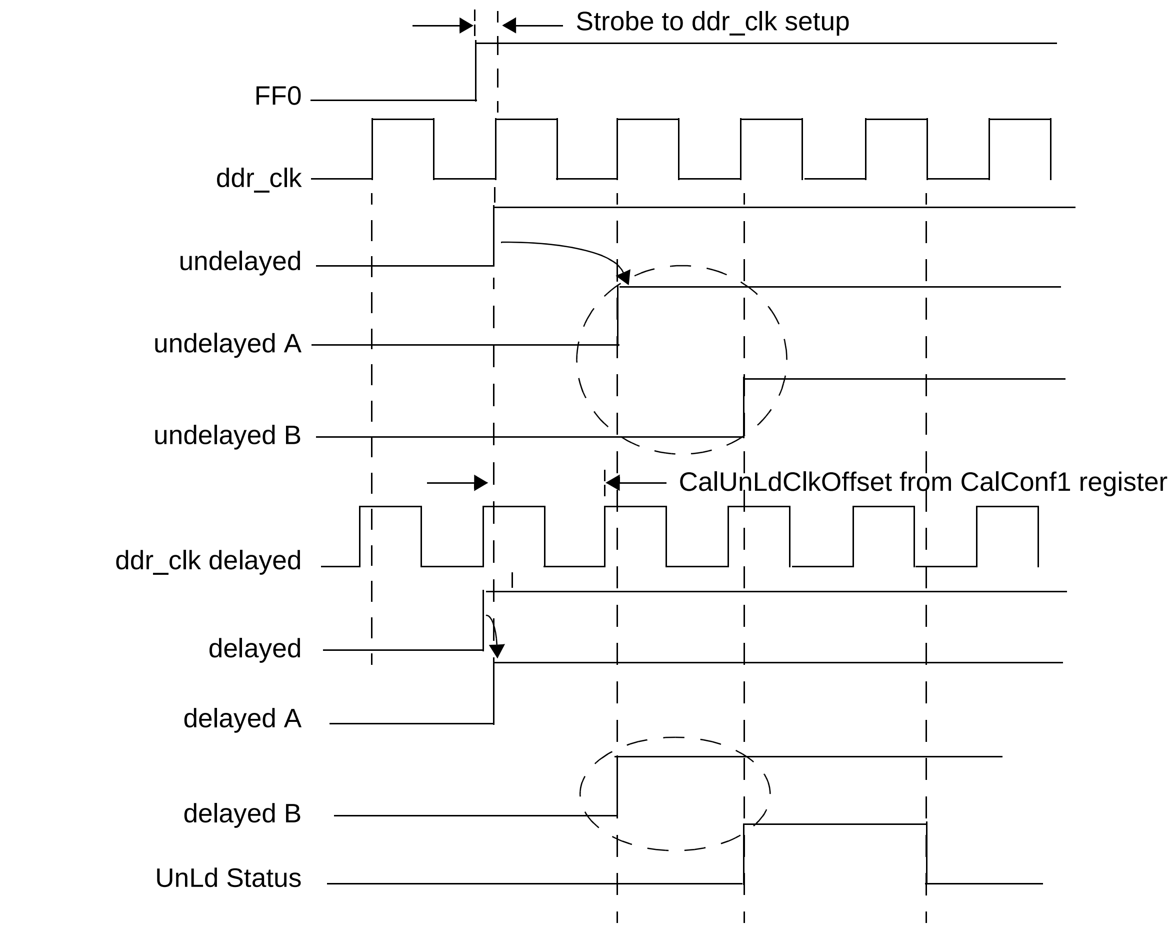


Figure 7-29 shows an example timing in which the AND condition *is* satisfied. It has identically the approximate 1/4-bit time setup from the FF0 capture latch output changing to the rising edge of the internal ddr_clk. What is different is that a much larger CalUnLdClkOffset value has been used to delay ddr_clk to the “delayed” flop. So much delay has been added that a rising edge of this delayed clock is now clocking the FF0 output into the “delayed” flop *earlier* than the normal ddr_clk is clocking the FF0 output into the “undelayed” flop.

In this scenario the “delayed” flop changes just before ddr_clk, so “delayed A” changes one cycle earlier than “undelayed A” and “delayed B” changes one cycle earlier than “undelayed A.” Now, for one cycle, the AND condition is satisfied. The hold latch will go high and stay high. The monitor result bit (the sticky latch) stays high (assuming the UnLd Calibration Sense Mode bit = 0, which resets the sticky latch at the beginning of the calibration step to be high).

Note the confusing nomenclature: the “delayed” flop which is running off an internally delayed clock, actually changes earlier than the “undelayed” flop which is running off the undelayed clock, when there is a large amount of delay.

Also note that if FF1 or FF3 is selected, the signal levels of the capture latch FF, the undelayed and delayed flops, and the four A and B flops will all be inverted compared to selecting FF0 or FF2. However, the bottom AND, with its inverted inputs, will be selected. The results will be the same: the UnLd Status output will either be low, or it will pulse high for one cycle.

Roughly speaking, the amount of delay programmed into the UnLdClk vernier delay is equal to the delay from the output of a capture latch to the next rising edge of the internal ddr_clk. There is an inverse relationship: if there is a small setup time between the capture latch FF output and ddr_clk then a large delay will have to be programmed into the delayed clock to detect this condition. If there is a large setup time then the programmed delay will be small.

The definition of “pass” or “fail” is up to the user. If we note that a changing capture latch output is the same as the rising or falling edge of DQS (ignoring the propagation time of the capture latch and other delays in the unload monitor) and if we consider that the “measuring point” is the amount of delay dialed into the UnLd-Offset Clock, a “0” indicates that the measuring point is to the left (earlier) than the DQS edge and a “1” indicates that the measuring point is to the right (later) than the DQS edge.

For use with autocalibration, as described in *Section 7.31.9* the FSM will increment the delay for every step that passes, and stop when it finds the passing to failing point. Therefore for autocalibration using the unload monitor “0” means “pass” and “1” means “fail.”

7.31.6 Jitter Considerations

The purpose of the sticky latches in the load and unload monitors, and the reason for providing for a stream of reads, is to deal with jitter.

If the load monitor is being used and there is jitter on the DQS, the sticky latches in the load monitor will detect the earliest DQS that causes a pass condition to become a fail. For example, suppose ResetLdEnable is set to some passing condition and gradually adjusted to the right until the load monitor records a fail. If a series of reads is performed, and if the initial rising edge of DQS jitters among the reads, the read with the earliest DQS rising edge is the one that will create the fail, the sticky latch will change to a fail state, and it will continue to stick at that fail. The user will then know the latest that ResetLdEnable can arrive and still operate with jittering arrival times on DQS, and from that infer an optimum amount of time that ResetLdEnable should be backed off to put it in the middle of the read preamble.

Similar considerations apply to adjusting the Read Strobe Offset Deltas to affect the arrival times of the delayed DQS rising and falling edges. In all cases the load monitor measurements indicate the *earliest edges* of a jittering DQS, without discovering any information regarding the *amount* of jitter.

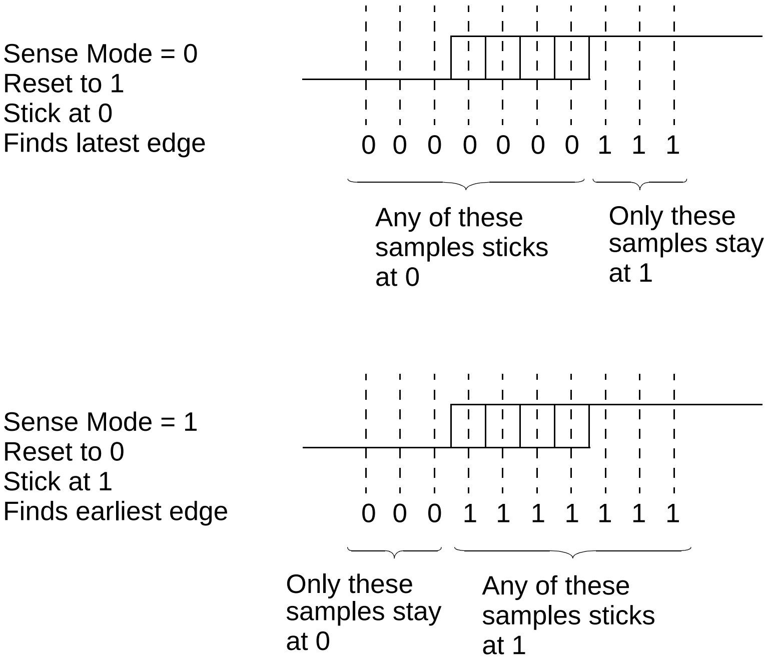
The unload monitor has a programmable sticky latch using the UnLd Calibration Sense Mode bit in the CalConf0 register. This allows both the earliest and latest edges to be detected, and hence, the amount of jitter.

The unload monitor detects the output of a capture latch FF going from one level to its opposite, that is, the leading edge of when the capture latch FF is clocked, that is, when there is a rising or falling edge of DQS. As described above in *Section 7.31.5* the unload monitor has a signal UnLd Status which stays low when the “measuring point” (the delay programmed into the UnLdClkOffset) is to the left of the this DQS edge. UnLd Status pulses high when the “measuring point” is the right of the DQS edge.

Suppose there is jitter on DQS. If the UnLd Calibration Sense Mode equals ‘1’ which equals Early Mode, the sticky latch will be programmed to reset to a ‘0’ and stick at ‘1’ if a ‘1’ is sampled. Suppose the measuring point is well to the left and moved to the right. At first the UnLd Status will always be ‘0’, and the sticky latch will always sample ‘0’ and have an output of ‘0’. When the measuring point is moved enough to the right that it is in the vicinity of the jittering DQS, as soon the measuring point moves past the left most (earliest) rising edge the UnLd Status signal will pulse and the sticky latch will go to a ‘1’ and stick there, even if subsequent samples are to the left of DQS. Thus, for UnLd Calibration Sense Mode equals ‘1’ which equals Early Mode, the unload monitor acts the same as the load monitor to find the earliest DQS edge.

If the UnLd Calibration Sense Mode equals ‘0’ which equals Late Mode, the sticky latch will be programmed to reset to a ‘1’ and stick at ‘0’ if a ‘0’ is sampled. If the measuring point is to the left of the jittering DQS the UnLd Status signal will be ‘0’, and the sticky latch output will go from ‘1’ to ‘0’ on the first sample and stick there. If the measuring point is in the middle of the jittering DQS, at least one edge of DQS will be to the left of the measuring point, and the sticky latch will go to 0 and stick there. It is only when the measuring point moves to the right of the right most (latest) DQS edge that UnLd Status pulses high on every read, and the sticky latch remains at its reset value of ‘1’. Thus for UnLd Calibration Sense Mode equals ‘0’ which equals Late Mode, the unload monitor finds the latest DQS edge.

Figure 7-30. UnLd Calibration Sense Mode



Each vertical dashed line indicates multiple samples of a jittering signal.

Therefore, by making two measurements, early and late, the amount of jitter (or conversely the size of the “eye”) of DQS can be determined.

The user is interested in characterizing the DDR2 interface (how much variance is there from byte to byte; how much jitter is on the bus), and determining optimized values for the timing controls that yield reliable read operation for that user’s board design, and over a variety of conditions.

- The ResetLdEnable timing is determined using the load monitor. This timing must be valid before using the unload monitor.

- The unload operation (the time when the data is captured versus the timing of the internal ddr_clk) is determined using the unload monitor.
- For timing the ring and falling edges of the internally delayed strobes, the user has the choice of using the load monitor or the unload monitor, or both. The load monitor yields information about the absolute limits of operation (how far the edges can go before the load fails); the unload monitor yields information about the quality (jitter) of the interface.

7.31.7 Calibration Setup

Before using the DDR2 PHY calibration logic, the external DDR2 memories must be capable of doing reads. The bulk of the control registers such as RAS/CAS Timing, DIMM Configuration, IO Pad Control and so on, must be properly programmed. The usual initialization sequence (*Memory Device Initialization* on page 151) must be applied. At the board level, the clocks (CK) must be adjusted such that the commands issued to memories will be reliably clocked into the memory chips. (The initialization sequence must be properly received by the memory chips, as well as the subsequent read commands used by the calibration steps.)

Since the calibration logic only does reads, but does not use the read data, the write timings do not need to be correct. (However the user might wish to tune these settings first. If for some reason the CK clocks have to be adjusted as part of the write tuning, this would invalidate any read tunings done previously.)

Once the external memories have been properly brought up, the following conditions must be set up or followed prior to any calibration operation:

- Since the calibration bit has a 4-stage FIFO, the read operation must be 4-beats. Systems using the 128-bit bus configuration always use a burst length of 4. Systems using a 64-bit bus configuration must temporarily re-program the burst length to 4 when using the DDR2 PHY calibration logic, and re-program the length to 8 for normal operation.
- Dynamic CKE must be disabled when using the DDR2 PHY Calibration logic. (Set MemModeCntl Register 0xF8002500 bit 6 = 0.)
- To avoid erratic behavior, do not override the DQS glitch filter. (Set MemBusConfig 0xF80022D0 bit 20 = 0.)
- The MemInit registers must be reprogrammed with a sequence that contains reads for the calibrating operation. Typically the user will set up a Row Activate command to open a row to a bank in some rank (for example, bank 0 and rank 0), followed by several Read commands to that rank/bank, and ending with a Precharge command to close the row.

  This sequence of commands can then be executed as many times as desired. The contents of memory, and the bank, row and column addresses do not matter since the calibration logic is only dealing with DQS. The rank usually matters, however, since the delays on DQS are a function of board wiring length, which usually varies with rank.

  If calibration is intended to be performed using pulse mode, delays must be coded between reads. These delays are not needed for continuous mode.

7.31.8 Single Step Mode

7.31.8.1 Single Step Use Summary

Program CalConf0 0xF80029B0.

- Set Read Stream Mode bit 8 to pulsed mode or continuous mode. If pulsed mode, the read operations in the MemInit registers must have delays coded.
- Set Measurement Reset Latency bits 16:19. For continuous mode, a value of 0 is okay. For pulsed mode, a value of -1 (0x9) or -2 (0xA) is preferable.
- Set Measurement Result Latency bits 20:23. For continuous mode, a value of 6 or 7 is okay. For pulsed mode, a value of 8 is desirable. These times are relative to ResetLdEnable; they must be adjusted if Measurement Reset Latency is not 0. For example, if the desired value Result Latency is 8 (0x8) and the Reset Latency is -1, then Reset Latency must be 8 + 1 = 9 (0x9).
- If the Unload Monitor is being used set UnLd Calibration Select bits 29:30 to select which of the 4 capture latches to be observed. Set UnLd Calibration Sense Mode bit 31 to select Early mode sensing or late mode sensing. If the Unload Monitor is not being used these bits are don’t care.
- The remaining bits in CalConf0 are don’t care for Single Step mode.

After configuring for Single Step mode, perform a series of calibration steps. For each step:

- Write bit 0, Reset Calibration Registers, in each of the 4 Calibration Control and Delay Measurement Registers 0xF80028F0, 0xF80029F0, 0xF8002AF0, 0xF8002BF0.
- Write MemProgCntl register 0xF80020B0 bit 0, Init Start, to kick off the reads.
- Poll MemProgCntl register bit 1, InitComplt until the reads are complete.
- Read the Calibration Read Margin Result Registers 0xF80028E0, 0xF80029E0, 0xF8002AE0, 0xF8002BE0.

  If you are interested in the Load Monitor results the output values will be in bits 0:15. If using ECC the results will be in bits 0:23 for registers CalC1 and CalC3.

  If you are interested in the Unload Monitor results the output values will be in bits 24:27. If using ECC the results will be in bits 24:28 for registers CalC1 and CalC3.

Between each step, vary the timing control of interest.

7.31.9 Autocalibration Mode

7.31.9.1 Autocal Overview

As described previously each cluster has a FSM machine and associated logic that independently performs a series of calibration steps. The FSM automatically kicks off a set of read operations (previously programmed in the MemInit registers). When the reads have completed, the FSM examines the pass/fail results for a single bytelane within the cluster. Depending on the result, the FSM changes the value of one vernier control (out of four) and kicks off another calibration step, or it stops. In general all 4 clusters are operated in parallel, yielding calibration results for 4 byteslanes at a time (2 byteslanes for ECC).

Typically four autocalibration sequences are run with all 4 clusters operated in parallel, to get results for:

1. Bytelanes 0, 4, 8, 12
2. Bytelanes 1, 5, 9, 13
3. Bytelanes 2, 6, 10, 14
4. Bytelanes 3, 7, 11, 15

For systems with ECC, an additional autocalibration sequence is run with just clusters 1 and 3 to get results for:

5. Bytelanes 16 and 17 (check bytes 0 and 1)

Compared to single step mode, additional setup is required to select the bytelane to be calibrated, select the control vernier to be manipulated, specify the maximum control value to be used, and define the condition that is a “pass.” These values are programmed into fields in the CalConf0 register at 0xF80029B0 that are don’t care in single step mode.

7.31.9.2 Bytelane Selection

The bytelane is specified using bits 24:26, Bytelane Select, in the CalConf0 Register.

7.31.9.3 Calibration Mode Selection

The FSM manipulates the control value for 1 of 4 verniers, using internal multiplexers to temporarily replace the programmed control value for a vernier with a control value generated by the FSM. The substituted values are one of:

- A RdStrOffset_DelaLR value located in the Read Data Strobe Control registers (Read Strobe Rising).
- A RdStrOffset_DelaLF value located in the Read Data Strobe Control registers (Read Strobe Falling).
- A ResetLdEn Offset value ResetLdEn Offset Delay registers.
- The CalUnLdClkOffset value in the CalConf1 register.

The vernier is selected using bits 27:28, Calibration Mode Select, in the CalConf0 Register, with values 00, 01, 10, 11 selecting Read Strobe Rising, Read Strobe Falling, ResetLdEn Offset, and CalUnLdClkOffset, respectively.

When Read Strobe Rising, Read Strobe Falling or ResetLdEn Offset is selected (00, 01, or 10), the output of the load monitor is selected for determining pass/fail status.

When CalUnLdClk Offset is selected (11), the output of the unload monitor is selected for determining pass/fail status.

7.31.9.4 Generated Vernier Control Values

The FSM maintains an 8-bit control value which, as described above, is applied to one of four verniers. This control value is manipulated as 2, 4-bit nibbles. The FSM initializes the control value to 0x10 and increments the upper nibble between calibration steps as long as the results of the steps produce a "pass" result. Once a "fail" is seen, the FSM switches to manipulating the lower nibble, using a binary search algorithm. For example, if the largest value that produces a "pass" value is 0x40, the FSM will generate the sequence: 0x10 (pass), 0x20 (pass), 0x30 (pass), 0x40 (pass), 0x50 (fail), 0x48 (fail), 0x44 (fail), 0x42 (fail), 0x41 (fail). At this point the FSM stops, and sets the value of 0x40 into bits 8:15, CalFSMSetting Result, in the Control and Delay Measurement Register.

Following an autocalibration sequence, the value read in bits 8:15, CalFSMSetting Result, in the Control and Delay Measurement Register will be the largest value that produces a "pass" result. Adding 1 to this value should produce a "fail" result.

It might happen that the first calibration step produces a "fail" with a vernier value of 0x10. For this case the FSM will immediately switch manipulating the lower nibble. If each step produces a fail, the FSM sequences through 0x08, 0x04, 0x02 and stops at 0x00. It is a quirk of the autocalibration logic that if it returns a value of 0x00, that might indicate that the FSM never found a "pass" value (the calibration fails with a vernier value of 0x00), or it might indicate that the FSM found that 0x00 passes while 0x01 fails. There is no way to tell.

There are 2 values which indicate an error:

- 0xFE indicates a timeout condition. At the start of a calibration step the FSM enables a 32-bit counter to start counting from 0, incrementing by 1 every ddr_clk. If a value of 0xFFFF is reached, indicating (2^{32}) -1 ddr_clks have passed, the autocalibration sequence is terminated and a value of 0xFE is placed in the Result field.

  This condition could arise, for example, if the MemInit registers had no read commands specified, or the commands looped beyond a very long time.
- 0xFF indicates the FSM exceeded a maximum value when incrementing the 8-bit control value.

By default the FSM tries to restrict its "sweep range" to a full bit time. It does this by monitoring the 1/2-bit time delay that is fed to the DQS Verniers (*Figure 7-8 1/2 Bit Time Averager* on page 218). The FSM multiples this value by 2 (shifts left by 1) to obtain the full bit time. This value is used as a maximum: if it is exceeded then the autocalibration sequence is terminated and 0xFF is returned in the Result field.

The FSM can temporarily exceed this value in applying its search algorithm. For example suppose the 1/2-bit time is 0x32, so the limit = the full bit time = 0x64.

- Suppose the largest "pass" value = 0x63. The FSM will generate 0x10 (pass), 0x20 (pass), 0x30 (pass), 0x40 (pass), 0x50 (pass), 0x60 (pass), 0x70 (fail, and exceed maximum), 0x68 (fail, and exceed maximum), 0x64 (fail), 0x62 (pass). Returned value = 0x63.
- On the other hand, suppose the largest "pass" value = 0x65. The FSM generates 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70 (fail, exceed maximum), 0x68 (fail, exceed maximum), 0x64 (pass. at maximum). Returned value = 0xFF.

The maximum value can be changed using bits 9:15, Delay Offset Limit, in CalConf0. This is a 1-bit sign, 6-bit magnitude value, which is added to the 1/2-bit time value. That is, the range of the maximum value used by the FSM is the 1/2-bit time value -0x3F through +0x3F.

The user should read the 1/2-bit time value, bits 24:31 in the CntlDlyMeasC0 Register 0xF80028F0 and multiply the value by 2 to learn the maximum value that will be used by the FSM. If this maximum is not desired use the Delay Offset Limit in CalConf0 to raise or lower the maximum.

As explained above CalConf0 bits 27:28, Calibration Mode Select, select 1 of 4 verniers to be controlled by the autocalibration FSM. For values 10 and 11, the ResetLdEn Offset and CalUnLdClk Offset verniers, the control values generated by the FSM are applied directly to the verniers. The verniers will see a sweep of 0x10, 0x20, 0x30, ... up to some fail or maximum value, or 0x10 down to 0x00 if the pass value is small.

For Calibration Mode Select value 00 and 01, the Read Strobe Rising and Falling edges, the control value is calculated in a different manner, but the results are the same.

Figure 7-15 Read DQS on page 234 shows the DeltaR and DeltaF sign/magnitude values being fed to an adder that adds or subtracts the register value to the 1/2-bit time value to form the value controlling the delay vernier.

The autocalibration logic inserts multiplexers between the DeltaR/DeltaF register values and the +/- Delta adder circuit. If the FSM generated a sequence of 0x10, 0x20, 0x30...0x80, 0x90, ... the Delta adder would produce an undesired sweep sequence to the delay vernier that would start with 1/2-bit delay, increment, then suddenly switch back to 1/2-bit delay and decrement when the high-order bit became high.

Therefore the FSM generates a modified sequence that, when added by the hardware to the 1/2-bit time value, produces a value *at the vernier* that is the desired sweep of 0x10, 0x20, 0x30...and so on.

The reported result is the value as it would appear at the delay vernier. The user must take this into account to determine the value that would be programmed in the Read Data Strobe Control Register. For example, if the 1/2-bit delay value = 0x32, and an autocalibration sequence returned the result of 0x4D, the user could produce the same delay for Read Strobe rising edge by setting the value 0x4D - 0x32 = 0x1B in the RdStrOffset_DelaLR and RdStrOffset_DelaUR fields of the Read Data Strobe Control Register. As noted above the 1/2-bit delay value is available in bits 24:31 in the CntlDlyMeasC0 Register 0xF80028F0.

7.31.9.5 FSM Pass/Fail Determination

As described above the FSM will use either the load monitor or unload monitor for pass/fail determination, as appropriate: the unload monitor is used when the UnLdClk Offset vernier is selected, otherwise the load monitor is used.

CalConf0 bits 0:3, Calibration Pass Result, and bits 4:7, Calibration Pass Mask are used to interpret the outputs of the selected monitor. The output of the monitor is compared, bit by bit, to bits 0:3. If all 4 bits of the monitor output are identical to the to 4 Calibration Pass Result bits, the result is a “pass.” If any of the bits miscompare, the result is a “fail.” The Calibration Pass Mask bits can be used to remove one or more bits from the compare operation. For example, if the user specifies that a Pass Result is 0xA but the load monitor output is 0x2, the result will still be considered a “pass” if the Pass Mask is coded to 0x7 to mask out bit 0, the bit that miscompares.

For the load monitor the usual Pass Result is 0xA and the usual Pass Mask is 0xF. One example of when the Pass Mask might be used to exclude bits in the comparison is when the Read Strobe Rising or Falling edge is being adjusted. The problem here is that the autocalibration FSM is only adjusting one of the edges. A fail might be produced by the fact that 2 edges are widely separated, not because the edge being adjusted is too far in a given direction. A user might find, for example the rising edge adjustment, might not yield any results for a Pass Mask of 0x8 (the first rising edge), but might give a usable result for a Pass Mask of 0x2 (the second rising edge).

For the unload monitor the usual Pass Result is 0x0 and the usual Pass Mask is 0x8.

7.31.9.6 Control and Delay Measurement

After setup, an autocalibration sequence is kicked off by writing to bit 1, Start Auto Calibration State Machine, in the Control and Delay Measurement registers (0xF80028F0, 0xF80029F0, 0xF8002AF0, 0xF8002BF0). Bit 3, Auto Calibration Done, changes from 0 to 1 when the FSM has completed the autocalibration sequence. The results (the final control value = the delay measurement) will be in bits 8:15.

7.31.9.7 Autocalibration Use Summary

Program CalConf0 0xF80029B0.

- Set Read Stream Mode bit 8 to pulsed mode. The read operations in the MemInit registers must have delays coded.
- Same as for Single Step mode, set the Measurement Reset and Result Latencies.
- Same as for Single Step mode, if using the Unload Monitor set the UnLd Calibration Select bits and the UnLd Calibration Sense Mode bit.
- Set the byte lane to be adjusted, using bits 24:26 Bytelane Select.
- Set the adjustment to be varied, using bits 27:28 Calibration Mode Select.
- If the UnLdClk Offset is the adjustment selected, set the capture latch to be monitored using bits 29:30 UnLd Calibration Select[0:1] and bit 31 UnLd Calibration Sense Mode (same as Single Step mode). Nothing has to be programmed into CalConf1, since the FSM will generate the CalUnLdClkOffset register.
- Set the desired Calibration Pass Result, bits 0:3 and Calibration Pass Mask, bits 4:7. As discussed above, this can vary depending on which measurement is selected, and might have to be determined by trial and error.
- If desired, change the maximum control value generated, plus or minus, using the Delay Offset Limit bits 9:15. This might have to be determined by trial and error.

After setting up the configuration in CalConf0, kick off the autocalibration sequence by writing to bit 1, Start Auto Calibration State Machine, in the Control and Delay Measurement registers 0xF80028F0, 0xF80029F0, 0xF8002AF0, 0xF8002BF0. Poll bit 3, Auto Calibration Done. When the bit changes to 1 in all four registers the FSM in each cluster has completed the autocalibration sequence. The results for the 4 selected bytes will be in bits 8:15.

As discussed above this sequence is usually repeated 4 times, each time changing the bytelane. If ECC is used the sequence is repeated one more time, using just clusters 1 and 3, with the ECC check byte selected.

8. I^{2}C Interfaces

8.1 Overview

The CPC945 has two independent I^{2}C interfaces, a single port I^{2}C Slave interface and a dual port I^{2}C Master interface. These two interfaces are completely independent should not be confused.

The I^{2}C Slave Interface is used to read and write the CPC945 internal control registers or, indirectly, read or write any address in the 36 bit memory space. The master for the CPC945 I^{2}C slave port is typically the external System Management Processor (SMP).

The two port CPC945 I^{2}C master interface is intended to be connected to the I^{2}C slave ports on the DRAM DIMMs so that the CPC945 can read or write to the configuration registers on each DIMM. The master interface control logic can be switched between the two ports under control of the I^{2}C Controller MODE Register. Only one port can be controlled at a time.

8.2 I^{2}C Slave Interface

The CPC945 provides a standard two wire I^{2}C slave interface for the serial exchange of data between the external master and the CPC945. For the CPC945 the two signals are PI_ISCL for the clock line and PI_ISCA for the data line. Programming of the CPC945 is accomplished using standard I^{2}C read and write transactions on this slave interface.

8.2.1 I^{2}C Slave Interface Transactions

There are two different types of I^{2}C transaction types depending on the intent of the request. The first transaction type is used for reading and writing the CPC945 control registers. This transaction uses a direct path into the register space and is used for system initialization, debug and diagnostics.

A second indirect method is used for reading or writing any address within the 36 bit physical address space of the CPC945 including the control registers.

Note: This transaction actually uses the same internal path as a similar request from the main processor so the request will be snooped. This makes this type of transaction reliant on a fully operational system. If the processor bus or processors are not operational, this type of transaction will hang the CPC945 because the snoop cannot be completed.

8.2.1.1 Control Register transaction types

All I^{2}C transactions are initiated by a start, followed by a 4-byte I^{2}C write. The rest of the I^{2}C transaction depends upon whether a write or read transaction is being performed.

The I^{2}C slave expects 4 bytes of command/address followed by the data bytes. A write always has 8 bytes of data. A read is unlimited in the number of bytes being transferred, and is terminated only by the master issuing a stop or abort condition. In this way an entire sequence of control registers can be dumped with a single command.

The first byte of a transaction is the address. The CPC945 has a hardcoded 7 bit I^{2}C address: 7 b1100000. When the address is combined with the eighth bit, the READ/WRITE bit, the result is two different 8 bit words used for addressing the slave interface:

WRITE: 0xC8

READ: 0xC9

If the first byte after the start of an I^{2}C operation is not a 0xC8 or a 0xC9, the CPC945 will ignore the operation.

The second byte of a transaction is always a sub-address indicating that it is a command byte, the type of command - R/W, and for WRITEs it contains the Write Byte Enables (WBE). While this byte must always be present, the WBE are only needed for writing to certain HyperTransport and PCI Express registers.

Table 8-1. Twenty-two Bit I^{2}C System Command Request Definition.

	[0] Reserved	[1] Reserved	[2] Command	[3] R/W	[4:7]
Read Request Command	1b0	1b0	1b1	1b1	4b0000
Write Request Command	1b0	1b0	1b1	1b0	WBE [0:3]

This results in the general format of this second byte as:

WRITE: 0x2F <-- 0x2 || Write Byte Enables[0:3]

READ: 0x30

For a limited number of HT and PCIe express registers the write byte enables can be different from 0xF (or 0b1111). Each bit is associated with its respective byte in a 32 bit word.

The third and fourth bytes of the I^{2}C transaction are two bytes containing the control register address. The control registers in the CPC945 are located in the 16 Mbit address space 0x0F8xx xxxx. Within that space only 16 bits are required to address any control register. Therefore, a register in the system control register address space is addressed with a 16 bit field called the target address (Target Address[0:15]). The target address to system address translation is accomplished within the CPC945 by the following relationship.

System Address [0:35] <-- 0x0F80 || Target Address[0:15] || 0x0.

The remaining bytes of the transaction are specific to the type of transaction: READ or WRITE.

Control Register WRITE Transaction Detail

The I^{2}C interface transaction is built up from a set of smaller byte by byte data exchanges. Each byte data exchange is completed with a byte acknowledge (ACK). The I^{2}C cycle by cycle write transaction description is shown in Table 8-2 I^{2}C Write Transaction.

Note that all writes have eight bytes (64 bits) of data but not all registers are 64 bits wide. The eight bytes of data sent into the slave I^{2}C are formed into a 64 bit group. For a 32 bit write, bits 0:31 of the incoming data are loaded into bits 0:31 of the 32 bit register. Since 64 bits must still be written, bits 32:63 should be a duplicate of the lower 32 bits. For a 16 bit write, bits 16:31 are loaded.

For a 32 bit read, 32 bits of data are duplicated to form 64 bits of returning data. In case of 16 bit reads, bits 0:15 and bits 32:47 are forced to zero and the 16 bits of data are duplicated in the remaining bits. This is shown below:

16-bit register:16’h0000 || RdDt[0:15] || 16’h0000 || RdDt[0:15]

32-bit register: RdDt[0:31] || RdDt[0:31]

64-bit register: RdDt[0:63]

Table 8-2. I^{2}C Write Transaction (Includes Number of Bits Transmitted for Each Action).

8 bits	1	8	1	8	1	8	1	8	1	8	1	45	8	1	1
Slave Address, WrBit	Ack	Addr 1	Ack	Addr 2	Ack	Addr 3	Ack	Data 1	Ack	Addr 2	Ack	-	Data B	Ack	Stop

The sequence of bits on the I^{2}C bus looks like the following for a control register WRITE. For these transactions, the 8 databytes contain the actual control register contents on a write.

Table 8-3. Bit Sequence on the I^{2}C Bus for a Control Register Write Transaction.

ISCL Clock Cycle	Action	Direction	Comments
0	START	Master -> Slave	
1 ~ 7	7'b110010	Master -> Slave	I^{2}C Write Request: I^{2}C Slave Address concatenated with the R/W bit = 0xC8
8	R/W = 1b0	Master -> Slave	
9	ACK	Slave -> Master	
10 ~ 17	0x2F	Master -> Slave	Write command with all WBE enabled
18	ACK	Slave -> Master	
19 ~ 26	Target Address [0:7]	Master -> Slave	High order byte of the 16bit register address
27	ACK	Slave -> Master	
28 ~ 35	Target Address [8:15]	Master -> Slave	Low order byte of the 16bit register address
36	ACK	Slave -> Master	
37 ~ 44	Data Byte 0	Master -> Slave	
45	ACK	Slave -> Master	
46 ~ 53	Data Byte 1	Master -> Slave	
54	ACK	Slave -> Master	
55 ~ 62	Data Byte 2	Master -> Slave	
63	ACK	Slave -> Master	
64 ~ 71	Data Byte 3	Master -> Slave	
72	ACK	Slave -> Master	
73 ~ 80	Data Byte 4	Master -> Slave	
81	ACK	Slave -> Master	
82 ~ 89	Data Byte 5	Master -> Slave	
90	ACK	Slave -> Master	
91 ~ 98	Data Byte 6	Master -> Slave	
99	ACK	Slave -> Master	

Table 8-3. Bit Sequence on the I²C Bus for a Control Register Write Transaction.

ISCL Clock Cycle	Action	Direction	Comments
100 ~ 107	Data Byte 7	Master -> Slave	
108	NACK	Slave -> Master	
109	STOP	Master -> Slave	

Control Register READ Transaction Detail

The READ transactions are a variant of a WRITE transaction. A read request consists of a the first 4 bytes of a write transaction (as described above), followed by a (re)start, followed by a read command consisting of the CPC945 I^{2}C address, with the read/write_bit = “1”. The Slave interface logic sends serially the first byte from the register on the PI_ISCA pin. An ACK from the master indicates the data has been accepted. Unlimited read bursts are allowed to help reading an entire set of registers with one request. The master can continue to read (by sending an ACK to the Slave interface after every byte), and sending the STOP signal after the last byte to terminate the transaction. The sequence of bytes on the I^{2}C bus looks like the following for a control register READ. For these transactions, the 8 databytes are the actual control register contents from the read.

Table 8-4. Bit Sequence on the I2C Bus for a Control Register Read Transaction.

ISCL Clock Cycle	Action	Direction	Comments
0	START	Master -> Slave	
1 ~ 7	7'b110010	Master -> Slave	I^{2}C Write Request: I^{2}C Slave Address concatenated with the R/W bit = 0xC8
8	R/W = 1b0	Master -> Slave	
9	ACK	Slave -> Master	
10 ~ 17	0x30	Master -> Slave	Read command, WBE are not used
18	ACK	Slave -> Master	
19 ~ 26	Target Address [0:7]	Master -> Slave	High order byte of the 16bit register address
27	ACK	Slave -> Master	
28 ~ 35	Target Address [8:15]	Master -> Slave	Low order byte of the 16bit register address
36	ACK	Slave -> Master	
37	START	Master -> Slave	
38 ~ 44	7'b110010	Master -> Slave	I^{2}C Read Request: I^{2}C Slave Address concatenated with the R/W bit = 0xC9
45	R/W = 1b1	Master -> Slave	
46	ACK	Slave -> Master	
47 ~ 54	Data Byte 0 [0:7]	Slave -> Master	
55	ACK	Master -> Slave	
56 ~ 63	Data Byte 1 [8:15]	Slave -> Master	
64	ACK	Master -> Slave	
65 ~ 72	Data Byte 2 [16:23]	Slave -> Master	

Table 8-4. Bit Sequence on the I2C Bus for a Control Register Read Transaction.

ISCL Clock Cycle	Action	Direction	Comments
73	ACK	Master -> Slave	
74 ~ 81	Data Byte 3 [24:31]	Slave -> Master	
82	ACK	Master -> Slave	
83 ~ 90	Data Byte 4 [32:39]	Slave -> Master	
91	ACK	Master -> Slave	
92 ~ 99	Data Byte 5 [40:47]	Slave -> Master	
100	ACK	Master -> Slave	
101 ~108	Data Byte 6 [48:55]	Slave -> Master	
109	ACK	Master -> Slave	
110 ~ 117	Data Byte 7 [56:63]	Slave -> Master	
118	NACK	Master -> Slave	
119	STOP	Master -> Slave	

Examples of Control Register Addressing for I^{2}C Slave Requests

Example 1:
The default value in the Clock Control Register at system address
0x0F8000800 is 0x000803BC00000000
To change the DDR2 frequency from 400 MHz to 533 MHz in the Clock Control Register.
This is the desired result:
0x0F8000800 to 0x001003BC00000000,
To do this a WRITE is issued to the Target Address bits 16:31 of the full 36 bit address or 0x0080.
The sequence of command bytes transmitted on I^{2}C is (in hex):
C8 2F 00 80 00 10 03 BC 00 00 00 00

Example 2:
The sequence of bytes on I^{2}C for reading the new result in the same register is (in hex)
C8 30 00 80 C9 00 10 03 BC 00 00 00 00
Although the Clock Control Register is defined as a 64-bit register, bits 32:63 are reserved and read as all zero. The READ transaction could be terminated by the master after the first 4 bytes of data.

8.2.1.2 I^{2}C Transactions to any Physical Memory Location

The slave I^{2}C port can be used to write or read any memory location within the physical address space. This is done by a series of writes and reads to specific command registers using the operations described above. The details on how this is done is described in the following sections.

WRITE Transaction to Memory Detail

It takes multiple command sequences to execute a WRITE to an arbitrary memory location. First are WRITEs to the four System Command Data Registers (SysCmdDt[0:3]: 0xF8030240-0xF803270) to hold the data to be written. Next are control register WRITEs to the two PI System Command Registers (SysCmdCntl0: 0xF8030200 and SysCmdCntl1: 0xF8030210) with the data bytes being the address of the actual memory location where the data is to be written.

Essentially the two 32 bit System Command Registers registers form a 64 bit register. The first control register to be written to is SysCmdCntl1: 0xF8030210 which will hold the 32 low order bits (32:63)of the system address to be written to. The second register to be written is SysCmdCntl0: 0xF8030200 which will contain the four high order bits (28:31) of the system address to be written to. Immediately after the second register is written, the CPC945 executes the requested WRITE operation at the specified address. This order must be followed for the command to work correctly.

The sequence of bytes on the I^{2}C bus for writing the SysCmdCntl Registers are shown in *Table 8-5* and *Table 8-6* on page 273.

Table 8-5. Byte Sequence on the I^{2}C Bus for Writing the SysCmdCntl1 Register .

ISCL Clock Cycle	Action	Direction	Comments
0	START	Master -> Slave	
1 ~ 7	7b110010	Master -> Slave	I^{2}C Write Request: I^{2}C Slave Address concatenated with the R/W bit = 0xC8
8	R/W = 1b0	Master -> Slave	
9	ACK	Slave -> Master	
10 ~ 17	0x2F	Master -> Slave	Write command with all WBE enabled
18	ACK	Slave -> Master	
19 ~ 26	0x30	Master -> Slave	The target address for SysCmdCntl1 is 3021
27	ACK	Slave -> Master	
28 ~ 35	0x21	Master -> Slave	
36	ACK	Slave -> Master	
37 ~ 44	0x00	Master -> Slave	Writes must be 8 bytes but register is 4 bytes
45	ACK	Slave -> Master	
46 ~ 53	0x00	Master -> Slave	
54	ACK	Slave -> Master	
55 ~ 62	0x00	Master -> Slave	
63	ACK	Slave -> Master	
64 ~ 71	0x0 || [Addr bits 28:31]	Master -> Slave	High 4 bits of 36 bit address to be written
72	ACK	Slave -> Master	
73 ~ 80	0x00	Master -> Slave	
81	ACK	Slave -> Master	
82 ~ 89	0x00	Master -> Slave	
90	ACK	Slave -> Master	

Table 8-5. Byte Sequence on the I²C Bus for Writing the SysCmdCntl1 Register (Continued).

ISCL Clock Cycle	Action	Direction	Comments
91 ~ 98	0x00	Master -> Slave	
99	ACK	Slave -> Master	
100 ~ 107	0x0 || [Addr bits 28:31]	Master -> Slave	Repeated for 32 bit register write
108	NACK	Slave -> Master	
109	STOP	Master -> Slave	

Table 8-6. Byte Sequence on the I²C Bus for Writing the SysCmdCntl0 Register .

ISCL Clock Cycle	Action	Direction	Comments
0	START	Master -> Slave	
1 ~ 7	7b110010	Master -> Slave	I²C Write Request: I²C Slave Address concatenated with the R/W bit = 0xC8
8	R/W = 1b0	Master -> Slave	
9	ACK	Slave -> Master	
10 ~ 17	0x2F	Master -> Slave	Write command with all WBE enabled
18	ACK	Slave -> Master	
19 ~ 26	0x30	Master -> Slave	The target address for SysCmdCntl0 is 3020
27	ACK	Slave -> Master	
28 ~ 35	0x20	Master -> Slave	
36	ACK	Slave -> Master	
37 ~ 44	[Addr bits 32:39]	Master -> Slave	This is the low 32 bits of the register to be written.
45	ACK	Slave -> Master	
46 ~ 53	[Addr bits 40:47]	Master -> Slave	
54	ACK	Slave -> Master	
55 ~ 62	[Addr bits 48:55]	Master -> Slave	
63	ACK	Slave -> Master	
64 ~ 71	[Addr bits 56:63]	Master -> Slave	
72	ACK	Slave -> Master	
73 ~ 80	[Addr bits 32:39]	Master -> Slave	Repeated for 32 bit register write.
81	ACK	Slave -> Master	
82 ~ 89	[Addr bits 40:47]	Master -> Slave	
90	ACK	Slave -> Master	
91 ~ 98	[Addr bits 48:55]	Master -> Slave	
99	ACK	Slave -> Master	
100 ~ 107	[Addr bits 56:63]	Master -> Slave	
108	NACK	Slave -> Master	
109	STOP	Master -> Slave	

As noted earlier, this type of request participates in the Snoop reflection protocol on the PI bus and returns an Accumulated Snoop Status in the SysCmdStat register. If the status indicates a retry, the request will be reissued the number of times specified in the NumRetry field in that register. The number of times the request was retried is indicated in the RetryCnt field. If the request comes back with a non-retry status or the RetryCnt exceeds the NumRetry value then the StatDone bit is set. A NumRetry value of zero indicates that the request should be reissued until non-retry status is received (RetryCnt is ignored). The RetryCnt will stay at its maximum value if it is reached.

On a write operation the data that is placed in the SysCmdData registers is written to the requested destination when the status indicated is non-retry. When the non-retry status is seen, StatDone is set and the write request is sent. When the data operation has completed the DataDone bit is set. If the RetryCnt exceeds the NumRetry value, the StatDone is set and DataAbort is set indicating that the operation has completed without the data being written.

On a read operation the StatDone bit is set if the RetryCnt exceeds the NumRetry value, the status returned is Intervention, or a valid (Null or Shared) response is received. Only with a Null or Shared response does the read request continue to its destination. When the data is returned from the Read request, it is placed in SysData registers and the DataDone bit is set completing the operation. If the Read request is not sent, and a StatDone condition occurs, the DataAbort bit is set indicating the operation has completed.

Read Transaction from Memory Details

Similar to WRITE transactions, it takes multiple command sequences to execute a READ to an arbitrary memory location. The format is actually a sequence of two control register WRITEs to the two PI System Command Registers (SysCmdCntl0: 0xF8030200 and SysCmdCntl1: 0xF8030210) with the data bytes being the address of the actual memory location and an operation field indicating a READ.

The two SysCmdCntl registers are written using the same method as described for a WRITE operation. First control register SysCmdCntl1: 0xF8030210 is written with the 32 low order bits 32:63 of the system address to be read from. Then control register SysCmdCntl0: 0xF8030200 is written with the four high order bits 28:31 of the system address to be read from. Immediately after the second register is written, the CPC945 executes the requested operation.

The CPC945 then retrieves the data from the requested location and puts the data into the System Command Data Registers (SysCmdDt[0:3]: 0xF8030240-0xF803270). A subsequent standard I^{2}C READ operation is used to read the data from these registers. Because these registers are sequential, this can be a continuous read that is terminated by the Master when all bytes have been read.

8.3 I2C Master Interface

8.3.1 Overview

The I^{2}C Master Controller is designed to drive two Master I^{2}C ports, one at a time, as determined by a bit in the I^{2}C Mode Register (0xF8001000). These ports are standard bi-directional I/O pads that interface the system DRAM I^{2}C interfaces to the CPC945. Each of the two I^{2}C Master Interfaces consists of two wires, serial data (SYS_ISCAx) and serial clock (SYS_ISCLx), to transfer information between the devices connected to the I^{2}C bus. The protocol on the interface is the standard I^{2}C protocol. Be careful not to confuse the I^{2}C Master ports operation with the I^{2}C Slave Interface. They are two independent functions.

The I^{2}C is used to obtain configuration information from the DIMMs to determine which memory mode to operate in.

The I^{2}C master interface has the following features:

- Master only operation
- 7-bit addressing
- Slave clock stretching support
- 100 kHz, 50 kHz, and 25 kHz bit timing modes
- Registers for 7-bit address and 8-bit subaddress
- Support for Automatic Address Phase Generation in standard, standard with subaddress, and combined modes
- Bus status indicator (busy bit)
- Maskableinterrupts for: start condition, address, and stop conditions transmitted
- Maskable interrupt on data byte and Ack transmitted or received

The following features are not supported:

- Slave support
- Arbitration
- Fast mode
- 11-bit or general call addressing
- Clock synchronization/multimaster support

8.3.2 I^{2}C Master Control Registers

There are eleven CPC945 DRAM I^{2}C Master Controller Registers. They include: MODE, CNTRL, STATUS, ISR, IER, ADDR, SUBADDR, DATA, REV, RISETIMECNT, and BITTIMECNT Registers. Usage information follows with additional information on the registers provided in the programmer’s interface (See *Section 12.8 DRAM I2C Master Controller Registers* on page 398). These registers are used to set the various modes and controls for the I^{2}C master interface and also to hold status and data on the transfers. Reading and writing these registers is actually done using the main processor or using the I^{2}C Slave interface.

8.3.2.1 MODE Register Usage

The MODE register (0xF8001000) is used to control the bit rate of the I^{2}C interface, to select the address phase mode, and to select one of the two I^{2}C ports. The bit rate of the interface can be set to 25, 50, and 100 kb/s. This feature is provided so that the bit timing can be derated from the standard 100 kb/s, because some slaves can not keep up with the standard rate. The I^{2}C cell provides four options for the address phase generation. The first option is manual mode in which the START condition, address bits / R/W_ bit, each data byte, and the STOP condition are sent on the I^{2}C interface by the separate commands of START, DATA, and STOP. The remaining three options are variations of an automatic address phase sequence. The automatic modes provide for transfers with Standard 7-bit addresses, Standard 7-bit address with 8-bit sub-address, and a Combined mode packet that first uses a 7-bit address to write an 8-bit sub-address followed by a repeated START condition and a Standard 7-bit address packet to the same slave to read or write that slave device.

The I^{2}C cell drives two electrically independent I^{2}C interfaces, however it can only drive one of the interfaces at a given time. A bit in the MODE register is used to select which interface the cell will use for transmission. The MODE register should only be programmed when the I^{2}C cell is IDLE. The cell is IDLE when the BUSY bit in the STATUS register is cleared.

8.3.2.2 CNTRL Register Usage

The CNTRL register (0xF8001010) is used to initiate and terminate actions on the I^{2}C interface. In manual mode, the START control bit is used to initiate a transfer, and the STOP control bit is used to end a write transfer. The AAK control bit is used to send a Not Acknowledge bit following the last byte of a read transfer, and a STOP condition will automatically be sent after the Not Acknowledge bit. Following the transmission of a START condition, the interface will pause in the BUSY state until a data byte is loaded (address, or write data) and the ISTART interrupt is cleared. The interface will then pause in the BUSY state, following the transfer of each data byte, until the IDATA interrupt is cleared. The STOP control bit is used to send a STOP condition on the bus and terminate a write transfer. In automatic address phase mode, the XADDR control bit is used to initiate a transfer on the I^{2}C interface. When the XADDR bit is set, all portions of an address phase will be sent in sequence on the I^{2}C interface, and the interface will then pause in the BUSY state until a data byte is loaded (on a write) and the IADDR interrupt is cleared. From this point forward the transmission continues as in a manual mode transfer. The interface will then pause in the BUSY state, following the transfer of each data byte, until the IDATA interrupt is cleared. The STOP control bit is used to send a STOP condition on the bus and terminate a write transfer.

8.3.2.3 STATUS Register Usage

The STATUS register (0xF8001020) allows the software to determine the current status of the selected I^{2}C interface. It shows whether the interface is busy or idle, the state of the last AAK bit that was transmitted or received, the state of the last R/W_ bit that was transmitted, and the current levels present on the SCL and SCA lines.

8.3.2.4 ISR & IER Register Usage

The ISR and IER registers (0xF8001030, 0xF8001040) are used to convey interrupt conditions to the control software. These interrupt conditions are used to orchestrate a transfer on the I^{2}C interface. The ISTART status bit is set when a START condition has been sent on the interface in manual mode. The IADDR status bit is set when an address phase has been completed in one of the automatic address phase modes. The ISTOP status bit is set when a STOP condition has been sent on the interface, either due to the STOP control bit being set in the CNTRL register, or the automatic termination of a transfer because a Not Acknowledge was received. The IDATA status bit is set whenever the I^{2}C cell is waiting for the latest read data byte to be unloaded from the DATA register or the next write data byte to be loaded into the DATA register. These four interrupt status bits will cause the Int_ output from the I^{2}C cell to be asserted if the corresponding enable bit in the IER is also set. The control software is therefore able to decide which conditions it wants to detect by interrupt or by polling of the various registers.

8.3.2.5 ADDR, SUBADDR, & DATA Register Usage

The ADDR register (0xF8001050) is used to store the 7-bit address and the R/W_ bit for the automatic address modes of operation. The SUBADDR register (0xF8001060) is used to store the 8-bit sub-address that is used in the Standard w/Sub-Address and Combined address phase modes. The DATA register (0xF8001070) is actually the I/O shift register that is used to hold data that is transmitted or received directly from the I^{2}C interface on the SCA line. The ADDR and SUBADDR registers should only be loaded when the I^{2}C interface is idle or transferring data bytes. The DATA register should be read or written when the I^{2}C interface is paused between byte transfers. The DATA register is used during address phases, so it cannot be preloaded before the automatic address phase is completed.

8.3.2.6 REV Register

The REV Register (0xF8001080) contains the 8-bit I^{2}C Cell Revision Number for the CPC945: 0xA2.

8.3.2.7 RISETIMECNT Register

The RISETIMECNT Register (0xF8001090) contains the 10-bit ISCL rise time count value. This register determines the time period that the I^{2}C cell waits after tristating the ISCL output before it stalls its bit time counter because of slave clock stretching. The value programmed into this register is the number of periods, minus one, of the I^{2}C CLK input that corresponds to the desired rise time period. The desired rise time period is the worst possible case, which is 110 ns for this design. The I^{2}C CLK input is the DDR2 clock input. For example if the DDR2 clock was running at 100 MHz, the register is initialized to 0x00A, representing 11 clock periods, or 110 ns. The initial values for a 333 MHz DDR2 clock (167 MHz Memory Controller Clock). These values should be changed by software if a higher DDR2 clock is used.

8.3.2.8 BITTIMECNT Register

The BITTIMECNT Register (0xF80010A0) contains the 10-bit I^{2}C bit time count value. The value programmed into this register is the number of periods, minus one, of the I^{2}C Clk input that corresponds to one fourth of the desired bit time period or 2.5 µs. The I^{2}C CLK input is the DDR2 clock input. For example if the DDR2 clock was running at 100 MHz, the register is initialized to 0x0F9, representing 250 clock periods, or 2.5 µs. The initial values are for a 333 MHz DDR2 clock (167 MHz Memory Controller Clock). These values should be changed by software if a higher DDR2 clock is used.

9. MPIC

9.1 Feature Summary

Features of the MPIC (Multiprocessor Programmable Interrupt Controller) are:

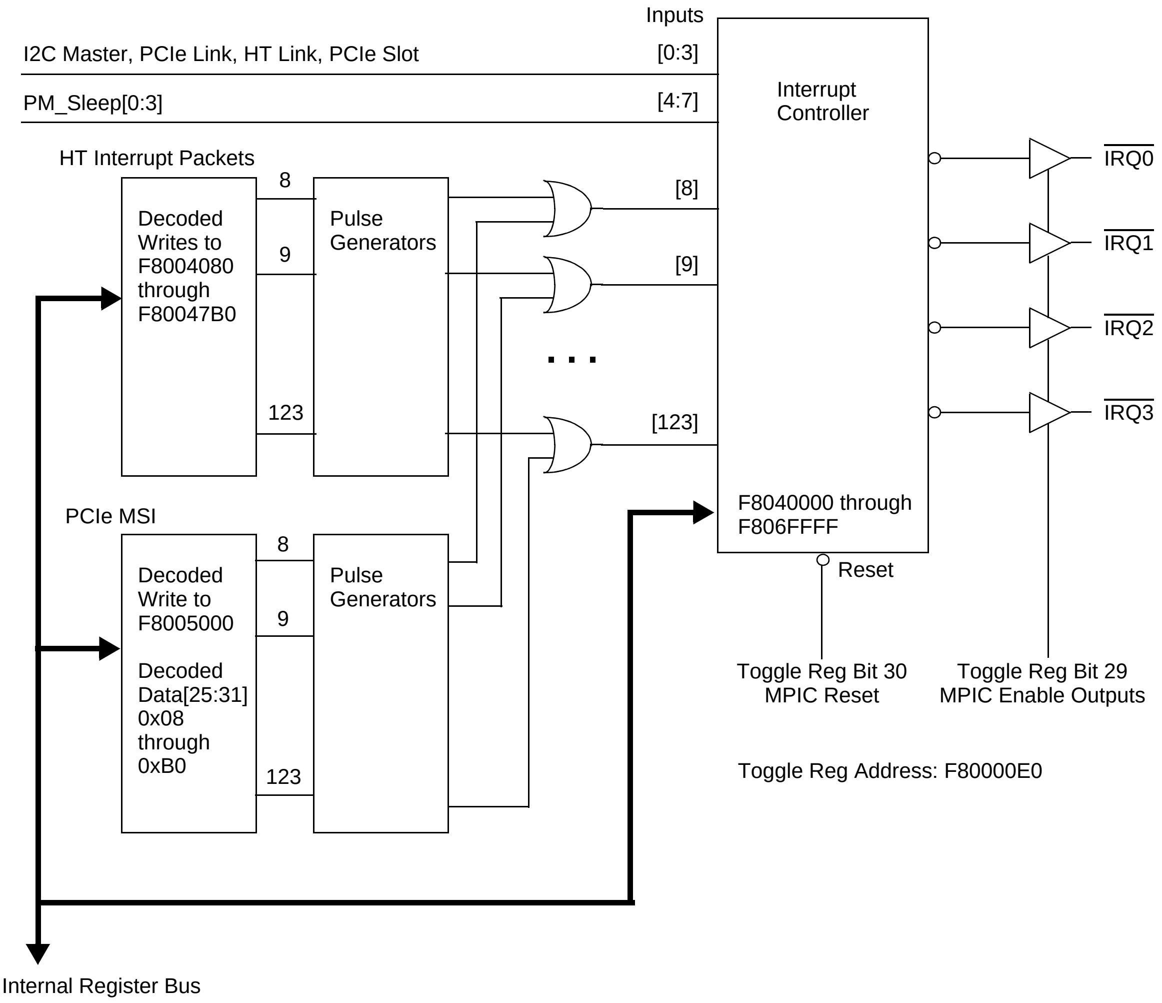
- A total of 124 I/O interrupt sources:
  - Eight internal interrupts
  - A total of 116 external I/O interrupts consisting of a mixture of HT posted write interrupts and PCIe message signaled interrupts
- Four IPI (interprocessor interrupts)
- Generates one external interrupt per CPU for a total of four
- The 124 I/O interrupts can each be programmed as distributed (sent to one of multiple CPUs using a fair distribution algorithm) or directed mode to a single specified processor.
- The interprocessor interrupts use directed mode, in which a given IPI can be directed to a single, specified CPU, or to multiple CPUs at the same time (multicast).
- I/O interrupt inputs are programmable as level sensitive or edge triggered. All level sensitive interrupts are active low. All edge triggered interrupts are positive edge triggered.
- Programmable interrupt source vector.
- Programmable interrupt source priority and processor current task priority.
- Fully nested interrupt support.
- Programmed using registers from 0xF8040000 through 0xF806FFFF. There are per CPU registers and global registers.
  - Reset and I/O Enable controlled by Toggle Register at 0xF80000E0.
  - HT posted write interrupts presented using register addresses 0xF8004xx0.
  - PCIe message signaled interrupts presented using register address 0xF8005000.
- Based on OpenPIC. *The Open Programmable Interrupt Controller (PIC) Register Interface Specification Revision 1.2*, Issue Date: October 1995, issued jointly by Advanced Micro Devices and Cyrix Corporation.

9.2 MPIC Organization

Figure 9-1 is a diagram of the MPIC which consists of the following:

- Interrupt input logic
- IRQ output drivers
- Programmable interrupt controller, an implementation of the OpenPIC specification

Figure 9-1. MPIC


Inputs
I2C Master, PCIe Link, HT Link, PCIe Slot
[0:3]
PM_Sleep[0:3]
[4:7]
Interrupt Controller
HT Interrupt Packets
8
9
123
Decoded Writes to F8004080 through F80047B0
Pulse Generators
[8]
[9]
. . .
[123]
$\overline{\text{IRQ0}}$
$\overline{\text{IRQ1}}$
$\overline{\text{IRQ2}}$
$\overline{\text{IRQ3}}$
F8040000 through F806FFFF
Reset
PCIe MSI
8
9
123
Decoded Write to F8005000
Decoded Data[25:31] 0x08 through 0xB0
Pulse Generators
Toggle Reg Bit 30 MPIC Reset
Toggle Reg Bit 29 MPIC Enable Outputs
Toggle Reg Address: F80000E0
Internal Register Bus

Note: Hypertransport Interrupt Packets to address FD_00xx0000 are converted to register writes to address F8004xx0

9.3 Interrupt Inputs

- Interrupts 0:3 are level, active low interrupts generated by miscellaneous units in the .
- Interrupts 4:7 are low-to-high edge trigger interrupts generated by the power manager.
- Interrupts 8:123 are low-to-high edge trigger interrupts generated by writes to specific register addresses (in support of HT posted write interrupts and PCIe message signaled interrupts).

Interrupts 0:3 must be programmed as level sensitive. Interrupts 4:127 must be programmed as edge triggered.

9.3.1 Interrupt 0, I2C Master

As discussed in *Section 8.3 I2C Master Interface* on page 275, the I2C Master can be enabled to generate an interrupt for several conditions using the I2C Controller Interrupt Enable Register (IER) at 0xF8001040. When an interrupt is generated, it is presented as an active low level to on interrupt input number 0. The interrupt must be cleared by writing to the appropriate bit in the I2C Controller Interrupt Status Register (ISR) at 0xF8001030. This will cause the signal presented to MPIC interrupt 0 to become high again.

9.3.2 Interrupt 1, PCIe Link Error

As discussed in *Section 5.1.2 Addressing* on page 97, the PCIe unit can signal an interrupt upon the detection of several error conditions. This interrupt is presented as an active low level interrupt on interrupt input number 1. The interrupt must be cleared by writing to the appropriate PCIe configuration register. See *Section 12.11 PCI Express Registers* on page 505 for more details on the PCIe link error interrupt.

9.3.3 Interrupt 2, HT Link

The Hypertransport unit can signal an interrupt for a variety of detected conditions. This interrupt is presented as an active low level interrupt on interrupt input number 2. The interrupt must be cleared by writing to the appropriate HT configuration register. See *Section 12.12.11 Error Handling/Enumeration Scratchpad Register (ErrCtrl/Enum)* on page 630 for more details on the HT link interrupt.

9.3.4 Interrupt 3, PCIe Slot

The original PCI bus supports four “slot” interrupts, INT A, B, C and D. PCIe supports these interrupts in the form of an upstream “message interrupt requester” transaction. The TAG HDW field denotes the interrupt source (A,B,C,D) and whether the interrupt is asserted or deasserted. When the PCIe logic receives a message interrupt request it drives interrupt input number 3. If the PCIe transaction is tagged as an asserted interrupt the interrupt input to MPIC is driven low (to present the interrupt), and if a deasserted interrupt is denoted the interrupt input to MPIC is driven high (to remove the interrupt). See *Section 12.11 PCI Express Registers* on page 505, for more details on the PCIe slot interrupt.

9.3.5 Interrupts 4:7, PM_Sleep[0:3]

The power manager contains a CPU Power Management Register at 0xF8000820. Bits 20:23 are the PM_Sleep bits [3:0]. These bits are set to 1 when it is safe to power off a given processor.

These 4 bits are also wired to interrupt inputs 4:7.
(Note the bit reversal: 0xF8000820 bits [20:23] = PM_Sleep[3:0] = interrupts 7:4)

It is intended that the interrupt be generated (if enabled) when the PM_Sleep bit changes from 0 to 1, therefore interrupts 4:7 should be programmed as edge triggered.

9.3.6 Interrupts 8:123, HT Posted Write Interrupts, and PCIe Message Signaled Interrupts

9.3.6.1 HT Posted Write Interrupts

Downstream hypertransport devices that generate interrupts signal these interrupts by sending an “interrupt request” packet upstream.

An interrupt request packet is a posted write sent to the address range 0xFD_0000_0000 through 0xFD_F8FF_FFFF. Interrupts 8 through 123 can be generated with these interrupt request packets.

The HT unit in the converts posted writes to the address range 0xFD_0000_0000 through 0xFD_F8FF_FFFF into writes to the address 0xF8004xx0. (Little-endian) Hypertransport address bits 23:16 become (big-endian) address bits 20:27. For example, HT posted writes to addresses 0xFD_00xx_0000 become writes to addresses 0xF8004xx0.

Addresses starting with 0xF8xxxxxx are reserved for the CPC945 internal control registers, so the CPC945 puts the converted writes on its internal register bus. MPIC has logic that responds to accesses on the internal register bus for addresses 0xF8004xxx. If the access is a write to addresses 0xF8004080 through 0xF800407B0, then a “one hot” decode is performed on address bits 20:27. The generated one hot signal drives a pulse generator which drives an interrupt input. Since address bits 20:27 are one hot decoded for the range 0x08 through 0x7B, an interrupt input from number 8 through number 123 is pulsed.

Writes in which address bits 20:27 are not in the range 0x08 through 0x7B are discarded.

The Hypertransport specification requires that interrupt packets have a count field of 0, meaning a data packet of 4 bytes follows. The 4 bytes of data indicate interrupt information. Additional interrupt information is contained in the interrupt request packet. The CPC945 discards all of this interrupt information, except for the fact that the address starts with 0xFD.

The Hypertransport specification describes an EOI packet that can be sent downstream to an interrupting device. The CPC945 does not have provisions for generating this EOI packet.

9.3.6.2 PCIe Message Signaled Interrupts

In addition to the “slot” interrupt described above (interrupt number 3), the PCIe specification has provision for “message signaled interrupts” (MSI). The CPC945 supports MSI by having PCIe devices write a DWORD (x86 4 bytes) to address 0xF8005000. Interrupts 8 through 123 can be generated with this write.

Addresses starting with 0xF8xxxxxx are reserved for the CPC945 internal control registers, so the CPC945 puts the write from PCIe on its internal register bus. MPIC has logic that responds to accesses on the internal register bus for address 0xF8005000. If the access is a write to addresses 0xF8005000, then a “one hot” decode is performed on (big-endian) data bits 25:31. The generated one hot signal for values 0x08 through 0x7B drives a pulse generator which drives an interrupt input. Therefore an interrupt input from number 8 through number 123 is pulsed.

Writes in which (big-endian) data bits 25:31 are not in the range 0x08 through 0x7B are discarded. Data bits 0:24 are discarded.

PCIe is little-endian and the CPC945 byte swaps internally to match the big-endian registers and memory. Therefore the PCIe DWORD write to (little-endian) bits 31:0 should use bits 30:24 to specify the generated interrupt.

9.3.6.3 Merging of HT and PCIe Interrupts

The interrupts 8 though 123 produced by HT posted writes are merged with the interrupts 8 through 123 produced by PCIe MSIs by bit-wise ORing to produce a single group of input interrupts 8 through 123 to the interrupt controller. Software is responsible for assigning HT and PCIe interrupts such that both devices do not share a given interrupt.

9.3.6.4 Generating Interrupts with Register Writes

Interrupts 8 through 123 are produced by performing writes to specified addresses in the CPC945 control register space. For HT the writes are done to 0xFD_00xx_0000 through 0xFD_F8xx_FFFF, which the CPC945 internally converts to address 0xF8004xx0. For PCIe this is simply a matter of writing directly to the specified address of 0xF8005000.

These HT and PCIe interrupts can be emulated in software by having any source with access to the CPC945 register space (such as the CPUs and I2C slave) perform a 4-byte write to the addresses used by the interrupt controller for HT and PCIe interrupts.

HT interrupts 8 through 123 can be emulated by a 4-byte write to addresses 0xF8004xx0, where xx is in the range 0x08 through 0x7B. The write data is “don’t care.”

PCIe interrupts 8 through 123 can be emulated by a 4-byte write to address 0xF8005000, with write data values of 0x00000008 through 0x0000007B.

A read from address 0xF8004xx0 returns all 0s. A read from address 0xF8005000 returns the last value written.

Note: For register address decoding, the interrupt controller hardware actually has only 2 fundamental decodes: 0xF8004xxx and 0xF8005xxx. (As described above the HT interrupt source logic further decodes address bits 20:27 [actually 21:27] as the decoded interrupt number.)

So, for example, accesses to 0xF8004000 are the same as accesses to 0xF8004004, and accesses to 0xF8005000 are the same as accesses to 0xF8005FFC. However, users should restrict access to 0xF8004xx0 and 0xF8005000 to guard against future changes.

9.3.7 Interrupt Input Summary

The CPC945 I^{2}C master is assigned to interrupt number 0. The PCIe link error interrupt is assigned to interrupt number 1. The HT link status interrupt is assigned to interrupt number 2. These interrupts are level sensitive.

The CPC945 power manager generates 4 interrupts using the PM_Sleep bits from the CPU Power Management Register at 0xF8000820. These are interrupts number 4 through 7, and they are edge triggered.

Hypertransport devices can generate interrupt numbers 8 through 123. Software must set up the HT devices to generate interrupt request packets directed to addresses 0xFD_0000_xx00, where xx = 0x08 through 0x7B (8 through 123). The generated interrupts are edge triggered.

PCIe devices can generate interrupts in 2 ways:

- Legacy interrupts (INT A, B, C, D) are supported. These are sent on PCIe using Message Transaction Packets. The CPC945 converts Messages of type “Interrupt” to a level sensitive, active low interrupt assigned as interrupt number 2.
- Message Signaled Interrupts are supported simply by having the PCIe device perform a posted x86 DW write to address 0xF8005000. Interrupts 8 through 123 are generated by setting the high order byte to values of 0x08 through 0x7B. The generated interrupts are edge triggered, low-to-high.

The 116 possible interrupts generated by HT interrupt packets are bit-wise ORed with the 116 possible interrupts generated by PCIe MSIs. Software is responsible for assigning these 116 interrupts (8 through 123) such any given interrupt is not assigned to both HT and PCIe.

Table 9-1. Interrupt Input Summary

Interrupt Number	Level or Edge	Source
0	Level	I2C Master
1	Level	PCIe Link Error
2	Level	HT Link Status
3	Level	PCIe “Slot” INT
4	Edge	PM_Sleep0 = CPU Pwr Mngt Reg[23]
5	Edge	PM_Sleep1 = CPU Pwr Mngt Reg[22]
6	Edge	PM_Sleep2 = CPU Pwr Mngt Reg[21]
7	Edge	PM_Sleep3 = CPU Pwr Mngt Reg[20]
8:123	Edge	Mixture of HT Interrupt Packet[8:123] and PCIe MSI[8:123]. Bit-wise ORed [8:123]. Software must manage.

9.4 Interrupt Outputs

The 4 interrupts driven off-chip to the CPUs use the same I/O circuits as used for the Processor Interfaces. The interrupts are high when not asserted, and low when asserted. The default output state is the tri-state condition; therefore *external pull-up resistors must be provided* so that the CPUs do not receive interrupts when the CPC945 interrupt drivers are disabled.

The tri-state enable is controlled by bit 29, MPICEnableOutputs of the Toggle Register at address 0xF80000E0. The default state of this bit is 0 (outputs disabled = tri-stated), therefore this bit must be set so that the MPIC can present interrupts to the CPUs.

The MPICEnableOutputs bit is intended to be used when putting the processors to sleep. Before the processors are put in sleep, interrupts from the CPC945 are blocked by setting the MPICEnableOutput bits to 0. While the processors are asleep, interrupts can continue to arrive and be processed by the MPIC. After the processors are woken up, the MPICEnableOutputs is set back to 1 to allow the interrupts to propagate to the CPC945 IOs and out to the CPUs.

If the MPIC has no interrupt to send to a given CPU it will drive the IRQ signal high. It drives the IRQ low when it receives an enabled interrupt. The IRQ stays low until software resets the interrupt by writing the appropriate register in MPIC. If no other interrupt is pending, the IRQ goes back to a high level. The IRQ stays low if some other interrupt is pending (or the original interrupt is not cleared).

9.5 Interrupt Controller

9.5.1 Global Reset/Enable

9.5.1.1 Toggle Register, MPICReset Bit

The interrupt controller has a reset input which is controlled by bit 30, MPICReset, of the Toggle Register at 0xF80000E0. When MPICReset is 0, the interrupt controller is held in a reset state. No interrupts are processed. MPICReset = 0 is the default state, so this bit must be set = 1 following chip reset so that the MPIC is enabled.

If the interrupt controller has been enabled and MPICReset is subsequently set = 0, all interrupt processing in progress is lost and all MPIC registers are set to their reset state. While MPICReset is set = 0, the MPIC registers cannot be accessed.

9.5.1.2 MPIC Global Configuration Register, Reset Controller Bit

Bit 31 (little-endian OpenPic = big-endian bit 0) of the MPIC Global Configuration Register at 0xF8041020 is the Reset Controller bit. Normally this bit is 0. If the bit is set to 1, MPIC is reset, and bit 0 is reset back to 0.

When the Reset Controller bit is set, all interrupts in process are lost and all MPIC registers are set to their reset state. Following the reset, the registers are available to be written and read.

9.5.1.3 MPIC Global Configuration Register, 8259 Pass Through Enable bit

Bit 29 (little-endian OpenPic = big-endian bit 2) of the MPIC Global Configuration Register at 0xF8041020 is the 8259 Pass Through Enable bit. OpenPIC specifies this bit = 0 to Enable 8259 Pass Through and = 1 to Disable 8259 Pass Through. The CPC945 does not support 8259 Pass Through so this bit must be set = 1 to disable the pass through feature and allow MPIC to process its 124 interrupts sources.

The reset value of this bit = 0, so following a CPC945 reset this bit must be set = 1 to enable interrupts.

9.5.2 Global Enable Summary

Following a CPC945 reset, external interrupts from the CPC945 to the PowerPC processors are disabled. In addition to enabling individual inputs (both in MPIC and in the interrupting devices), the following bits must be set to globally enable interrupts:

- MPICReset bit in the Toggle Register at 0xF80000E0.
- MPICEnableOutputs bit in the Toggle Register at 0xF80000E0.
- 8259 Pass Through Enable bit MPIC Global Configuration Register at 0xF8041020 (= disable pass through).

9.5.3 Registers

Only a general description on the MPIC registers is provided here. For MPIC register details, see *Section 12.6 MPIC Registers* on page 359.

Note: OpenPIC uses little-endian notation for register bits 31:0, whereas PowerPC uses big-endian notation [0:31]. The bit notation for the register descriptions in *Section 12.6 MPIC Registers* on page 359 is little-endian, to match that of OpenPIC.

9.5.3.1 OpenPIC

The OpenPIC Register Interface Specification lists requirements for two groups of registers: Per CPU registers and Global registers. The Per CPU registers have the quality that a given register has multiple instances, one per CPU, but only one address. When a CPU accesses a Per CPU at its specified address the instance unique to that processor is written or read.

The Per CPU registers are: IPI Dispatch Command Ports, Current Task Priority, Who Am I (a single register which returns different results depending on which processor reads the register; optional for PowerPC), Interrupt Acknowledge (optional for x86, required for PowerPC), and End Of Interrupt (EOI).

The Global registers are: Feature Reporting (register 0 defined, register 1 reserved), Global Configuration (register 0 defined, register 1 reserved), Vendor Specific (optional), Vendor Identification, Processor Initialization, IPI Vector/Priority, Spurious Vector, Global Timer (Timer Frequency, Global Timer 0, 1, 2, 3 Current Count, Global Timer 0, 1, 2, 3 Base Count, Global Timer 0, 1, 2, 3 Vector/Priority), and Interrupt Source (Vector/Priority, and Destination).

9.5.3.2 OpenPIC Compliance

The CPC945 MPIC implements:

- The Per CPU registers: IPI Dispatch Command Ports, Current Task Priority, Interrupt Acknowledge, End of Interrupt. (Note: With the exception of the IPI Dispatch Command Ports, these registers deviate from the OpenPIC specification, because the multiple instances of a given register each have unique addresses.)

  The Who Am I register is also implemented, four instances at a single address, but the register is not in MPIC. It is a general control register at 0xF8000050.

- The Global registers: Feature Reporting Register 0, Global Configuration Register 0, Vendor Identification, Spurious Vector, Interrupt Source. Note that the Interrupt Source registers come in pairs (Vector/Priority, and Destination).

OpenPIC allows for a implementation specific number of interrupt inputs up to a total of 2048. The CPC945 MPIC implements 124 interrupt inputs.

OpenPIC allows for an implementation specific number of processor, up to a total of 32. The CPC945 MPIC implements interrupts to 4 processors.

9.5.3.3 Deviations from the OpenPIC Specification

- The Global Timers are not implemented because the equivalent function can be found on the PowerPC processors. Therefore there are no Timer Frequency, Global Timer 0, 1, 2, 3 Current Count, Global Timer 0, 1, 2, 3 Base Count, Global Timer 0, 1, 2, 3 Vector/Priority Registers.

- The Per CPU Processor Initialization register is implemented but has no useful function. This register is intended to drive initialization lines to the processors, but the PowerPC 970xx does not implement these signals. (OpenPIC also specifies that writing to the Processor Initialization register to restart a CPU will reset the current task priority of that CPU to 15 to inhibit interrupts to that CPU. The CPC945 MPIC implements this function.)

- OpenPIC assigns register addresses in the form xFCaaaaa, where “x” is a base address. The value of x is obtained from the Base Address Relocation bits in the Global Configuration Register 0, bits 19:0. The default value of bits 19:0 = 000F. With this value the default register address value (for a 32-bit address space) becomes FFCaaaaa.

  The CPC945 MPIC uses register addresses that fall within the F8aaaaaa space assigned to all CPC945 control registers. The Base Address Relocation bits are not used. An attempt is made to preserve the lower 12 or 16 address bits to match those of OpenPIC:

  - The Global registers (except for Interrupt Source) with OpenPIC addresses xFC01aaa are assigned to CPC945 addresses F8041aaa.

  - The Interrupt Source registers with OpenPIC addresses xFC10000 through xFC100F70 are assigned CPC945 addresses F8050000 through F8050F70. The Vector/Priority registers use addresses 0xF8050000, F8050020, F8050040, ... F8050F60, whereas the Destination registers use addresses 0xF8050020, F8050030, F8050050, ... F8050F70.

  - The per CPU registers with OpenPIC addresses xFC00040 through xFC000B0 are assigned CPC945 addresses F8060040 through F80600B0.

- OpenPIC provides for backward compatibility with SLiC dual processor interrupt controller by providing shadow copies of the IPI 0 registers. The IPI Dispatch Command register at xFC00040 is shadowed at xFC00000. The IPI Vector 0 register at xFC010A0 is shadowed at xFC00008.

  The CPC945 MPIC does not support shadows of the IPI 0 registers.

- For the Per CPU registers, there are multiple instances of a given register, one instance per CPU. OpenPIC specifies 2 addresses per instance. One address is common to all CPUs, the other address is unique to a CPU. Address bits 12:19 are used to distinguish among the address. For all CPUs to access a given register using the same address, address bits [12:19] = 0x00. For the CPUs to use unique addresses, address bits [12:19] = 0x20, 0x21, 0x22, ... 0x3F (for CPUs 0, 1, 2, ... 31).

  As noted above the CPC945 only supports unique addresses for four instances of each register. The CPC945 uses address bits 16:19 to address the separate registers.

  - The four IPI Command Dispatch Registers at OpenPIC addresses xFC00040, 50, 60, 70 becomes sixteen CPC945 registers at addresses F8060040, 50, 60, 70, F8061040, 50, 60, 70, F8062040, 50, 60, 70 and F8063040, 50, 60, 70. (But there are only 4 registers. See *Section 9.5.12 Interprocessor Interrupts* on page 294.)

  - The Current Task Priority Register at OpenPIC address xFC00080 becomes four CPC945 registers at addresses F8060080, F8061080, F8062080 and F8063080.

  - The Interrupt Acknowledge Register at OpenPIC address xFC000A0 becomes four CPC945 registers at addresses F80600A0, F80610A0, F80620A0 and F80630A0.

  - The End of Interrupt (EOI) Register at OpenPIC address xFC000B0 becomes four CPC945 registers at addresses F80600B0, F80610B0, F80620B0 and F80630B0.

  The WhoAmI Register which is implemented on the CPC945 outside of MPIC, has just the single address of F8000050 for all CPUs.

A comparison of OpenPIC registers to CPC945 registers is shown in *Table 9-2*.

Table 9-2. Register Summary

Register	OpenPIC Address		CPC945 Address	Page
Feature 0	xFC01000		F8041000	360
Global Configuration 0	xFC01020		F8041020	361
Vendor Identification	xFC01080		F8041080	362
Processor Initialization	xFC01090		F8041090	362
IPI 0 Vector/Priority IPI 1 Vector/Priority IPI 2 Vector/Priority IPI 3 Vector/Priority	xFC010A0 xFC010B0 xFC010C0 xFC010D0		F80410A0 F80410B0 F80410C0 F80410D0	363
Spurious Vector	xFC010E0		F80410E0	363
Interrupt Source	xFC10000 - xFC10F60		F8050000 - F8050F70	364
IPI 0 Dispatch Command	xFC00040 or	xFC20040 (CPU 0) xFC21040 (CPU 1) xFC22040 (CPU 2) xFC23040 (CPU 3)	F8060040 (CPU 0) F8061040 (CPU 1) F8062040 (CPU 2) F8063040 (CPU 3)	366
IPI 1 Dispatch Command	xFC00050 or	xFC20050 (CPU 0) xFC21050 (CPU 1) xFC22050 (CPU 2) xFC23050 (CPU 3)	F8060050 (CPU 0) F8061050 (CPU 1) F8062050 (CPU 2) F8063050 (CPU 3)	366
IPI 2 Dispatch Command	xFC00060 or	xFC20060 (CPU 0) xFC21060 (CPU 1) xFC22060 (CPU 2) xFC23060 (CPU 3)	F8060060 (CPU 0) F8061060 (CPU 1) F8062060 (CPU 2) F8063060 (CPU 3)	366
IPI 3 Dispatch Command	xFC00070 or	xFC20070 (CPU 0) xFC21070 (CPU 1) xFC22070 (CPU 2) xFC23070 (CPU 3)	F8060070 (CPU 0) F8061070 (CPU 1) F8062070 (CPU 2) F8063070 (CPU 3)	366
Current Task Priority	xFC00080 or	xFC20080 (CPU 0) xFC21080 (CPU 1) xFC22080 (CPU 2) xFC23080 (CPU 3)	F8060080 (CPU 0) F8061080 (CPU 1) F8062080 (CPU 2) F8063080 (CPU 3)	367
Who Am I	xFC00090 or	xFC20090 (CPU 0) xFC21090 (CPU 1) xFC22090 (CPU 2) xFC23090 (CPU 3)	F8000050	334
Interrupt Acknowledge	xFC000A0 or	xFC200A0 (CPU 0) xFC210A0 (CPU 1) xFC220A0 (CPU 2) xFC230A0 (CPU 3)	F80600A0 (CPU 0) F80610A0 (CPU 1) F80620A0 (CPU 2) F80630A0 (CPU 3)	368
End Of Interrupt (EOI)	xFC000B0 or	xFC200B0 (CPU 0) xFC210B0 (CPU 1) xFC220B0 (CPU 2) xFC230B0 (CPU 3)	F80600B0 (CPU 0) F80610B0 (CPU 1) F80620B0 (CPU 2) F80630B0 (CPU 3)	369

9.5.4 Interrupt Setup

As discussed above, the MPIC must be globally enabled by setting the MPICReset and MPICEnableOutputs bits in the Toggle Register at 0xF80000E0 and the MPIC Global Configuration Register at 0xF8041020. To fully set up the MPIC:

- The mask, vector and priority fields of the interrupts must be set. The Vector/Priority registers of the Interrupt Source Register pairs for the I/O interrupts; the IPI Vector/Priority registers for the Interprocessor Interrupts, and the Spurious Vector Register. For I/O interrupts the sense bit must also be set.
- For each CPU, the priority level of the task running on that CPU must be set in the corresponding Current Task Priority Register.
- For the I/O interrupts, the destination registers of the Interrupt Source Register pairs must be set.

9.5.4.1 Mask Bits

Each of the 124 I/O Interrupt Source Vector/Priority registers and each of the 4 IPI Vector/Priority registers contains a Mask bit. If the Mask bit equals ‘1’ the interrupt is disabled; if the Mask bit equals ‘0’ the interrupt is enabled. Following reset, all Mask bits that equal ‘1’ are disabled; therefore, the software must set the Mask bits equal to ‘0’ for all interrupts that are to be enabled.

9.5.4.2 Sense Bits

Each of the 124 I/O Interrupt Source Vector/Priority registers contains a Sense bit. This bit must be set equal to ‘0’ for edge triggered interrupts and equal to ‘1’ for level sensitive interrupts.

9.5.4.3 Vectors

Vector numbers are assigned by the programmer to the individual interrupt sources. When an interrupt is taken the software must read the Interrupt Acknowledge Register. MPIC will copy the vector field from the Vector/Priority register belonging to the interrupt source and return it as Interrupt Acknowledge Register read data. Using the returned vector number the software knows which source caused the interrupt.

As discussed in *Section 9.5.4.4* there might be interrupts pending from multiple sources. MPIC will return the vector from the source that has the highest priority in its Vector/Priority Register.

The vector fields are 8 bits. With 124 I/O + 4 IPI + 1 spurious interrupt = 129 sources, and 2^8 (256) possible vector values, there are enough vector values to uniquely identify each interrupt source.

9.5.4.4 Priorities

Priority fields are 4 bits, having values from 0 (lowest priority) to 15 (highest priority). Interrupt sources are assigned priorities by software using the Interrupt Source Vector/Priority Registers and IPI Vector/Priority Registers. Each CPU also has a task priority assigned by software using the Current Task Priority Registers.

The priority fields have two purposes:

- Interrupts can be blocked if they are “not more important” than the current task being executed.
- If multiple sources cause an interrupt, once the interrupt is taken software is given the vector of the source that is the “most important.”

When an interrupt event occurs, the interrupt from that source becomes “pending.” A pending interrupt might or might not be asserted on the IRQ signal, depending on the priority level of that source, The priority field of the source (from the corresponding Vector/Priority Register) is compared to the priority field of the Current Task Priority register. (One register for distributed mode or directed mode to a single processor; more than one register for direct mode multicast) If the source priority is greater than the task priority then an interrupt will be asserted; if the source priority is less than or equal to the task priority then the interrupt is not asserted.

Between the time that an interrupt is asserted on the IRQ signal until the software reads the Interrupt Acknowledge Register, interrupts from multiple sources might have become pending. When the Interrupt Acknowledge Register is read, the vector that is returned is the vector corresponding to the source with the highest priority.

If there are multiple pending interrupts that have the same highest priority, one of them is selected using a “tie breaker” algorithm.

If a CPU is assigned a task priority of 15 it can never be interrupted by MPIC because no source can have a higher priority than 15.

If an interrupt source is assigned a priority of 0 it can never interrupt, because 0 can never be higher than the lowest task priority of 0.

IPI Priority levels have restrictions. See *Section 9.5.12.1 IPI Priority Level Restrictions* on page 295.

9.5.5 Changing the Interrupt Setup

The contents of the Current Task Priority Registers can be changed at any time.

For the interrupt sources, it is not safe to change the contents of the Interrupt Source Registers or IPI Vector Priority Registers while the interrupts are unmasked. In order to change the vector, priority, sense, or destination of an active (unmasked) interrupt source, the following sequence should be performed:

- Mask the source using the MASK bit in the vector/priority register
- Wait for the activity bit (ACT) for that source to be cleared
- Make the desired changes
- Unmask the source

9.5.6 Interrupt Sequence

As explained above, when an enabled interrupt event happens (an I/O source programmed as level sensitive is low; an I/O source programmed as event triggered goes from low to high; an IPI Dispatch Command port is written), the interrupt for the given source becomes “pending.”

If MPIC determines that the pending interrupt should be asserted on an IRQ (based on source priority, task priority, and destination field (for I/O) or destination write data (for IPI)), the interrupt is said to be “dispatched.”

As explained above, when a CPU responds to an IRQ assertion it is required to read the Interrupt Acknowledge Register and obtain the interrupt vector. When a CPU reads the Interrupt Acknowledge Register, a pending interrupt is said to be “delivered.” The interrupt event that had been pending is now “in-service.” The IRQ signal is deasserted.

Also as explained above, interrupts from multiple sources might have been pending. It is the pending interrupt with the highest priority that changes from pending to in-service; the lower priority interrupts remain pending.

For an interrupt that is in-service, the software is required at some point (at the end of the interrupt routine) to write the End Of Interrupt (EOI) Register. This resets the “in-service” status of the interrupt and allows pending lower priority interrupts (if any) to assert IRQ for a new interrupt.

(As explained below in “Spurious Interrupts” the EOI Register must *not* be written if the returned vector is the Spurious vector. This is because spurious interrupts do not become in-service.)

Notes:

1. It is not legal for a CPU to access a Per CPU register that does not “belong” to it. For example, CPU 1 must not read the Interrupt Acknowledge Register for CPU 0 (or 2 or 3), and it must not write the EIO Register for CPU 0 (or 2 or 3).
2. Level sensitive interrupts will continue to appear to be “pending” as long as the are asserted (low).
3. OpenPIC (and MPIC) builds on concepts from the original 8259A interrupt controller. The 8259A allowed for EOI commands to be “specific” (terminating a given in-service interrupt) or “non-specific” (terminating the highest level in-service interrupt of a fully nested stack). Write data of all 0’s is non-specific; non-zero data is specific. OpenPIC requires a fully nested stack, therefore the EOI command must be non-specific; therefore the write data must be all zeros.
4. The x85 and x86 processors used with the 8259A generate an Interrupt Acknowledge bus cycle when an interrupt is serviced. The 8259A responds to this type of bus cycle by automatically returning the interrupt vector, and internally moving from the “dispatched” state to the “in-service” state. This is why OpenPIC specifies that the Interrupt Acknowledge Register is optional for x86 interrupt controllers based on OpenPIC.

   The PowerPC 970xx has no Interrupt Acknowledge type of bus cycle; this is why OpenPIC requires (and MPIC implements) the Interrupt Acknowledge Register and the need for the software to read this register to get the interrupt vector and move the interrupt from the dispatched state to the in-service state.

9.5.7 Nesting of Interrupt Events

As required by OpenPIC, MPIC supports a fully nested interrupt mechanism.

Consider a pending interrupt which is dispatched. IRQ to a CPU is asserted. The CPU reads the Interrupt Acknowledge Register. The interrupt has been delivered. IRQ to the CPU is deasserted.

Some time later the CPU writes the EOI. But before that happens, an interrupt of higher priority might become pending. If this happens, MPIC will assert IRQ. Once again, the CPU is required to read the Interrupt Acknowledge Register. The second, higher priority interrupt is delivered. IRQ is deasserted. From a CPU viewpoint, a higher priority interrupt has preempted the lower priority interrupt.

There are now 2 interrupts that are in-service. If an even higher priority interrupt becomes pending then it will preempt the second interrupt and 3 interrupts will be in-service. Thus, a fully nested stack of interrupts can be in-service, from lowest priority to highest priority.

A CPU will never have an in-service interrupt preempted by an equal or lower priority source.

When the EOI Register is written it will reset the in-service status of the highest priority interrupt of the nested stack. (This is why the EOI write data must be zero. The OpenPIC does not permit a “specific” in-service interrupt in the stack to be terminated. Only the non-specific (highest priority) in-service interrupt can be terminated.)

As usual, multiple EOI writes must be performed, one for each Acknowledged interrupt, to terminate all of the in-service interrupts in the stack.

9.5.8 Spurious Interrupts

As discussed above, a pending interrupt becomes dispatched at some time; at a later time the software reads the Interrupt Acknowledge register to learn the vector of this pending interrupt. But it is possible that by the time the Interrupt Acknowledge register is read, there is no valid vector to return. In this circumstance, the MPIC will return the contents of the Spurious Interrupt Register.

Following are the cases that will cause the spurious vector to be returned:

- Interrupt request is asserted in response to a level triggered interrupt source which is deasserted before interrupt acknowledge.
- Interrupt request is asserted for an interrupt source which is masked by an increase in current task priority level for the interrupted processor before interrupt acknowledge.
- Interrupt request is asserted for an interrupt source which is masked using the mask bit in the source configuration register before interrupt acknowledged.

For correct operation the EOI register should not be written in response to the spurious vector.

In all cases the spurious vector will not be returned if there is another pending interrupt that has sufficient priority to interrupt that processor. If such an interrupt is available, the vector for that source will be returned.

9.5.9 Delivery Modes

The concept of delivery modes is a consequence of having interrupt request for multiple processors. There are two interrupt delivery modes: distributed and directed.

9.5.9.1 Directed Mode

For directed mode, the interrupt is delivered to a specific processor (single destination) or to multiple processors (multicast) as specified by the destination field value.

9.5.9.2 Distributed Mode

For distributed mode, interrupt events from a particular source are distributed among a group of processors specified by the destination field value, using a fair distribution algorithm.

9.5.9.3 Exactly Once Delivery

Distributed mode interrupts are delivered “exactly once.” This means that an interrupt event will never be pending or active on more than one processor at a time and will be delivered only once to the selected destination processor. Also, once an interrupt is dispatched to a particular destination, further events from that source will not be dispatched to any other processor until processing of the original event is terminated by an EOI.

9.5.10 Processor Identification

The WhoAmI register returns a value that is unique to the processor reading the register. This value must be used by the processors when accessing MPIC to determine which of the multiple instances of the Per CPU registers to use, for setting the destination bits in the Interrupt Source Destination registers, and for writing the destination data for IPI Command Dispatch Port access.

9.5.11 I/O Interrupts

The 124 I/O interrupts have 124 interrupt source register pairs in which the second register of the pair is the Destination Register. The destination register has 4 bits corresponding with the 4 processors.

For a given interrupt, if only a single bit is set in the destination register than that interrupt uses directed mode.

If more than one bit is set, then distributed mode is used, with the interrupts distributed among the processors whose corresponding bits are set in the destination register.

9.5.12 Interprocessor Interrupts

Interprocessor interrupts are available for one processor to interrupt another processor. They can also be used for a processor to interrupt itself (self interrupt).

Four IPI “channels” are available (four IPI interrupts that can have different vectors and priorities). The four channel vectors and priorities are set in the four IPI Vector/Priority registers.

Interprocessor interrupts events are created by writing to one of the four IPI Command Dispatch Command ports (one Command Dispatch port per channel).

In the general case, the Per CPU registers have multiple instances of a given register, one per processor. As specified by OpenPIC this is not true for the IPI Dispatch Command Registers. For the this means:

- Addresses F8060040, F8061040, F8062040 and F8063040 all map to the same register.
- Addresses F8060050, F8061050, F8062050 and F8063050 all map to the same register.
- Addresses F8060060, F8061060, F8062060 and F8063060 all map to the same register.
- Addresses F8060070, F8061070, F8062070 and F8063070 all map to the same register.

Multiple IPI events generated on a selected processor’s incoming channel will not be queued. The vector and priority for each of the IPI channels will be the same for all processors (the values programmed in the IPI Vector/Priority registers).

Interprocessor interrupts always use Directed Mode. The IPI Command Dispatch ports used to generate an IPI also use the write data to serve as the destination field. Writing to an IPI Command Dispatch port creates an interrupt event; four bits of write data specify to which processor(s) the event is directed. If only one bit is written, then the interrupt is directed to a single destination - the processor corresponding to the bit that is written. If more than one bit is written than the interrupt is multicast - directed to the multiple processors corresponding to the bits written.

9.5.12.1 IPI Priority Level Restrictions

The priority value programmed for each active IPI channel must be a unique priority on a per CPU basis.

- For a given CPU, the priority levels for IPI channels targeted to that CPU must be distinct from the priority levels used for all other interrupt sources.

- Multiple IPIs can be programmed to the same priority levels if they specify mutually exclusive CPU targets (when writing the destination data to the IPI Command Dispatch ports).

10. System Initialization Sequence

10.1 Introduction

This chapter gives an overview of power on considerations and the actions required to boot a system with a CPC945 bridge and memory controller.

In general, the steps required to fully initialize the CPC945 are:

1. Enable power to the system.
2. Enable the clock generators and PLLs of the CPC945.
3. Release reset to the CPC945.
4. Initialize the CPC945.
5. Perform the processor interface alignment procedure.
6. Configure the HyperTransport bus and boot Flash and NVRAM.

10.2 Power Sequencing

While the CU-11 process used for CPC945 has no power sequencing requirements, experience has shown that a module power sequencing requirement exists and should be followed before any attempt is made to communicate with the CPC945 over the slave I^2C interface. Correct I^2C slave interface operation depends on stable CPC945 core, processor I/O, and CPC945 I/O voltages.

The CPC945 has separate power rails for the core logic and I/O buffers. The core power rail should be raised and stabilized prior to raising the I/O power rails. Additionally, the reset signal to the CPC945 chip should be asserted the whole time that the power rails are being raised and stabilized. No voltage should be applied to an I/O pad if the associated power supply is not on.

10.3 Power-On Reset

The CPC945 signals that are used in the power-on reset process are described in *Table 10-1*.

Table 10-1. CPC945 System Support Signal Pins

Signal Name	Signal Description	Signal Type
NORTH_BRIDGE_RESET_L	Hardware reset for the CPC945. This active-low signal is the system hardware reset input to the CPC945. This is the hard-reset signal.	Input
SUSPENDREQ_L	Suspend request. This active-low signal is sent from the power management unit (under software control) to the CPC945 to request that the device stop all activity and enter the suspend (sleep) state.	Input
SUSPENDACK_L	Suspend acknowledgement. The CPC945 asserts this active-low signal back to the power management unit to indicate the suspension request is complete.	Output
HT_PWROK	HyperTransport power-OK signal.	I/O
HT_RESET_L	Active-low HyperTransport reset signal.	I/O
HT_LDTSTOP_L	Active-low HyperTransport power down request signal.	I/O

10.3.1 Hardware Reset Sequence

The following procedure outlines the steps required to bring a system through the initial power-on reset (POR) sequence:

1. The SPU powers up all devices and subsystems, asserting all resets (assert SUSPENDREQ_L and assert NORTH_BRIDGE_RESET_L).
2. In response to reset, the CPC945 resets and forces internal PLL clock generators to bypass, asserts SUSPENDACK_L, holds all the buses in the sleep state, deasserts HT_PWROK, and asserts HT_RESET_L. The CPC945 operates off of its bypass clocks during this phase of the bring up.
3. At this point in the reset sequence the SPU can start the PC board system clock generators, including the 66 MHz clocks to the HyperTransport.
4. Now the SPU waits an appropriate time for the power and clocks to stabilize.
5. Next the SPU deasserts NORTH_BRIDGE_RESET_L to the CPC945, and the resets to the HyperTransport Tunnel, and the southbridge.
6. In response to the deassertion of NORTH_BRIDGE_RESET_L the CPC945 stops using its bypass clocks, and starts using its phase-locked loops (PLL), and internal clocking structures.
7. Now the SPU can deassert SUSPENDREQ_L to the CPC945.
8. When all of its PLLs and clocks are running and SUSPENDREQ_L is deasserted the CPC945 starts waking its buses and the memory controller, and starts an internal reset state machine procedure.
9. After one millisecond, the reset state machines in the CPC945 asserts HT_PWROK, the reset state machine then waits another millisecond, and deasserts HT_RESET_L.
10. When the CPC945 detects HT_PWROK asserted, and both HT_RESET_L and HT_LDTSTOP_L deasserted, the CPC945 starts the HyperTransport interface.
11. Next the CPC945 signals to the SPU the deassertion of SUSPENDACK_L.

12. When all southbridge and HyperTransport tunnel PLLs and clocks and HyperTransport busses are running, the SPU can then deassert $\overline{\text{HT\_RESET\_L}}$ to the CPC945, the HyperTransport tunnel and the southbridge.
13. With the deassertion of $\overline{\text{HT\_RESET\_L}}$ the HyperTransport tunnel and southbridge start coming out of reset and the HyperTransport bus transmitters go active.
14. The SPU releases the processor resets and the processors start booting up.

10.3.2 CPC945 Initialization

After completion of the initial hardware reset sequence, the service processor begins initialization of the CPC945 through the I^{2}C slave interface. This I^{2}C interface can be used by the SPU to generate read or write cycles to any address in the system. The System Command Registers allow the I^{2}C slave interface to provide read and write access to any 36-bit CPC945 physical address.

The initialization consists of the following steps:

1. First write the Clock Control Register for the appropriate core speed and then the PLL1 Control Register to reflect the proper configuration for the attached processors via the PI interface. Note that the CPC945 PLLs are preloaded at reset time to default values that are not necessarily optimized for a specific system configuration. The service processor (SPU) can adjust the PLL settings by writing to the PLLn Control Register. For more detail see *Section 10.3.3 CPC945 Clocking Initialization* on page 300.
2. Next through the SPU I^{2}C slave interface disable all exceptions by programming the Processor Interface Exception Mask Register. See *Section 12.11.3.1 CORE_X: PI Core Interface Parameters Register* on page 586.
3. Also via the SPU I^{2}C slave interface disable exceptions in the Chip Fault Mask Register. See *Section 12.11.3.1 CORE_X: PI Core Interface Parameters Register* on page 586.
4. After disabling all exceptions, train the PI interface. This involves setting up the appropriate alignment of data and clocks at the physical layer of the interface. See *Section 4.1 Processor Interface Alignment Procedure* on page 78.
5. Once the PI has been trained, set up the Processor Interface (PI) Bus timing parameters. An understanding of the card delays is required to set up the appropriate bus delays. These parameters include snoopacc, snooplat, snoopwin, paamwin, and statlat. See *Section 4.1.1 Determining PI Bus Parameters* on page 79 on determining bus timing parameters.
6. After the PI Bus timing parameters are programmed set up the PI core interface parameters. The core interface registers configure the size of the various queues for the cpu and I/O. See *Section 4.1.4 API Programming Procedure* on page 84.
7. Next Initialize the HyperTransport Interface. See *Section 6.2.1 Programming the HyperTransport core* on page 117.
8. After initializing the HyperTransport, initialize the PCI Express (optional).
9. Once the initialization of the PCI Express is complete initialize the DDR2 interface. See *Section 7.7.2 Memory Controller Bring-up Summary* on page 136.
10. Next enable all exceptions using the Processor Interface Exception Mask Register and the Chip Fault Mask Register.
11. The CPC945 is now ready to function in the system.

10.3.3 CPC945 Clocking Initialization

The default power on reset values of the clocks are shown in *Section 12.5.6 PLL1 Control Register*, *Section 12.5.7 PLL2 Control Register*, *Section 12.5.8 PLL3 Control Register*, and *Section 12.5.9 PLL4 Control Register*.

The PLL control registers can be accessed through the SPU I^{2}C slave interface. Each control register allows control of the PLL associated dividers, tune bits, as well as the R, F, L, and P control bits.

The Power Management Clock (PMR_Clk) comes directly into the chip through the I/O (300 MHz) with no PLL clock multiplication. Consequently no PLL control is required for this clock tree.

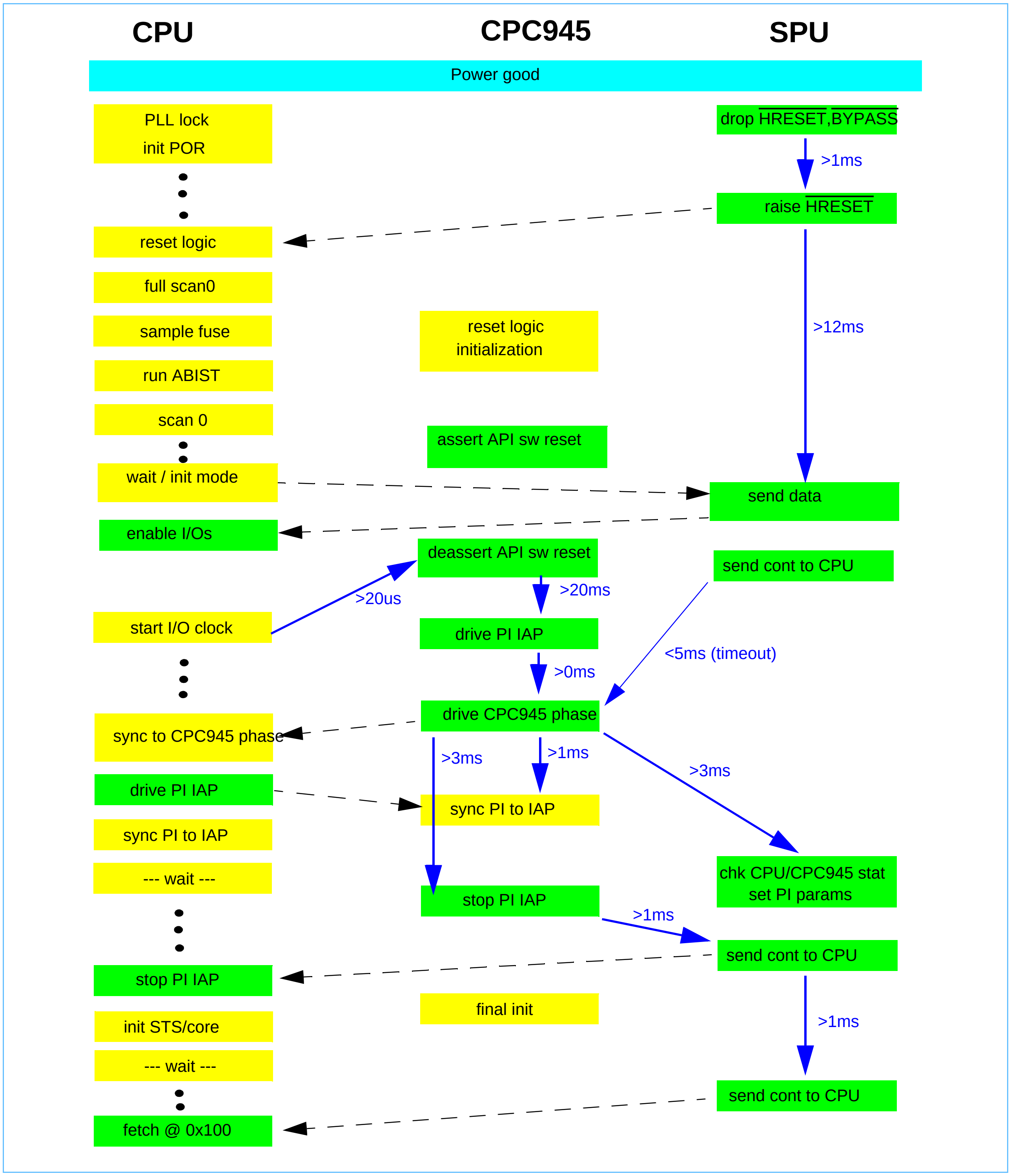
PLL1 Control Register controls the behavior of the Processor Interface PLL clock generator. This register is located at system address 0xF8000850. Setup values for this control register are dependent on the system configuration. For more detail on the specific setup values given the system configuration, see *Section 12.5.6 PLL1 Control Register* on page 346.

PLL2 Control Register controls the behavior of the DDR2 SDRAM PLL clock generator. This register is located at system address 0xF8000860. For typical usage it should not be necessary to change any of the initial bit values in this register. However if it is necessary to change values in this register see a more detailed specification of this control register, fields, and steps to alter their value in *Section 12.5.7 PLL2 Control Register* on page 350.

PLL3 Control Register controls the behavior of the PCI Express PLL clock generator. This register is located at system address 0xF8000870. For typical usage it should not be necessary to change any of the initial bit values in this register. However if it is necessary to change values in this register see a more detailed specification of this control register, fields, and steps to alter their value in *Section 12.5.8 PLL3 Control Register* on page 352.

PLL4 Control Register controls the behavior of the HyperTransport PLL clock generator. This register is located at system address 0xF8000880. Setup values for this control register are dependent on the system configuration. For more detail on the specific setup values given the system configuration see *Section 12.5.9 PLL4 Control Register* on page 355.

Figure 10-1. CPC945 Power On Reset Procedure



11. Power Management and Clocks

11.1 Introduction

In order to optimize the electrical power consumption and thermal performance of computer systems built with the CPC945, there is specific power management logic in the bridge that switches off clocks to various parts of the chip when those parts are not needed. There is also logic to control the speeds at which different interfaces operate, allowing additional power savings and configuration control. In addition to saving power in the CPC945, these modes save significant system power by managing and optimizing the power used by the PowerPC 970xx family processor.

11.2 System Power Management

The CPC945 Power Manager (PMR) logic can be used to control the idling of all buses and internal operations when the 970-based system goes in and out of the system sleep state. This is typically done in tandem with another system unit such as a Power Management Unit (PMU) or Service Processor or System Management Unit (SMU). The CPC945 can be programmed to implement a two-wire handshake interface consisting of the external signals SUSPENDREQ_L and SUSPENDACK_L. Using these signals and the bits in the PwrSystem Register, CPC945 safely suspends all internal operations, quiesces all external buses, and places the memory subsystem in the self-refresh state.

11.2.1 CPC945 and Processor State Definitions

Table 11-1. CPC945 and Processor Power Management State Definitions.

	Run	Doze	Nap	DeepNap	Power Off
Processor Core	Active, Dynamic Clock Stopping Logic	Clocks Partially Stopped	Clocks Stopped	Clocks Stopped	Power Down Single Core (on 970MP multiprocessor only)
Caches	Active	Snoop On, Caches Preserved	Snoop Off, Caches Preserved	Snoop Off, Caches Preserved	Powered Off
CPU Timers/Interrupts/PLLs/IOs/Pervasive/ MachineCheck	Active	Active	Active	Active	Powered Off
CPC945	Active	Active	Active	On, Retains State	Off
DRAM	Active	Active for DMA Traffic	On, Self-Refresh	Off	Off
Frequency Scaling	Full (F), Half (F/2), Quarter (F/4)	Full (F), Half (F/2), Quarter (F/4)	Full (F), Half (F/2), Quarter (F/4)	PowerSave (F/64)	Powered Off
Coherency Logic	Active	Active	Off, QAck transition to Doze	Off	Powered Off
Snoop Penalty	No	No	Yes	No	No

Note:

1. In values listed in the *Frequency Scaling* column, ’F’ is frequency.
2. In *Frequency Scaling/DeepNap* cell, F/64 applies only to the processor, not to the CPC945.

Figure 11-1 shows the basic logic flow used during the run to sleep transition.

Figure 11-1. Run to Sleep Transition

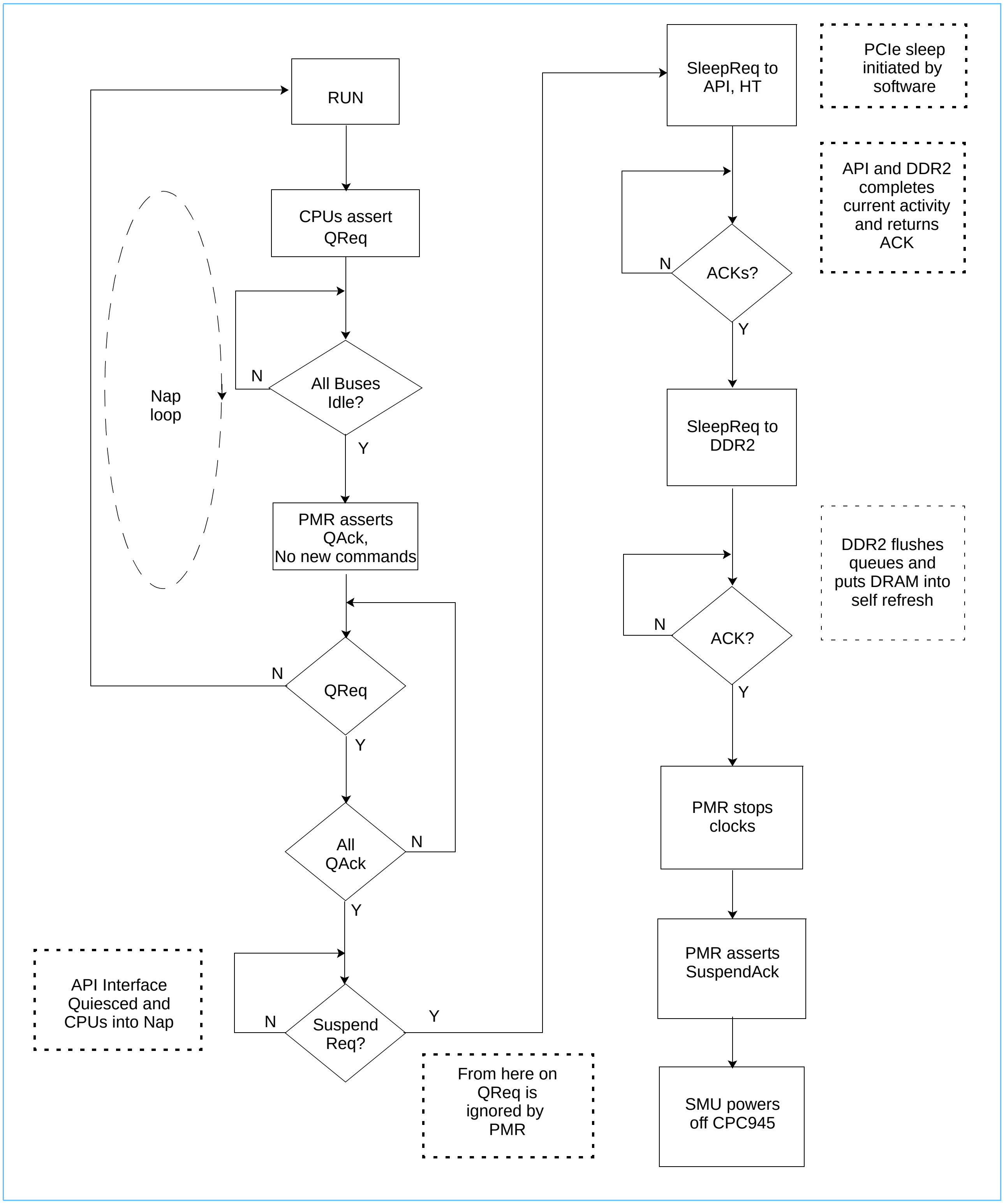
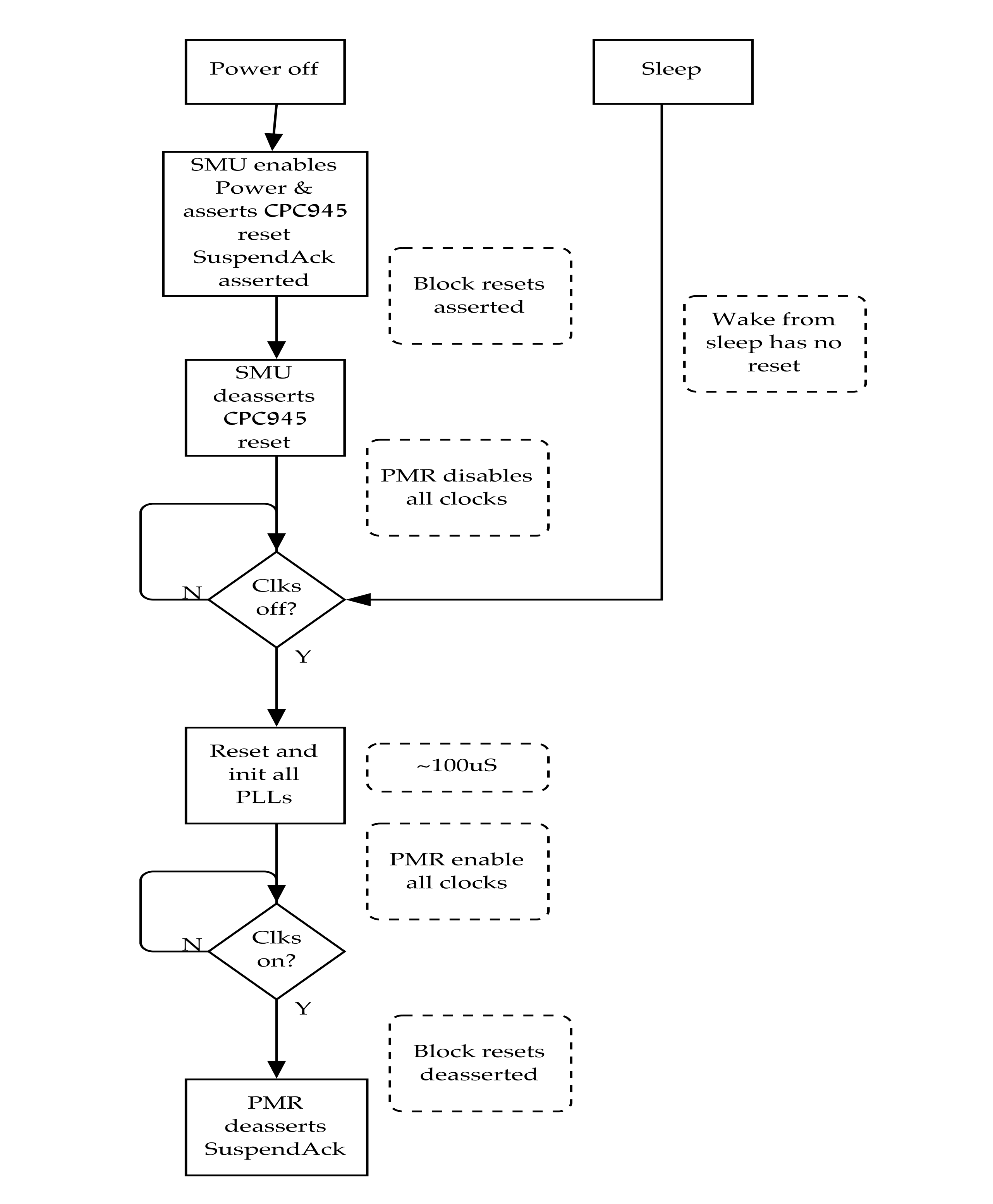

RUN
CPUs assert QReq
Nap loop
N
All Buses Idle?
Y
PMR asserts QAck, No new commands
N
QReq
Y
All QAck
N
Y
API Interface Quiesced and CPUs into Nap
N
Suspend Req?
Y
From here on QReq is ignored by PMR
SleepReq to API, HT
PCIe sleep initiated by software
N
ACKs?
Y
API and DDR2 completes current activity and returns ACK
SleepReq to DDR2
N
ACK?
Y
DDR2 flushes queues and puts DRAM into self refresh
PMR stops clocks
PMR asserts SuspendAck
SMU powers off CPC945

Figure 11-2 show the matching logic flow used for the wake sequence.

Figure 11-2. Wake Sequence



11.2.2 CPC945 Top Level Power Manager

The CPC945 top level power manager controls the logic used to coordinate activity that occurs between CPC945 and the rest of the system when moving from one power management state to another.

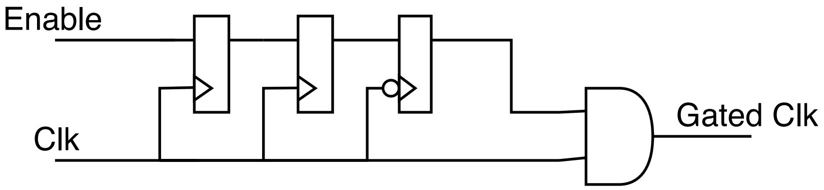
11.2.3 PLLs

The clocks in CPC945 are generated by on-chip PLLs. These PLLs are used for two functions; to multiply the input clock frequency to a higher frequency and to remove clock tree insertion delay on the interfaces where insertion delay is important. The four PLLs in CPC945 generate all the clocks for the chip. Each PLL is surrounded by a power management wrapper, which controls the PLL through reset, system sleep, and software-enabled dynamic power management. All four PLL[n] Control registers (which are a part of this wrapper) are accessible via memory-mapped accesses and I^2C accesses. I^2C access is required for the cases where the SPU must modify a PLL's configuration before releasing the processors from reset.

11.2.4 Clock Stoppers

In order to save power, the clocks to most parts of CPC945 can be stopped. The CPC945 PMR uses clock stoppers to synchronously stop and control the various clocks at the base of the clock trees. Additionally, the PMR uses clock stoppers to isolate the PMR logic from the PLL outputs during times when the PLL or input reference clock is not stable. *Figure 11-3* illustrates the logic typically used for clock stoppers.

Figure 11-3. Clock Stopper Logic



The first two flops synchronize the enable to the clock the third flop generates a gating signal that changes only during the clock low time, so that the clock gating does not create any clock glitches.

Since the stopping and starting of clocks causes significant changes in the current consumed by the CPC945, there is a clock stopping sequencer which allows only one clock to start or stop within a given time interval. *Figure 11-3* shows the basic logic in the PMR clock stop sequencer.

11.3 CPU Power Management

11.3.1 CPU Power Manager

Historically, PowerPC processors had three power management states: doze, nap, and sleep. The PowerPC 970xx processor implements doze and nap, but does not implement the sleep state. In sleep mode, the processors caches have been flushed and it is not required to participate in data coherency operations.

The CPC945 simulates sleep for the PPC970xx processor. Sleep is the same as nap as far as the PowerPC 970xx is concerned, but when sleep is entered, software first flushes the caches, so that it is not necessary to move a processor that is in the sleep state to doze to do snooping. The sleep state is exited by either using a reset or an interrupt. The system power manager should be able to remove the power from a processor and put the CPC945 into the sleep state, which will save the leakage power. If the power is removed, it will be necessary to reinitialize the processor interconnect interface when power is restored.

Implementing the sleep state in the power manager of the CPC945 is straightforward. The per-processor sleep bit is used to indicate that a particular processor should enter the Sleep state when all required operations for that processor have completed, and one QAck signal is provided for each processor. The Sleep mode is the same as the Nap mode as far as the processor is concerned. The only difference is in Sleep mode, the CPC945 does not de-assert QAck when snooping is required since the processor in the Sleep state has empty caches. When a processor is in the Sleep state, it can be powered down. Each processor has a PwrDnEnabled bit. A CPU’s PM_SLEEP bit should be asserted whenever the processor’s sleep state machine is in the “sleeping” state and the PwrDnEnabled bit is set.

Since the CPC945 has multiple Processor Interconnect interfaces and supports multiple CPUs per interface, generation of QAck is not as simple as it might seem. This is further compounded by the fact that the QReq and QAck signals are asynchronous to the operation of the Processor Interconnect interface. Each CPU has its own state machine for controlling the Quiesce state of that CPU. Each CPU is capable of independently moving in and out of the Nap and Sleep states without explicit involvement by other CPUs. The CPC945 takes advantage of idle periods on the bus to signal QAck to a requesting processor. This requires holding other bus traffic for the time it takes the CPU to move in and out of the low power state.

Figure 11-4. CPU Power Manager (1 of 2)

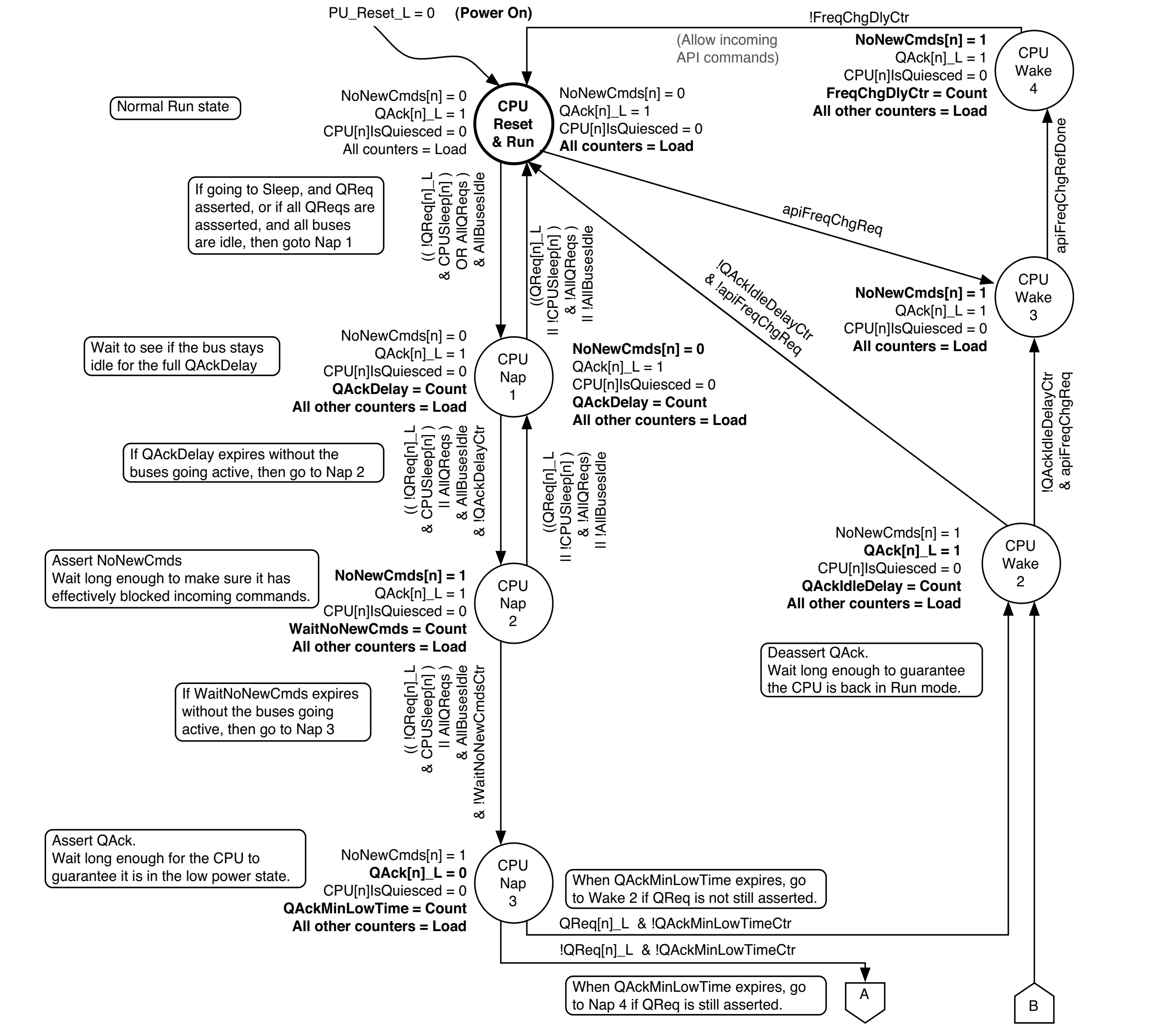
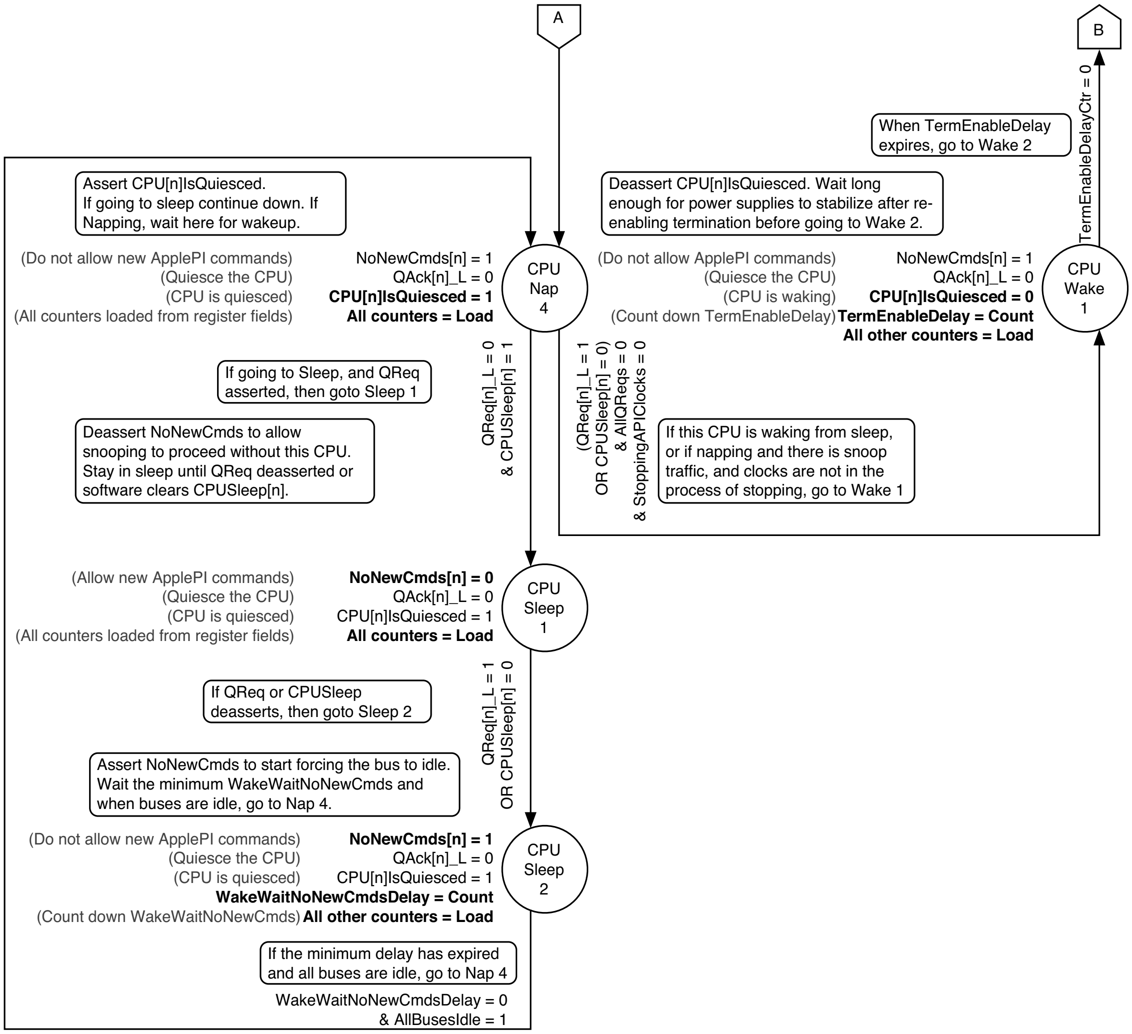

PU_Reset_L = 0
(Power On)
!FreqChgDlyCtr
(Allow incoming API commands)
Normal Run state
NoNewCmds[n] = 0
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
All counters = Load
CPU Reset & Run
NoNewCmds[n] = 0
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
All counters = Load
NoNewCmds[n] = 1
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
FreqChgDlyCtr = Count
All other counters = Load
CPU Wake 4
apiFreqChgRefDone
apiFreqChgReq
If going to Sleep, and QReq asserted, or if all QReqs are assserted, and all buses are idle, then goto Nap 1
((!QReq[n]_L & CPUSleep[n]) OR AllQReqs) & AllBusesIdle
((QReq[n]_L II !CPUSleep[n]) & !AllQReqs) II !AllBusesIdle
!QAckIdleDelayCtr & !apiFreqChgReq
NoNewCmds[n] = 1
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
All counters = Load
CPU Wake 3
Wait to see if the bus stays idle for the full QAckDelay
NoNewCmds[n] = 0
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
QAckDelay = Count
All other counters = Load
CPU Nap 1
NoNewCmds[n] = 0
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
QAckDelay = Count
All other counters = Load
If QAckDelay expires without the buses going active, then go to Nap 2
((!QReq[n]_L & CPUSleep[n]) II AllQReqs) & AllBusesIdle & !QAckDelayCtr
((QReq[n]_L II !CPUSleep[n]) & !AllQReqs) II !AllBusesIdle
!QAckIdleDelayCtr & apiFreqChgReq
NoNewCmds[n] = 1
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
QAckIdleDelay = Count
All other counters = Load
CPU Wake 2
Assert NoNewCmds
Wait long enough to make sure it has effectively blocked incoming commands.
NoNewCmds[n] = 1
QAck[n]_L = 1
CPU[n]IsQuiesced = 0
WaitNoNewCmds = Count
All other counters = Load
CPU Nap 2
Deassert QAck.
Wait long enough to guarantee the CPU is back in Run mode.
If WaitNoNewCmds expires without the buses going active, then go to Nap 3
((!QReq[n]_L & CPUSleep[n]) II AllQReqs) & AllBusesIdle & !WaitNoNewCmdsCtr
Assert QAck.
Wait long enough for the CPU to guarantee it is in the low power state.
NoNewCmds[n] = 1
QAck[n]_L = 0
CPU[n]IsQuiesced = 0
QAckMinLowTime = Count
All other counters = Load
CPU Nap 3
When QAckMinLowTime expires, go to Wake 2 if QReq is not still asserted.
QReq[n]_L & !QAckMinLowTimeCtr
!QReq[n]_L & !QAckMinLowTimeCtr
When QAckMinLowTime expires, go to Nap 4 if QReq is still asserted.
A
B

Figure 11-5. CPU Power Manager (2 of 2)


A
B
When TermEnableDelay expires, go to Wake 2
TermEnableDelayCtr = 0
Assert CPU[n]IsQuiesced. If going to sleep continue down. If Napping, wait here for wakeup.
Deassert CPU[n]IsQuiesced. Wait long enough for power supplies to stabilize after re-enabling termination before going to Wake 2.
(Do not allow new ApplePI commands)
(Quiesce the CPU)
(CPU is quiesced)
(All counters loaded from register fields)
NoNewCmds[n] = 1
QAck[n]_L = 0
CPU[n]IsQuiesced = 1
All counters = Load
CPU Nap 4
(Do not allow new ApplePI commands)
(Quiesce the CPU)
(CPU is waking)
(Count down TermEnableDelay)
NoNewCmds[n] = 1
QAck[n]_L = 0
CPU[n]IsQuiesced = 0
TermEnableDelay = Count
All other counters = Load
CPU Wake 1
If going to Sleep, and QReq asserted, then goto Sleep 1
QReq[n]_L = 0
& CPUSleep[n] = 1
(QReq[n]_L = 1
OR CPUSleep[n] = 0)
& AllQReqs = 0
& StoppingAPIClocks = 0
Deassert NoNewCmds to allow snooping to proceed without this CPU. Stay in sleep until QReq deasserted or software clears CPUSleep[n].
If this CPU is waking from sleep, or if napping and there is snoop traffic, and clocks are not in the process of stopping, go to Wake 1
(Allow new ApplePI commands)
(Quiesce the CPU)
(CPU is quiesced)
(All counters loaded from register fields)
NoNewCmds[n] = 0
QAck[n]_L = 0
CPU[n]IsQuiesced = 1
All counters = Load
CPU Sleep 1
If QReq or CPUSleep deasserts, then goto Sleep 2
QReq[n]_L = 1
OR CPUSleep[n] = 0
Assert NoNewCmds to start forcing the bus to idle. Wait the minimum WakeWaitNoNewCmds and when buses are idle, go to Nap 4.
(Do not allow new ApplePI commands)
(Quiesce the CPU)
(CPU is quiesced)
(Count down WakeWaitNoNewCmds)
NoNewCmds[n] = 1
QAck[n]_L = 0
CPU[n]IsQuiesced = 1
WakeWaitNoNewCmdsDelay = Count
All other counters = Load
CPU Sleep 2
If the minimum delay has expired and all buses are idle, go to Nap 4
WakeWaitNoNewCmdsDelay = 0
& AllBusesIdle = 1

There is one CPU Power Manager for each of the four CPUs the CPC945 supports. Any CPU Power Manager is capable of stalling all incoming transactions on all CPC945 Processor Interconnect interfaces. The Processor Interconnect PHY block takes several actions when the bus is quiesced:

- The Processor Interconnect PHY must not issue any snoop transactions to its outgoing bus. Instead, an encoded zero must be sent on the outbound bus. This must be a static state as clocks to the Processor Interconnect PHY logic might be removed.
- The Processor Interconnect PHY must not accept or respond to any incoming traffic on its incoming bus. This includes not generating any error conditions based on incoming bus signals as the incoming bus might be tristated.
- The Processor Interconnect PHY must present a static snoop response of null to the internal snoop logic. This must be a static state as clocks to the Processor Interconnect PHY logic might be removed.

In addition, the Processor Interconnect PHY has a signal that tells it to tristate the output drivers on the outbound bus. This signal is asserted when all CPUs on the bus have the CPUPwrDnEnabled[n] bits set in the PwrCPU Register, the TristateEnable bit is set in the PwrCPU Register, and all CPUs are in the sleep state.

11.3.2 Processor Interconnect Power Manager

The power management logic for the Processor Interconnect busses works with the clock control logic to support dynamic clock stopping as well as the safe stopping and starting of clocks as the system enters and exits the sleep state. The Processor Interconnect Power Manager can reduce power consumption by removing clocks from circuitry not in use. Specifically, it can dynamically remove clocks going to the Processor Interconnect interface when it is quiesced. To do this, the power management logic must ensure QAck is not removed while Processor Interconnect is stopping its clocks. Likewise, the power management logic must restart clocks whenever any source wants to awaken from the Quiesced state.

11.3.2.1 PLL1

PLL1 generates the bus rate and DDR2 bit rate clocks used by the Processor Interconnect Interface. The bus rate is a maximum of 625 MHz and the DDR2 bit rate clock is twice this frequency, or 1250 MHz. This PLL accepts a reference clock (EI_PLL_CLK) that is at the bus rate (for CPUs with a 3:1 or 4:1 ratio between the core clock and the DDR2 bit rate) or 1/2 the bus rate (for CPUs with a 2:1 ratio between the core clock and the DDR2 bit rate). PLL1 multiplies this clock to generate the required frequencies.

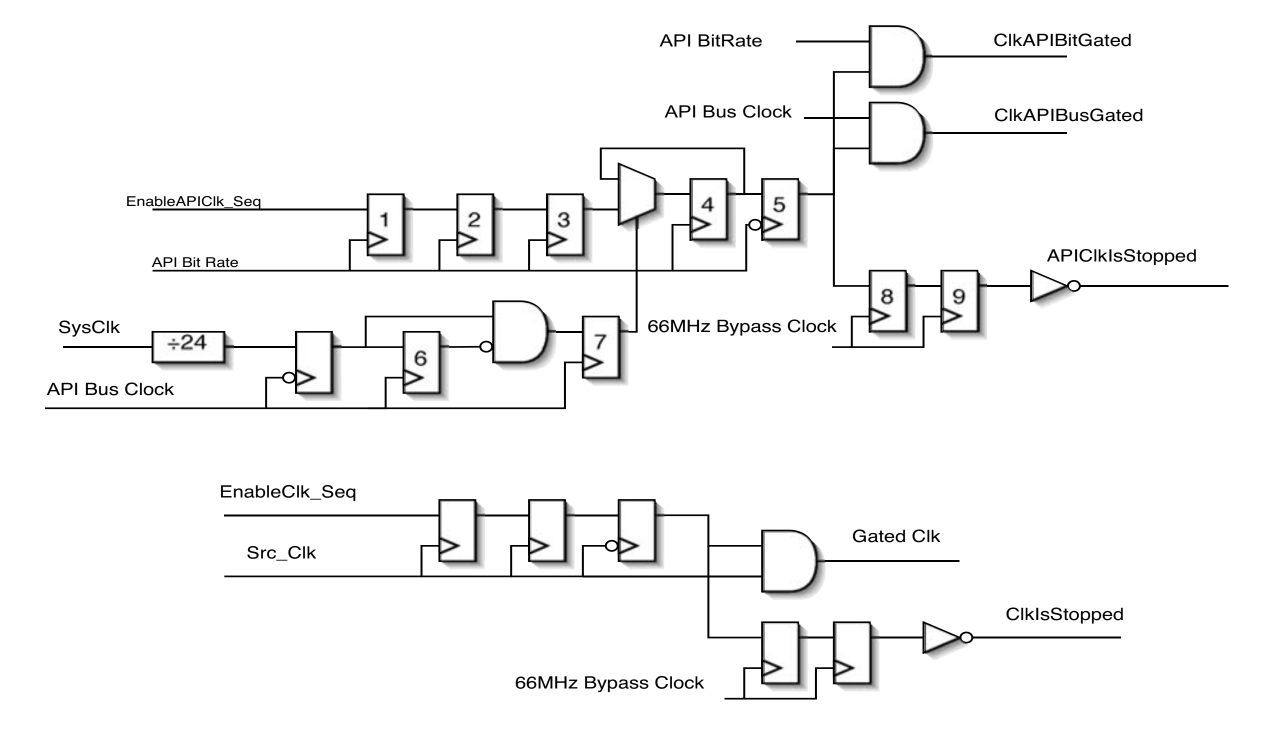
PLL1 is required to support spread spectrum clocking. The reference clock for PLL1 is spread at rates between 10 kHz and 40 KHz, ±1.25%.

After power-on-reset, the service processor unit (SPU) uses I^2C to configure PLL1. This is required due to the variance of bus speed with processor speed. PLL1 requires different settings to support different speed CPUs and different reference clock to CPU ratios (the PowerPC 970xx core runs at either six or eight times the reference clock input). To support this requirement, the PLL1 Control Register Values are available for reading and writing via the I^2C slave port and the memory-mapped register space.

In order to power manage the PowerPC 970xx processor it is necessary to dynamically change the PPC970xx core frequency. The processor frequency ranges between full speed and half speed. Since the core frequency is directly tied to the reference clock within the PPC970xx, the reference frequency changes one octave under software control. The clock chip limits the rate of frequency slewing to be slower than the rate used by spread spectrum clocking. At its fastest, spread spectrum moves 2.5% in 16.67 microseconds, so the fastest clock chip will move 50% (from full to half speed) in 333 microseconds. PLL1 is required to track the reference frequency moving at this rate.

Because PLL1 has certain limits, it is necessary to change the settings of PLL1 in order to enable full octave frequency shifts for a variety of processor speeds and CPU to Bus ratios. To support these changes, all PLL1 control settings are brought out to the PLL1 Control Register. This register is used by software to set PLL1 appropriately for the frequency it is being fed as a reference. Since changing the controls to the PLL breaks the output clock's lock, it is necessary to define a safe mechanism for these signals to be updated.

Figure 11-6. PLL1 Clock Stopper



During system operation, the PLL settings cam be set dynamically. If the APILogicStopEnable and EnablePLL1Shutdown bits are set in the Clock Control Register then the Processor Interconnect clocks are stopped and the PLL is shut down when all four processors are quiesced. So, in order to load new values into the PLL, first software loads the proper values into the PLL1 Control Register with the ForcePLLReset and ForcePLLLoad bits set to zero. Then all the CPUs in the system are quiesced. When the system wakes from sleep, the PLL is updated with the new value. Software verifies that the update has occurred by checking the PLLLoaded bit in PLL1Control. If software clears the PLLLoaded bit when programming the PLL1Control register, then software can tell the update occurred by checking the PLLLoaded bit later. PLLLoaded is set to a one by the hardware when the PLL updates the configuration latch.

One effect of the update procedure for PLL1 is that the clocks to the Processor Interconnect interface are completely stopped. This means that the outgoing source-synchronous clocks to the PPC970xx are stopped. The PPC970xx uses these clocks to maintain certain clock relationship information which is critical to proper operation of Processor Interconnect. In order not to break the interface, it is necessary that the number of dropped clocks occupy an amount of time exactly equal to a multiple of twenty-four times the processor interconnect reference clock. The circuit in *Figure 11-6* shows how this is accomplished. The logic to do this works with inputs and outputs synchronized to the 66 MHz bypass clock while controlling clocks synchronous to the processor interconnect bus clock.

The values set by software in the PLL1Control Register are translated to be intelligible. See *Section 12.5.6 PLL1 Control Register* on page 346.

11.3.3 DDR2 Power Management (PLL2)

PLL2 generates the CPC945 core clock. This clock is used to generate the clock that runs the DDR2 SDRAM controller and all the inter-block connection paths.

The core clock is synchronous to the DDR2 SDRAM interface clock. Since different systems have different SDRAM interface speeds, the core/SDRAM clock rate is controllable externally.

Software can reconfigure PLL2 in order to change the core speed to match the operating speed of the memory system. However, this must be done at power-up time before the memory controller has been activated.

The initial state of the PLL2 operating point is set by interpreting the values of the CoreSpeed[2:0] pins which are latched at Reset. The mappings are shown in *Table 12-6 PLL2 DDR2 Core Speed* on page 349.

11.3.4 PCI Express Power Management

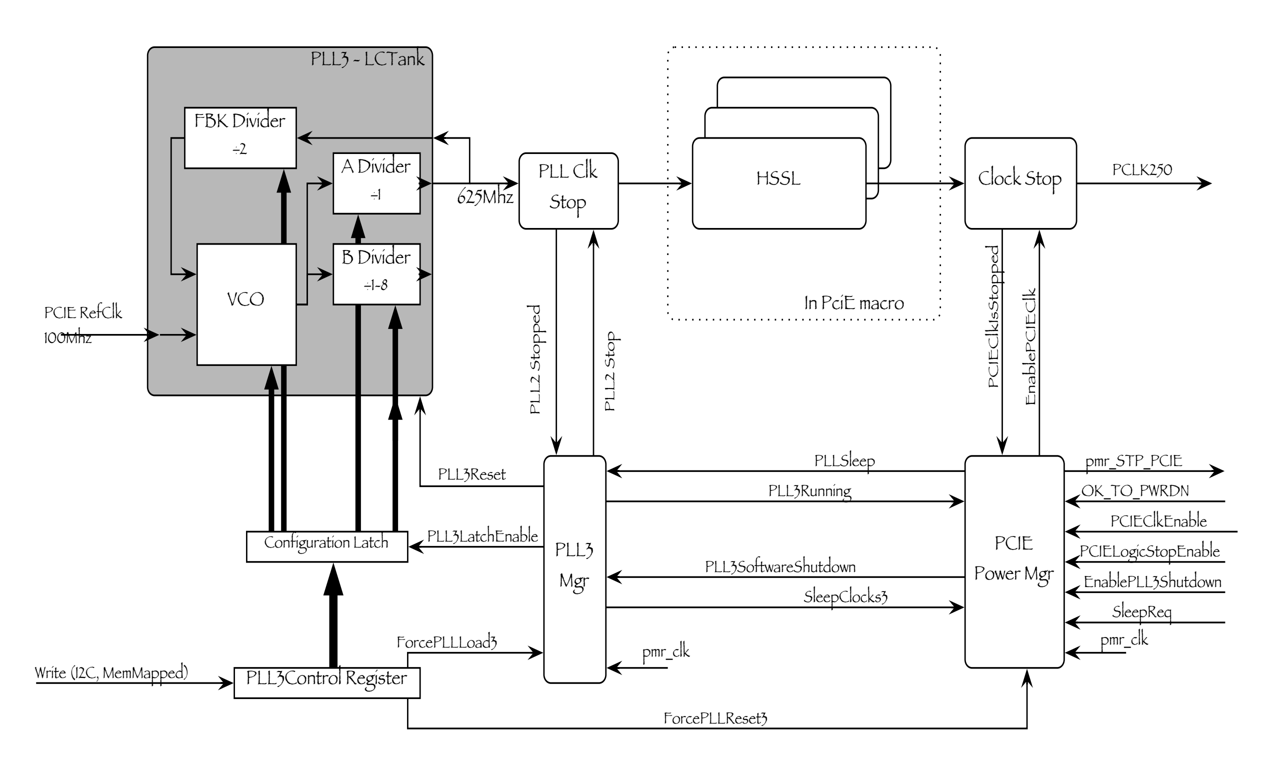
In order to save power, it is possible to quiesce the PCI Express bus using the power management features of the PCI Express architecture. The PCI Express bus architecture defines power management states which include D0 through D3. D0 state is the normal operating mode for the bus and represents the highest power consumption mode for PCI Express. D3 state is the quiesced state at which power can be removed from the PCI Express logic. If the PCI Express bus is quiesced and reaches D3 state, then the CPC945 is capable of both gating off clocks to the PCI Express logic and stopping PLL3.

Note that the PMR logic has no direct means to place the PCI Express logic into the D3 state to be ready for the power down state. Software must write the appropriate control registers to place the PCI Express logic in the power down state. This means that for sleep wake, the system software/firmware must bring down the PCI Express interface prior to asserting the SUSPENDREQ_L signal to the CPC945.

11.3.4.1 PLL3

The PCI Express clock is actually generated by a cascade of two PLLs. The first PLL is the Intermediate Frequency PLL which is in the PMR and referred to as PLL3. PLL3 has an input reference clock of 100 MHz which is used to generate a 625 MHz output clock. The 625 MHz clock is used as a reference clock for the second level PLL. The second PLL is in the PCI Express HSSL macros and generates a 250 MHz clock which is used by the protocol stack. *Figure 11-7* illustrates the two PLLs and associated control logic:

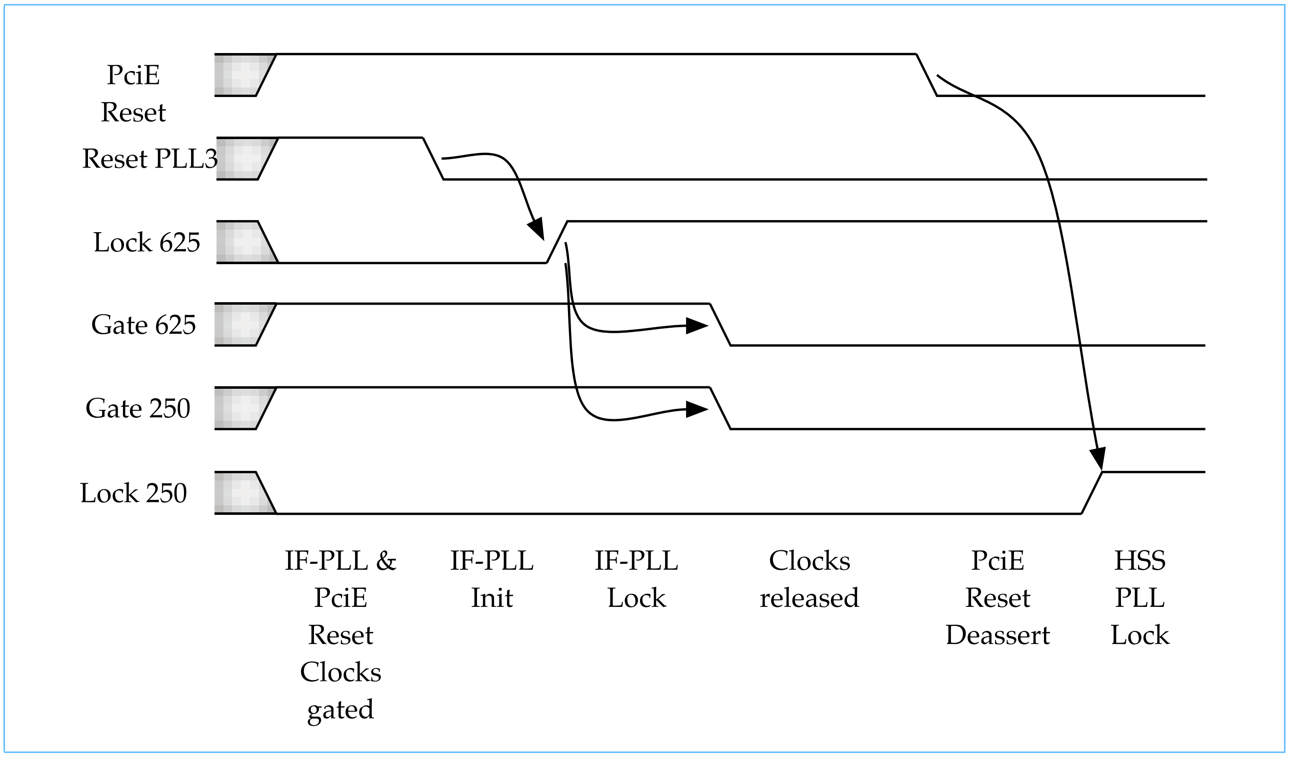
Figure 11-7. PLL3



PLL3 has the typical PMR control and config register arrangement which can be used to set the PLL3 control bits. The control logic is managed by the PCI Express power manager state machine which can be used to stop clocks to the PCI Express logic to support sleep/wake or power down modes.

Because PCI Express has two PLLs, the startup is a bit unlike the other PLLs. First PLL3 is initialized and locked. Once PLL3 has achieved lock, the output from PLL3 is enabled and the clocks are allowed to run and flow through the HSS and PCI Express stack. The PCI Express reset is then deasserted, causing the HSS PLL to initialize and lock. *Figure 11-8* illustrates the reset sequence for PLL3 and PCI Express.

Figure 11-8. PLL3 Startup Sequence



11.3.5 HyperTransport Power Management

11.3.5.1 LDTReq and LDTStop Generation

In order to save power, it is possible to quiesce the HyperTransport bus using the LDTReq and LDTStop protocol. LDTReq is a request to keep the bus active and powered. HT_LDTSTOP_L is a signal that tells all devices on the bus to disconnect their links and go into a power saving state. HT_LDTSTOP_L can also be used to reconfigure the interface when resizing the link width, changing the link speed, or both. In CPC945-based systems, HT_LDTSTOP_L can either be driven externally to CPC945, or the CPC945 can generate it whenever HTClockEnable is cleared in the PwrSystem Register. HT_LDTSTOP_L is generated by the CPC945 whenever the HTClockEnable bit in the ClockControl Register is turned off. HT_LDTSTOP_L is also generated in response to a system sleep request in order to quiesce the HyperTransport interface and tristate all the transmitters on the link. The CPC945 generates an internal LDTReq signal which is driven externally by the HT_LDTREQ_L pin. HT_LDTREQ_L is an open-collector signal which can be driven by multiple sources. If any source is asserting HT_LDTREQ_L, this signals that there is traffic either needing or actually using the HyperTransport bus.

The CPC945 uses the signal coming in from the HT_LDTREQ_L pin as an input to the LDTStop generation logic. Before CPC945 generates an LDTStop condition, it pauses a programmed delay period from the de-assertion of HT_LDTREQ_L. This delay works to filter the number of times the bus is stopped in an attempt to prevent rapid stop/start cycling which saves no power and costs the system in terms of performance.

It is important to note that both LDTStop and warm reset stop the PLL for some amount of time. It is even more important to not release these signals until the PLL is up and running. This is necessary to guarantee that the HyperTransport logic can come out of the stop and reset states within the time limits of the HyperTransport specification. As such, if the CPC945 samples either HT_LDTSTOP_L or ldtLinkReset asserted, it must hold these signals asserted until PLL4 is running again.

ForceLDTStop is a bit in the PwrHT Register which forces the generation of the HyperTransport LDTStop sequence at any time, regardless of the state of the LDTReq signal. This can be used to guarantee that the bus stops at a given time. Its expected use is as a debugging aid for thrashing the LDTStop protocol. HT_StopEnable is a signal which simply stops the HT interface from stopping between reset and the point at which the HT clocks are running.

The HyperTransport interface is responsible for acknowledging that an LDTStop sequence has disconnected and tristated the HyperTransport interface.

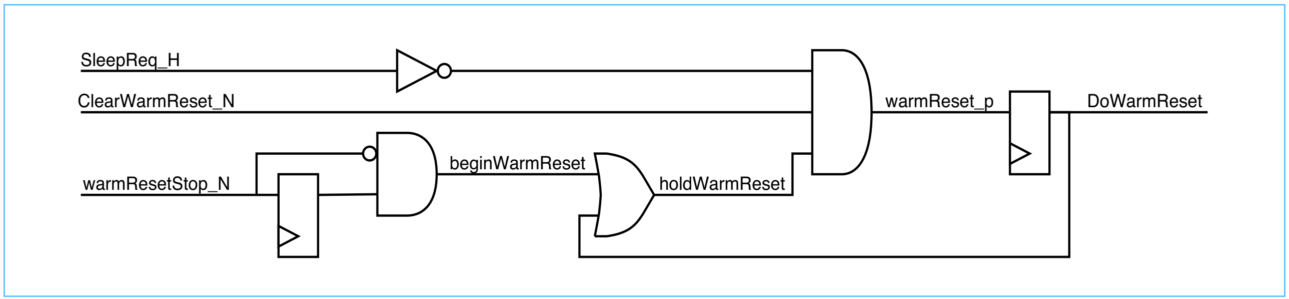
The CPC945 can optionally force the HyperTransport interface into the cold reset state during system sleep. This is required in systems which remove power from any HyperTransport device.

11.3.5.2 HyperTransport Power Manager

Independent of who generates LDTStop, if the bus is quiesced the CPC945 is capable of both gating off clocks to the HyperTransport logic and stopping PLL4.

In *Figure 11-9*, there is a signal labeled DoWarmReset. This signal is asserted on the falling edge of warmResetStop_N from HyperTransport. DoWarmReset is deasserted once the PLL has been reconfigured and relocked as required whenever performing a HyperTransport warm reset sequence. This signal guarantees that a PLL relock does not occur for any assertion of warm reset which happens as part of the process of entering sleep. DoWarmReset signal also guarantees that a PLL relock occurs once and only once for any other assertion of warm reset. The logic for DoWarmReset is shown in *Figure 11-9*.

Figure 11-9. HT Warm Reset

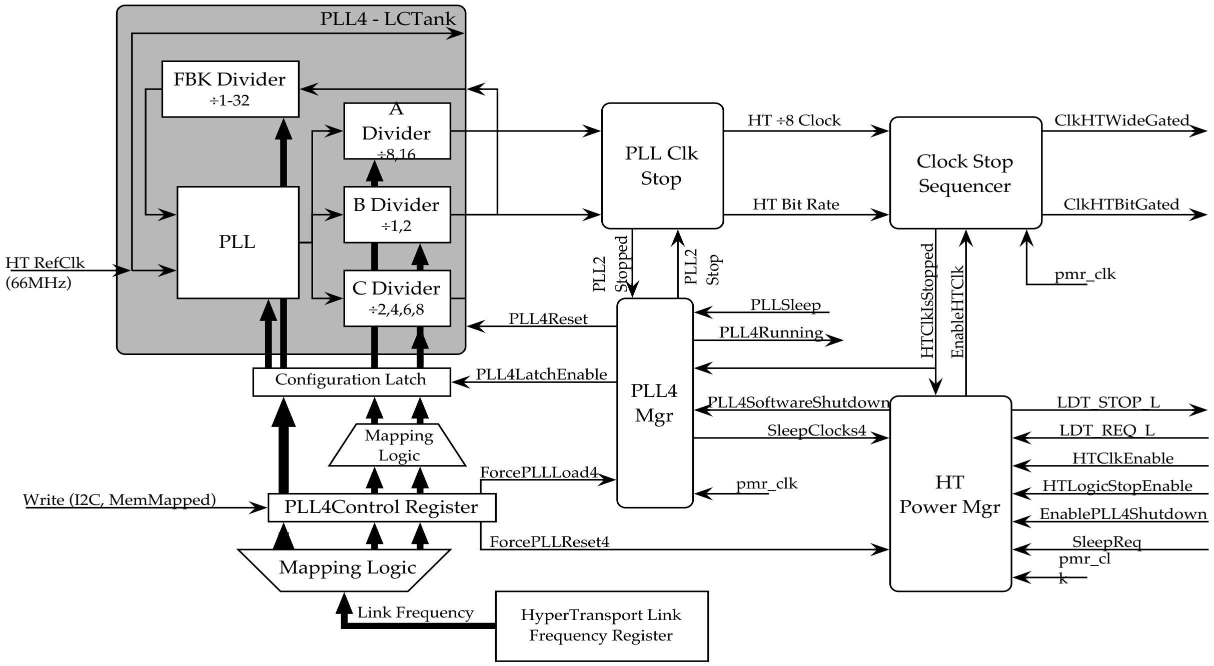


11.3.5.3 PLL4

PLL4 generates the 600 MHz (or lower) clock used by the HyperTransport interface. This PLL accepts a reference clock of 66.67 MHz (the same frequency is applied to all HyperTransport devices in the system). The multiplication ratio depends on the desired operating frequency of the HyperTransport interface - it could be as low as 400 MHz and as high as 600 MHz.

The actual speed of PLL4 is controlled by the Link Frequency Register in HyperTransport configuration space. When the Link Frequency Register is modified, it automatically loads the PLL4Control Register with the appropriate values. PLL4 is actually updated using the standard PLL update mechanism, which is tied into the HyperTransport defined mechanisms for stopping clocks before changing frequencies. If the values automatically loaded into PLL4 by writing to the Link Frequency Register need modification before being used by the PLL, then software can modify them appropriately in the PLL4Control Register before starting the HyperTransport link speed change sequence.

Figure 11-10. PLL4



11.4 Power Tuning

Power tuning is used as another power savings method implemented in the CPC945. With power tuning the frequency of the Processor Interconnect interfaces can be switched from full speed to half or quarter speed. This effectively reduces the frequency of the interface, reducing the power consumed by those interfaces.

The power tuning logic in CPC945 is partially contained in the PI and partially contained in the PMR. The PMR contains the logic which quiesces the PI interfaces prior to switching the clocks. The PMR also contains the logic that switches the clocks.

Power tuning operations are initiated by a CPU which sends a power tuning transaction on the PI to the CPC945. The bridge responds by reflecting the power tuning transaction to all CPUs present. This causes all processors to quiesce current activity and prepare to change bus frequency as specified in the power tuning transaction. Each processor core signals its quiesced state to the bridge by asserting its Qreq signal. When the bridge detects that all CPU cores have asserted their respective Qreq signals, it responds by asserting Qack to each CPU core. The assertion of the Qack signals to the CPU cores is the indication that the CPU cores should now switch bus frequency as specified in the power tuning transaction. The CPC945 also switches bus frequency at this time. When each processor has completed the bus frequency change, the processor deasserts its Qreq signal. When the bridge has completed its bus frequency transition it looks for the Qreqs to be deasserted and responds by deasserting the matching QAck. Once all the QAcks have been deasserted the CPU cores and the CPC945 resumes normal operation at the new bus speed.

The speed selection is handled by an interface between PMR and PI logic. When the PI interfaces have been quiesced and the clocks need to be switched, the PI/PMR logic sends a speed select to the PMR's PI clock logic. The power tuning logic also sends a signal called PrmUpdtBusPrm to indicate that a new speed has been requested.

Figure 11-11 illustrates a power tuning operation as seen by the various interfaces.

Figure 11-11. Power Tuning Sequence

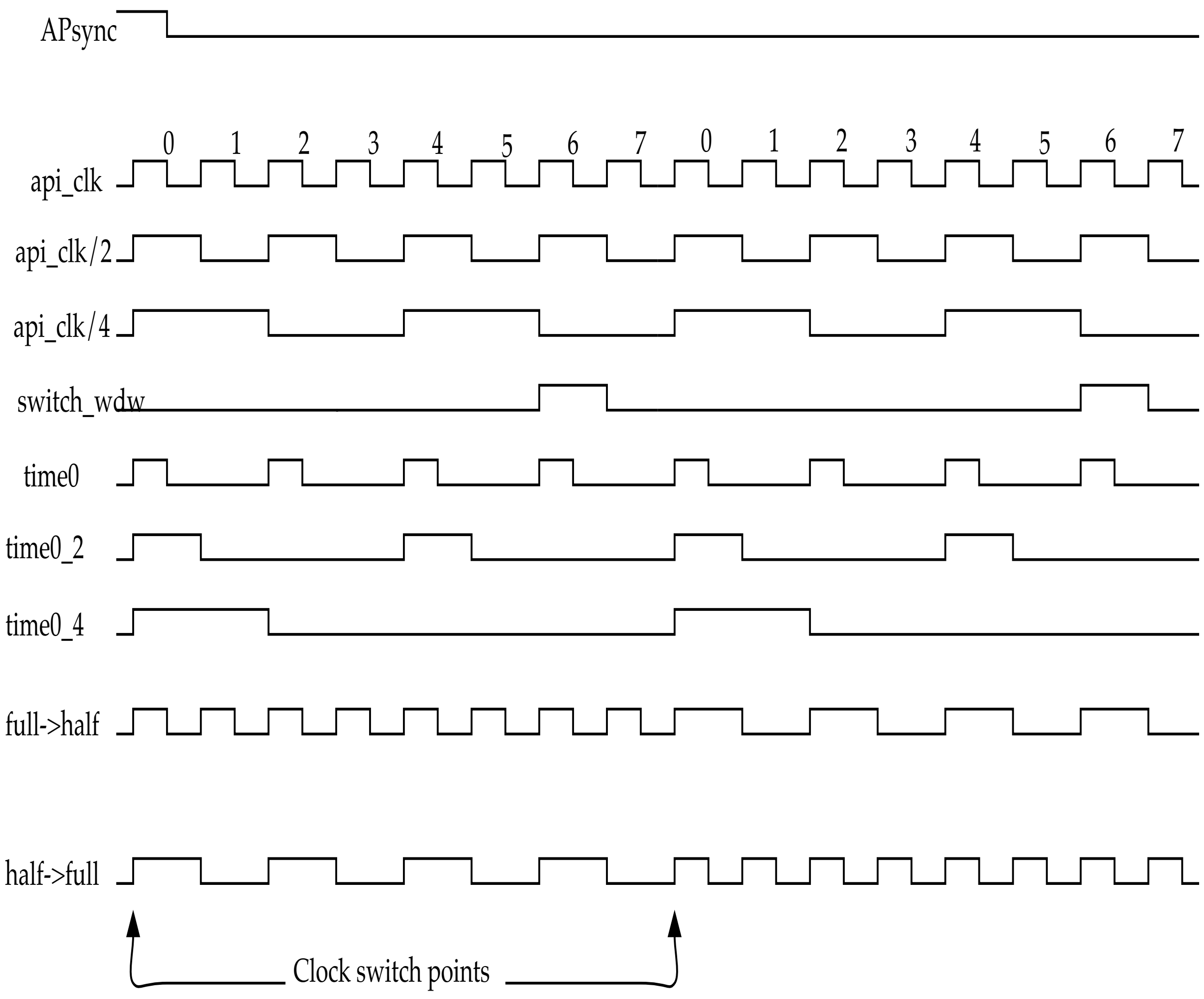
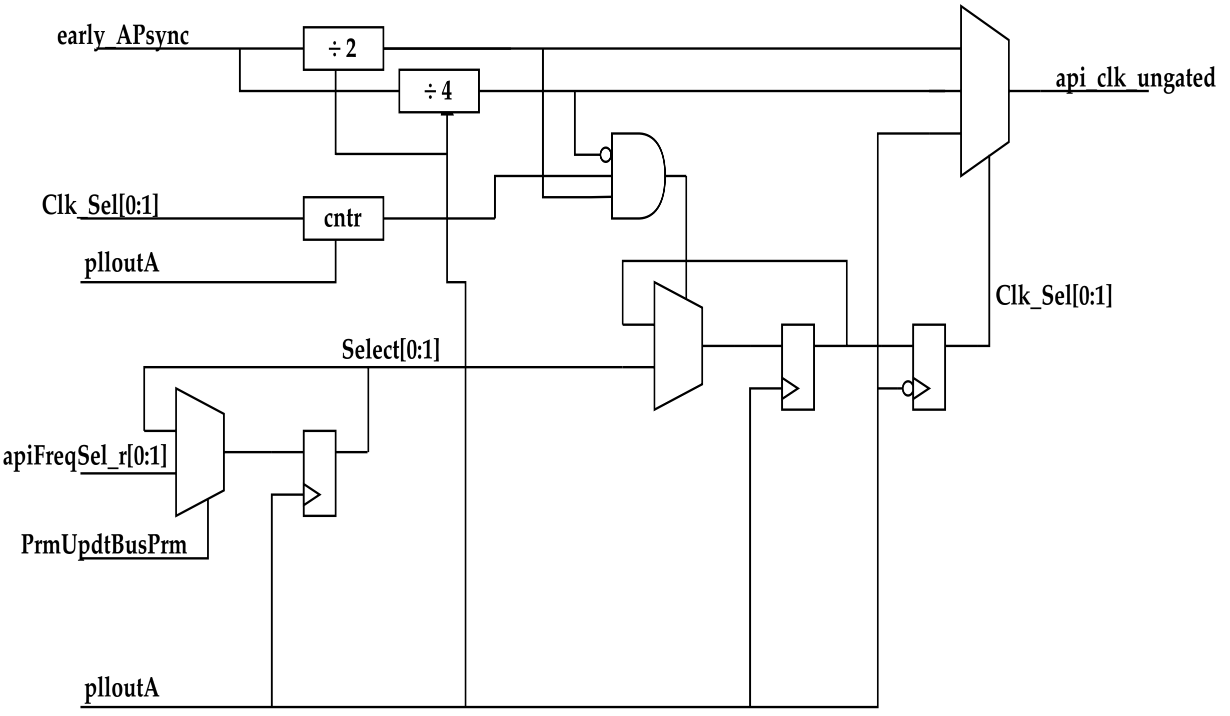


Figure 11-12 is a block diagram of the Power tuning logic contained within the PMR for switching the clock speeds.

Figure 11-12. Power Tuning Logic



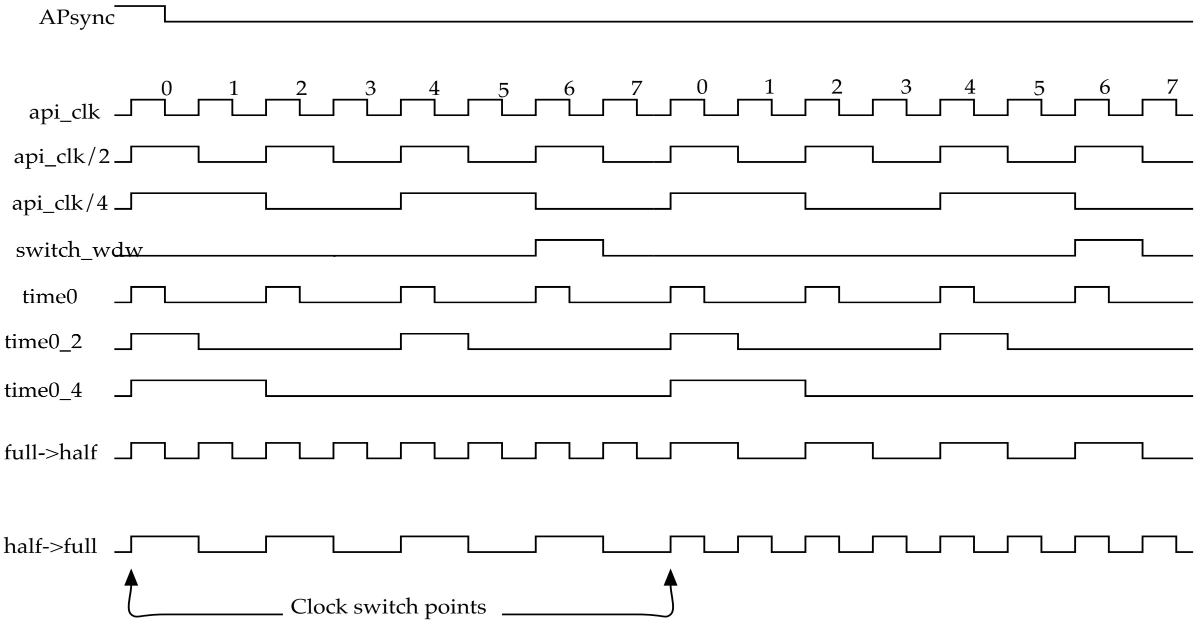
The PMR contains two clock dividers which generate the half and quarter speed clocks. These two clocks and the full speed clock from the PLL are then muxed to generate the api_clk into the clock stoppers at the base of the api_clk tree.

The PMR logic is designed so that the clock output multiplexer can only be switched when time0 is aligned for all three clock speeds. This is critical because the Processor Interconnect interface contains a four deep FIFO that is constantly sampling the bus. The FIFO is loaded by the incoming bus clock, but the FIFO unload is done at the local bus speed. Between the bridge and the CPU there exists a situation where one side is sampling at twice the speed of the other side and the FIFO write pointer can be passing the FIFO read pointer.

This would normally cause a problem were it not for two mechanisms that are employed in all power tuning operations. First the bus is constantly driven with an unchanging pattern of null characters. This means the FIFO data is always null data. The second mechanism is that the bus frequency is always switched at a time0 cycle. Time0 is defined as occurring every four bit times. Switching bus frequency at time0 assures that while the FIFO pointer relationship can be momentarily disturbed, the original alignment is restored when both sides have performed their respective frequency change. Changing bus frequency at time0 assures that the FIFO pointers are unaligned by a distance that is modulo 4 which is the exact size of the FIFO. This brings the FIFO pointers back to their original alignment.

Figure 11-13 shows the time0 marker at full, half and quarter speed as well as the alignment used to switch the clocks.

Figure 11-13. Power Tuning Timing



The clock switching logic has a counter that keeps track of when time0 occurs in each clock speed, as well as when the three clocks all have a common time0. Only when all three clocks are about to reach time0 does the speed selection multiplexer switch to the new clock selection. The speed select is flopped once then a second time with an inverted clock. This second latch is used to make sure the multiplexer speed select changes when all three clocks are low just before the rising edge of time0. By switching the multiplexer during the clock low time, any clock glitches or spikes that might be generated by the switch are avoided.

11.5 PI Frequency Change Operation

Whenever an application does not require the maximum performance from the processor(s), power consumption can be reduced by decreasing the processor clock frequency. When the processor frequency is reduced, CPC945's frequency is reduced by the same percentage. To let CPC945 know a frequency change is desired, the processor can write a configuration value to the processor's Power Control Register (PCR). The write to the PCR generates a new transaction type (0x05) that CPC945 consumes to begin an PI frequency change. Imbedded in the new transaction is the new frequency (full, half, quarter, or no change) and voltage to be switched to as well as the processor's new STATLAT, SNOOPLAT, SNOOPACC. In addition, the transaction is reflected by CPC945 to any other processors to make sure that all processors and CPC945 are operating with the same timing and voltage levels.

In detail, the actual frequency change operation consists of a frequency change command being sent from the processor to the CPC945. The parameters sent with this command are examined in the apiCommandBuf logic to determine whether it should be forwarded to the apiSnpPipe block. Once the apiSnpPipe block receives the frequency change command, it sets the frequency change status to in-progress. apiSnpPipe block also notifies the PMR block of the frequency change command. At this time PMR asserts its QAck signals to wake up any napping CPUs. Once all the CPU's are awake and ready to see the reflected command on the PI bus, the PMR block acknowledges the frequency change command. The command is ready to be reflected on the Processor Interconnect bus. After the command is reflected, PMR monitors the QReqs from the CPUs, once all CPUs have been quiesced as determined by the QReqs signals going inactive, PMR instructs the PI block to update the bus parameters to the new values from the appropriate BUSCONF register pair. The PMR then changes the PI clock frequency. Meanwhile CPUs are also changing their clock frequencies. Once the CPUs are done with the frequency change operation, they assert their respective QReq signals. In response, the PMR asserts the QAcks and bus operations resume on the Processor Interconnect bus. When all QReqs and QAcks become active again, PMR notifies the PI logic to update the frequency change status from in-progress to idle.

Figure 11-14. Timing Diagram for Frequency Change Operation in PI

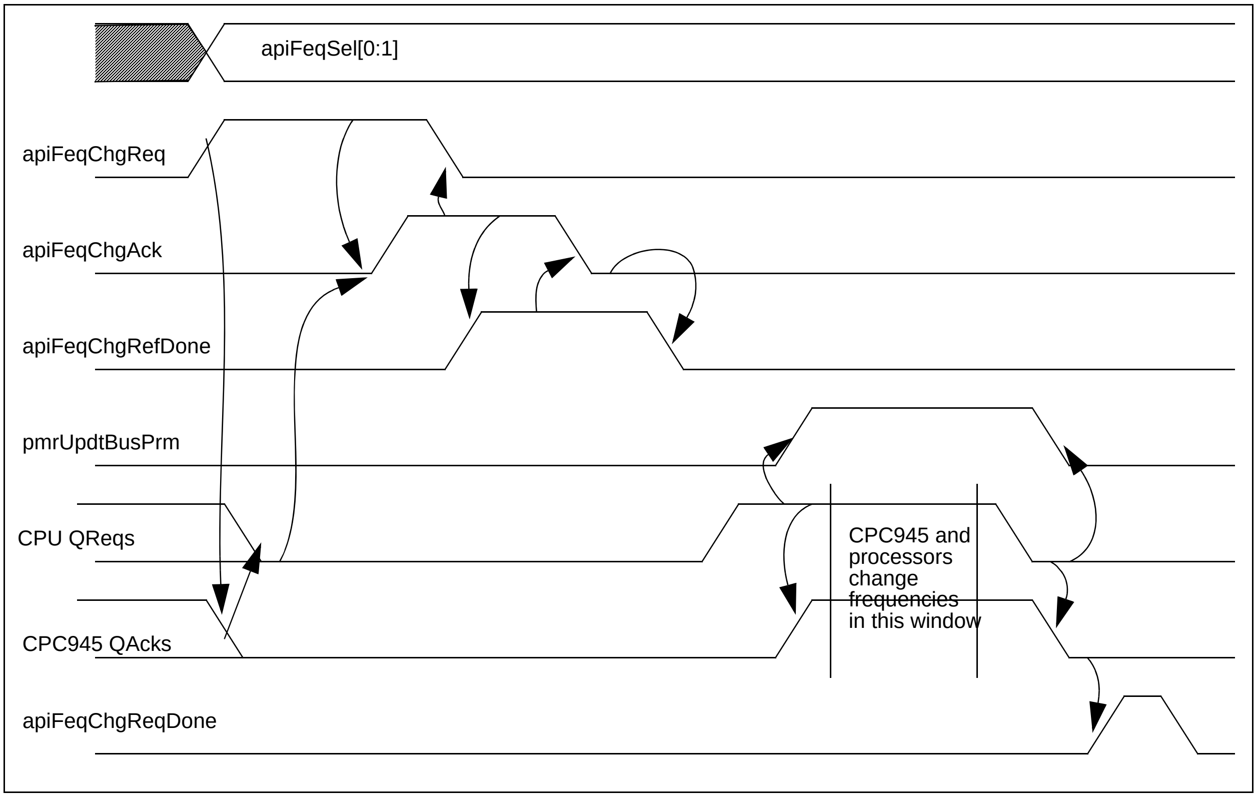
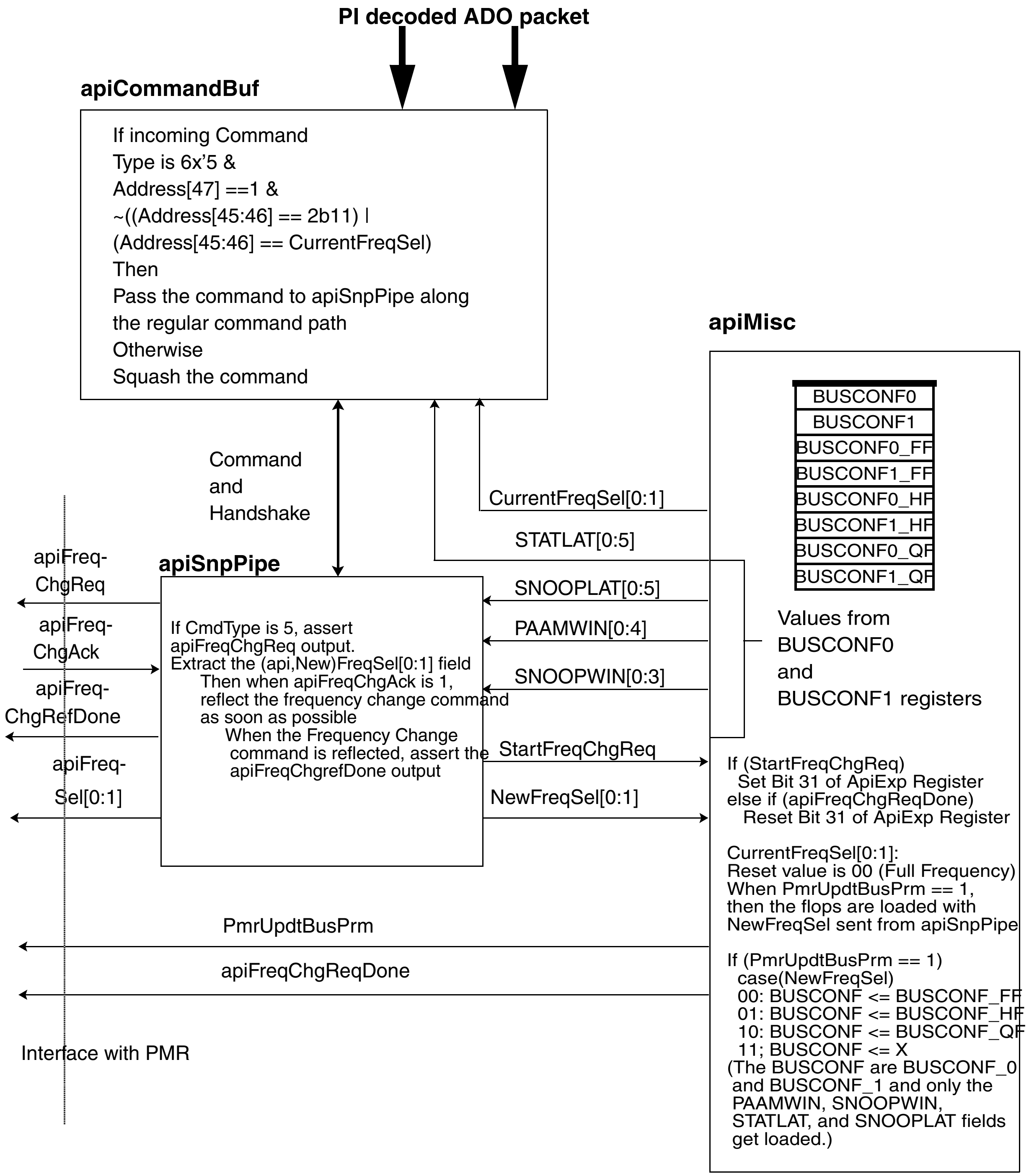


Figure 11-15. Block Diagram for a Frequency Change Operation in PI



```
If incoming Command
Type is 6x'5 &
Address[47] ==1 &
~((Address[45:46] == 2b11) |
(Address[45:46] == CurrentFreqSel)
Then
Pass the command to apiSnpPipe along
the regular command path
Otherwise
Squash the command
```

```
If CmdType is 5, assert
apiFreqChgReq output.
Extract the (api,New)FreqSel[0:1] field
     Then when apiFreqChgAck is 1,
     reflect the frequency change command
     as soon as possible
          When the Frequency Change
           command is reflected, assert the
           apiFreqChgrefDone output
```

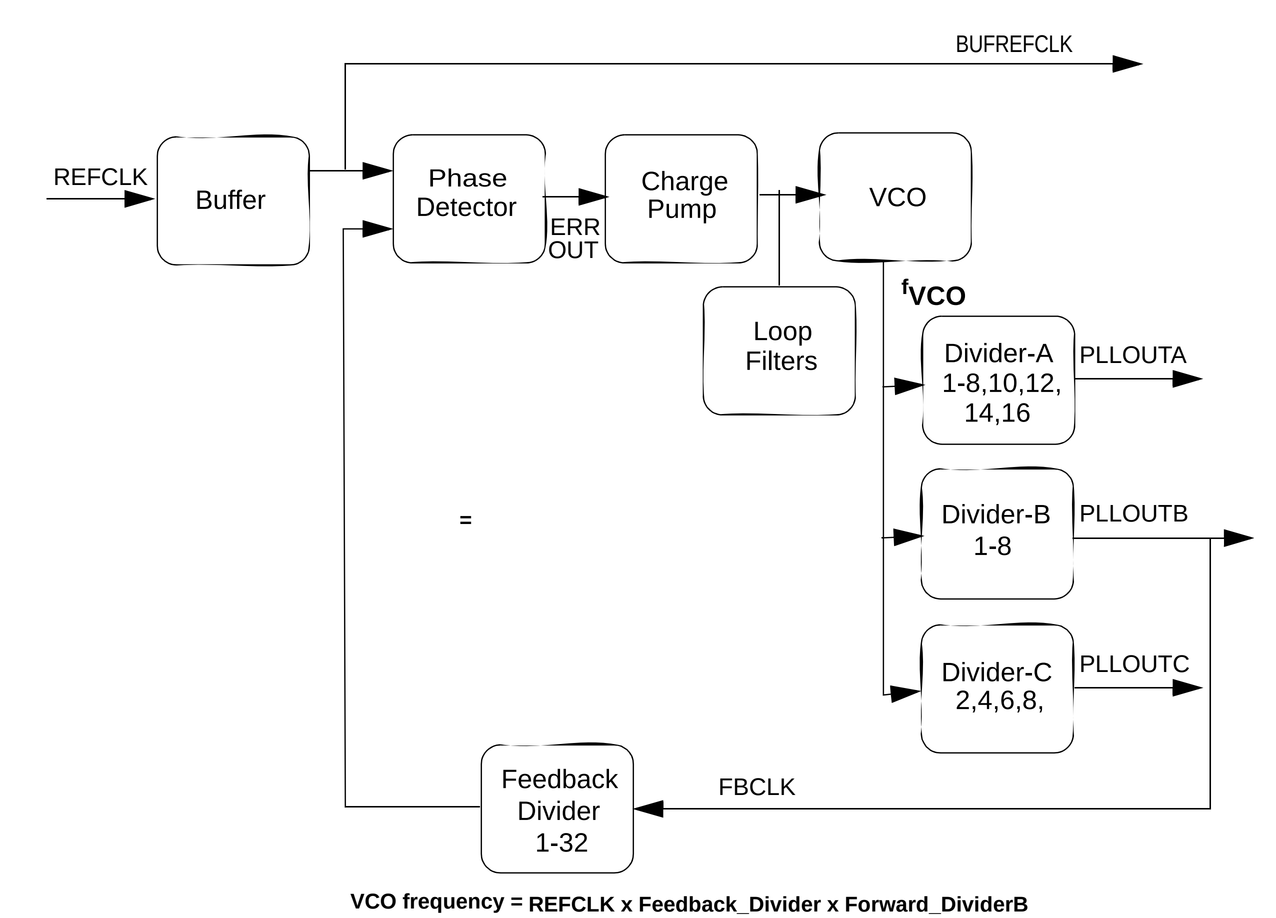
```
If (StartFreqChgReq)
  Set Bit 31 of ApiExp Register
else if (apiFreqChgReqDone)
   Reset Bit 31 of ApiExp Register

CurrentFreqSel[0:1]:
Reset value is 00 (Full Frequency)
When PmrUpdtBusPrm == 1,
then the flops are loaded with
NewFreqSel sent from apiSnpPipe

If (PmrUpdtBusPrm == 1)
  case(NewFreqSel)
  00: BUSCONF <= BUSCONF_FF
  01: BUSCONF <= BUSCONF_HF
  10: BUSCONF <= BUSCONF_QF
  11; BUSCONF <= X
(The BUSCONF are BUSCONF_0
 and BUSCONF_1 and only the
 PAAMWIN, SNOOPWIN,
 STATLAT, and SNOOPLAT fields
 get loaded.)
```

11.6 PLL Programming

11.6.1 PLL1 and PLL2

PLL1 and PLL2 both use the same type of PLL macro. These PLLs are used to generate the PI and DDR2 clocks respectively. *Figure 11-16* illustrates the major blocks within the PI/DDR2 PLL.

Figure 11-16. PLL1/ PLL2 Internal Block Diagram



VCO frequency = REFCLK x Feedback_Divider x Forward_DividerB

For the CPC945 application the feedback clock (FBCLK) is connected to the PLLOUTB pin of the PLL. This closes the PLL feedback loop and defines how the PLL dividers should be programmed for a given reference clock and target PLL output.

To determine the VCO frequency, we use the following equation:

```
VCO frequency = REFCLK x Feedback_Divider X Forward_DividerB
```

The resulting VCO frequency should be between 600 MHz and 1334 MHz. With the wide range of programmability offered on the outputs, several integer and non-integer relationships can be realized between the PLLOUTA and PLLOUTB outputs by controlling the A and B dividers. The frequency relationship is given as: The PLLOUTA and PLLOUTB outputs are always synchronized to the rising edge (that is, the PLLOUTA rising edge coincides with the PLLOUTB rising edge at the start of the cycle). The PLLOUTC output is not used or connected in the CPC945.

The PLL tuning bits, TUNE[9:0], are used to modify the PLL loop parameters by modifying the internal gains of the charge pumps. This external control allows the PLL to be stable over a wide range of frequencies and multiplication factors. The following tables describes the proper settings for the PLL tune bits:

Table 11-2. PLL1 and PLL2 Tune Bit Settings, 1 of 2.

M = Div-A * Fbk	Tune bits									
	9	8	7	6	5	4	3	2	1	0
$2 \leq M \leq 3$	0	1	-	0	1	1	0	0	1	0
$3 < M \leq 6$	0	1	-	0	1	1	0	1	0	0
$6 < M \leq 10$	0	1	-	0	1	1	0	1	1	0
$10 < M \leq 22$	1	0	-	0	1	1	1	0	0	0
$22 < M \leq 40$	1	1	-	0	1	1	1	1	0	0

Table 11-3. PLL1 and PLL2 Tune Bit Settings, 2 of 2.

VCO Frequency	Tune bits									
	9	8	7	6	5	4	3	2	1	0
600 MHz < VCO Frequency ≤ 900 MHz	-	-	0	-	-	-	-	-	-	-
900 MHz < VCO Frequency ≤ 1334 MHz	-	-	1	-	-	-	-	-	-	-

The key constraints in choosing values for the dividers are:

- Divider ratios A must be 1 - 8, 10, 12, 14, or 16
- Divider ratios B must be 1 - 8
- Divider ratio C must be 2, 4, 6, or 8
- Feedback ratio M must be 1 - 32
- Tune bits set according to tables above
- REFCLK must be 33 - 500 MHz
- VCO frequency f_{VCO} must be 600 - 1334 MHz

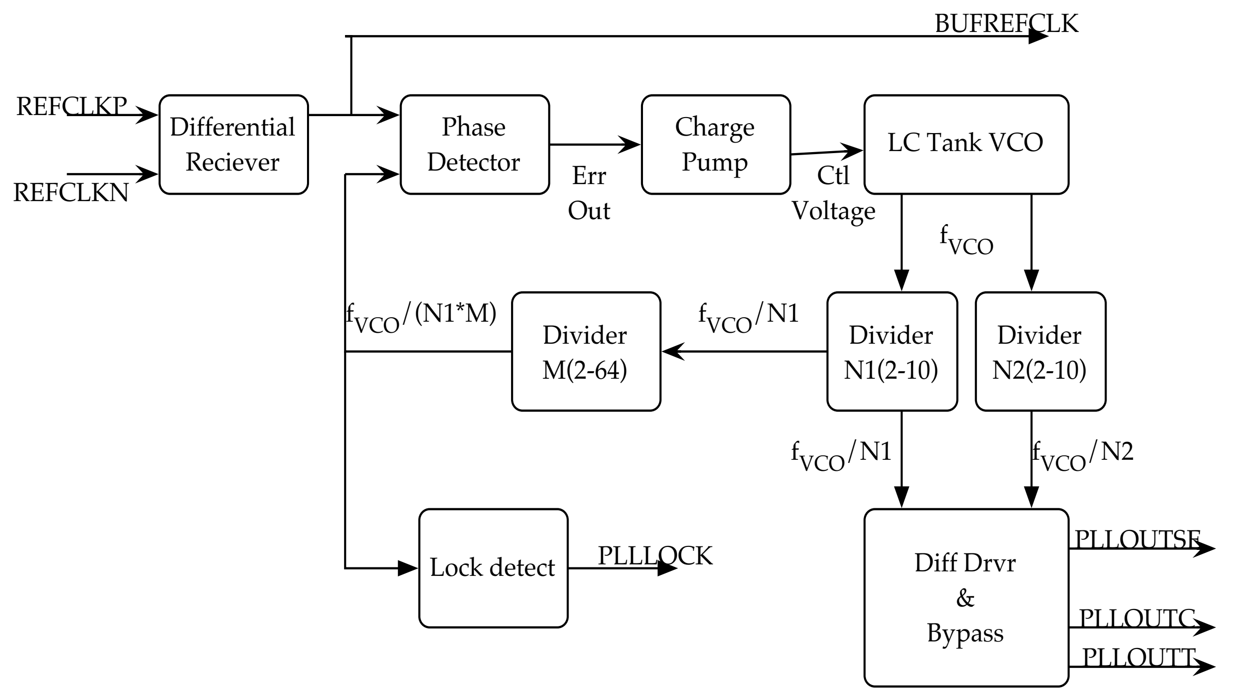
See *Section 12.5.6 PLL1 Control Register* on page 346 and *Section 12.5.7 PLL2 Control Register* on page 350 for programming details.

11.6.2 PLL3

PLL3 is used to generate the clocks for the PCI Express HSSL logic. The PLL used is the LC-Tank PCE which is designed specifically for use with the PCI Express HSSL macro. The following diagram illustrates the major blocks within the LC-Tank PCE PLL.

See *Section 12.5.8 PLL3 Control Register* on page 352 for programming details.

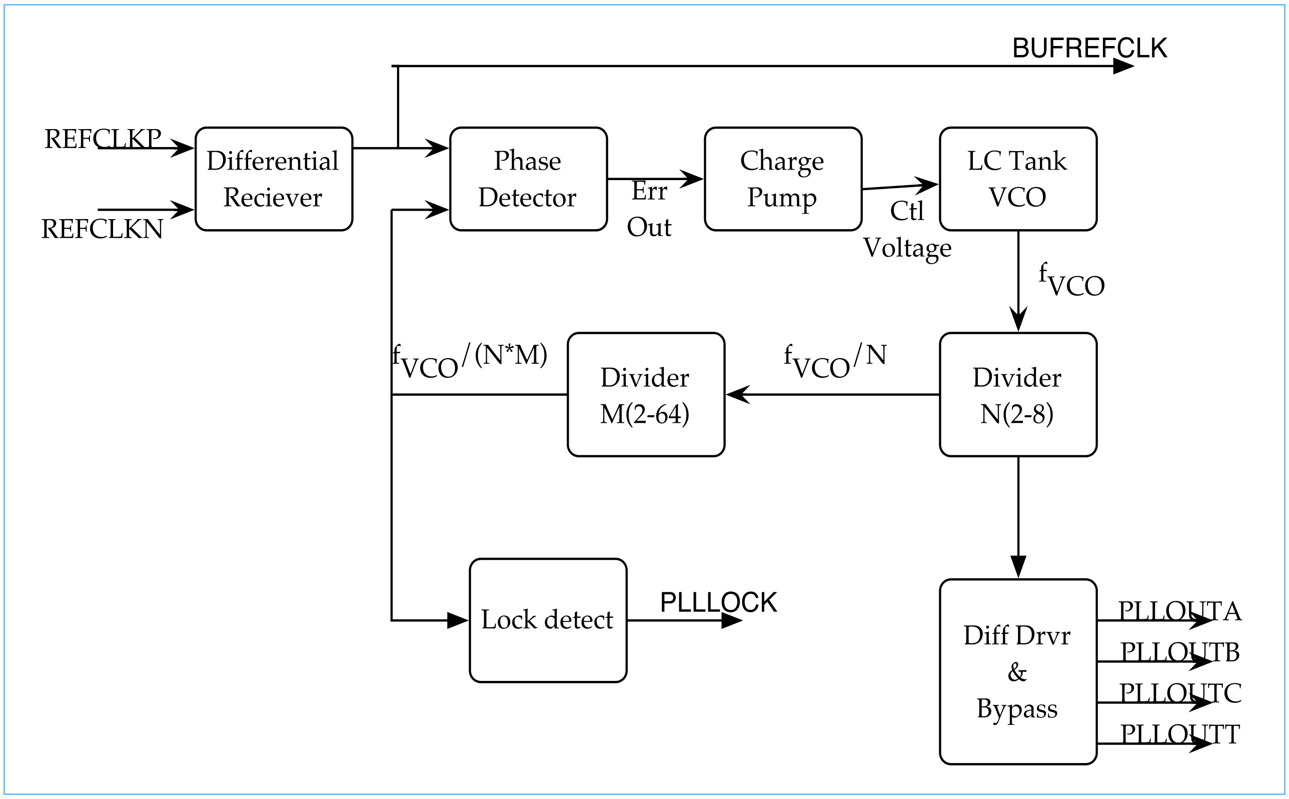
Figure 11-17. PLL 3 Internal Block Diagram



11.6.3 PLL4

PLL4 is used to generate the clocks for the HyperTransport logic. The PLL used is the LC-Tank HTT which is designed specifically for use with the HyperTransport. The following diagram illustrates the major blocks within the LC-Tank HTT PLL.

Figure 11-18. PLL 4 Internal Block Diagram



The PLL LC Tank core is programmed by means of the selection of the M and N multipliers. The key equation describing the output frequency at PLLOUTA is:

```
PLLOUTA = REFCLK * M
```

where REFCLK is the frequency of the input signal and M is the multiplier ratio (selected via the M[0:5] input pins). Note that the frequency at the differential outputs PLLOUTT and PLLOUTC is equal to the frequency at PLLOUTA.

The VCO frequency f_{VCO} is defined as:

f_{VCO} = REFCLK * M * N

The key constraints in choosing values for the dividers are:

- The range of divider ratio N must be 2 - 8.
- The range of multiplier ratio M must be 2 - 64.
- The range of REFCLK must be 25 - 320 MHz.
- The range of the VCO frequency f_{VCO} must be 3 - 3.6 GHz

The PLLOUTB output frequency is programmed by choosing the appropriate value for the P divider. The frequency at this output is the PLLOUTA frequency divided by the value of the P divider.

See *Section 12.5.9 PLL4 Control Register* on page 355 for programming details.

12. Programmer’s Interface

12.1 Memory Map

The CPC945 core logic uses the 36-bit extended memory map shown in *Table 12-1*. If a name includes the word “alias” then any access to the aliased memory space will access the same offset into the memory space being aliased. For example, the “HT1 16 MB Alias” memory space is a window into the address spaces with the names “HyperTransport1” and “HT1 I/O Space”. In addition, any unused memory space (for example HT2 and HT3) should not be accessed as this can produce undefined results that might cause harmful effects to the hardware.

Table 12-1. 36-bit Extended Memory Map.

Start Address	End Address	Name	Comments	Page
0x000000000	0x07FFFFFFF	Main Memory	2 GB of main memory, linearly accessed.	
0x080000000	0x0EFFFFFFF	PCI Bus Memory Space	2 GB - 256 MBytes of PCI Express/HyperTransport (HT) memory space for I/O devices.	
0x0F0000000	0x0F1FFFFFF	PCI Express	PCI Express in CPC945 contains the video expansion slot. 0x0F0000000-0x0F07FFFFF: PCIe I/O Space 0x0F0800000-0x0F0FFFFFF: PCIe Config. Space 0x0F1000000-0x0F1FFFFFF: PCIe Direct Access to first 256 bytes of Configuration Register Space	505
0x0F2000000	0x0F4FFFFFF	HyperTransport1	HyperTransport bridge 1 is used to connect to I/O devices outside of CPC945. There are no other bridges in CPC945.	614
0x0F5000000	0x0F7FFFFFF		Reserved.	
0x0F8000000	0x0F8FFFFFF	CPC945 Control Registers	CPC945 Control Registers are located here. See *Table 12-2*.	332
0x0F9000000	0x0F9FFFFFF	Undefined		
0x0FA000000	0x0FEFFFFFF	Undefined		
0x0FF000000	0x0FFFFFFFF	ROM Space	16 MB of executable ROM space. The reset vector is at 0xFFF00100.	
0x100000000	0xFFFFFFFFF	Main Memory	60 GB of main memory, linearly addressed.	

CPC945 powers up with fourteen hard-decoded spaces:

1.	0x000000000 to 0x07FFFFFFF:	2 GB Memory Space
2.	0x0F08XXXXX:	PCIe Config Address Register
3.	0x0F0CXXXXX:	PCIe Config Data Register
4.	0x0F2XXXXXX:	HT1 Configuration Command Type 0
5.	0x0F3XXXXXX:	HT1 Configuration Command Type 1
6.	0x0F4000000 to 0x0F43FFFFF:	HT1 I/O Space
7.	0x0F4400000 to 0x0F47FFFFF:	HT1 EOI Space
8.	0x0F4800000 to 0x0F4FFFFFF:	HT1 Reserved Space
9.	0x0F8000000 to 0x0F8FFFFFF:	CPC945 Control Space
10.	0x0F9000000 to 0x0F9FFFFFF:	Undefined
11.	0x0FC400000 to 0x0FC7FFFFF:	HT1 I/O Access Space
12.	0x0FE400000 to 0x0FE7FFFFF:	HT1 EOI Space
13.	0x0FF000000 to 0x0FFFFFFFF:	Executable ROM Space
14.	0x100000000 to 0xFFFFFFFFF:	60 GB Memory Space

Note: The entire HyperTransport I/O and EOI Space is located at 0x0FC000000 to 0x0FCFFFFFF and 0x0FE000000 to 0x0FEFFFFFF respectively. Only the HyperTransport 1 Bridge is used in CPC945, so only the space for that bridge is decoded.

The ROM space requires special setup when the processor is brought out of reset. This is because the PPC970xx fetches its restart vector from the address 0x0000000100, which must be in the ROM space. As a part of the Service Processor controlling the PPC970xx, the ROM accesses will be mapped using the Hypervisor registers. This will convert the access from 0x000000100 to 0x0FFF00100.

12.2 Memory-Like Space

Any size operand up to 32 bytes can be used to access memory-like space. Memory-like space is defined as DRAM and ROM. Only DRAM and ROM space are kept coherent by the system hardware (CPC945 and the microprocessor). It is the responsibility of the operating system software to maintain the cache coherence of expansion slot spaces.

Although they are mentioned in this section, the CPC945 Control Registers do not act like memory in that they do not accept burst transactions and do not accept partial word writes. They are accessible in the memory map by any device in the system.

12.2.1 DRAM

The 36-bit Extended address map allocates 62 GBytes of space for DRAM memory. This space is responded to on the Processor Interconnect (PI) by the CPC945. Memory transactions that appear on PCI interfaces with addresses in this space generate snoop requests on PI (to check for coherency) after being remapped using the DMA Relocation Table (DART) prior to being serviced by the memory controller in CPC945. Up to 62 GBytes of memory can be installed. DRAM memory banks are stitched contiguously.

12.2.2 Noncoherent DRAM Access

Below is the code used in most drivers after writing data to noncoherent memory spaces and immediately before actually submitting an I/O request to hardware:

```
{
        SInt32 s32End = (word_count << 2) - 32;

        for(s32Temp = 0; s32Temp < s32End; s32Temp += 32)
            __dcbst(virtual_offset, s32Temp);

        __sync();
        __isync();
        __dcbf(virtual_offset, s32Temp);
        __sync();
        __isync();
        glkRead32(virtual_offset, s32Temp);
        __isync();
}
```

This code prevents any code reordering problems that could be introduced by the PPC970xx's ability to reorder instructions. The Syncs ensure the last write in program order is the last write on the bus. The core logic operation ensures the read will not complete until the write is complete. Even if writes are reordered, execution will not continue until all writes have gone to memory.

12.2.3 ROM

The CPC945 36-bit Extended Address Map allocates 16 MB of space for ROM. This space is responded to on PI by the CPC945. Memory transactions that appear on the PCI interfaces with addresses in this space are serviced by the CPC945. The 1Mx8 ROM is aliased 16 times within this 16 MB address space. At Reset, the processor accesses ROM starting at address 0x0FFF00100.

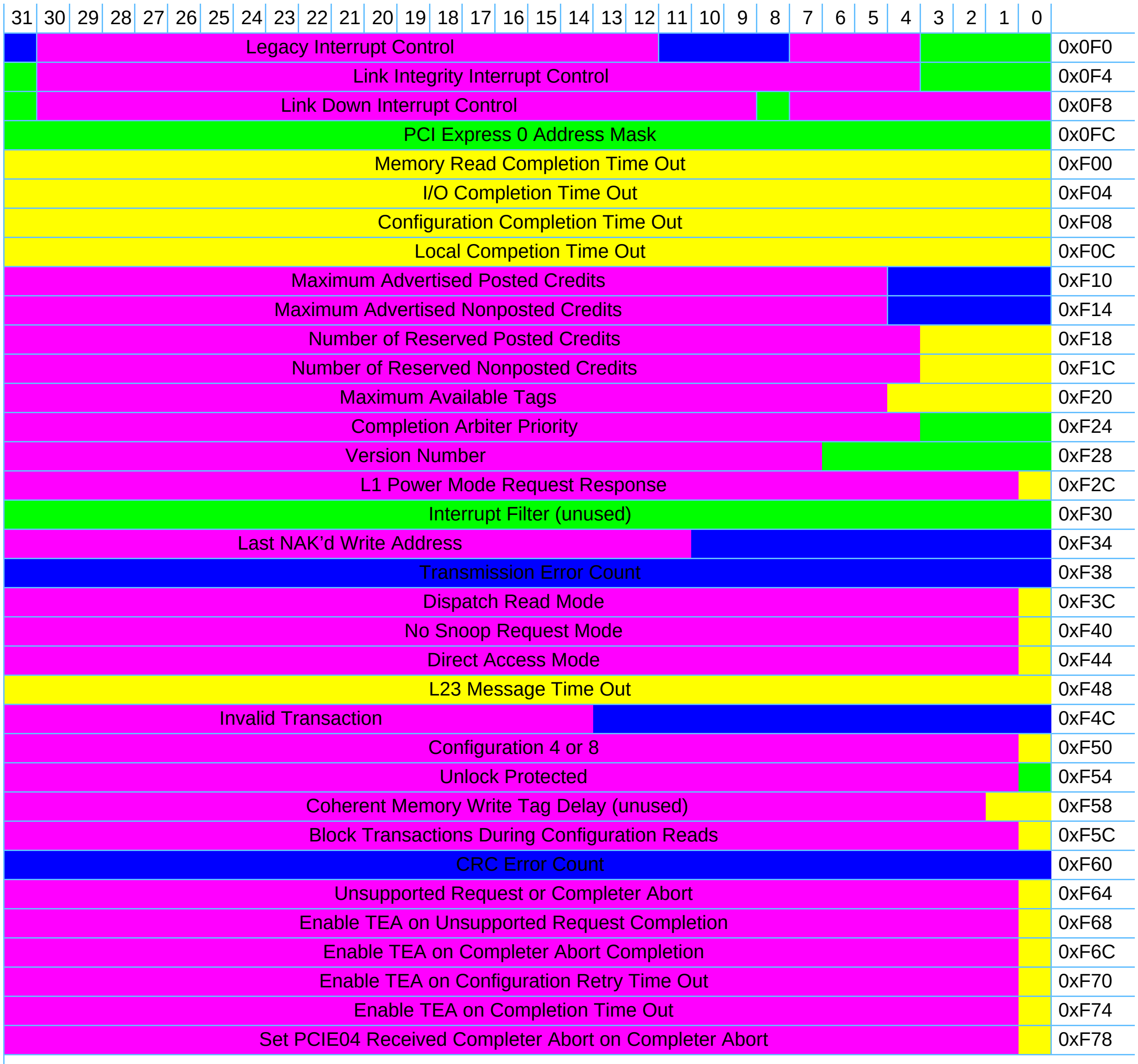
12.2.4 Control Registers

The CPC945 36-bit Extended Address Map allocates 16 MB of space for System Control. This space is responded to on PI by the CPC945. Memory transactions that appear on the PCI interfaces with addresses in this space are serviced by the CPC945. The accesses do not need to be remapped via the DART, as they reside within the first 2 GB of addressable memory. The TEA signal will be asserted in response to any burst accesses to this address space. CPC945 does not support partial writes of any internal registers. If the write data does not cover all bits in the register, the results will be indeterminate. This means that a register can be written with a byte operand if all defined bits are covered by that byte. If a register contains reserved fields, those fields must be included in the size of the register and should always be written with zeros. If a register contains undefined fields, those fields can be neglected when determining the width of the register.

12.2.5 PCI Express Configuration Space

The PCI Express configuration space is the memory range from 0xF0BFF000 to 0xF0FFFFFF, which is populated by two types of registers: PCIe configuration registers in the memory range from 0xF0BFF000 to 0xF0BFFFFF, and XBus configuration registers in the memory range 0xF0C00000 to 0xF0FFFFFF. The PCIe configuration registers are described in *Section 12.11.1* on page 505 and the XBus PCIe configuration registers are described in *Section 12.11.2* on page 551.

Figure 12-1. PCIe XBus Configuration Space



Legacy Interrupt Control	0x0F0
Link Integrity Interrupt Control	0x0F4
Link Down Interrupt Control	0x0F8
PCI Express 0 Address Mask	0x0FC
Memory Read Completion Time Out	0xF00
I/O Completion Time Out	0xF04
Configuration Completion Time Out	0xF08
Local Competion Time Out	0xF0C
Maximum Advertised Posted Credits	0xF10
Maximum Advertised Nonposted Credits	0xF14
Number of Reserved Posted Credits	0xF18
Number of Reserved Nonposted Credits	0xF1C
Maximum Available Tags	0xF20
Completion Arbiter Priority	0xF24
Version Number	0xF28
L1 Power Mode Request Response	0xF2C
Interrupt Filter (unused)	0xF30
Last NAK'd Write Address	0xF34
Transmission Error Count	0xF38
Dispatch Read Mode	0xF3C
No Snoop Request Mode	0xF40
Direct Access Mode	0xF44
L23 Message Time Out	0xF48
Invalid Transaction	0xF4C
Configuration 4 or 8	0xF50
Unlock Protected	0xF54
Coherent Memory Write Tag Delay (unused)	0xF58
Block Transactions During Configuration Reads	0xF5C
CRC Error Count	0xF60
Unsupported Request or Completer Abort	0xF64
Enable TEA on Unsupported Request Completion	0xF68
Enable TEA on Completer Abort Completion	0xF6C
Enable TEA on Configuration Retry Time Out	0xF70
Enable TEA on Completion Time Out	0xF74
Set PCIE04 Received Completer Abort on Completer Abort	0xF78

🟩 Read/Write 🟪 Unused 🟨 Protected 🟦 Read Only

12.3 Control Register Memory Map

CPC945 contains a number of control registers to report board and chip configurations. All the control registers are 1 - 4 bytes wide and are accessed as 4-byte quantities. The registers are grouped into logical collections and are provided with their own offsets within the 16 MByte space at 0xF8xxxxxx. Most of these registers are aligned to 16-byte boundaries, but it is possible for registers to be 4-byte aligned and packed. The CPC945 register map is shown in *Table 12-2*. 0xF8xxxxxx is a 32 bit address space, CPC945 uses 36-bit addressing, and the PPC970xx uses 64-bit addressing. To keep the explanation simple, all address bits specified for a CPC945 control register are for 32-bit addressing. If 64-bit addressing is wanted, add 32 to each bit location. This does not affect the bit locations within the registers, just the address.

Table 12-2. Control Register Memory Map.

Address	Register Group	Page
0xF8000000- 0xF80007FF	CPC945 Control Registers	332
0xF8000800- 0xF8000FFF	CPC945 Clocks and Power Management Registers	338
0xF8001000- 0xF8001FFF	DRAM I^{2}C Master Controller Registers	398
0xF8002000- 0xF8002FFF	Memory Control Registers	447
0xF8004000- 0xF8004FFF	The HyperTransport block converts incoming posted writes associated with an interrupt request to a write in this space.	283
0xF8005000- 0xF8005FFF	Reserved for PCIe Message Signaled Interrupts (MSI), which are converted into a write to 0xF8005000	
0xF8006000- 0xF800FFFF	Not used - reserved	
0xF8010000- 0xF801FFFF	Not used - reserved	
0xF8020000- 0xF8021FFF	Not used - reserved	
0xF8022000- 0xF8022FFF	PI Physical Interface Registers 0	370
0xF8023000- 0xF8023FFF	PI Physical Interface Registers 1	370
0xF8024000- 0xF8024FFF	Not used - reserved	
0xF8028000- 0xF802DFFF	Not used - reserved	
0xF802E000- 0xF802EFFF	Reserved	
0xF802F000- 0xF802FFFF	Not used - reserved	
0xF8030000- 0xF803FFFF	Advanced Processor Interconnect Registers	408
0xF8040000- 0xF806FFFF	MPIC Registers	359

Table 12-2. Control Register Memory Map.

Address	Register Group	Page
0xF8070000- 0xF807FFFF	HyperTransport Registers	614
0xF8080000- 0xF808FFFF	PCI Express Registers	505
0xF8090000- 0xF809FFFF	PCI Express Configuration Registers	505
0xF80A0000- 0xF8FFFFFF	Not used - reserved	

12.4 CPC945 Control Registers

Table 12-3. CPC945 Control Register Addresses.

Address	Register Name	Description	Page
0xF8000000- 0xF8000FFF	See below	CPC945 Control Registers	
0xF8000000	UniRevision	CPC945 Revision Number Register	333
0xF8000010	Undefined		
0xF8000020		No Longer Implemented (Reads as 0x00000000)	
0xF8000030		No Longer Implemented (Reads as 0x00000000)	
0xF8000040		No Longer Implemented (Reads as 0x00000000)	
0xF8000050	WhoAmI	Who am I Bus Master ID register	334
0xF8000060	MPSemaphore	Processor Semaphore Register	335
0xF8000070	HWInitState	Hardware Initialization State Register	336
0xF8000080		No Longer Implemented (Reads as 0x00000000)	
0xF8000090		No Longer Implemented (Reads as 0x00000000)	
0xF80000A0		No Longer Implemented (Reads as 0x00000000)	
0xF80000B0		No Longer Implemented (Reads as 0x00000000)	
0xF80000C0		No Longer Implemented (Reads as 0x00000000)	
0xF80000D0		No Longer Implemented (Reads as 0x00000000)	
0xF80000E0		CPC945 Toggle Register	337

12.4.1 CPC945 Revision Register

This register contains revision information specific to the given CPC945 implementation. The CPC945 will contain 0x30 in this register.

Reset Value Chip Specific

Address xF800 0000

Access Type Read Only

Reserved																								UNType				RevNum			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:23	Reserved	Reserved.	R	0x00 0000
24:27	UNType		R	0x4
28:31	RevNum	This field specifies the revision level for the CPC945. 0x2: CPC945 DD 1.2 0x4: CPC945 DD 2.0	R	Chip Specific

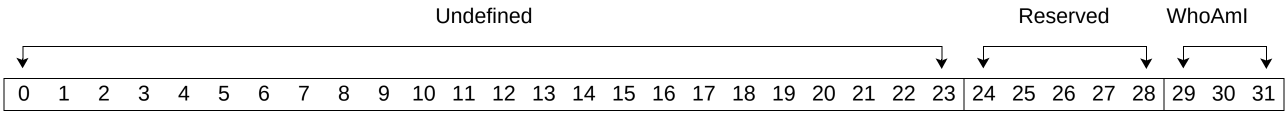
12.4.2 Who Am I Bus Master ID Register

The WhoAmI register returns a value that indicates which bus master has accessed the register. This allows a processor to determine to which BR/BG pair it is connected.

Reset Value x0000 000n

Address xF800 0050

Access Type Read Only

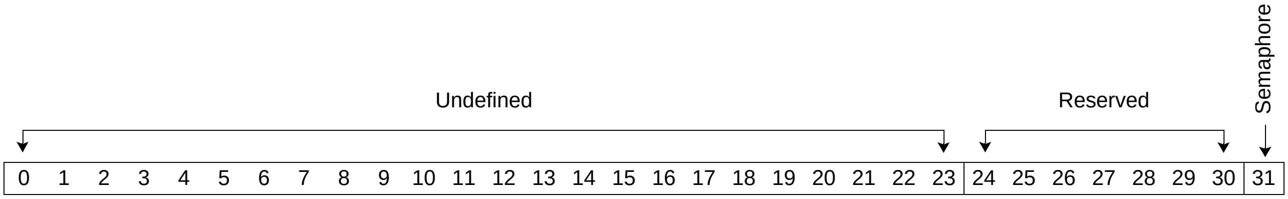


Bits	Field Name	Description	Access	Reset
0:23	Undefined		R	0x000000
24:28	Reserved		R	0b000000
29:31	WhoAmI	This field specifies the accessing master as follows: WhoAmI[0:2] = 0b000: External Master 0 WhoAmI[0:2] = 0b001: External Master 1 WhoAmI[0:2] = 0b010: External Master 2 WhoAmI[0:2] = 0b011: External Master 3 WhoAmI[0:2] = 0b100: Internal Master 0 (PCI0) WhoAmI[0:2] = 0b101: Internal Master 1 (HT1) WhoAmI[0:2] = 0b110: N/A WhoAmI[0:2] = 0b111: N/A	R	N/A

12.4.3 Processor Semaphore Register

This register is used to identify the primary processor. The first processor to read this register is the primary processor.

Reset Value	x0000 0001
Address	xF800 0060
Access Type	Read Only

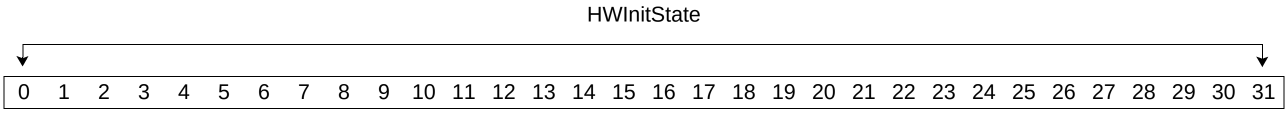


Bits	Field Name	Description	Access	Reset
0:23	Undefined	This field is all 0’s	R	x00 0000
24:30	Reserved	This field is all 0’s	R	x00
31	Semaphore	This bit is set to 1 whenever Reset_ or SUSPEND_REQ_L are asserted. After Reset_ and/or SUSPEND_REQ_L are deasserted, it will return a 1 on the first read access, and a 0 on any future access until Reset_ or SUSPEND_REQ_L are again asserted.	R	1

12.4.4 Hardware Initialization State Register

This scratch pad register can be set to any desired value.

Reset Value	x0000 0000
Address	xF800 0070
Access Type	Read and Write

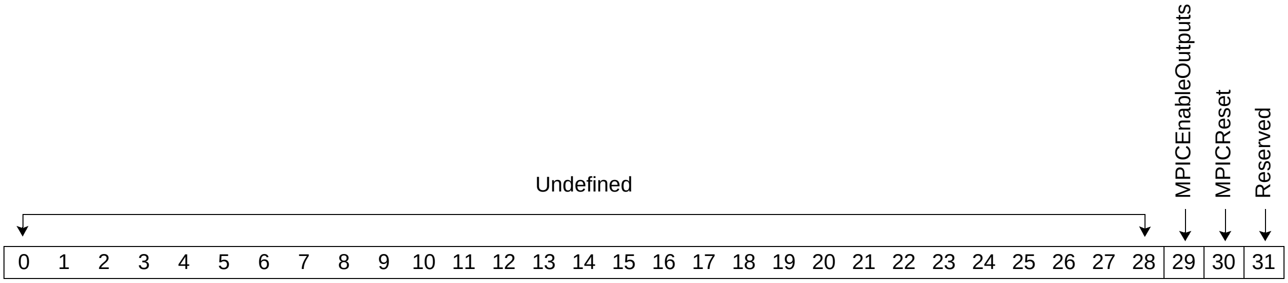


Bits	Field Name	Description	Access	Reset
0:31	HWInitState	This register is initialized to 0x00000000 when NORTH_BRIDGE_RESET_L is asserted. After NORTH_BRIDGE_RESET_L is deasserted, it can be set to any value that you want and that value will be maintained as long as CPC945 remains powered.	R/W	x0000 0000

12.4.5 CPC945 Toggle Register

This register contains bits to toggle (start/stop) actions within the IBM CPC945 Bridge and Memory Controller. Toggle bits should be reset after use.

Reset Value	x0000 0000
Address	xF800 00E0
Access Type	Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
0:28	Undefined	This field is all zeros.	R	x0000 0000
29	MPICEnableOutputs	This bit enables the MPIC register block to notify processors of pending interrupts. When cleared, this bit blocks all interrupt outputs from exiting CPC945. It is cleared by the assertion of NORTH_BRIDGE_RESET_L and SUSPEND_REQ_L. It is the responsibility of software to clear the bit. This bit will prevent all CPUs from receiving interrupts after reset and after exiting system sleep. Software can clear this bit at the same time it sets the Sleep bit to guarantee incoming interrupts cease. Software will set this bit after reset and exiting sleep once it is prepared to handle incoming interrupts. 0 Mask Interrupts 1 Enable CPU interrupts	R/W	0
30	MPICReset	This bit holds the MPIC in reset and does not clear the pending interrupts. To get MPIC out of reset, this bit must be set to 0b1. 0 Hold MPIC in reset 1 Enable MPIC to run	R/W	0
31	Reserved	Reserved.	–	0

12.5 Clocks and Power Management Registers

In order to optimize the electrical power consumption and thermal performance of computer systems built with CPC945, there is specific power management logic in CPC945 which switches off clocks to various parts of the chip when those parts are not needed. There is also logic to control the speeds at which different interfaces operate, allowing additional power savings and configuration control. In addition to saving power in the CPC945, these modes save significant system power by managing and optimizing the power used by the PPC970xx processor. The registers which control this behavior are documented in this section.

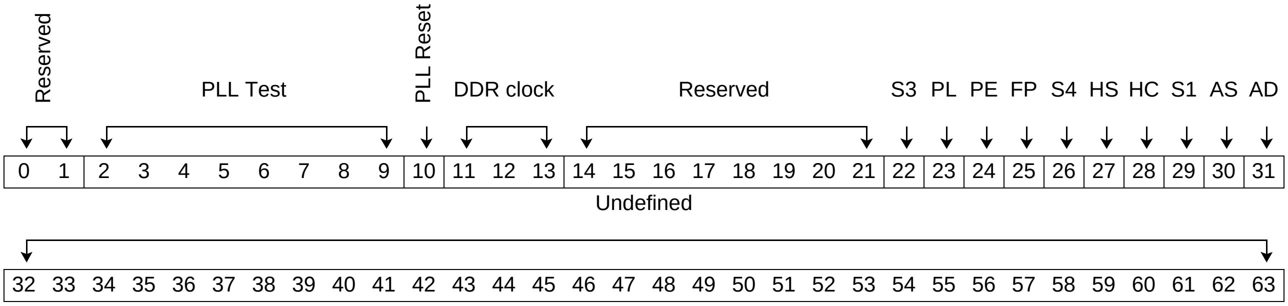
Table 12-4. PMR Address Space.

Address	Reg	Description	Page
0xF8000800	ClockControl	Clock Control Register	338
0xF8000810	PwrSystem	System Power Management Register	341
0xF8000820	PwrCPU	CPU Power Management Register	342
0xF8000830	PwrCPUQuiesce	Power Management System Quiesce Parameters	345
0xF8000840	PwrHT	HyperTransport Power Managment Register	345
0xF8000850	PLL1Control	PLL1 Control Register (PI)	346
0xF8000860	PLL2Control	PLL2 Control Register (DDR2)	350
0xF8000870	PLL3Control	PLL3 Control Register (PCI Express)	352
0xF8000880	PLL4Control	PLL4 Control Register (HT)	355
0xF8000890	PLLVis	PLL/Clock Visibility Control	358

12.5.1 Clock Control Register

The Clock Control Register provides software access for managing power by turning off unused clocks and PLLs. All clocks and PLLs are stopped and started cleanly (no spikes or short cycles), so logic does not have to be reset after stopping and restarting its clock.

Reset Value	0x000803BC 00000000
Address	0xF8000800
Access Type	Read/Write, Read Only



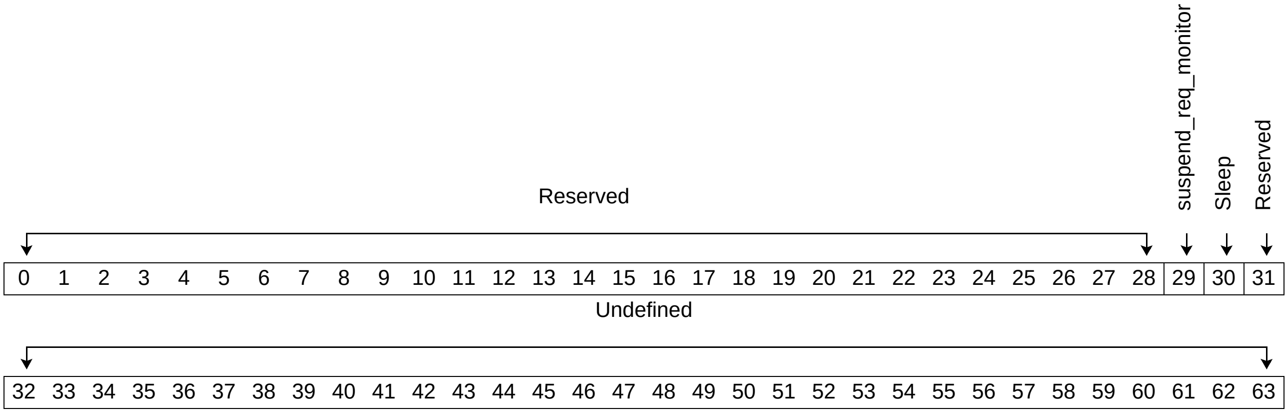
Bits	Field Name	Description	Access	Reset
0:1	Reserved	This field reads all 0’s	R/W	0x00
2:9	PLL Test	This field reads all 0’s	R	0x00
10	PLL Reset	For Manufacturing Test use only. Must be left at ’0’ for normal (nonmanufacturing test) operation.	R/W	0b0
11:13	DDR clock	The four bits map to a core speed as documented in the DDR2 clock speed table (*Table 12-6* on page 349). Writing a new value to this register will load the PLL2 Control Register with the value appropriate for the desired core speed. The update of the PLL2 Control Register will also clear the PLLLoaded bit in that register. In order to change the PLL speed, software is required to set either the ForcePLLReset or ForcePLLLoad bit in the PLL2 Control Register before the change is forced into the PLL, changing the core speed. When the PLL speed has been changed, the PLLLoaded bit in PLL2 Control will be set by the PLL2 Power Manager.	R/W	0b010
14:21	Reserved	This field will read all 0’s.	R	0x00
22	S3	EnablePLL3Shutdown- Shutdown used to enable the shutdown of PLL3 whenever all associated output clocks are stopped. After shutdown, PLL3 requires 100us to relock. PLL3 is automatically shutdown in Sleep regardless of the setting of this bit. 1 Shutdown PLL3 whenever the clocks to PCIe are stopped 0 Shutdown PLL3 only in sleep	R/W	0b1
23	PL	PciELogicStopEnable	R/W	0b1
24	PE	PciEClkEnable used to enable and disable clocks on the PCI Express bus. When enabled, PCI Express clocks are always running. When disabled The PMR logic stops the PCI Express clocks as soon as the PCIe logic indicates it has reached L23State.	R/W	0b1
25	FP	PciE_ForcedL23LinkState indicates that during the process of reaching L2/3 Ready (that is, PciE_InL23LinkState active), the Endpoint did not complete the process by sending a PME_To_Ack message and therefore the shutdown process timed out. In this event, the power can be removed immediately as it is assumed something has gone drastically wrong with the Endpoint (that is, the state is irrecoverable). 1 Forced to L23 State. 0 Normal L23 State.	R	n/a
26	S4	EnablePLL4Shutdown - Shutdown Used to enable the shutdown of PLL4 whenever all associate output clocks are stopped. After shutdown, PLL4 requires 100 ms to relock. PLL4 is automatically shut down in Sleep, regardless of the setting of this bit. 1 Shut down PLL4 whenever the clocks to HyperTransport are stopped by software (see Bit 27, below) 0 Shut down PLL4 only in Sleep	R/W	0b1

Bits	Field Name	Description	Access	Reset
27	HS	HTLogicStopEnable - Used to enable and disable stopping the bus rate and bit rate clocks to the internal HyperTransport bridge logic when the LDTSTOP_L sequence to the external HyperTransport devices should have resulted in the external devices stopping their internal use of the HyperTransport clock. This bit has no effect unless HT_LDTSTOP_L is asserted either by an external device or by setting HTClockControl to 0. If an external device requires use of the HyperTransport bus while HT_LDTSTOP_L is asserted, it must assert HT_LDTREQ_L in order to cause a deassertion of HT_LDTSTOP_L. If the CPU or PCI0 attempt to access HyperTransport while HT_LDTSTOP_L is asserted, then CPC945 will assert an internal LDTREQ_L signal, which will cause the clocks to restart. The clock to HyperTransport logic is stopped automatically when the system goes to sleep, regardless of the setting of this bit. 1 Stop the bus rate and bit rate clocks to HyperTransport whenever the LDTSTOP_L sequence should have caused all external HyperTransport devices to stop their internal use of the HyperTransport clock. 0 Keep the internal HyperTransport clock running, even if the LDTSTOP_L sequence should have caused all external HyperTransport devices to stop their internal use of the HyperTransport clock	R/W	0b1
28	HC	HTClkEnable - Used to enable and disable clocks on the HyperTransport bus. When enabled, HyperTransport clocks are always running. When disabled, CPC945 generates the HyperTransport protocol for clock stopping, signaling LDTSTOP_L to all other HyperTransport devices whenever the bus is idle. Other devices can use HT_LDTREQ_L to signal to CPC945 that they need the HyperTransport bus enabled, which will override HTClkEnable. CPC945 does not control the external HyperTransport clock but uses the handshake to tell the external HyperTransport devices when to start and stop their internal uses of the HyperTransport clock. The LDTSTOP_L sequence is used automatically when the system goes to sleep, regardless of the setting of this bit. 1 Keep the HyperTransport clock running in all external HyperTransport devices whenever the system is awake. When this bit is a 1, software can always access any external HyperTransport device, and the clock to CPC945's internal HyperTransport logic is always on (except in sleep).	R/W	0b1
29	S1	EnablePLL1 Shutdown - Used to enable the shutdown of PLL1 whenever all of its associated output clocks are stopped. After a shutdown, PLL1 requires 100 ms to relock. PLL1 is automatically shut down in Sleep, regardless of the setting of this bit. 1 Shut down PLL1 whenever the clocks to the PI interface is stopped by software. 0 Shut down PLL1 only in Sleep.	R/W	0b1
30	AS	APILogicStopEnable - Used to enable and disable stopping the bus rate and bit rate clocks to the internal PI interface logic when the QReq/ QAck sequence has quiesced all CPUs. If any CPU deasserts its QReq signal, or any other device (PCI Express, HyperTransport) needs to start a snooped I/O cycle, then the clocks will restart automatically. The clock to the PI logic is stopped automatically when the system goes to Sleep, regardless of the setting of this bit. 1 Stop the bus rate and bit rate clocks to PI logic whenever the QReq/QAck sequence has completed and all CPUs attached to this bus are quiesced. 0 Keep the internal PI clocks running, even if the QReq/ QAck sequence indicates that all CPUs are quiesced.	R/W	0b0
31	AD	APIDebugClkEnable - Used to enable and disable the clock to the PI debug logic. The clock to PI debug logic is stopped automatically when the system goes to Sleep, regardless of the setting of this bit. 1 Keep the clocks to the PI debug logic running. 0 Stop all clocks to the PI debug logic.	R/W	0b0
32:63	Undefined	This field will read all 0's.	R	0x000000

12.5.2 System Power Management Register

The system power management register contains information used by CPC945 to determine whether it should go to the system sleep state when the power manager deasserts the SUSPEND_REQ_L signal.

Reset Value	0x00000000 00000000
Address	0xF8000810
Access Type	Read/Write, Read Only

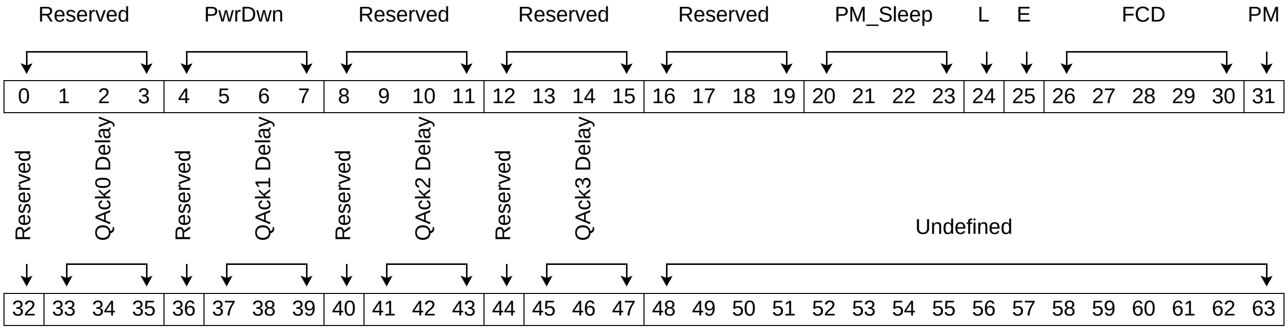


Bits	Field Name	Description	Access	Reset
0:28	Reserved	This field will read all 0’s.	R	0x00000000
29	suspend_req_monitor	This bit is cleared when CPC945 receives a suspend_req request from the PMU and can be used to monitor the occurrence of suspend_req_l. The bit can be set or cleared by software and controls no function.	R/W	0b0
30	Sleep	This field controls whether CPC945 will enter the Sleep power saving state when the SUSPEND_REQ_L signal from the Power Manager is asserted. This bit must be set by the processor but will be cleared by CPC945 when the Power Manager deasserts the SUSPEND_REQ_L signal.	R/W	0b0
31	Reserved	This field will read all 0’s.	R	0b0
32:63	Undefined	This field will read all 0’s.	R	0x00000000

12.5.3 CPU Power Management Register

The CPU Power Management Register controls the per-CPU behavior for power management.

Reset Value	0x00000054 00000000
Address	0xF8000820
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:3	Reserved	This field will read all 0’s.	R	0b0000
4:7	PwrDwn	Power Down Enable Bit for CPUs. This bit controls whether the CPU will assert PM_SLEEP3 when the CPU is safely in the sleep state. Bit [4] = Power Down Enable bit for CPU3 Bit [5] = Power Down Enable bit for CPU2 Bit [6] = Power Down Enable bit for CPU1 Bit [7] = Power Down Enable bit for CPU0	R/W	0b0000
8:11	Reserved	This field will read all 0’s.	R	0x0
12:15	Reserved	This field will read 0b0000.	R	0b0000
16:19	Reserved	This field will read all 0’s.	R	0x0
20:23	PM_Sleep	This bit provides software the ability to check if it is OK to power off CPUs. It reflects the current state of the PM_SLEEP signal, which is asserted only when CPU is in the “Sleeping” state and CPUPwrDnEnabled3 (bit 4) is set. This bit is asserted at the same time as QAck is asserted. The assertion of QAck also causes an internal CPC945 interrupt that will be latched and sent to the processor through MPIC. Bit [20] = PMSLEEP for CPU3 Bit [21] = PMSLEEP for CPU2 Bit [22] = PMSLEEP for CPU1 Bit [23] = PMSLEEP for CPU0	R	0b0000
24	L	The APsync_lock bit is used to indicate that the CPC945 PMR has successfully locked on to the externally supplied APsync, and all internal time0 or APsync timing should now be synchronous with the external APsync pulse. 1 = APSync Locked	R	0b0

Bits	Field Name	Description	Access	Reset
25	E	This bit indicates where the APsync signal is generated. If set, the APsync is generated externally to CPC945 and is an input to CPC945. The power manager logic will sync up CPC945 clock controls and internal APsync pulses to the external APsync input. If cleared, the power manager will internally generate the APsync and the pin on CPC945 is used as an output to drive the APsync signal.	R/W	0b1
26:30	FCD	This field specifies that after a power tuning transaction has been reflected to all CPUs, CPC945 will wait 1-64 PI bus clock cycles prior to looking for a QReq_l signal assertion from the CPUs. This time allows the CPUs to deassert any current QReq and then reassert the QReq signal for the power tuning transaction. The number of bus cycles to delay is actually the field value plus 1. The default value of 0b01010 will cause a delay of 10 + 1 or 11 PI bus cycles.	R/W	0b01010
31	PM	This bit indicates that the system will remove power to CPUs whose PM_Sleep signal is asserted. If set, CPC945 will tristate all output buffers to any interface that has all its associated PM_Sleep signals asserted. If set, CPC945 will also disable all input buffers on a slept interface.	R/W	0b0
32	Reserved	This field will read all 0’s.	R	0b0
33:35	QAck0 Delay	QAck0 assertion delay - This bit field is used to add an additional delay to the assertion of QAck0. In reality the entire waveform is delayed, resulting in adding equal delay to the deassertion as well.	R/W	0b000
36	Reserved	This field will read all 0’s	R	0b0
37:39	QAck1 Delay	QAck1 assertion delay - This bit field is used to add an additional delay to the assertion of QAck1.	R/W	0b000
40	Reserved	This field will read all 0’s.	R	0b0
41:43	QAck2 Delay	QAck2 assertion delay - This bit field is used to add an additional delay to the assertion of QAck2.	R/W	0b000
44	Reserved	This field will read all 0’s.	R	0b0
45:47	QAck3 Delay	QAck3 assertion delay - This bit field is used to add an additional delay to the assertion of QAck3.	R/W	0b000
48:63	Undefined	This field will read all 0’s.	R	0x00000000

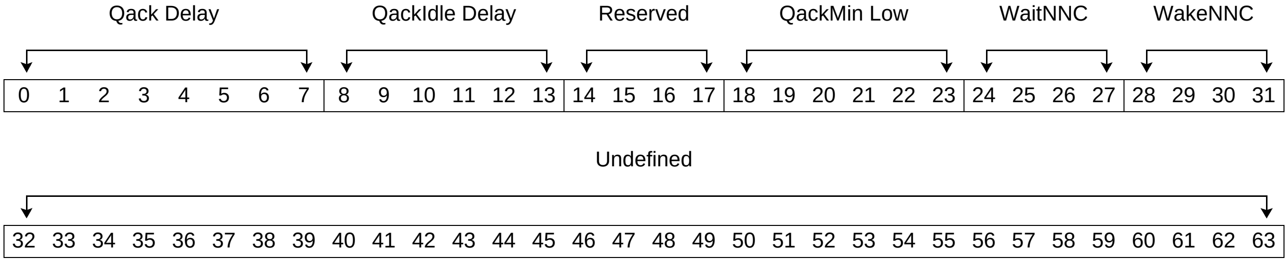
12.5.4 CPU Quiesce Timing Register

This register is intended to put all the System Quiesce parameters in one place. The delay counters count at the bus clock rate, which is 1/2 the PI bit rate.

Reset Value 0xFFFFFFFF 00000000

Address 0xF8000830

Access Type Read/Write, Read Only



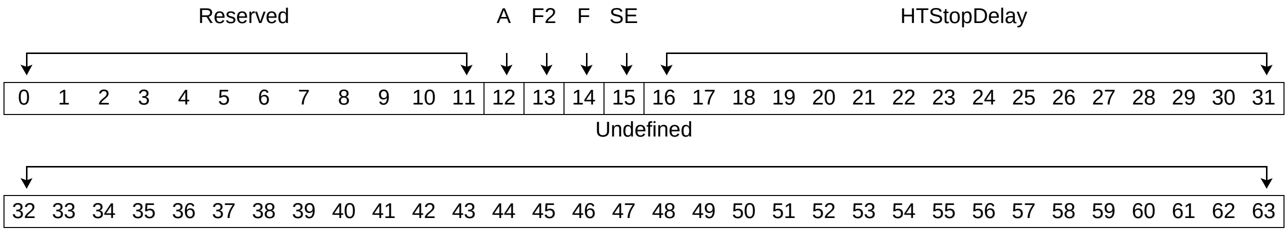
Bits	Field Name	Description	Access	Reset
0:7	Qack Delay	This field specifies that CPC945 will wait 16 to 256 PI bus clock cycles before asserting QAck_ after the bus is idle and a processor's QReq_ has been asserted. This feature is used to keep processor bus masters from encountering a delay following the QAck_ de-assertion. The number of bus cycles to delay is the field value plus 1. The value 0x0f, for example, encodes a QAckDelay of 16. Since 16 is the minimum delay allowed, values between 0x00 and 0x0e are restricted in this field.	R/W	0xFF
8:13	QackIdle Delay	This field specifies that CPC945 will wait 8 to 64 PI bus clock cycles after QAck_ is deasserted before the bus allows any snoop transactions to be presented. The number of bus cycles to delay is the field value plus 1. The value 0b000111, for example, encodes a QAckIdleDelay of 8. Since 8 is the minimum delay allowed, values between 0b000000 and 0b000110 are restricted in this field.	R/W	0b111111
14:17	Reserved	This field will read 0b1111	R	0b1111
18:23	QackMin Low	This field specifies that CPC945 will wait 8 to 64 PI bus clock cycles after QAck_ is asserted before it considers the CPU to actually be in any Quiesced state. This is to guarantee that the PPC970xx actually has the time it needs to descend into its Nap mode before QAck is deasserted. Otherwise, bad things happen. The number of bus cycles to delay is actually the field value plus 1. The value 0b000111, for instance, encodes a QAckMinLowTime of 8. Since 8 is the minimum delay allowed, values between 0b000000 and 0b000110 are restricted in this field.	R	0b111111
24:27	WaitNNC	This field specifies that CPC945 will wait 2 to 16 PI bus clock cycles after a CPU state machine has told the system to not accept new bus commands before it actually believes that the bus logic has heard it. If the system presents a snoop to the CPU within the WaitNoNewCmds window, the CPU still accepts the snoop and restarts the process of finding an idle bus. This is to guarantee that the bus is idled cleanly and there are no transactions in progress when QAck is asserted. The number of bus cycles to delay is actually the field value plus 1. The value 0b0001, for instance, encodes a WaitNoNewCmds value of 2. Since 2 is the minimum delay allowed, the value of 0b0000 is restricted in this field.	R/W	0b1111

Bits	Field Name	Description	Access	Reset
28:31	WakeNNC	This field is identical in description and usage to the WaitNoNewCmds field. The only difference is that it specifies the delay for a CPU exiting the sleep state rather than one entering either the nap or sleep state.	R/W	0b1111
32:63	Undefined	This field will read all 0’s.	R	0x00000000

12.5.5 HyperTransport Power Management Register

The HyperTransport Power Management Register controls the amount of time the HyperTransport bus must be idle before CPC945 will generate an LDTSTOP.

Reset Value	0x0001012F 00000000
Address	0xF8000840
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:11	Reserved	This field will read all 0’s.	R	0x000
12	A	This field reflects the state of the LDTStopAck signal from the HT logic block.	R	0b0
13	F2	Setting this bit asserts HT_LDTStop_L on the HyperTransport interface (after HTStopDelay expires). It overrides all other controls and ignores the HT_LDTREQ_L signal. The bit and HT_LDTStop_L signal are reset when the LDTStopAck signal is asserted by the HT logic.	R/W	0b0
14	F	Setting this bit asserts HT_LDTStop_L on the HyperTransport interface (after HTStopDelay expires). It overrides all other controls and ignores the HT_LDTREQ_L signal. The bit and HT_LDTStop_L remains asserted until such time as the bit is cleared by a register write.	R/W	0b0
15	SE	SleepEnable - This bit generates a HyperTransport cold reset during system sleep. Cold reset is required for the case where any HyperTransport device loses power during system sleep. If all HyperTransport devices remain powered during system sleep, then the LDTSTOP protocol will keep these devices quiesced and they will not lose link state during system sleep. 1 Generate HyperTransport cold reset during system sleep. 0 Do not generate HyperTransport cold reset during system sleep.	R/W	0b1
16:31	HTStopDelay	This field controls the number of 300 MHz reference clocks the HyperTransport bus must remain idle before CPC945 will generate an LDTSTOP to the bus. The actual delay is (HTStopDelay + 1) × 3.3 ns. The maximum delay programmable in CPC945 is 218.43 µs. The default delay is 1 µs [actually (0x0012F+1) × 3.3ns = 1.0013 µs].	R/W	0x012F
32:63	Undefined	This field will read all 0’s.	R	0x00000000

12.5.6 PLL1 Control Register

The PLL1 Control Register configures the PI PLL. It is set at reset to support a CPU running a 3:1 ratio between its core clock and the PI bit rate. The CPU core runs between 2400 MHz and 2664 MHz. The PLL1 Control Register is reloaded during processor reset by the SPU. Its value is initialized at that point to reflect the proper configuration for the CPUs attached to CPC945. At the same time, the SPU sets the ForcePLL-Reset bit to reset the PLL concurrently with the loading of new parameters.

When performing dynamic speed control of the processors, it might be necessary for system software to change the configuration of PLL1. This procedure guarantees a stable PLL1 transition:

1. Software naps or sleeps all but one processor.
2. The last processor loads a new value into PLL1Control, clearing PLLLoaded.
3. The last processor sets APILogicStopEnable and EnablePLL1Shutdown in the clock control register, if not already set.
4. The last processor sets the decrementer to wake it. The minimum delay is the time required for the CPU to quiesce and CPC945 to assert QAck. This delay is affected by parameters in the PwrCPU-Quiesce register.
5. The last processor naps.
6. CPC945 asserts QAck to the last processor (or all processors if they are Napping).
7. CPC945 sees all processors quiesced and APILogicStopEnable set, so it stops the PI clocks.
8. CPC945 sees the PI clocks stopped and EnablePLL1Shutdown set, so it stops PLL1 and loads the new configuration from PLL1Control.
9. The last processor wakes from its decrementer interrupt.
10. CPC945 sees the last processor de-assert QReq, and starts waking PLL1 with the new configuration.
11. When PLL1 is re-locked, CPC945 restarts clocks to the PI interface.
12. When all clocks are running, CPC945 deasserts QAck and the last processor resumes operation.
13. Last processor checks PLLLoaded bit to make sure that PLL1 was reloaded with the new configuration.
14. Last processor wakes other processors, if desired.

The values which go into the PLL1 Control Register depend upon the speed of the PI interface and the ratio between the reference clock input and the PI interface. The range of frequencies available for different processor and PI bus ratios are shown in *Table 12-5 PLL1 Clock Settings*. *Table 12-5* documents the PLL settings for CPUs running between 600 MHz and 2700 MHz at all three ratios: 2:1, 3:1, and 4:1.

Table 12-5. PLL1 Clock Settings.

CPU to Bus Ratio	Reference Clock		CPU [3] Frequency		Bus Frequency (Bus Clock)		Bit Rate [2] (Data Clock)		PLL1 VCO		Divisor		Feedback	Control Register Setting	Tune
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	A	B			
2:1	125	150	1000	1200	250	300	500	600	1000	1200	4	8	1	x8400 11B6	1B6
2:1	150	225	1200	1800	300	450	600	900	600	900	2	4	1	x8240 1134	134
2:1	225	333	1800	2664	450	666	900	1332	900	1332	2	4	1	x8240 11B4	1B4
3:1	100	112.5	1200	1350	200	225	400	450	800	900	4	8	1	x8400 1136	136[4]
3:1	112.5	150	1350	1800	225	300	450	600	900	1200	4	8	1	x8400 11B6	1B6[4]
3:1	150	225	1800	2700	300	450	600	900	600	900	2	4	1	x8240 1134	134 [4]
4:1	125	150	1000	1200	125	150	250	300	1000	1200	8	8	1	x8800 11B6	1B6
4:1	150	225	1200	1800	150	225	300	450	600	900	4	4	1	x8440 1134	134
4:1 [5]	225	333	1800	2664	225	333	450	666	900	1332	4	4	1	x8440 11B4	1B4
4:1	300	337.5	2400	2700	300	337.5	600	675	600	675	2	2	1	x8220 1132	132

Notes:

1. VCO = Ref clk × B divisor × Feedback.
2. Bus is a double data rate so Bit Rate is 2x the Bus Frequency.
3. The CPU frequency = BitRate × CPU to Bus Ratio.
4. For the 3:1 bus ratio, alternatives using slower reference clocks are shown.
5. Reset default value.

The PLL tuning bits modify the PLL loop parameters by modifying the internal gains of the charge pumps. This external control allows the PLL to be stable over a wide range of frequencies and multiplication factors. Programmability helps ensure proper PLL operation with no hardware impact, should changes be required.

Reset Value	0x344011B4 04000000
Address	0xF8000850
Access Type	Read/Write, Read Only

R	F	L	P	ADiv	Reserved	BDiv	Reserved	Feedback	Reserved	Tune
0	1	2	3	4 5 6 7	8	9 10 11	12 13 14	15 16 17 18 19	20 21	22 23 24 25 26 27 28 29 30 31
				FSM State					Reserved	

32 33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

Bits	Field Name	Description	Access	Reset
0	R	This field specifies that the PLL will be immediately forced through a reset cycle where the values in this register will be loaded into the PLL. **The ForcePLLReset function performs no synchronization or protocol stopping, so setting this bit on an active bus is highly likely to lose data.** This bit is intended to be used only by the SPU for setting the initial state of PLL1. After the PLL is updated, this bit is reset to 0, indicating that the update has occurred.	R/W	0b0
1	F	This field specifies that the PLL will be immediately updated with the values in this register. No PLL reset will occur. **This can destabilize the PLL and should not be used without sufficient testing to determine that it is safe.** After the PLL is updated, this bit is reset to 0, indicating that the update has occurred. If new values are loaded in this register and neither ForcePLLReset nor ForcePLLLoad are set, they will be loaded into the PLL the next time it is stopped using the power management mechanism. For PLL1, that means that EnablePLL1Shutdown and APILogicStopEnable must be set in the ClockControl Register (0xF8000020). All CPUs must be quiesced as well. The contents of this register will also be loaded into PLL1 while it is stopped during system sleep.	R/W	0b0
2	L	This field will reflect the value of PLL1's PLLLock output at all times. It tells whether or not the PLL is locked to its programmed frequency. Whenever the PLL is in reset (states RESET and PLL SHUTDOWN) PLLLock is deasserted. When reset is deasserted, the PLL starts tracking in on lock. PLLLock will assert within 100 µs of reset deasserting. This signal is not used because as the PLL is locking, the lock output can toggle back and forth until the PLL is finally locked. **This register is to be used only as feedback for debug to verify that the PLL believes itself to be locked.**	R/W	0b0
3	P	This bit is set to a one whenever the PLL control latch is loaded with a new value. If software desires confirmation that the new values in this register have been loaded into the PLL, this bit must be cleared when loading a new value into this register.	R/W	0b0
4:7	ADiv	This field holds the A Divider control bits for PLL1. The A divider takes the PLL VCO frequency and divides it down to create the mem_clk. The value for this divider should be taken from the *Table 12-5 PLL1 Clock Settings* on page 347.	R/W	0b0100
8	Reserved	This field will read all 0's.	R	0b0
9:11	BDiv	This field holds the B Divider control bits for PLL1. The B Divider takes the PLL frequency and divides it down to generate the ddr_clk rate which is also used as the feedback clock. The value for the B divider should be taken from the *Table 12-5 PLL1 Clock Settings* on page 347.	R/W	0b010
12:14	Reserved	This field will read all 0's	R	0b000
15:19	Feedback	This field holds the feedback divider control bits for PLL1. The FeedBack divider divides down the feedback clock in the PLL to be used by the phase comparator against the reference clock. The settings for the FeedBack divider should be taken from the *Table 12-5 PLL1 Clock Settings* on page 347.	R/W	0b00001
20:21	Reserved	This field will read all 0's	R	0b0
22:31	Tune	This field holds the TUNE[9:0] control bits for PLL1. The suggested values are shown in the *Table 12-5 PLL1 Clock Settings* on page 347.	R/W	0b0110110100

Bits	Field Name	Description	Access	Reset
32:42	FSM State	This field is used to monitor the state of the PLL state machine which control the PLL. The bits are defined as follows: Bit [32] = wake1 Bit [33] = shutdown2 Bit [34] = shutdown1 Bit [35] = sleep Bit [36] = load Bit [37] = run Bit [38] = wake4 Bit [39] = wake3 Bit [40] = wake2 Bit [41] = wake0 Bit [42] = reset	R	0x001
43:63	Reserved	This field will read all 0’s.	R	0x000000

Table 12-6. PLL2 DDR2 Core Speed.

Core Speed	Ref Clk min (slew)	ddr _clk min (slew)	FB	A Divisor	B Divisor	M	PLL VCO	mem _clk(A)	ddr _clk(B)	CNTL Register Setting	Tune
000	56.25	225	4	8	4	16	1066.67	133.33	266.67	884042B8	2B8
001	45.00	225	5	8	4	20	1333.34	166.67	333.34	884052B8	2B8
010[1]	50.00	300	6	6	3	18	1200.01	200.00	400.00	863062B8	2B8
011	64.29	450	7	4	2	14	933.34	233.33	466.67	842072B8	2B8
100	56.25	450	8	4	2	16	1066.67	266.67	533.34	842082B8	2B8
101	50.00	450	9	4	2	18	1200.01	300.00	600.00	842092B8	2B8
110	45.00	450	10	4	2	20	1333.34	333.34	666.67	8420A2B8	2B8
111	54.55	600	11	2	1	11	733.34	366.67	733.34	8210B238	238

Note:

1. Reset default value.

12.5.7 PLL2 Control Register

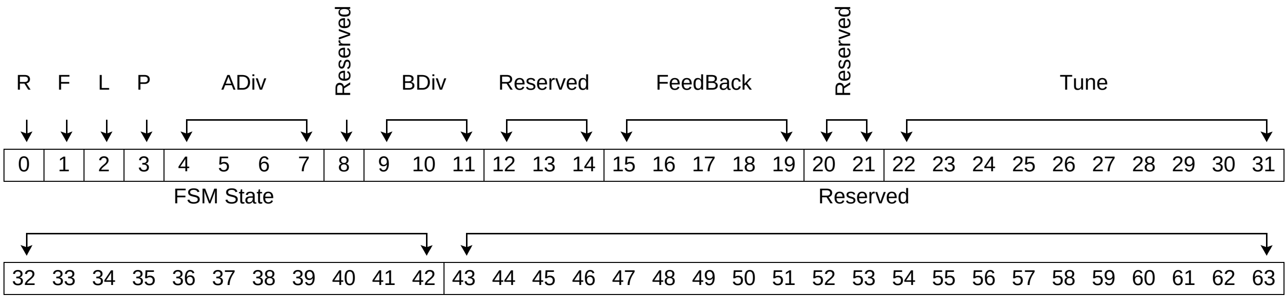
PLL2 generates the clocks for the DDR2 SDRAM memory interface. Its speed is set to 400 MHz at power-on reset.

It is not expected that software will modify the PLL2 Control Register. If software does modify PLL2Control, the only safe method of updating is to load in a new value and then take the system all the way into and out of the system sleep state. It is possible to load a new value and set the ForcePLLLoad bit, which will update the settings of the PLL. However, this is not guaranteed to be safe. It might be possible for the SPU to perform this operation at boot time without unduly confusing CPC945.

It is possible for software to modify the PLL2 Control Register and change the core frequency of CPC945. However, this facility is designed only to support this function when CPC945 is idle and the memory controller is not activated. To use this facility, read the SPD ROMs on the SDRAM DIMMs to determine DIMM speed. Then, if the core/SDRAM speed requires changing, set the appropriate three bit code in the Clock Control Register. This puts the hardware default values into the PLL2 Control Register. If these values need modifying, a read-modify-write should be performed on the appropriate bit fields. After the appropriate value is in the PLL2 Control Register, a RMW of the register to set the ForcePLLReset bit is performed. At this point, software should ensure the next set of instructions are present in the instruction cache. Software should execute some form of loop which keeps it from accessing CPC945 for the next 200 µs. This will prevent any traffic from entering CPC945 while the core clock is stopped. Once the 200 µs timer has expired, software can check the PLLLoaded bit to verify that the new core frequency has been set. Once CPC945 is at the correct frequency for the memory, the Hardware Init code can initialize the memory controller as appropriate.

The PLL tuning bits are used to modify the PLL loop parameters by modifying the internal gains of the charge pumps. This external control allows the PLL to be stable over a wide range of frequencies and multiplication factors. Programmability helps ensure proper PLL operation with no hardware impact, should changes be required.

Reset Value	0x363062B8 04000000
Address	0xF8000860
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	R	This field specifies that the PLL will be immediately forced through a reset cycle where the values in this register will be loaded into the PLL. **The ForcePLLReset function performs no synchronization or protocol stopping, so setting this bit on an active bus is highly likely to lose data.** This bit is intended to be used only by the SPU for setting the initial state of PLL2. After the PLL is updated, this bit is reset to 0, indicating that the update has occurred.	R/W	0b0
1	F	This field specifies that the PLL will be immediately updated with the values in this register. No PLL reset will occur. **This can destabilize the PLL and should not be used without sufficient testing to determine that it is safe.** After the PLL is updated, this bit is reset to 0, indicating that the update has occurred. If new values are loaded in this register and neither ForcePLLReset nor ForcePLLLoad are set, they will be loaded into the PLL the next time it is stopped using the power management mechanism. For PLL1, that means that EnablePLL1Shutdown and APILogicStopEnable must be set in the ClockControl Register (0xF8000800). All CPUs must be quiesced as well. The contents of this register will also be loaded into PLL2 while it is stopped during System Sleep.	R/W	0b0
2	L	This field will reflect the value of PLL2's PLLLock output at all times. It tells whether or not the PLL is locked to its programmed frequency. Whenever the PLL is in reset (states RESET and PLL SHUTDOWN) PLLLock is deasserted. When reset is deasserted, the PLL starts tracking in on lock. PLLLock will assert within 100us of reset deasserting. This signal is not used because as the PLL is locking, the lock output can toggle back and forth until the PLL is finally locked. **This register is to be used only as feedback for debug to verify that the PLL believes itself to be locked.**	R/W	0b0
3	P	This bit is set to a one whenever the PLL control latch is loaded with a new value. If software desires confirmation that the new values in this register have been loaded into the PLL, this bit must be cleared when loading a new value into this register.	R/W	0b0
4:7	ADiv	This field holds the A Divider control bits for PLL2. The A divider takes the PLL VCO frequency and divides it down to create the memory clock. The value for this divider should be taken from the PLL2 clock settings table. (See *Table 12-6* on page 349.)	R/W	0b0100
8	Reserved	This field will read all 0's.	R	0b0
9:11	BDiv	This field holds the B Divider control bits for PLL2. The B Divider takes the PLL frequency and divides it down to generate the ddr_clk rate which is also used as the feedback clock for the PLL. The value for the B divider should be taken from the PLL2 clock settings table. (See *Table 12-6* on page 349.)	R/W	0b010
12:14	Reserved	This field will read all 0's.	R	0b000
15:19	FeedBack	This field holds the feedback divider control bits for PLL2. The FeedBack divider divides down the feedback clock in the PLL to be used by the phase comparator against the reference clock. The settings for the feedback divider should be taken from the PLL2 clock settings table. (See *Table 12-6* on page 349.)	R/W	0b00001
20:21	Reserved	This field will read all 0's.	R	0b00
22:31	Tune	This field holds the TUNE[9:0] control bits for PLL2. The suggested values are shown in the PLL2 clock settings table. (See *Table 12-6* on page 349.)	R/W	0b01101101 00

Bits	Field Name	Description	Access	Reset
32:42	FSM State	This field is used to monitor the state of the PLL state machine which control the PLL. The bits are defined as follows: [32] = wake1 [33] = shutdown2 [34] = shutdown1 [35] = sleep [36] = load [37] = run [38] = wake4 [39] = wake3 [40] = wake2 [41] = wake0 [42] = reset	R	0b00000000 001
43:63	Reserved	This field will read all 0’s.	R	0x0000000

12.5.8 PLL3 Control Register

Table 12-7. PLL3 Default Settings.

Pdiv	Mdiv	N1div	N2div	PLL VCO	F(n1)	F(n2)	pllout	V	O
1	5	5	4	2500	500	625	625.00	1	1

The PLL is programmed by selection of the M, N1, N2, and P multipliers/dividers and the NS select pin. RefClk is the frequency of the input signal, M, N1, N2, and P are the multiplier/divider ratios (selected via the appropriate input pin/bit settings), and the VCO frequency f_{VCO} is defined as:

$$f_{VCO} = (RefClk \times M \times N1)/P$$

The key equations describing the output frequencies at the PLL output (pllout) are:

$pllout = f_{VCO}/N1$ if the N1 divider is selected to control the PLL output

$pllout = f_{VCO}/N2$ if the N2 divider is selected to control the PLL output

The O bit controls whether the N1 or N2 divider is selected to control the PLL output.

The key constraints in choosing values for the dividers are:

- Divider ratios N1 and N2 must be 2 – 10
- Divider ratio P must be 1 – 8
- Multiplier ratio M must be 2 – 63
- REFCLK must be 44 – 350 MHz
- VCO frequency f_{VCO} must be 2.3 – 2.9 GHz

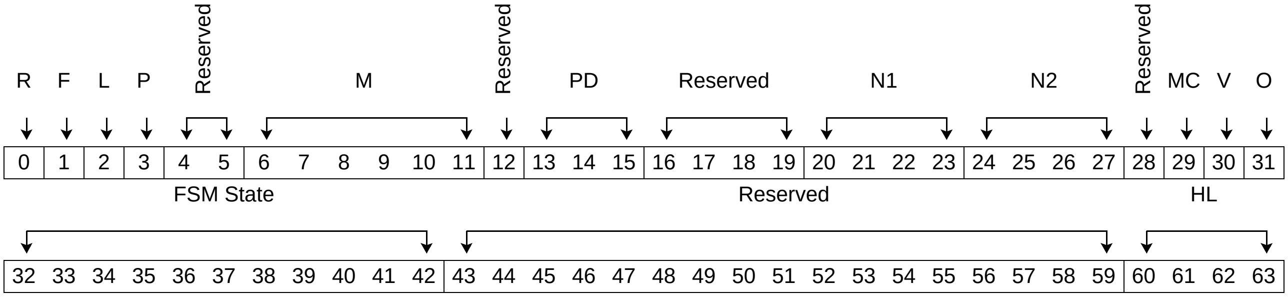
In general it is best to keep the product M x N1 as low as possible and the VCO frequency as high as possible for low output jitter.

The procedure for selecting the appropriate values can be summarized as:

1. Select an input clock frequency (REFCLK) as high as possible. This allows for a higher VCO frequency as well as a lower M × N1 product.

2. Select the appropriate values for P, M, N1, and N2 based on the desired output frequency. Select a low value for N1 and check that f_{VCO} is in the appropriate range (2.3 – 2.9 GHz). It is desirable to keep the VCO frequency at the top of the range, or as close to the top as possible, while also minimizing the product of the M and N1 divide ratios, M × N1.
3. Repeat steps 1 and 2, if needed.
4. Select the appropriate settings for the dividers from the tables in *Table 12-7 PLL3 Default Settings* on page 352.

Reset Value	0X30510543 04000007
Address	0xF8000870
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	R	This field specifies that the PLL will be immediately forced through a reset cycle where the values in this register will be loaded into the PLL. **The ForcePLLReset function performs no synchronization or protocol stopping, so setting this bit on an active bus is highly likely to lose data.** This bit is intended to be used only by the SPU for setting the initial state of PLL3. After the PLL is updated, this bit is reset to 0, indicating that the update has occurred.	R/W	0b0
1	F	This field specifies that the PLL will be immediately updated with the values in this register. No PLL reset will occur. **This can destabilize the PLL and should not be used without sufficient testing to determine that it is safe.** After the PLL is updated, this bit is reset to 0, indicating that the update has occurred. If new values are loaded in this register and neither ForcePLLReset nor ForcePLLLoad are set, they will be loaded into the PLL the next time it is stopped using the power management mechanism. For PLL1, that means that EnablePLL1Shutdown and APILogicStopEnable must be set in the ClockControl Register (0xF8000020). All CPUs must be quiesced as well. The contents of this register will also be loaded into PLL3 while it is stopped during System Sleep.	R/W	0b0
2	L	This field will reflect the value of PLL3's PLLLock output at all times. It tells whether or not the PLL is locked to its programmed frequency. Whenever the PLL is in reset (states RESET and PLL SHUTDOWN), PLLLock is deasserted. When reset is deasserted, the PLL starts tracking in on lock. PLLLock will assert within 100 µs of reset deasserting. This signal should not be used by software as an indicator, because as the PLL is locking, the lock output can toggle back and forth until the PLL is finally locked. **This register is to be used only as feedback for debug to verify that the PLL believes itself to be locked.**	R/W	0b0
3	P	This bit is set to a one by hardware whenever the PLL control latch is loaded with a new value. If software clears this bit (to zero) when loading a new PLL setting into this register, a subsequent read and test of this bit can determine if the new values were loaded.	R/W	0b0

Bits	Field Name	Description	Access	Reset
4:5	Reserved	This field will read all 0’s.	R	0b00
6:11	M	This field is used to set up the multiply factor for the M multiplier of the PLL; a factor of between 2 and 63 is provided.	R/W	0b000101
12	Reserved	This field will read all 0’s.	R	0b0
13:15	PD	This field is used to set up the divider ratio for the P divider; a factor of between 2 and 8 is provided.	R/W	0b001
16:19	Reserved	This field will read all 0’s.	R	0x0
20:23	N1	This field controls the N1 divider which divides down the PLL VCO frequency; a factor of between 2 and 10 is provided. However this divider is not used in this application.	R/W	0b0101
24:27	N2	This field controls the N2 divider which divides the PLL VCO frequency down to generate the 625 MHz clock used to drive the PCI Express HSS logic; a factor of between 2 and 10 is provided.	R/W	0b0100
28	Reserved	This field will read all 0’s.	R	0b0
29	MC	The MC field is used to override the lookup table for the M-divider setting. Due to area and timing constraints not all of the possible bit mappings for the M-Divide field were coded into the RTL. The RTL lookup table only supports divider settings from 2-10. For values greater than 10 the Mdivider setting should be looked up in the PLL spec and entered into the M field. Additionally, the MC bit must be set to 1b1 to override the table lookup. 1 Override Mdivider lookup table 0 Use Mdivider lookup table	R/W	0b0
30	V	This field is used to set up the VCO for the desired frequency range according to desired VCO frequecncy. 1 VCO frequency 2.3 – 2.6 GHz 0 VCO frequency 2.6 – 2.9 GHz	R/W	0b1
31	O	This bit is the output select for the PLL. The PLL output can be driven by the output of N1 or N2 dividers. This bit controls the multiplexer which selects which divider will drive the clock output of the PLL. 1 N2 divider drives PLL output 0 N1 divider drives PLL output	R/W	0b1
32:42	FSM State	This field is used, primarily during system bring up and debug, to monitor the state of the PLL state machine which controls the PLLs. The bits are defined as follows: Bit [32] = wake1 Bit [33] = shutdown2 Bit [34] = shutdown1 Bit [35] = sleep Bit [36] = load Bit [37] = run Bit [38] = wake4 Bit [39] = wake3 Bit [40] = wake2 Bit [41] = wake0 Bit [42] = reset	R	0x001
43:59	Reserved	This field will read all 0’s.	R	0x000000
60:63	HL	There are three additional PLLs in PCIe, one in each of the three PCIe PHY macros. This field reflects the state of the PLL lock signals from those PLLs. A ’0111’ indicates that all 3 are locked. 1 PLL locked 0 PLL not locked Bit 60 should always be ’0’.	R	0b0000

12.5.9 PLL4 Control Register

PLL4 generates the clocks for HyperTransport. The speed of the link is controlled by the Link Frequency Register in HyperTransport configuration space.

When the Link Frequency Register is modified, it loads the PLL4 Control register with the appropriate values. PLL4 is updated using the standard PLL update mechanism, which is tied into the HyperTransport defined mechanisms for stopping clocks before changing frequencies. If the values automatically loaded into PLL4 by writing to the Link Frequency Register need modification before being used by the PLL, then software can modify them appropriately in the PLL4 Control Register before starting the HyperTransport link speed change sequence.

The PLL V bits modify the PLL loop parameters by modifying the internal gains of the charge pumps. This external control allows the PLL to be stable over a wide range of frequencies and multiplication factors. Programmability helps ensure proper PLL operation with no hardware impact, should changes be required.

Table 12-8. PLL4 Control Register Default Values.

LinkFreq Code	Pdiv	Mdiv	Ndiv	V	PLL VCO	htbit (A)	htwlnk(B)	Ext Div	Bus Clk	ht_core_clk	CNTL Reg
000[1]	2	6	8	1	3200	400	50	4	200	200	80006281
001	2	9	6	0	3600	600	75	4	300	300	80009260
010	4	12	4	1	3200	800	100	2	400	200	8000C441
011	4	15	3	1	3000	1000	125	2	500	250	8000F431
100	4	18	3	0	3600	1200	150	2	600	300	80012430
101	4	24	2	1	3200	1600	200	2	800	400	80018421
110[2]	4	27	2	0	3600	1800	225	2	900	450	8001B420

Notes:

1. Reset default value.
2. Warning: Do not use. This selection is not supported in CPC945. PLL4 does not support 1.8 GHz mode.

Reset Value 0x30006281 04000000

Address 0xF8000880

Access Type Read/Write, Read Only

R	F	L	P	Reserved	M	PD	N	Reserved	V
0	1	2	3	4 5 6 7 8 9 10 11 12 13	14 15 16 17 18 19	20 21 22 23	24 25 26 27	28 29 30	31

FSM State	Reserved
32 33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

Bits	Field Name	Description	Access	Reset
0	R	This field specifies that the PLL will be immediately forced through a reset cycle where the values in this register will be loaded into the PLL. **The ForcePLLReset function performs no synchronization or protocol stopping, so setting this bit on an active bus is highly likely to lose data.** This bit is intended to be used only by the SPU for setting the initial state of PLL4. After the PLL is updated, this bit is reset to 0, indicating that the update has occurred.	R/W	0b0
1	F	This field specifies that the PLL will be immediately updated with the values in this register. No PLL reset will occur. **This can destabilize the PLL and should not be used without sufficient testing to determine that it is safe.** After the PLL is updated, this bit is reset to 0, indicating that the update has occurred. If new values are loaded in this register and neither ForcePLLReset nor ForcePLLLoad are set, they will be loaded into the PLL the next time it is stopped using the power management mechanism. For PLL1, that means that EnablePLL1Shutdown and APILogicStopEnable must be set in the ClockControl Register (0xF8000020). All CPUs must be quiesced as well. The contents of this register will also be loaded into PLL4 while it is stopped during System Sleep.	R/W	0b0
2	L	This field reflects the value of PLL4's PLLLock output at all times. It tells whether or not the PLL is locked to its programmed frequency. Whenever the PLL is in reset (states RESET and PLL SHUTDOWN), PLLLock is deasserted. When reset is deasserted, the PLL starts tracking in on lock. PLLLock asserts within 100us of reset deasserting. This signal is not used because as the PLL is locking, the lock output can toggle back and forth until the PLL is finally locked. **This register is to be used only as feedback for debug to verify that the PLL believes itself to be locked.**	R/W	0b0
3	P	This bit is set to a one whenever the PLL control latch is loaded with a new value. If software desires confirmation that the new values in this register have been loaded into the PLL, this bit must be cleared when loading a new value into this register.	R/W	0b0
4:13	Reserved	This field will read all 0's.	R	0x000
14:19	M	This field is used to set up the multiply factor for the M multiplier of the PLL; a factor of between 2 and 64 is provided.	R/W	0b001011
20:23	PD	This field is used to set up the divider ratio for the P divider; a factor of between 2 and 8 is provided.	R/W	0b000
24:27	N	This field controls the N divider which divides the PLL VCO frequency down to generate the 625 MHz clock used to drive the PCI Express HSS logic; a factor of between 2 and 10 is provided	R/W	0b0110

Bits	Field Name	Description	Access	Reset
28:30	Reserved	This field will read all 0’s.	R	0b000
31	V	This field is used to set up the VCO for the desired frequency range according to desired VCO frequecncy. 1 VCO frequency 3.0 – 3.5 GHz 0 VCO frequency 3.5 – 3.6 GHz	R/W	0b1
32:42	FSM State	This field is used to monitor the state of the PLL state machine which control the PLL. The bits are defined as follows: Bit [32] = wake1 Bit [33] = shutdown2 Bit [34] = shutdown1 Bit [35] = sleep Bit [36] = load Bit [37] = run Bit [38] = wake4 Bit [39] = wake3 Bit [40] = wake2 Bit [41] = wake0 Bit [42] = reset	R	0b00000000001
43:63	Reserved	This field will read all 0’s.	R	0x000000

12.5.10 PLL/Clock Visibility and Test

The PLLVis is test circuitry providing external visibility to examine PLL stability and operation. The circuitry consists of a multiplexer which allows software to select one signal to provide to a single external pin for lab debug. The signal to be provided is selected by the PLLVis register.

Reset Value 0x00000001 00000000

Address 0xF8000890

Access Type Read/Write, Read Only

Signal Selection

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Undefined

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

Bits	Field Name	Description	Access	Reset
0:31	Signal Selection	Enables the specified signal to be driven to the PMR observability pin. Enables are defined as: [0:14] = undefined [15] = real_api_plllock, real PLL1 lock signal [16] = real_ddr_plllock, real PLL2 lock signal [17] = real_pcie_plllock, real PLL3 lock signal [18] = real_ht_plllock, real PLL4 lock signal [19] = api_plloutA, PLL1 output A [20] = ddr_plloutA, PLL2 output B [21] = pcie_plloutse, PLL3 output [22] = ht_plloutB, PLL4 output B [23] = PI_clk [24] = ddr_clk [25] = mem_clk [26] = PCLK250 [27] = ht_clk [28] = pmr_clk [29] = NoIOReqs, NoIO DMA request from PI to PMR during nap. [30] = SDA_sampled, I2C data as sampled by CPC945 logic [31] = SCL_sampled, I2C clock as sampled by CPC945 logic These visibility bits should only have one bit set at a time. However, there is no logic to prevent that and if multiple bits were to be set, the bits would be logically ORed.	R/W	0x00000001
32:63	Undefined	This field will read all 0’s.	R	0x00000000

12.6 MPIC Registers

CPC945 contains a number of registers for software access to its internal MPIC interrupt controller. The MPIC register block lies at an address base of 0x50000. The reset and enable output bits for MPIC are located in the CPC945 Toggle Register (0xF80000E0). All MPIC registers are 32 bits wide and are accessed with 4-byte big-endian (that is, byte-reversed) operands. These registers are mapped to exist on 16-byte boundaries.

Table 12-9. Register Addresses .

Address	Register Name	Description	Page
0xF8041000		MPIC Feature Reporting Register	360
0xF8041020		MPIC Global Configuration Register	361
0xF8041080		MPIC Vendor ID Register	362
0xF8041090		MPIC Processor Initialization Register (not supported, reserved)	362
0xF80410A0 0xF80410B0 0xF80410C0 0xF80410D0		MPIC IPI (0,1,2,3) Vector/Priority Register	363
0xF80410E0		MPIC Spurious Vector Register	363
0xF8050000 0xF8050020, 0xF8050040 ... 0xF8050FE0		MPIC Interrupt Source 0-124 Vector/Priority Register	364
0xF8050030, 0xF8050050 ... 0xF8050FF0		MPIC Interrupt Source 0-124 Destination Registers	365
0xF8060040, 0xF8061040, 0xF8062040, 0xF8063040, 0xF8060050, 0xF8061050, 0xF8062050, 0xF8063050, 0xF8060060, 0xF8061060, 0xF8062060, 0xF8063060, 0xF8060070, 0xF8061070, 0xF8062070, 0xF8063070		MPIC CPU(0-3) IPI(0-3) Dispatch Command Registers	366
0xF8060080, 0xF8061080, 0xF8062080, 0xF8063080		MPIC CPU(0-3) Current Task Priority Registers	367
0xF80600A0, 0xF80610A0, 0xF80620A0, 0xF80630A0		MPIC CPU(0-3) Interrupt Acknowledge Registers	368
0xF80600B0, 0xF80610B0, 0xF80620B0, 0xF80630B0		MPIC CPU(0-3) End-of-Interrupt Registers	369

12.6.1 MPIC Feature Reporting Register

This register contains information about the number of interrupt sources and microprocessors supported by this implementation of MPIC.

Reset Value x007B0302
Address 0xF8041000
Access Type Read Only

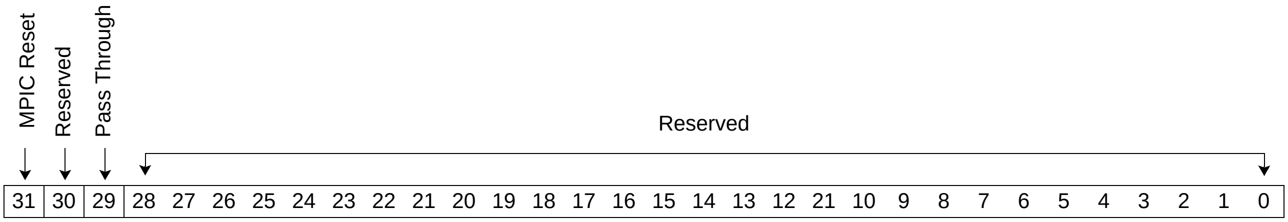
Reserved	NumIRQSources	Reserved	NumCPU	VersionID
31 30 29 28 27	26 25 24 23 22 21 20 19 18 17 16	15 14 13	12 21 10 9 8	7 6 5 4 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:27	Reserved	Always returns zero when read.	R	0x00
26:16	NumIRQSources	The number of the highest IRQ source supported.	R	0d123 (0x7B)
15:13	Reserved	Always returns zero when read.	R	0b00
12:8	NumCPU	The number of the highest physical CPU supported. CPC945's MPIC supports 4 microprocessors, numbered 0 to 3.	R	0x03
7:0	VersionID	This value will report the level of OpenPIC specification supported by an implementation. Values are: 1 OpenPIC version 1.0 compliant 2 OpenPIC version 1.2 compliant All others reserved.	R	0x02

12.6.2 MPIC Global Configuration Register 0

This register resets the MPIC logic and sets up the MPIC for cascading an external 8259 pair.

Reset Value	x0000 0000
Address	0xF8041020
Access Type	Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31	MpicReset	This bit resets the MPIC including clearing all pending interrupts. After the MPIC resets, the MPIC starts running again and this bit resets itself to 0b0. If the goal is to hold the MPIC in reset, use the MPICReset in the *Section 12.4.5 CPC945 Toggle Register* on page 337 instead. 0 MPIC Running 1 Reset MPIC	R/W	0b0
30	Reserved	Always returns zero when read.	R	0b0
29	Pass Through	OpenPIC 8259 pass through enable bit. 0 Pass through enabled 1 Pass through disabled **Note:** The CPC945 has no pass through function, so this but must always be set to 1 to allow the 124 I/O interrupt sources to be seen by the interrupt controller.	R/W	0b0
28:0	Reserved	Always returns zero when read.	R	0x00000000

12.6.3 MPIC Vendor ID Register

This register uniquely identifies the MPIC implementation.

Reset Value x1446 0000

Address 0xF8041080

Access Type Read Only

Reserved	Reserved	Device ID	Vendor ID
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 21 10 9 8	7 6 5 4 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:24	Reserved	Always returns zero when read.	R	0x00
23:16	Reserved	Always returns zero when read.	R	0x00
15:8	Device ID	Vendor-specified identifier for this device (MPIC-2A)	R	0x46
7:0	Vendor ID	Manufacturer ID. **Note:** This ID was not assigned by OpenPIC.	R	0x14

12.6.4 MPIC Processor Initialization Register

This register is not supported and left as reserved. The MPIC Register contains legacy logic that implements a register decode for the Processor Initialization Register specified by OpenPIC. The intended mainline function of this register, which is to activate an initialization of the attached processors, is not implemented on the CPC945 because this function is not supported by the PowerPC 970xx family of microprocessors.

Reset Value x0000 0000

Address 0xF8041090

Access Type Read Only

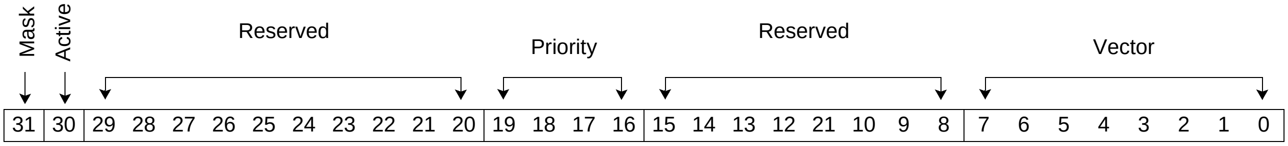
Reserved
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 21 10 9 8 7 6 5 4 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:0	Reserved	Always returns zero when read.	R	0x0000 0000

12.6.5 MPIC IPI (0,1,2,3) Vector/Priority Registers

This set of 4 registers contains the masking bit, status bit, priority level, and interrupt vector for IPI interrupts 0 through 3.

Reset Value	0x0000 0001
Address	0xF80410A0, 0xF80410B0, 0xF80410C0, 0xF80410D0
Access Type	Read Only, Read/Write

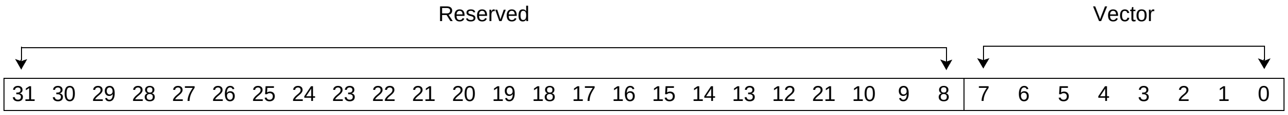


Bits	Field Name	Description	Access	Reset
31	Mask	A logic one on this bit disables future interrupts from this source.	R/W	0b1
30	Active	This bit is asserted (=1) when an interrupt is active for this source.	R	0b0
29:20	Reserved	Always returns zero when read.	R	0x000
19:16	Priority	Interrupt priority. Priority levels of 0x0 to 0xF are supported, with 0x0 as the lowest and 0xF as the highest.	R/W	0x0
15:8	Reserved	Always returns zero when read.	R	0x00
7:0	Vector	Interrupt vector address.	R/W	0x00

12.6.6 MPIC Spurious Vector Register

This register contains the interrupt vector for spurious interrupts.

Reset Value	0xFF00 0000
Address	0xF80410E0
Access Type	Read Only, Read/Write

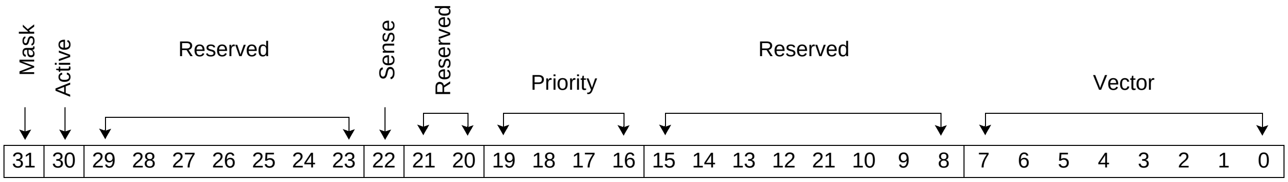


Bits	Field Name	Description	Access	Reset
31:8	Reserved	Always returns zero when read.	R	0x000000
7:0	Vector	Interrupt vector address.	R/W	0xFF

12.6.7 MPIC Interrupt Source 0-123 Vector/Priority Registers

This set of 124 registers contains the masking bit, status bit, sense, priority level, and interrupt vector for Interrupt Sources 0 through 123.

Reset Value	0x0000 0001
Address	0xF8050000, 0xF8050020, 0xF8050040, 0xF8050060, ..., 0xF8050F60
Access Type	Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31	Mask	A logic one on this bit disables future interrupts from this source.	R/W	0b1
30	Active	This bit is asserted (equal to 1) when an interrupt is active for this source.	R	0b0
29:23	Reserved	Always returns zero when read.	R	0x00
22	Sense	Interrupt line sense. 0 Positive Edge-Triggered 1 Negative Level-Sensitive	R/W	0b0
21:20	Reserved	Always returns zero when read.	R	0b0
19:16	Priority	Interrupt priority. Priority levels of 0x0 to 0xF are supported, with 0x0 as the lowest and 0xF as the highest.	R/W	0x0
15:8	Reserved	Always returns zero when read.	R	0x00
7:0	Vector	Interrupt vector address.	R/W	0x00

Note: MPIC contains legacy logic which implements Bit 23 equal to the polarity, for register 0xF805000 only. Unlike the other Interrupt Source Vector/Priority Registers (0xF8050020, ...), (for register 0xF805000) bit 23 can be set and read back. If set to one, this changes the default operation from sense from active low/negative edge to active high/positive edge. However, for proper operation in the CPC945, this bit must always be set equal to 0.

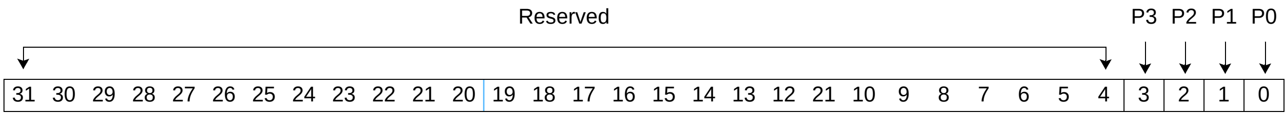
12.6.8 MPIC Interrupt Source 0-123 Destination Registers

This set of 124 registers contains the processor destination bits for all interrupt sources (0 to 123). If a single microprocessor is selected (directed delivery mode), then interrupts from this source are directed to that microprocessor. If multiple destination microprocessors are selected (distributed delivery mode), then interrupts from this source are distributed among the selected destination processors using an implementation-specific algorithm

Reset Value 0x0000 0001

Address 0xF8050030, 0xF8050050, 0xF8050070, ..., 0xF8050F70

Access Type Read Only, Read/Write

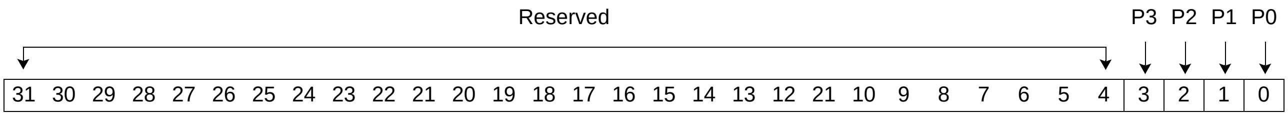


Bits	Field Name	Description	Access	Reset
31:4	Reserved	Always returns zero when read.	R	0x0000000
3	P3	CPU3 is the destination of the interrupt when this bit is set.	R/W	0b0
2	P2	CPU2 is the destination of the interrupt when this bit is set.	R/W	0b0
1	P1	CPU1 is the destination of the interrupt when this bit is set.	R/W	0b0
0	P0	CPU0 is the destination of the interrupt when this bit is set.	R/W	0b0

12.6.9 MPIC CPU(0-3) IPI(0-3) Dispatch Command Registers

This set of 16 registers is defined as write-only. Reads have no side effects and return 0x00000000. There are four IPI dispatch command registers for each microprocessor. Writing to an IPI dispatch command register causes an interprocessor interrupt request to be sent to one or more processors. A microprocessor is interrupted if the bit in the IPI dispatch command register corresponding to that microprocessor is set during the write. The IPI dispatch command registers are shared by all processors, but are located in the per-processor register space.

Reset Value	0x0000 0001
Address	0xF8060040, 0xF8061040, 0xF8062040, 0xF8063040, 0xF8060050, 0xF8061050, 0xF8062050, 0xF8063050, 0xF8060060, 0xF8061060, 0xF8062060, 0xF8063060, 0xF8060070, 0xF8061070, 0xF8062070, 0xF8063070
Access Type	Read Only, Write Only



Bits	Field Name	Description	Access	Reset
31:4	Reserved	Always returns zero when read.	R	0x0000000
3	P3	CPU3 is the destination of the interrupt when this bit is set.	W	0b0
2	P2	CPU2 is the destination of the interrupt when this bit is set.	W	0b0
1	P1	CPU1 is the destination of the interrupt when this bit is set.	W	0b0
0	P0	CPU0 is the destination of the interrupt when this bit is set.	W	0b0

12.6.10 MPIC CPU(0-3) Current Task Priority Registers

This set of 4 registers contains the minimum interrupt priority level needed to generate a microprocessor interrupt. There is one task priority register per microprocessor. Setting the task priority to 0xF masks all interrupts to a given microprocessor, since no interrupt source can have a priority higher than 0xF. It is recommended, however, that PowerPC microprocessors use the MSR[EE] bit rather than the task priority registers to disable internal interrupts.

Reset Value	0x0000 0000
Address	0xF8060080, 0xF8061080,0xF8062080,0xF8063080
Access Type	Read Only

Reserved																												Priority			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	21	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:4	Reserved	Always returns zero when read.	R	0x000000
3:0	Priority	Interrupt source priorities need to be higher than this value in order to generate a microprocessor interrupt. Minimum priority is 0x0, and maximum priority is 0xF.	R	0x00

12.6.11 MPIC CPU(0-3) Interrupt Acknowledge Registers

This set of 4 read-only registers contains the interrupt vector for the next pending microprocessor interrupt. There is one interrupt acknowledge register per microprocessor. Reading the Interrupt Acknowledge register returns the interrupt vector corresponding to the highest-priority pending interrupt in that microprocessor's Interrupt Request Register (IRR). It also transfers the highest-priority pending interrupt from that microprocessor's IRR to that microprocessor's In-Service Register.

Clears the bit in the Interrupt Pending Register (IPR) corresponding to the highest-priority pending interrupt in that microprocessor's IRR. This is only effective for edge-triggered interrupts. Level-triggered interrupts normally cause the bit in the IPR to be set every cycle until the device driver's interrupt service routine has cleared the interrupt source.

The IRR, IPR, and In-Service Registers are internal to MPIC and are not visible to software.

Reset Value x0000 0000
Address 0xF80600A0, 0xF80610A0, 0xF80620A0, 0xF80630A0
Access Type Read Only

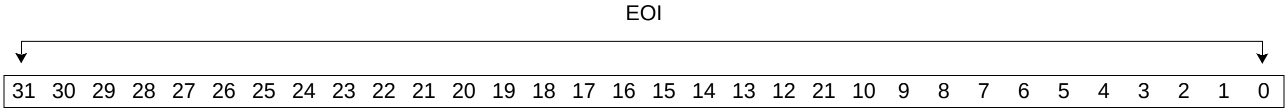
Reserved																								Vector							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	21	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:8	Reserved	Always returns zero when read.	R	0x000000
7:0	Vector	Interrupt vector address.	R	0x00

12.6.12 MPIC CPU(0-3) End-of-Interrupt Registers

Writing to this set of 4 write-only registers signals the end of processing for the highest-priority interrupt currently in service by the associated microprocessor. When this register is written with a value of 0x00000000, the highest-priority interrupt in the In-Service Priority Register is reset along with the corresponding bit in the interrupt source In-Service Register. **Writes of values other than 0x00000000 will not generate an EOI event.** Reads of this register will have no side effects, and will return 0x00000000.

Reset Value	0x0000 0000
Address	0xF80600B0, 0xF80610B0, 0xF80620B0, 0xF80630B0
Access Type	Write Only



Bits	Field Name	Description	Access	Reset
31:0	EOI	Writing 0x00000000 to this register will generate an EOI event for the associated microprocessor (see above). Writing a value other than 0x00000000 will have no effect. Reads will have no side effects, and will return 0x00000000.	W	0x0000 0000

12.7 PI Physical Interface Registers

12.7.1 APIPhy Command and Status Register Bus

The APIPhy Command and Status Register Bus is an internal configuration and debug bus that is inside the Processor Interconnect (PI) interface logic block. It is used to configure the APIPhy logic at boot time or during wake up, and can be used as a debug mechanism for checking the APIPhy status. The APIPhy receiver status register contains data from several internal sources. As detailed below, this consists of an I/O group select (iogrp_sel[0:3], command (command_data[0:14]), a command_load bit, a status data select (data_stat_sel [0:2]), and a status data register (data_stat[0:15]).

The iogrp_sel, command_data, and data_stat_sel Registers reside as bit fields in the register, *Section 12.7.4.4 APIPhy Receiver Mode and Command Register (APIPhyRcvModeCmd)* on page 385. The data_stat register resides as bit fields in the register, *Section 12.7.4.3 APIPhy Status 0 Register (APIPhySTAT0)* on page 382.

The I/O group select register, iogrp_sel, is a 4 bit read-or-write register that is used to select which I/O group's (15 possible I/O groups; 0 to 14) status registers are to be accessed and is also used to obtain the global status of the APIPhy receiver. In CPC945, the only active I/O group is group 0; All other I/O groups are reserved (1 to 14). The encodings for this register are as follows.

Table 12-10. I/O Group Select Register Settings, iogrp_sel[0:3]

iogrp_sel	Description
0000	Select I/O group 0
0001 - 1110	Reserved
1111	Select global status

Table 12-11. Status data select register, data_stat_sel[0:2] - io_group_sel = 0x0

data_stat_sel	Returned Data (data_stat[0:15])
000*	(0:7) I/O Clock Delay (8:15) IAP Flag 0
001*	(0:7) IAP Flag 1 (8:15) IAP Flag 2
010**	(0) Underflow of per-bit deskew from eical for selected group (1) Overflow of per-bit deskew from eical for selected group (2) Guardband delay underflow (gb = 0) for selected group (3) Guardband delay overflow (gb = maximum) for selected group (4) Guardband Threshold value exceeded (5:9) Guardband minimum for selected group (10:15) Reserved
011	Reserved
100	Reserved

Table 12-11. Status data select register, data_stat_sel[0:2] - io_group_sel = 0x0 (Continued)

data_stat_sel	Returned Data (data_stat[0:15])
101	Reserved
110	Reserved
111*	(0) IAP Status (0): Clock period too large (critical) (1) IAP Status (1): No IAP pattern detected (fatal) (2) IAP Status (2): Could not find trailing edge of latest bit in IAP pattern (critical) (3) IAP Status (3): At least one data bit was deskewed to maximum (warning) (4) IAP Status (4): No valid flag 0, that is, No valid IAP “all 1's” pattern found in deskewed bus (fatal) (5) IAP Status (5): No valid flag 1, that is, No start of trailing edge of IAP pattern found in deskewed bus (critical) (6) IAP Status (6): No valid flag 2, that is, No end of trailing edge of IAP pattern found in deskewed bus (critical) (7) IAP Status (7): Final values fail, that is, No valid IAP “all 1's” pattern found in fully aligned bus (fatal) (8) IAP Status (8): Final delay value negative-nonoptimal sampling point used (warning) (9:15) Reserved

Table 12-12. Status data select register, data_stat_sel[0:2] - io_group_sel = 0xF

data_stat_sel	Returned Data (data_stat[0:15])
000**	(0:7) I/O Clock Period (8:15) Reserved
001**	(0) Underflow of per-bit deskew from eical (1) Overflow of per-bit deskew from eical (2) Guardband delay underflow (gb = 0) (3) Guardband delay overflow (gb = max) (4) Guardband threshold value exceeded (5:9) Reserved (10:12) Learned Target Cycle (13:15) Learned Target Cycle Spread
010	Reserved
011**	(0) IAP Status nonzero fro Clock Group 0 (1:15) Reserved
100	Reserved

Table 12-12. Status data select register, data_stat_sel[0:2] - io_group_sel = 0xF (Continued)

data_stat_sel	Returned Data (data_stat[0:15])
101	Reserved
110	Reserved
111**	IAP status values below (bits 0 to 8) are a global logical OR of all I/O groups’ respective IAP status bits: (0) IAP Status (0): Clock period too large (critical) (1) IAP Status (1): No IAP pattern detected (fatal) (2) IAP Status (2): Could not find trailing edge of latest bit in IAP pattern (critical) (3) IAP Status (3): At least one data bit was deskewed to maximum (warning) (4) IAP Status (4): No valid flag 0, that is, No valid IAP “all 1's” pattern found in deskewed bus (fatal) (5) IAP Status (5): No valid flag 1, that is, No start of trailing edge of IAP pattern found in deskewed bus (critical) (6) IAP Status (6): No valid flag 2, that is, No end of trailing edge of IAP pattern found in deskewed bus (critical) (7) IAP Status (7): Final values fail, that is, No valid IAP “all 1's” pattern found in fully aligned bus (fatal) (8) IAP Status (8): Final delay value negative-nonoptimal sampling point used (warning) (9) IAP Done (on only when RIAP is asserted and IAP completed; disappears after RIAP is deasserted) (10) Learned Target cycle passed (11) Learned Target cycle failed (12) EST passed or RDT passing (13) EST or RDT failed (14:15) Reserved

The status data select register, data_stat_sel, is a 3 bit read-or-write register that is used to select which status data is to be read. This register is set after the I/O group select register has been set to either I/O Group 0 (0x0) or to global status (0xF). The data status register, data_stat, is a 16 bit read-only register that displays the returned status data that was selected by data_stat_sel. The encodings for the data_stat_sel register when the I/O group select register has been set to I/O Group 0 (0x0) and the returned data from the data_stat register are defined as follows:

* I/O group must be selected before IAP for returned status data to be valid

** I/O group can be selected at anytime before or after IAP for returned status data to be valid.

The encodings for the data_stat_sel register when the I/O group select register has been set to global status (0xF) and the returned data from the data_stat register are defined as follows:

* I/O group must be selected before IAP for returned status data to be valid

** I/O group can be selected at anytime before or after IAP for returned status data to be valid.

The command and status register bus also allows for changing the PI clock delay. To do this, the command bus register (command_data(0:14), command_load) must be used. The bus procedure for changing the PI clock delay is similar to the procedure for reading the status registers. First a target I/O group must be selected (only 1 target group exists: 0x0) via the I/O group select register (iogrp_sel(0:3)). Second the command_load bit is set and the desired clock delay value is written into the command bus register in the following format:

Table 12-13. Changing the PI clock delay (command_data[0:14],ncommand_load)

Field	Description
command_data[0:7]	PI clock delay value
command_data[8:14]	0x00
command_load	1

Finally, the command_load bit is cleared with a write of 0 to trigger the IAP Register state machine to update the new clock value.

Because shadow registers are used to capture clock-group-specific data and are only loaded during IAP, it is not possible to read back the value written to the PI clock delay registers. For example, after IAP the PI clock delay for clock group 0 is found to be 0x08 (set iogrp_sel to 0x0, data_stat_sel to 0x0 and read data_stat = 0x08). It is possible to change the PI clock delay by writing 0x10 to the command bus register, however, the value read back from data_stat after the change would be still be 0x08 and not 0x10. The logic would be updated, so the PI bus clock delay would be set to 0x10.

12.7.2 PI Physical Interface Registers

This section describes functional details for each register. The base address can be found in the *Section 12.3 Control Register Memory Map* on page 331.

Table 12-14. PI Physical Interface Register Address Space

System Bus Address	Register Name	Description	Page
offset 0x000	APIPhyDRVIAPPATMASK	APIPhy Driver IAP Pattern Mask	375
offset 0x010	APIPhyRCVIAPPATMASK	APIPhy Receiver IAP Pattern Mask	376
offset 0x020	APIPhyCONFIGREG0	APIPhy Configuration 0	377
offset 0x030	APIPhyCONFIGREG1	APIPhy Configuration 1	379
offset 0x040	APIPhySTR	APIPhy Shorts Test Configuration	381
offset 0x050	APIPhySTAT0	APIPhy Status 0	382
offset 0x060	APIPhyCLKSTAT	APIPhy Clock Status	385
offset 0x070	APIPhyRIAPSTATE	APIPhy Receiver IAP State	387
offset 0x080	DATAERROR0	APIPhy Data Error 0	388
offset 0x090	DATAERROR1	APIPhy Data Error 1	389
offset 0x0A0	DATAERROR2	APIPhy Data Error 2	390
offset 0x0B0	APIPhyDATAERROR3	APIPhy Data Error 3	391
offset 0x0C0	DATAERROR4	APIPhy Data Error 4	392
offset 0x0D0	DATAERROR5	APIPhy Data Error 5	393
offset 0x0E0	APIPhyIOCTRL	APIPhy I/O Control Register	394
offset 0x0F0	APIPhyPMRIOCTRL	APIPhy PMR I/O Control Register	396

12.7.3 APIPhy Configuration Registers

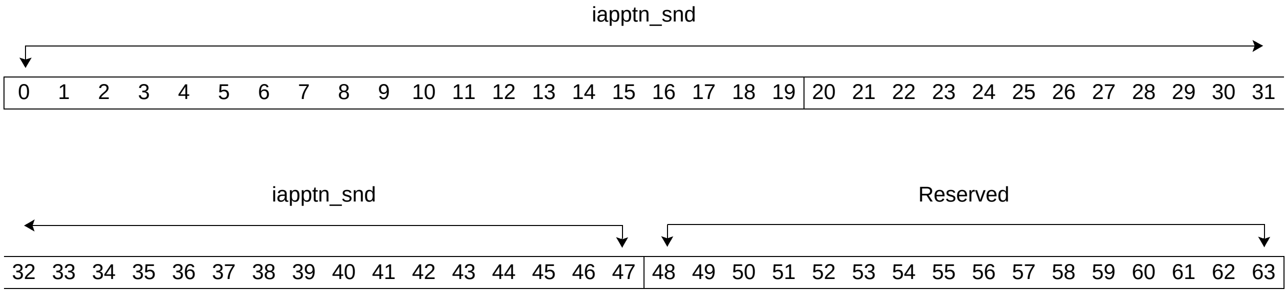
12.7.3.1 APIPhy Driver IAP Pattern Mask (APIPhyDRVIAPPATMASK)

This 44-bit mask value determines which driving lines of the AD bus will be inverted or not such that there are an equal number of ones and zeroes that are transmitted. This mask register controls the converting of the logical IAP signature to the physical IAP signature. The APIPhyDRVIAPPATMASK register of the PI Physical Interface in the transmitting chip must have the same value as the APIPhyRCVIAPPATMASK register of the PI Physical Interface in the receiving chip.

Reset Value N/A

Offset 0x000

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
0:47	iapptn_snd [0:47]	Driver IAP Pattern Mask Bits 0:43 for the AD bus and bits 44:47] for the snoop response bus {sr0, sr0_n, sr1, sr1_n}.	R/W	0xF21C 3BC8 70E5
48:63	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined

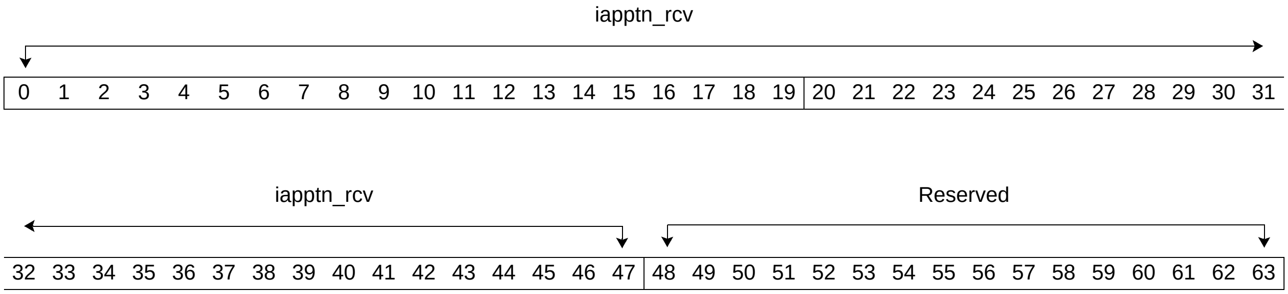
12.7.3.2 APIPhy Receiver IAP Pattern Mask (APIPhyRCVIAPPATMASK)

This 44-bit mask value determines which receiving lines of the AD bus will be inverted or not such that there are an equal number of ones and zeroes that are transmitted. This mask register controls the converting of the logical IAP signature to the physical IAP signature. The APIPhyDRVIAPPATMASK register of the PI Physical Interface in the transmitting chip must have the same value as the APIPhyRCVIAPPATMASK register of the PI Physical Interface in the receiving chip.

Reset Value N /A

Offset 0x010

Access Type Read/Write



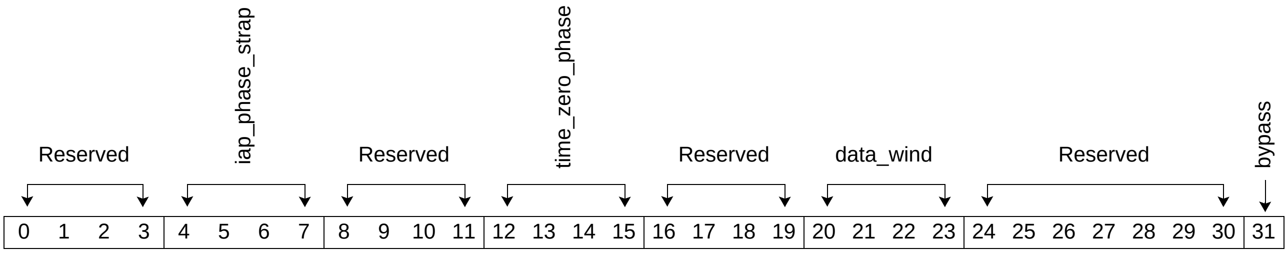
Bits	Field Name	Description	Access	Reset
0:47	iapptn_rcv [0:47]	Receiver IAP Pattern Mask Bits 0:43 for the AD bus and bits 44:47 for the snoop response bus {sr0, sr0_n, sr1, sr1_n}.	R/W	0xF21C 3BC8 70E5
48:63	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined

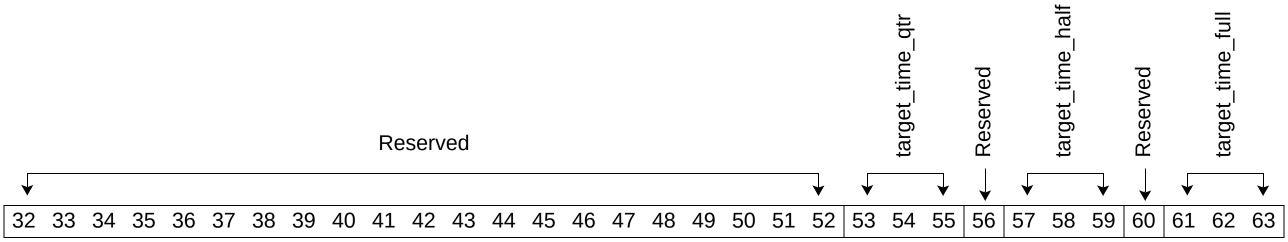
12.7.4 APIPhy Configuration 0 Register (APIPhyCONFIGREG0)

Reset Value N/A

Offset 0x020

Access Type Read/Write



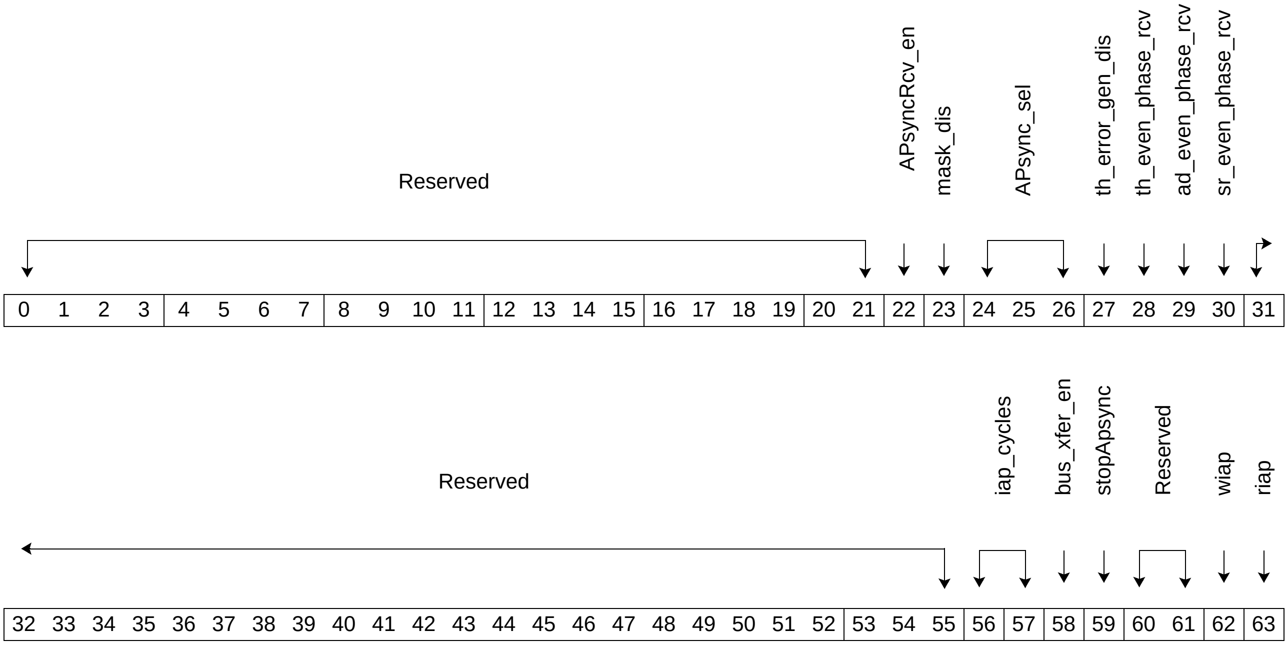


Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
4:7	iap_phase_strap	Phase adjustment of launching of IAP pattern 10001000.... by APIPhy driver. By default (4x0), “1” in the IAP pattern is launched at time zero.	R/W	0x0
8:11	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
12:15	time_zero_phase	Phase adjustment for time zero reset. By default (4x0), time zero reset is pulsed at system time zero.	R/W	0x0
16:19	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
20:23	data_wind	Data windage: Phase offset of the final data bus bits expressed in delay elements. Binary encoded from 0 - 15. The programming of the wind value allows the flexibility of modifying the data bus delay for “centering” as a function of bus speed, system configuration, process, etc.	R/W	0b000
24:30	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
31	bypass	Enable APIPhy bypass mode: Bypass the APIPhy physical layer (the data and clock deskew logic).	R/W	0b0 (disabled)

Bits	Field Name	Description	Access	Reset
32:52	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
53:55	target_time_qtr	Target beat number for quarter frequency: Also called the target cycle. The target cycle is programmed as the cycle the data is to be received. The target cycle is referenced to time = 0 and is mod 4 (The most significant bit, bit 53, is reserved; writes to this bit has no effect on hardware). The programming of the target value allows the flexibility of modifying the target cycle as a function of bus speed, system configuration, process, etc.	R/W	Undefined
56	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
57:59	target_time_half	Target beat number for half frequency: Also called the target cycle. The target cycle is programmed as the cycle the data is to be received. The target cycle is referenced to time = 0 and is mod 4 (The most significant bit, bit 57, is reserved; writes to this bit has no effect on hardware). The programming of the target value allows the flexibility of modifying the target cycle as a function of bus speed, system configuration, process, etc.	R/W	Undefined
60	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
61:63	target_time_full	Target beat number for full frequency: Also called the target cycle. The target cycle is programmed as the cycle the data is to be received. The target cycle is referenced to time = 0 and is mod 4 (The most significant bit, bit 61, is reserved; writes to this bit has no effect on hardware). The programming of the target value allows the flexibility of modifying the target cycle as a function of bus speed, system configuration, process, etc.	R/W	0b011 (target beat three)

12.7.4.1 APIPhy Configuration 1 Register (APIPhyCONFIGREG1)

Reset Value	N/A
Address	0x030
Access Type	Read/Write

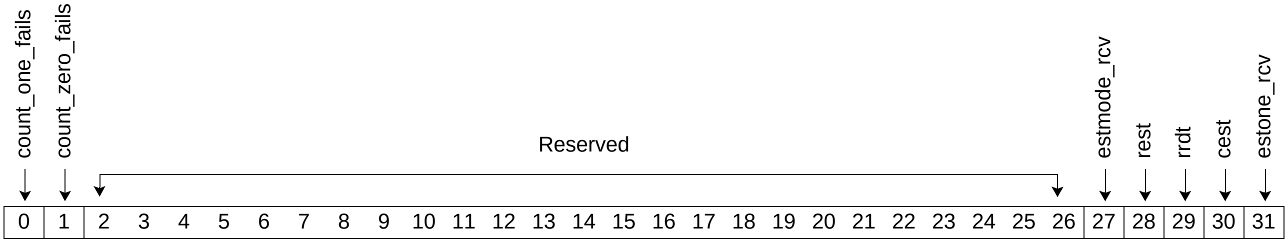


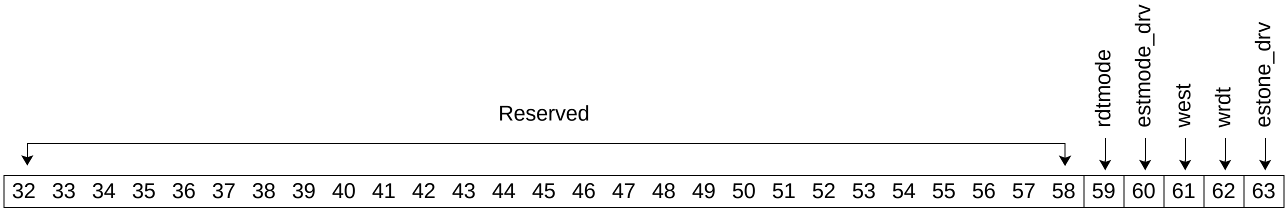
Bits	Field Name	Description	Access	Reset
0:21	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
22	APsyncRcv_en	External APsync enable. If system designed with external clock chip generating APsync, this bit needs to be set to 1 to put CPC945 in mode to use this externally generated APsync.	R/W	0x0
23	mask_dis	IAP pattern mask register disable. If set to 1, IAP pattern masks for both receiver and driver are disabled.	R/W	Undefined

Bits	Field Name	Description	Access	Reset
24:26	APsync_sel	FOR DEBUG PURPOSES ONLY. APsync pulse timing adjustment. By default APsync is launched from CPC945 at the rising edge of APclk (PI reference clock) and pulsed every 24 APclk clocks. APsync_sel is used to program the APIPhy to launch APsync earlier in half APclk increments according to the following table: APsync_sel 000 Reserved; aliases to 001 001 Default APsync 010 Launch APsync 1/2 APclk earlier 011 Launch APsync 1 APclk earlier 100 Launch APsync 1 1/2 APclks earlier 101 Launch APsync 2 APclks earlier 110 Reserved; aliases to 001 111 Reserved; aliases to 001	R/W	Undefined
27	th_error_gen_dis	Disable generation of transfer handshake parity error within APIPhy	R/W	0b0
28	th_even_phase_rcv	Receiver TH Beat Phase adjustment: If a 0, data TH beat is being received 180 degrees out of phase of the bus clock, else data is being received in phase with bus clock.	R/W	0b0
29	ad_even_phase_rcv	Receiver AD Beat Phase adjustment: If a 0, data AD beat is being received 180 degrees out of phase of the bus clock, else data is being received in phase with bus clock.	R/W	0b1
30	sr_even_phase_rcv	Receiver SR Beat Phase adjustment: If a 0, data SR bear is being received 180 degrees out of phase of the bus clock, else data is being received in phase with bus clock.	R/W	0b0
31:55	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	0b1
56:57	iap_cycles	IAP Pattern generation modes. iap_cycles 00 Generate 4-cycle pattern, (10001000...) 01 Generate 8-cycle pattern, (1000000010000000...) 10 Generate 12-cycle pattern, (100000000000100000000000...) 11 Generate 16-cycle pattern, (10000000000000001000000000000000...)	R/W	0b0
58	bus_xfer_en	Enable snoop, command, and data transactions from APIPhy to PI and from PI to APIPhy.	R/W	0b1
59	stopApsync	Stop APsync (disable APsync generation): If set to a value of 1, stop generating APsync pulse.	R/W	Undefined
60:61	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	0b00
62	wiap	WIAP Start Driver IAP sequence: If set to a value of 1, WIAP is enabled. The IAP signature of “1000” is continuously transmitted by the APIPhy. Should not be set to a 0 until the RIAP done signal on the receiving side is confirmed.	R/W	0b0
63	riap	RIAP Start Receiver IAP sequence: If set to a value of 1, RIAP is enabled. Should not be set to a 0 until RIAP done is confirmed. Note that the WIAP signal on the transmitting side must be active when RIAP is asserted.	R/W	0b0 (disabled)

12.7.4.2 APIPhy Shorts Test Configuration Register (APIPhySTR)

Reset Value	N/A
Offset	x‘040’
Access Type	Read/Write





Bits	Field Name	Description	Access	Reset
0	count_one_fails	Receiver control. enable error detection during RDT; an error is an expected zero detected as a one.	R/W	0b0
1	count_zero_fails	Receiver control. enable error detection during RDT; an error is an expected one detected as a zero.	R/W	0b0
2:26	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
27	estmode_rcv	Receiver control. Processor interface Shorts Test Mode. If 1, sets receiver in shorts test mode and enables use of rest command.	R/W	0b0
28	rest	Receiver control. Setting to 1 sends command to begin receiver processor interface shorts test when estmode_rcv is asserted.	R/W	0b0
29	rrdt	Receiver control. If 1, sends command to begin receiver random data test.	R/W	0b0
30	cest	Receiver control. If 1, sends command to check for errors (shorts from another bus’ (PPC970xx processor interface) EST test) on this bus (CPC945 processor interface).	R/W	0b0
31	estone_rcv	Receiver control. Determines type of REST or CEST. If 0, EST is walking ones test. If 1, EST is walking zeros test. This must be programmed to the same value as the estone_drv.	R/W	0b0
32:58	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined

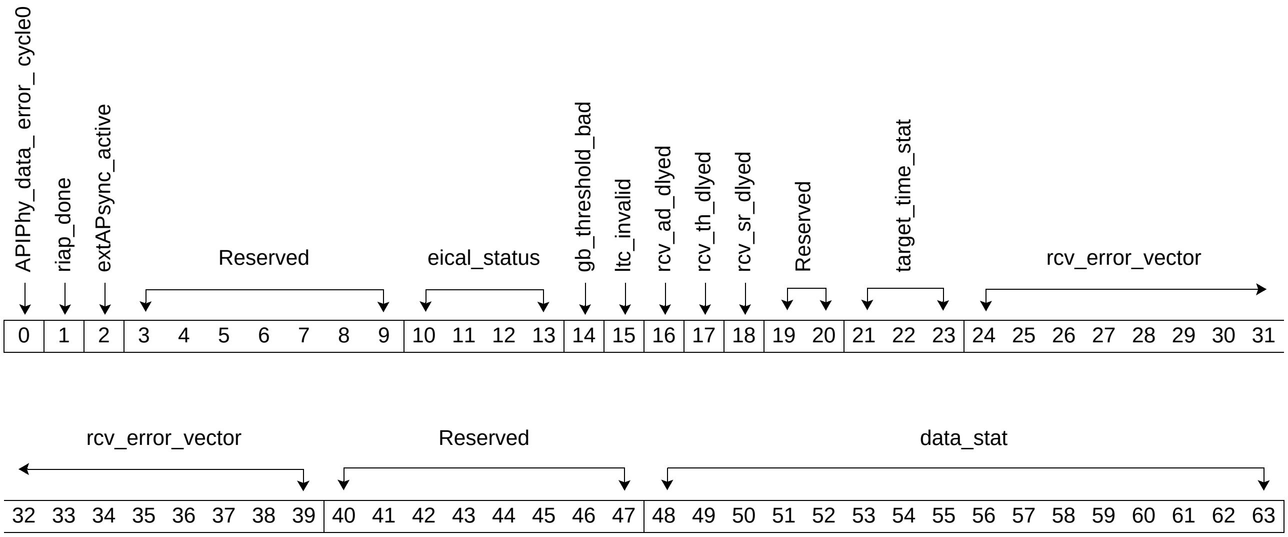
Bits	Field Name	Description	Access	Reset
59	rdtmode	Driver control. Used in conjunction with estmode. [estmode, rdtmode] 00 Nontest; functional mode 01 rdt mode 10 est mode 11 Invalid	R/W	0b0
60	estmode_drv	Driver control. Used in conjunction with rdtmode. [estmode, rdtmode] 00 Nontest; functional mode 01 rdt mode 10 est mode 11 Invalid	R/W	0b0
61	west	Driver control. If 1, send command to begin Write Processor Interface Shorts Test. Asserted after estmode/estone asserted.	R/W	0b0
62	wrdt	Driver control. If 1, send command to begin write random data test when rdtmode is asserted.	R/W	0b0
63	estone_drv	Driver control. Determines type of REST or CEST. If 0, EST is walking ones test. If 1, EST is walking zeros test. This must be programmed to the same value as the estone_rcv.	R/W	0b0

12.7.4.3 APIPhy Status 0 Register (APIPhySTAT0)

This register holds copies of various 1 to 4 bit status registers to allow software to access this data from diverse locations, via a single register read operation. The mappings follow:

Bit Field	Description
24:27	copy of eical_status[0:3] found in 0x205/0x305 bits 10:13 where bit 24 = Overflow of per-bit deskew from eical where bit 25 = Underflow of per-bit deskew from eical where bit 26 = Guardband delay underflow where bit 27 = Guardband delay overflow
28	copy of gb_threshold_bad found in 0x205/0x305 bit 14
29	copy of ltc_invalid found in 0x205/0x305 bit 15
30	copy if riap_done found in 0x205/0x305 bit 1
31	copy of data_stat[12] found in 0x205/0x305 bit 60
32	copy of data_stat[13] found in 0x205/0x305 bit 61
33:39	RESERVED (should read all zeros)

Reset Value	x‘0000 0000 0000 0000’
Offset	x‘050’
Access Type	Read Only

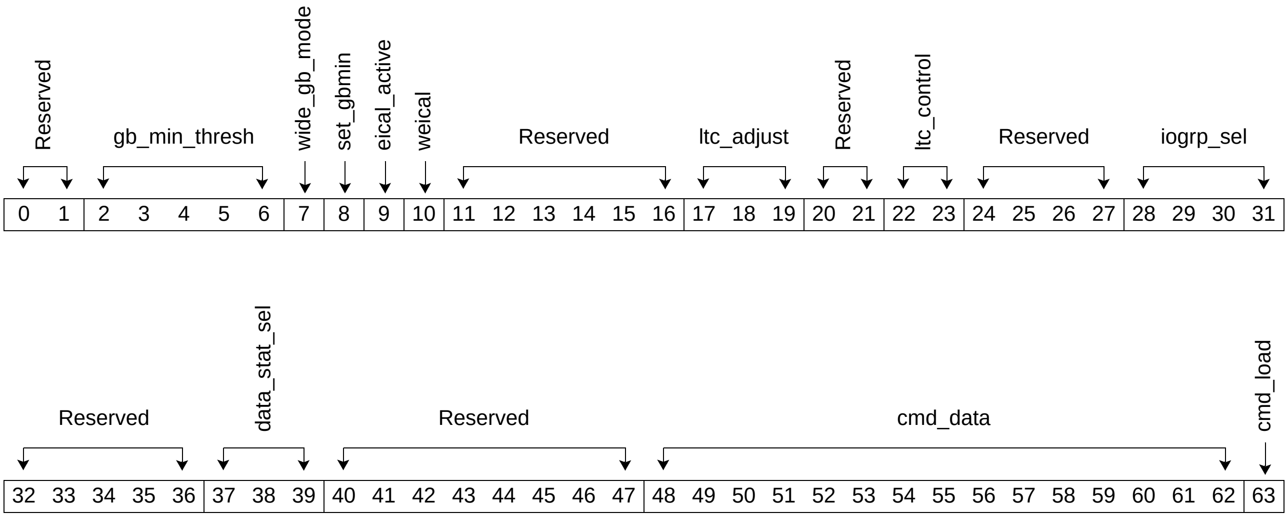


Bits	Field Name	Description	Access	Reset
0	APIPhy_data_ error_ cycle0	Parity or BCM data error status: if 1, a data error was detected else no error detected.	R	0b0
1	riap_done	RIAP_DONE: If value is a 1, then RIAP for APIPhy complete. If value is a 0, then RIAP is not complete 0 Receiver IAP sequence not complete 1 Receiver IAP sequence finished	R	0b0
2	extAPsync_enabled	External APsync enabled: if 1, APIPhy and the processors are using the external APsyncs generated off chip by the clock chip to determine time zero. If 0, APIPhy is generating its own APsync internally and generating the APsyncs for the processors to determine time zero. Mode of APsync is set in PMR Configuration Registers.	R	0b0
3:9	Reserved	Reserved	R	0x00
10:13	eical_status	PI calibration status.	R	0x0
14	gb_threshold_bad	If 1, guardband below threshold set by gb_min_thresh.	R	0b0
15	ltc_invalid	If 1, learned target cycle could not be determined. Need to manually set appropriate target cycle.	R	0b0
16	rcv_ad_dlyed	When read as a 0b1, the PI PHY receiver is delaying the AD (address/data) bus one PI PHY cycle to align it to the api_clk for the PI functional unit. This field adds observability into the PI PHY to see when the PI PHY even phase logic adds a delay to align odd aligned data and commands from the processor to the even edge of the PI clock.	R	0bX
17	rcv_th_dlyed	When read as a 0b1, the PI PHY receiver is delaying the TH (Transfer Handshake) bus one PI PHY cycle to align it to the api_clk for the PI functional unit. This field adds observability into the PI PHY to see when the PI PHY even phase logic adds a delay to align odd aligned data and commands from the processor to the even edge of the PI clock.		0bX

Bits	Field Name	Description	Access	Reset
18	rcv_sr_dlyed	When read as a 0b1, the PI PHY receiver is delaying the SR (Snoop Response) bus one PI PHY cycle to align it to the api_clk for the PI functional unit. This field adds observability into the PI PHY to see when the PI PHY even phase logic adds a delay to align odd aligned data and commands from the processor to the even edge of the PI clock.		0bX
19-20	Reserved	Reserved		0b00
21:23	target_time_stat	The currently loaded target beat (target cycle). The value here is the target cycle that has been chosen for the current power tuning mode (full, half, or quarter).	R	0b011
24:39	rcv_error_vector	Status of eical Error vector of various error/attention/interrupt signals. rcv_error_vector holds copies of various 1 to 4 bit status registers that reside in several places and puts it in one register. This way software did not have to do several register reads to collect this data. **Bit Field Description** 24:27 copy of eical_status[0:3] (found in 0x205/0x305, bits 10:13) bit 24 = Overflow of per-bit deskew from eical bit 25 = Underflow of per-bit deskew from eical bit 26 = Guardband delay underflow bit 27 = Guardband delay overflow 28 copy of gb_threshold_bad found in 0x205/0x305 bit 14 29 copy of ltc_invalid found in 0x205/0x305 bit 15 30 copy if riap_done found in 0x205/0x305 bit 1 31 copy of data_stat[12] found in 0x205/0x305 bit 60 32 copy of data_stat[13] found in 0x205/0x305 bit 61 33:39 RESERVED (should read all zeros)	R	0x0000
40:47	Reserved	Reserved	R	0x00
48:63	data_stat	The data status register, data_stat, is a 16 bit read-only register that displays the returned status data that was selected by data_stat_sel in the *Section 12.7.4.4 APIPhy Receiver Mode and Command Register (APIPhyRcvModeCmd)* on page 385.	R	0x0000

12.7.4.4 APIPhy Receiver Mode and Command Register (APIPhyRcvModeCmd)

Reset Value	N/A
Offset	x‘060’
Access Type	Read Only



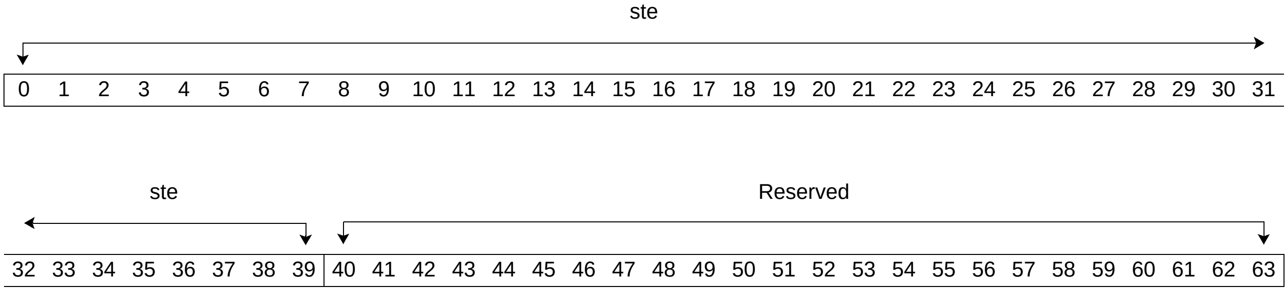
Bits	Field Name	Description	Access	Reset
0:1	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	0b0
2:6	gb_min_tresh[0:4]	Guardband minimum threshold value	R/W	0b0
7	wide_gb_mode	Guardband mode control. If 1, in wide guardband mode.	R/W	0b0
8	set_gbmin	If 1, restart the guardband minimum detector	R/W	0b0
9	eical_active	If 1, start processor interface calibration	R/W	Undefined
10	weical	If 1, use eical pattern instead of functional data during PI calibration	R/W	0b000
11:16	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
17:19	ltc_adjust[0:2]	Used when in learned target cycle mode. Target cycle windage	R/W	0b11
20:21	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
22:23	ltc_control[0:1]	learned target cycle control: ltc_control 00 search for minimum working target cycle 01 search for minimum working target cycle + 1 10 search for minimum working target cycle + 2 11 disable learned target cycle and set manually	R/W	0x0
24:27	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined

Bits	Field Name	Description	Access	Reset
28:31	iogroup_sel[0:3]	I/O group select. Used for selecting I/O group from which to read status or write configuration commands.	R/W	0b111
32:36	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.		Undefined
37:39	data_status_sel[0:2]	Status select. Used to select what status data to collect for the currently selected I/O group.		0x0000
40:47	Reserved	Reserved; Writes have no effect on hardware and read return undefined read data.		
48:62	command_data[0:14]	Bus used to load values into clock delay registers.		
63	command_load	Load command data into APIPhy register bus. Set to a 1 to load command data (located in bits 48:62 of this register: APIPhyRcvModeCmd Register). Follow with a clear of this bit (set to 0) to trigger the write of the command data.		

12.7.4.5 APIPhy Receiver IAP State Register (APIPhyRIAPSTATE)

This register allows software to sense the state of the IAP state machine. This feature can be used to determine if the state machine is running or stopped.

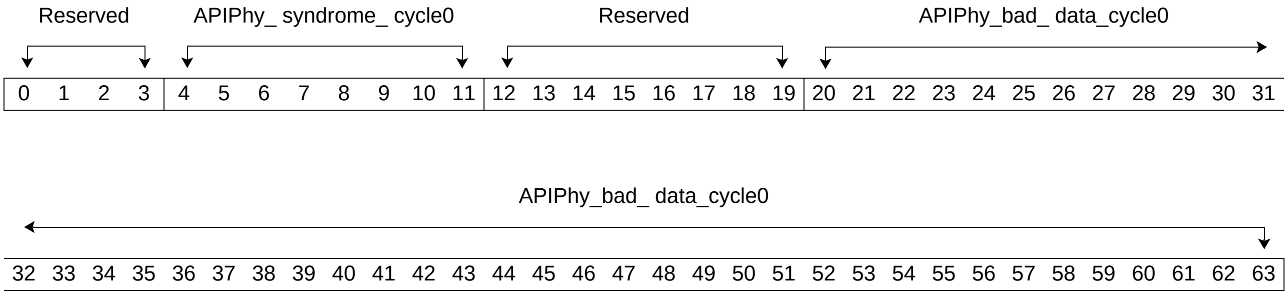
Reset Value	x‘8000 0000 0000 0000’
Address	x‘070’
Access Type	Read Only



Bits	Field Name	Description	Access	Reset
0:39	ste	Receiver IAP State Machine State (IAP_STATE). Reports the current state of the IAP state machine. Only one bit is active at a time. The bit position of the active bit corresponds to the current IAP state. (One hot-active bit position determines current state).	R	0x8000000000 (Ready State)
40:63	Reserved	Reserved	R	0x000000

12.7.4.6 APIPhy Data Error 0 Register (DATAERROR0)

Reset Value	x‘0000 0000 0000 0000’
Address	x‘080’
Access Type	Read Only



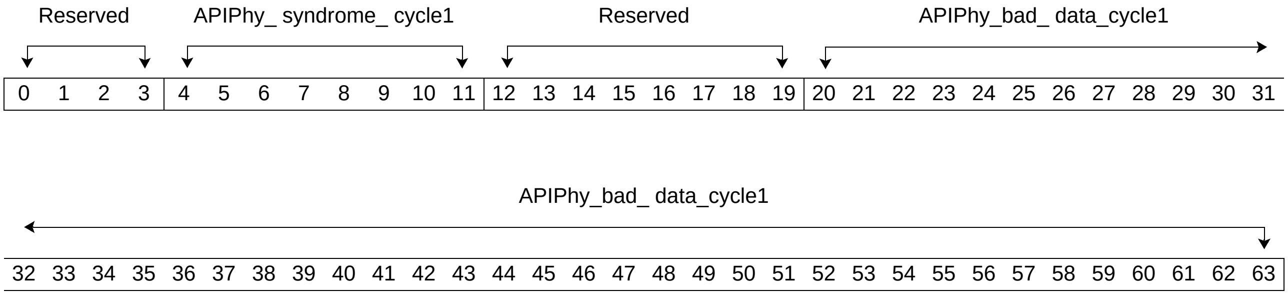
Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved	R	0000
4:11	APIPhy_ syndrome_ cycle0	Syndrome Cycle 0. This register holds syndrome of the initial beat that caused a parity error. Syndrome of the nonBCM mode (APSel = 0) parity error.	R	0x00
12:19	Reserved	Reserved	R	0x00
20:63	APIPhy_bad_ data_cycle0	Bad Data Cycle 0 (BAD_DATA_CYCLE 0). This register holds the initial beat of data that caused a parity or BCM error. If in nonBCM mode (APSel = 0), this register holds the 36 bit data and its corresponding 8 bit encoded parity that caused the parity error. If in BCM mode (APSel = 1), this register holds the 44 bit BCM encoded data that caused the data error.	R	0x0000

12.7.4.7 APIPhy Data Error 1 Register (DATAERROR1)

Reset Value x‘0000 0000 0000 0000’

Address x‘090’

Access Type Read Only



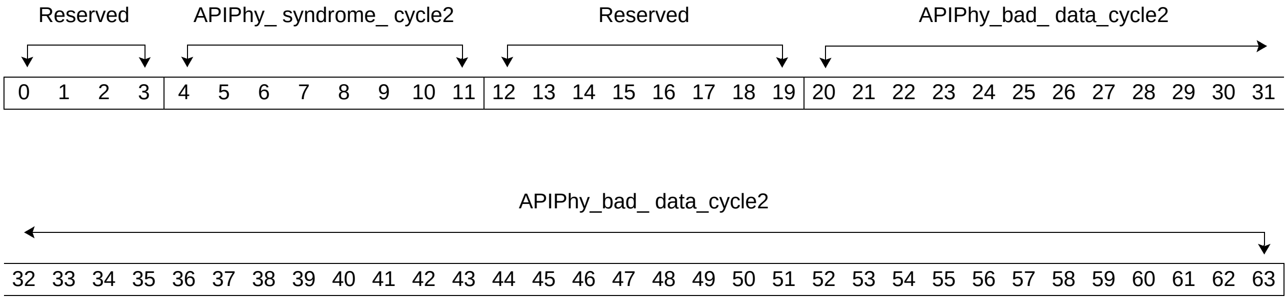
Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved	R	0000
4:11	APIPhy_ syndrome_ cycle1	Syndrome Cycle 1. This register holds the data of the 1st syndrome after the initial beat that caused a parity error. Syndrome of the non-BCM mode (APSel =0) parity error.	R	0x00
12:19	Reserved	Reserved	R	0x00
20:63	APIPhy_bad_ data_cycle1	Bad Data Cycle 1 (BAD_DATA_CYCLE 1). This register holds the data of the 1st data beat after the initial beat that caused a parity or BCM error. If in nonBCM mode (APSel = 0), this register holds the 36 bit data and its corresponding 8 bit encoded parity that caused the parity error. If in BCM mode (APSel = 1), this register holds the 44 bit BCM encoded data that caused the data error.	R	0x0000

12.7.4.8 APIPhy Data Error 2 Register (DATAERROR2)

Reset Value x‘0000 0000 0000 0000’

Address x‘0A0’

Access Type Read Only



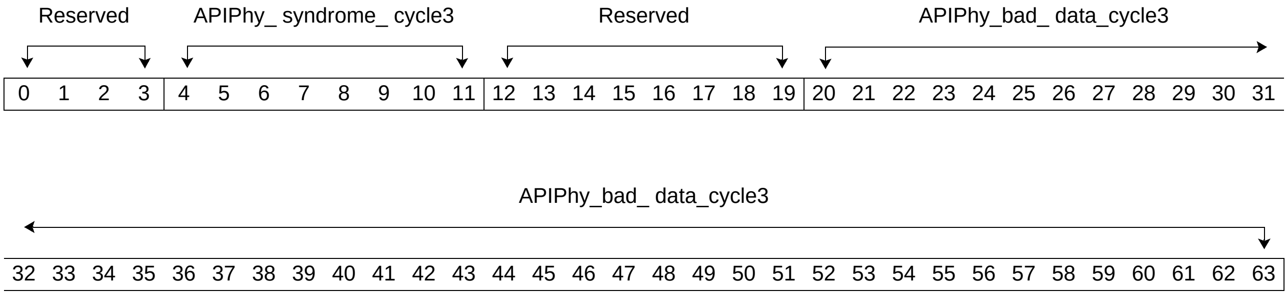
Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved	R	0000
4:11	APIPhy_ syndrome_ cycle2	Syndrome Cycle 2. This register holds the data of the 2nd syndrome after the initial beat that caused a parity error. Syndrome of the nonBCM mode (APSel =0) parity error.	R	0x00
12:19	Reserved	Reserved	R	0x00
20:63	APIPhy_bad_ data_cycle2	Bad Data Cycle 2 (BAD_DATA_CYCLE 2). This register holds the data of the 2nd data beat after the initial beat that caused a parity or BCM error. If in nonBCM mode (APSel = 0), this register holds the 36 bit data and its corresponding 8 bit encoded parity that caused the parity error. If in BCM mode (APSel = 1), this register holds the 44 bit BCM encoded data that caused the data error.	R	0x0000

12.7.4.9 APIPhy Data Error 3 Register (APIPhyDATAERROR3)

Reset Value x‘0000 0000 0000 0000’

Address x‘0B0’

Access Type Read Only



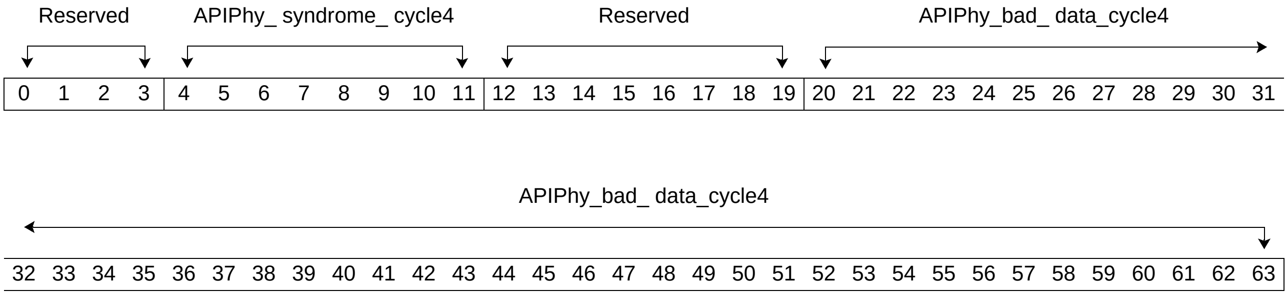
Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved	R	0000
4:11	APIPhy_ syndrome_ cycle3	Syndrome Cycle 3. This register holds the data of the 3rd syndrome after the initial beat that caused a parity error. Syndrome of the non-BCM mode (APSel =0) parity error.	R	0x00
12:19	Reserved	Reserved	R	0x00
20:63	APIPhy_bad_ data_cycle3	Bad Data Cycle 3 (BAD_DATA_CYCLE 3). This register holds the data of the 3rd data beat after the initial beat that caused a parity or BCM error. If in nonBCM mode (APSel = 0), this register holds the 36 bit data and its corresponding 8 bit encoded parity that caused the parity error. If in BCM mode (APSel = 1), this register holds the 44 bit BCM encoded data that caused the data error.	R	0x0000

12.7.4.10 APIPhy Data Error 4 Register (APIPhyDATAERROR4)

Reset Value x‘0000 0000 0000 0000’

Address x‘0C0’

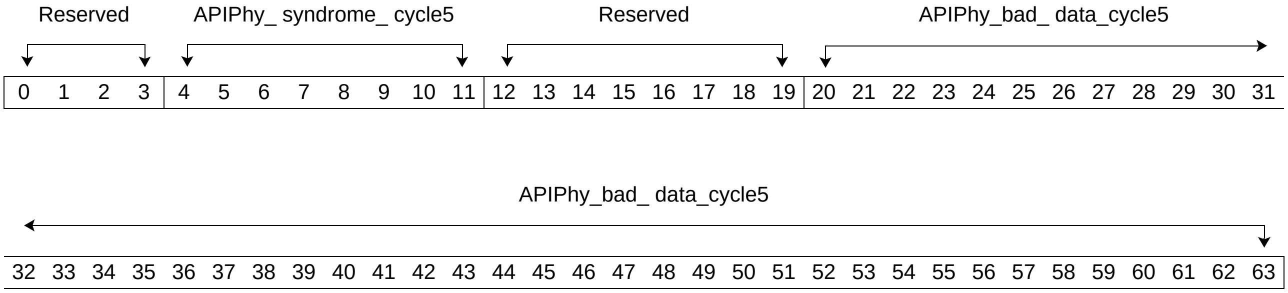
Access Type Read Only



Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved	R	0000
4:11	APIPhy_ syndrome_ cycle4	Syndrome Cycle 4. This register holds the data of the 4th syndrome after the initial beat that caused a parity error. Syndrome of the non-BCM mode (APSel =0) parity error.	R	0x00
12:19	Reserved	Reserved	R	0x00
20:63	APIPhy_bad_ data_cycle4	Bad Data Cycle 4 (BAD_DATA_CYCLE 4). This register holds the data of the 4th data beat after the initial beat that caused a parity or BCM error. If in nonBCM mode (APSel = 0), this register holds the 36-bit data and its corresponding 8-bit encoded parity that caused the parity error. If in BCM mode (APSel = 1), this register holds the 44-bit BCM encoded data that caused the data error.	R	0x0000

12.7.4.11 APIPhy Data Error 5 Register (APIPhyDATAERROR5)

Reset Value	x‘0000 0000 0000 0000’
Address	x‘0D0’
Access Type	Read only



Bits	Field Name	Description	Access	Reset
0:3	Reserved	Reserved	R	0000
4:11	APIPhy_ syndrome_ cycle5	Syndrome Cycle 5. This register holds the data of the 5th syndrome after the initial beat that caused a parity error. Syndrome of the non-BCM mode (APSel =0) parity error.	R	0x00
12:19	Reserved	Reserved	R	0x00
20:63	APIPhy_bad_ data_cycle5	Bad Data Cycle 5 (BAD_DATA_CYCLE 5). This register holds the data of the 5th data beat after the initial beat that caused a parity or BCM error. If in nonBCM mode (APSel = 0), this register holds the 36-bit data and its corresponding 8-bit encoded parity that caused the parity error. If in BCM mode (APSel = 1), this register holds the 44-bit BCM encoded data that caused the data error.	R	0x0000

12.7.4.12 APIPhy I/O Control Register (APIPhyIOCTRL)

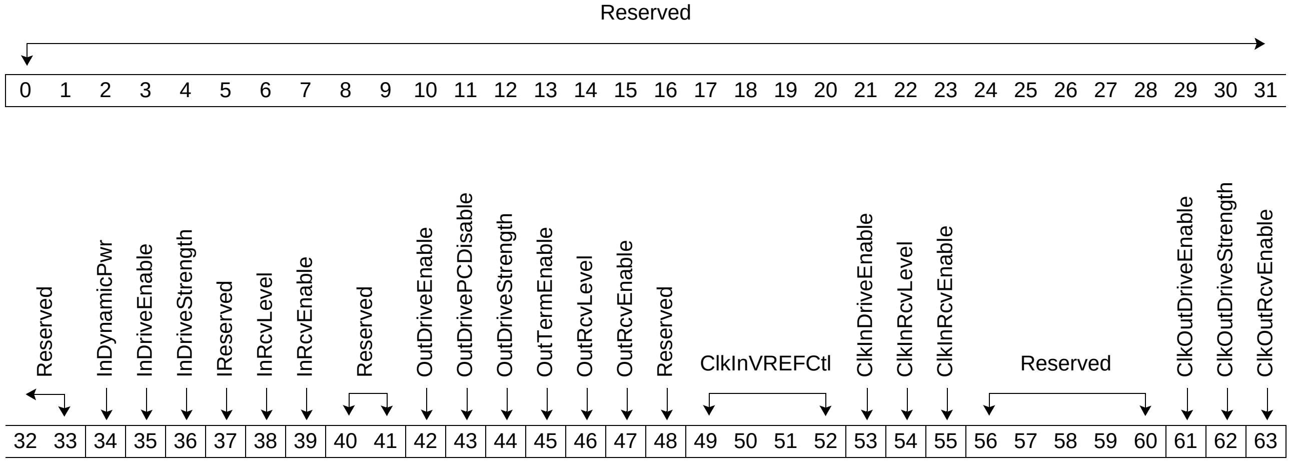
This register configures the APIPhy I/O pads for ADO, ADI, BCLKO, and BCLKI signals:

- The "ADO" inputs to the CPC945 use BBPICMP55TERM I/O *(1.5, 1.2, 1.0V Test Bus Pumped Processor Interface-2 Three State CIO)* configured as inputs using bits 34:39. The description is located in the IBM Standard Cell Nontest I/Os document (see *Section 13 References* on page 645).
- The "ADI" outputs of the CPC945 use BBPICMP55TERM configured as outputs using bits 42:47.
- The "BCLKO" input to the CPC945 uses IBPI55 I/O *(1.5, 1.2, 1.0V Nontest Bus Pumped Processor Interface-2 Differential Clock Receiver)* which is configured using bits 49:55.
- The "BCLKI" output of the CPC945 use OBP I/O *(1.5, 1.2, 1.0V Nontest Bus Pumped Processor Interface-2 Differential Clock Driver)* which is configured using bits 61:63.

Reset Value N/A

Address x‘0E0’

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
0:33	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
34	InDynamicPwr	1 Dynamic disable during bus quiesce as well as bus and system sleep 0 Dynamic disable only for bus and system sleep	R	0b1
35	InDriveEnable	1 Enable driver to pad except for bus and system sleep 0 Always tristate driver portion of buffer	R/W	0b0
36	InDriveStrength	1 Set driver output impedance to 20 Ω 0 Set driver output impedance to 40 Ω	R/W	0b1
37	Reserved	This field will read 0b1.	R/W	0b1

Bits	Field Name	Description	Access	Reset
38	InRcvLevel	1 Optimize for 1.2 V I/O 0 Optimize for 1.0 V I/O	R/W	0b1
39	InRcvEnable	1 Enable receiver on pad with dynamic disable 0 Disable receiver	R/W	0b1
40:41	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
42	OutDriveEnable	1 Enable driver to pad except for bus and system sleep 0 Always tristate driver portion of buffer	R/W	0b1
43	OutDrivePCDisable	1 Disable driver dynamic precompensation. Drive strength statically set. by register write to OutDriveStrength register. 0 Enable driver dynamic precompensation. Drive strength dynamically controlled by APIPhy unit	R/W	0b1
44	OutDriveStrength	1 Set driver output impedance to 20 Ω 0 Set driver output impedance to 40 Ω	R/W	0b0
45	OutTermEnable	1 Enable terminators at 90 Ω to Vdd/2 except for bus and system sleep 0 Disable terminators	R/W	0b0
46	OutRcvLevel	1 Optimize for 1.2V I/O 0 Optimize for 1.0V I/O	R/W	0b1
47	OutRcvEnable	1 Enable receiver on pad except for bus and system sleep 0 Disable receiver	R/W	0b0
48	Reserved	Reserved. Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
49:52	ClkInVREFCtl	0000 ZVref = 0.5 × V1.2/V1.0 (module test) 0001 ZVref = 0.48 × Vcm 0010 ZVref = 0.46 × Vcm 0011 ZVref = 0.44 × Vcm 0100 ZVref = 0.46 × Vcm 0101 ZVref = 0.44 × Vcm 0110 ZVref = 0.43 × Vcm 0111 ZVref = 0.41 × Vcm 1000 ZVref = 0.59 × Vcm 1001 ZVref = 0.57 × Vcm 1010 ZVref = 0.55 × Vcm 1011 ZVref = 0.54 × Vcm 1100 ZVref = 0.55 × Vcm 1101 ZVref = 0.54 × Vcm 1110 ZVref = 0.52 × Vcm 1111 ZVref = 0.5 × Vcm	R/W	0x0
53	ClkInDriveEnable	1 Enable driver to receiver pad except for bus and system sleep 0 Always tristate driver portion of buffer	R/W	0b0
54	ClkInRcvLevel	1 Optimize for 1.2V I/O 0 Optimize for 1.0V I/O	R/W	0b1
55	ClkInRcvEnable	1 Enable receiver on pad except for bus and system sleep 0 Disable receiver	R/W	0b1
56:59	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined

Bits	Field Name	Description	Access	Reset
60	Reserved	Reserved; Writes have no effect on hardware and Reads return undefined read data.	R/W	Undefined
61	ClkOutDriveEnable	1 Enable driver to pad except for bus and system sleep 0 Always tristate driver portion of buffer	R/W	0b1
62	ClkOutDriveStrength	1 Set driver output impedance to 20 Ω 0 Set driver output impedance to 40 Ω	R/W	0b1
63	ClkOutRcvEnable	1 Enable receiver on pad except for bus and system sleep 0 Disable receiver	R/W	0b0

12.7.4.13 APIPhy PMR I/O Control Register (APIPhyPMRIOCTRL)

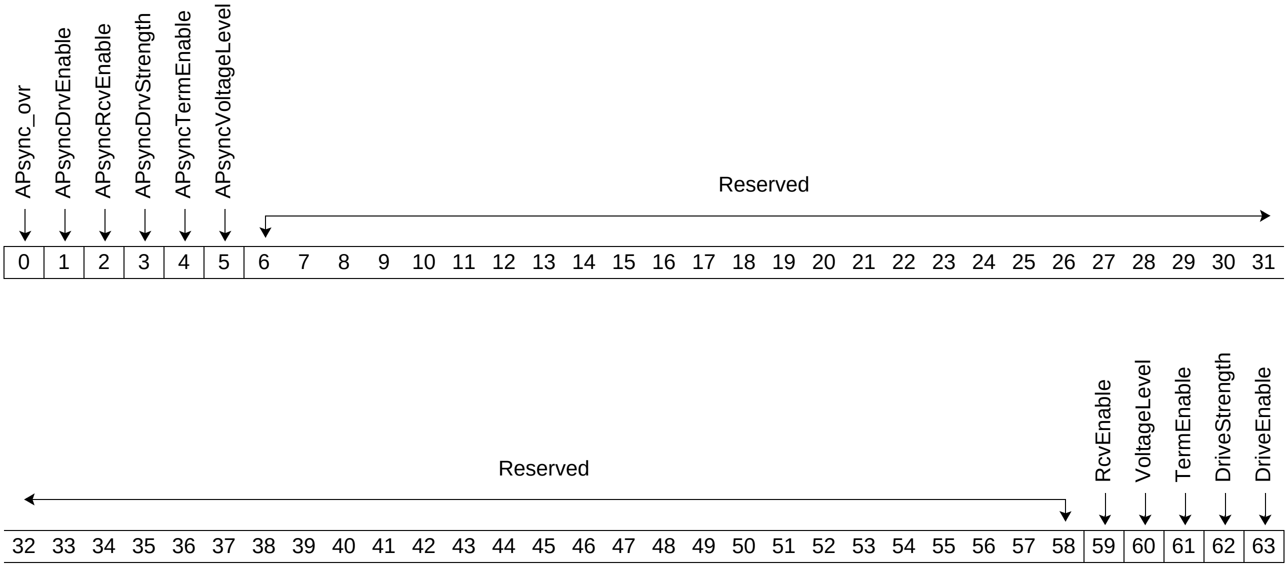
This APIPhy PMR I/O Control Register (APIPhyPMRIOCTRL) controls all I/O pins using the APIPhy I/O pads except the pins used by the actual PI Physical Interface. See *BBPICMPTERMT: 1.5, 1.2, 1.0V Test Bus Pumped Processor Interface-2 Three State CIO*. The description is located in the IBM Standard Cell Nontest I/Os document (see *Section 13 References* on page 645). This register configures the APIPhy I/O pads for APsync, QREQ, QACK, IRQ, PI_CSTP, API0_SE and API1_SE signals:

- The APsync signal of the CPC945 uses BBPICMP55TERM I/O *(1.5, 1.2, 1.0V Test Bus Pumped Processor Interface-2 Three State CIO)* configured as Inputs using bits 1:5.
- The PI_QREQ0, PI_QREQ1, PI_QACK0, PI_QACK1, IRQ0, IRQ1, and API0_SE signals of the CPC945 use the BBPICMP55TERM I/O configured using bits 59_63 at address 0xF802 20F0.
- The PI_QREQ2, PI_QREQ3, PI_QACK2, PI_QACK3, IRQ2, IRQ3, PI_CSTP and API1_SE signals of the CPC945 use the BBPICMP55TERM I/O configured using bits 59:63 at address 0xF802 30F0.

Reset Value N/A

Address x‘0F0’

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
0	APsync_ovr	If set to 1, APsync0 and APsync1 pads' driver enables (APsync_ts) and receiver enables (APsync_rg) can be set or disabled manually with a register write. If set to 0, then driver enables and receiver enables states are determined by APsync mode setting. If external APsync is enabled, APIPhy and the processors are using the external APsyncs generated off chip by the clock chip to determine timezero and APsync0 and APsync1 pads' driver enables are set to 0 and their receiver enables are set to 1. If not, APIPhy is generating its own APsync internally and generating the APsyncs for the processors to determine timezero. Also APsync0 and APsync1 pads' driver enables are set to 1 and their receiver enables are set to 0. Mode of APsync is set in PMR Configuration registers.	R/W	0b0
1	APsyncDrvEnable	1 Enable driver to pad 0 Always tristate driver portion of buffer	R/W	0b0
2	APsyncRcvEnable	1 Enable receiver on pad 0 Disable receiver	R/W	0b1
3	APsyncDrvStrength	1 Set driver output impedance to 20 Ω 0 Set driver output impedance to 40 Ω	R/W	0b0
4	APsyncTermEnable	1 Enable terminators at 55 Ω to Vdd/2 0 Disable terminators	R/W	0b0
5	APsyncVoltageLevel	1 Optimize for 1.2V I/O 0 Optimize for 1.0V I/O	R/W	0b1
6:58	Reserved	Reserved; Writes have no effect on hardware and reads return undefined read data.	R/W	Undefined
59	RcvEnable	1 Enable receiver on pad 0 Disable receiver	R/W	0b1
60	VoltageLevel	1 Optimize for 1.2V I/O 0 Optimize for 1.0V I/O	R/W	0b1
61	TermEnable	1 Enable terminators at 55 Ω to Vdd/2 0 Disable terminators	R/W	0b0
62	DriveStrength	1 Set driver output impedance to 20 Ω 0 Set driver output impedance to 40 Ω	R/W	0b0
63	DriveEnable	1 Enable driver to pad 0 Always tristate driver portion of buffer	R/W	0b1

12.7.5 Related Registers

12.7.5.1 Bus Encode Disable

Bit 31 of the CPC945 Configurable Timing Delay Parameter Register (BUSCONF). This register resides in PI. Set this bit to 1; APSel is the invert of this signal.

12.8 DRAM I^{2}C Master Controller Registers

Table 12-15. CPC945 I^{2}C Control Register Addresses

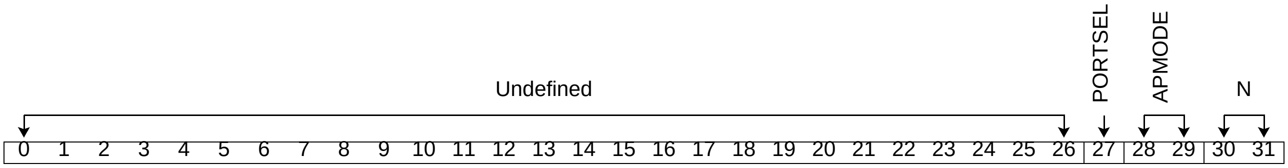
Address	Register Name	Description	Page
0xF8001000-0xF8001FFF	See below	I^{2}C Controller Registers	
0xF8001000	MODE	I^{2}C Mode Register	398
0xF8001010	CNTRL	I^{2}C Control Register	400
0xF8001020	STATUS	I^{2}C Status Register	401
0xF8001030	ISR	I^{2}C Interrupt Status Register	402
0xF8001040	IER	I^{2}C Interrupt Enable Register	403
0xF8001050	ADDR	I^{2}C Address Register	404
0xF8001060	SUBADDR	I^{2}C Sub-Address Register	404
0xF8001070	DATA	I^{2}C Data Transmit/Receive Register	405
0xF8001080	REV	I^{2}C Revision Register	405
0xF8001090	RISETIMECNT	I^{2}C ISCL Rise Time Count Register	406
0xF80010A0	BITTIMECNT	I^{2}C Bit Time Count Register	407

12.8.1 I^{2}C Controller MODE Register

The MODE register contains four mode bits used to configure the transmission mode of the I^{2}C cell and the data bit rate of the I^{2}C interface. The register is initialized to 0x00.

This register cannot be written when the I^{2}C interface is busy, as indicated by the BUSY bit in the STATUS register.

Reset Value	0x00000000
Offset	0xF8001000
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:26	Undefined	This field reads 0.	R	0x0000000
27	PORTSEL	This field is set by writing the desired value to its bit position. It controls the active I^{2}C port used by the I^{2}C cell: 0 The I^{2}C cell transmits and receives data on the ISCL0 and ISDA0 interface. 1 The I^{2}C cell transmits and receives data on the ISCL1 and ISDA1 interface.	R/W	0b0

Bits	Field Name	Description	Access	Reset
28:29	APMODE	This field is set by writing the desired value to its bit positions. This field controls the address transmission mode of the I^2C cell as follows: 00 The I^2C cell operates in dumb mode. In dumb mode, the transmissions are accomplished using the START and STOP control bits, and the DATA register. 01 The I^2C cell operates in Standard Address mode. Transmissions are accomplished using the XADDR control bit and the ADDR and DATA registers. 10 The I^2C cell operates in Standard Address with Sub-address mode. In Std Address with Sub-address mode, the transmissions are accomplished using the XADDR control bit, and the ADDR, SUBADDR, and DATA registers. 11 The I^2C cell operates in Combined mode. Transmissions are accomplished using the XADDR control bit, and the ADDR, SUBADDR, and DATA registers. In the three automated address transmission modes, the I^2C cell aborts the address transmission and sends a STOP condition if it receives a Not Acknowledge during the address transmission phase. This event is indicated by the BUSY status bit being cleared, the LASTAAK bit being cleared, and the IADDR interrupt bit being set.	R/W	0b00
30:31	N	The N field is set by writing the desired value to its bit positions. This field controls the transmission speed of the I^2C interface as follows: 00 100 kHz 01 50 kHz 10 25 kHz	R/W	0b00

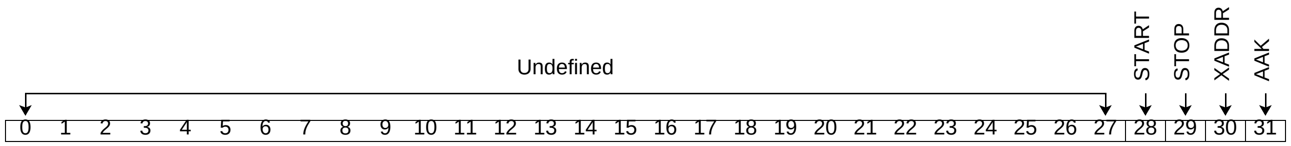
12.8.2 I^{2}C Controller CNTRL Register

The CNTRL register contains four bits used to initiate the various operations on the I^{2}C interface. All register bits are cleared to zero when the ResetClk_ signal is asserted.

Reset Value 0x00000000

Offset 0xF8001010

Access Type Read/Write, Read Only

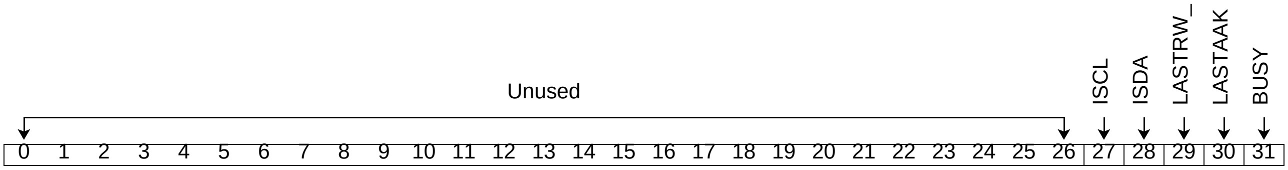


Bits	Field Name	Description	Access	Reset
0:27	Undefined	These fields read 0.	R	0x0000000
28	START	This bit is set by writing a one to its position. When this bit is set, a start condition is transmitted on the I^{2}C interface. When the start condition has been sent, the bit is cleared. Writing a zero to the START bit has no effect.	R/W	0b0
29	STOP	This bit is set by writing a one to its position. When this bit is set, a stop condition is transmitted on the I^{2}C interface. When the stop condition has been sent, the bit is cleared. Writing a zero to this bit has no effect.	R/W	0b0
30	XADDR	This bit is set by writing a one to its position. When this bit is set, an address phase is transmitted on the I^{2}C interface. The address mode is determined by bits in the MODE register. When the address phase has been sent, the bit is cleared. Writing a zero to this bit has no effect.	R/W	0b0
31	AAK	This bit is set by writing the desired value to its bit position. When this bit is set to one, an acknowledge is sent (low on SDA) during the acknowledge bit time following a received data byte. When this bit is set to zero, a Not Acknowledge is sent (high on SDA) during the acknowledge bit time following a received data byte.	R/W	0b0

12.8.3 I^{2}C Controller STATUS Register

The STATUS register contains the status bits that describe the state of the I^{2}C cell and the I^{2}C interface. The register is initialized to 0x00.

Reset Value	0x0000001A
Offset	0xF8001020
Access Type	Read Only

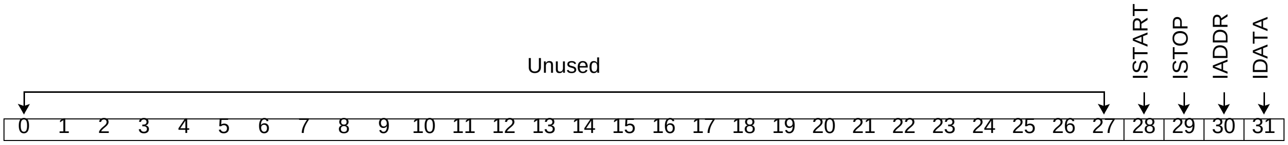


Bits	Field Name	Description	Access	Reset
0:26	Unused	These fields read 0.	R	0x0000000
27	ISCL	This bit is the double rank synchronized input from the ISCL line of the I^{2}C interface currently selected.	R	0b0
28	ISDA	This bit is the double rank synchronized input from the ISDA line of the I^{2}C interface currently selected.	R	0b0
29	LASTRW_	This bit indicates the value of the last R/W_ bit transmitted on the I^{2}C interface currently selected.	R	0b0
30	LASTAAK	This bit indicates the value of the last Acknowledge bit transmitted or received on the I^{2}C interface as follows: 1 An Acknowledge was transmitted. 0 A Not Acknowledge was transmitted.	R	0b0
31	BUSY	This bit indicates the state of the I^{2}C interface and the I^{2}C cell. If the BUSY bit is set, the I^{2}C interface is running properly. This bit is cleared when a STOP condition has been sent, and ISDA and ISCL are both high.	R	0b0

12.8.4 I^{2}C Controller Interrupt Status (ISR) Register

The Interrupt Status Register (ISR) contains the status bits for the four interrupt conditions that can occur in the I^{2}C cell. A status bit is cleared by writing a one to its bit position. The register is initialized to 0x00.

Reset Value	0x00000000
Offset	0xF8001030
Access Type	Read/Write, Read Only

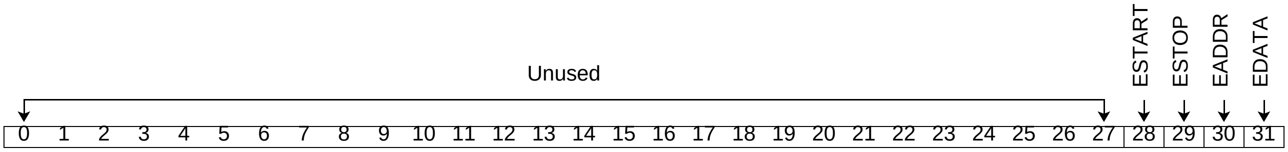


Bits	Field Name	Description	Access	Reset
0:27	Unused	These fields read 0.	R	0x0000000
28	ISTART	The start condition sent interrupt flag.	R/W	0b0
29	ISTOP	The stop condition sent interrupt flag.	R/W	0b0
30	IADDR	The address phase sent interrupt flag.	R/W	0b0
31	IDATA	The data byte sent or received interrupt flag.	R/W	0b0

12.8.5 I^{2}C Controller Interrupt Enable (IER) Register

The Interrupt Enable Register (IER) contains the enable bits that allow the four interrupt status conditions indicated in the ISR to cause assertion of the Int_ signal from the I^{2}C cell. When a bit is set in the IER, the corresponding bit in the ISR register, when set, will cause the Int_ signal to be asserted. The register is initialized to 0x00.

Reset Value	0x00000000
Offset	0xF8001040
Access Type	Read/Write, Read Only

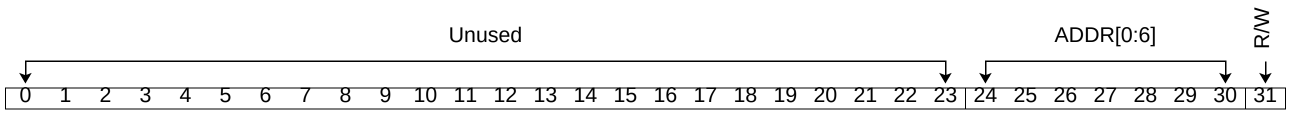


Bits	Field Name	Description	Access	Reset
0:27	Unused	These fields read 0.	R	0x0000000
28	ESTART	This bit enables start condition sent interrupt.	R/W	0b0
29	ESTOP	This bit enables stop condition sent interrupt.	R/W	0b0
30	EADDR	This bit enables address phase sent interrupt.	R/W	0b0
31	EDATA	This bit enables data byte sent or received interrupt.	R/W	0b0

12.8.6 I^{2}C Controller ADDR Register

The ADDR register contains the 7-bit Master address and the R/W_ bit. The R/W_ bit is sent following the 7-bit address, except during the first transmission of the 7-bit address in the combined mode. In the combined mode, the R/W_ bit following the 7-bit address is forced low, or to W_, on the first transmission. After the repeated start condition, the 7-bit address and R/W_ bit are sent as usual.

Reset Value	0x00000000
Offset	0xF8001050
Access Type	Read/Write, Read Only

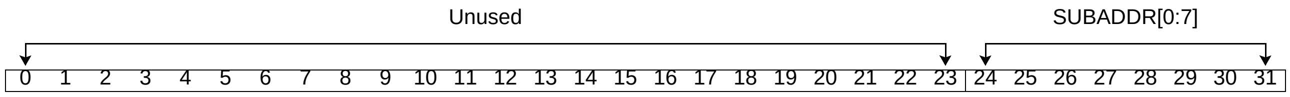


Bits	Field Name	Description	Access	Reset
0:23	Unused	This field will read all 0’s	R	0x000000
24:30	ADDR[0:6]	Master address.	R/W	0b0000000
31	R/W	This is the R/W bit.	R/W	0b0

12.8.7 I^{2}C Controller SUBADDR Register

The SUBADDR register contains the 8-bit Master Sub-Address that will be transmitted following the ADDR and R/W_ or W_ bits in either Std with Sub-address or Combined modes.

Reset Value	0x00000000
Offset	0xF8001060
Access Type	Read/Write, Read Only

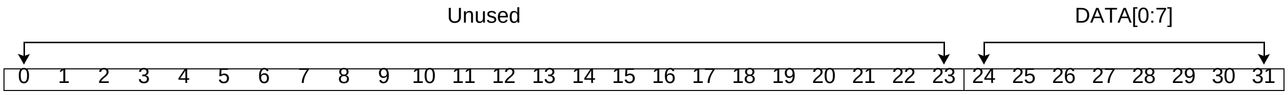


Bits	Field Name	Description	Access	Reset
0:23	Unused	This field will read all 0’s.	R	0x000000
24:31	SUBADDR[0:7]	Subaddress	R/W	0x00

12.8.8 I^{2}C Controller Data Transmit/Receive Register

The DATA register contains the byte of data to be transmitted in write mode, or the last byte of data received in read mode. This register is actually the I/O shift register. Following transmission, the register will contain the byte that was actually transmitted on the I^{2}C interface. During transmission, the value in the register is invalid.

Reset Value	0x00000000
Offset	0xF8001070
Access Type	Read/Write, Read Only

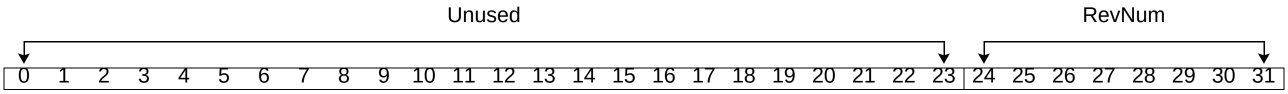


Bits	Field Name	Description	Access	Reset
0:23	Unused	This field will read all 0’s.	R	0x000000
24:31	DATA[0:7]	Data Transmit/Receive	R/W	0x00

12.8.9 I^{2}C Controller Revision Register

The REVNUM register contains the 8-bit I^{2}C Cell Revision Number.

Reset Value	0x000000A2
Offset	0xF8001080
Access Type	Read/Write, Read Only

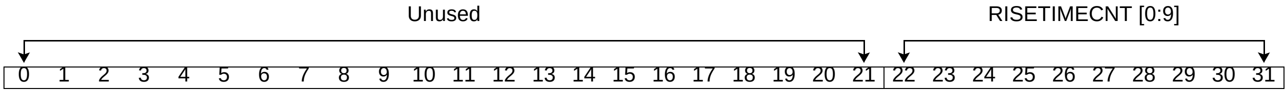


Bits	Field Name	Description	Access	Reset
0:23	Unused	This field will read all 0’s.	R	0x000000
24:31	RevNum	Revision number	R	0xA2

12.8.10 I^{2}C Controller RISETIMECNT Register

The RISETIMECNT register contains the 10-bit ISCL rise time count value. This register determines the time period that the I^{2}C cell waits after tri-stating the ISCL output before it stalls its bit time counter because of slave clock stretching. The value programmed into this register is the number of periods, minus one, of the I^{2}C CLK input that corresponds to the desired rise time period. The desired rise time period is the worst possible case, which is 110 ns for this design. The I^{2}C clock input is the ddr_clk input. For example if the ddr_clk was running at 100 MHz, the register is initialized to 0x00A, representing 11 clock periods, or 110 ns. The initial values entered below are for a 333 MHz ddr_clk (167 MHz memory controller clock). These values should be changed by software if a higher ddr_clk is used.

Reset Value	0x00000025
Offset	0xF8001090
Access Type	Read/Write, Read Only

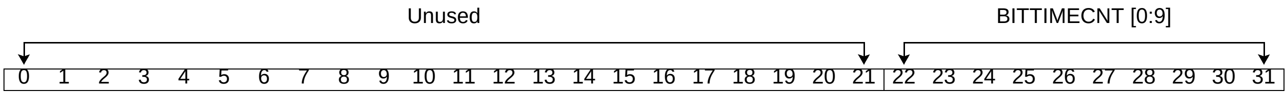


Bits	Field Name	Description	Access	Reset
0:21	Unused	This field will read all 0's.	R	0x000000
22:31	RISETIMECNT [0:9]	This is the ISCL rise time clock count value.	R/W	0x025

12.8.11 I^{2}C Controller BITTIMECNT Register

The BITTIMECNT Register contains the 10-bit I^{2}C bit time count value. The value programmed into this register is the number of periods, minus one, of the I^{2}C Clk input that corresponds to one fourth of the desired bit time period or 2.5 µs. The I^{2}C clock input is the ddr_clk input. For example if the ddr_clk was running at 100 MHz, the register is initialized to 0x0F9, representing 250 clock periods, or 2.5 µs. The initial values entered below are for a 333 MHz ddr_clk (167 MHz memory controller clock). These values should be changed by software if a higher ddr_clk is used.

Reset Value	0x00000342
Offset	0xF80010A0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:21	Unused	This field will read all 0’s.	R	0x000000
22:31	BITTIMECNT [0:9]	This is the bit time count.	R/W	0x342

12.9 Advanced Processor Interconnect Registers

Table 12-16. PI Registers.

System Bus Address	Register Name	Description	Page
0XF8030000	APIProcCmd	API Proc Command Slot Configuration	409
0XF8030010	APIIOPnd	API I/O Pending Queues Configuration	411
0XF8030020	APICmdArb	API Command Arbitration	412
0XF8030030	APITRqCfg	API Target Request Queue Configuration	414
0XF8030040	APITRspCfg	API Target Response Queue Configuration	415
0XF8030050	APIDtQCfg	API Data Queue Configuration	416
0XF8030060	APIWdbCfg	API Write Data Buffer Configuration	417
0XF8030070	APIIntCfg	API Intervention Buffer Configuration	418
0XF8030080	APIMemReqCfg	API Memory Request Queue Configuration	419
0XF8030090	APIMemRdCfg	API Memory Read Data Configuration	421
0XF80300A0	APIExcp	API Exception	423
0XF80300B0	APIMask0	API Mask 0	424
0XF80300C0	APIMask1	API Mask 1	426
0XF80300D0	APITRqGuar	API Target Request Queue Guarantees	427
0XF80300E0	APISnpSltCfg	API Snoop Slot Configuration	429
0XF8030100	APIPaamWin	API PAAM Window Values	430
0XF8030110	APISnoopWin	API Snoop Window Values	431
0XF8030120	APIIOSnoopWin	API I/O Snoop Window Values	432
0XF8030130	APIStatLat	API Handshake Status Latency Values	433
0XF8030140	APISnoopLat	API Snoop Latency Values	434
0XF80301C0	PSRO	PSRO Register	435
0XF8030200- 0XF8030210	SysCmdCntl[0:1]	System Command Control [0:1] Register	436
0XF8030220	SysCmdStat	System Command Status Register	436
0XF8030240- 0XF8030270	SysCmdDt[0:3]	System Command Data [0:3] Register	436
0XF8031000- 0XF8031050	Reserved	Reserved	
0XF8031080- 0XF80310D0	Reserved	Reserved	
0XF8033000	DARTCntl	DART Control Register	441
Note: Any register addresses in the F803xxxx range not listed here do not exist. Access to these registers is undefined.			

Table 12-16. PI Registers.

System Bus Address	Register Name	Description	Page
0XF8033010	DARTBase	DART Base Register	442
0XF8033020	DARTSize	DART Size Register	442
0XF8033030	DARTExcp	DART Exception Status Register	443
0XF8034000- 0XF8034FF0	DARTTag[0:3]	DART TLB Tag Array	445
0XF8038000- 0XF803BFF0	DARTData[0:3]	DART TLB Data Array	446
Note: Any register addresses in the F803xxxx range not listed here do not exist. Access to these registers is undefined.			

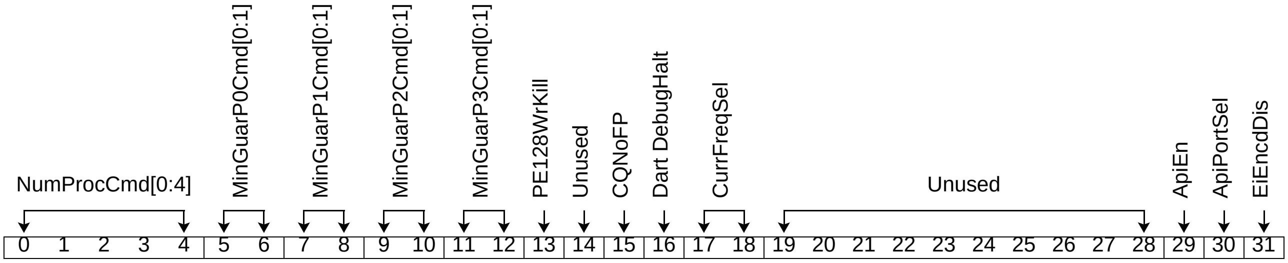
12.9.1 API Proc Command Slot Configuration Register (APIProcCmd)

This register defines the values associated with command slots.

Reset Value 0x85540000

Offset 0xF8030000

Access Type Read/Write, Read Only



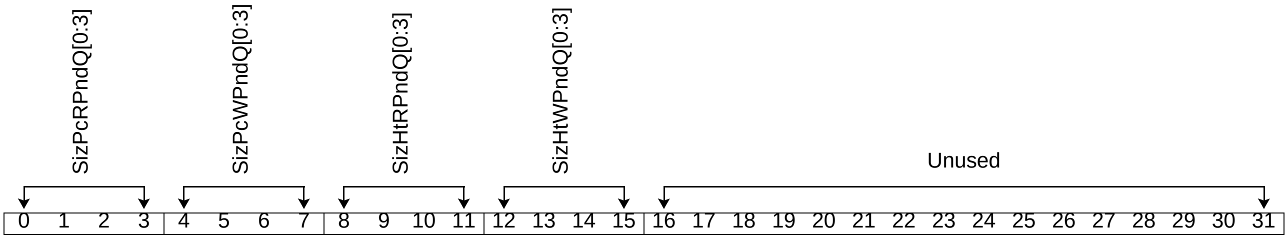
Bits	Field Name	Description	Access	Reset
0:4	Num ProcCmd [0:4]	Number of Processor command Slots. Minimum: 7, Maximum: 16, Default: 16 Must be greater than the sum of the MinGuarP[0,1,2,3] values plus 2.	R/W	0b10000
5:6	MinGuar P0Cmd [0:1]	Minimum guaranteed number of Command Slots for Proc 0. Minimum: 1, Maximum: 3, Default: 2 See NumProcCmd constraints.	R/W	0b10
7:8	MinGuar P1Cmd [0:1]	Minimum guaranteed number of Command Slots for Proc 1. Minimum: 1, Maximum: 3, Default: 2 See NumProcCmd constraints.	R/W	0b10
9:10	MinGuar P2Cmd [0:1]	Minimum guaranteed number of Command Slots for Proc 2. Minimum: 1, Maximum: 3, Default: 2 See NumProcCmd constraints.	R/W	0b10

Bits	Field Name	Description	Access	Reset
11:12	MinGuar P3Cmd [0:1]	Minimum guaranteed number of Command Slots for Proc 3. Minimum: 1, Maximum: 3, Default: 2 See NumProcCmd constraints.	R/W	0b10
13	PE128 WrKill	PCIe 128 byte write with kill support. PCIe interface to PI supports write with kill if 128-byte transfer and WrAll active, otherwise use writes with flush. 0 All writes with flush 1 Writes with kill	R/W	0b1
14	Unused	This field is not writable and will read all 0’s	R	0b0
15	CQNoFP	CmdQ no fast path. This bit disables a Fast Path from the Processor Interface to Command Queue Arbiter. 0 Fast enabled 1 Fast disabled	R/W	0b0
16	Dart Debug Halt	DART debug halt requests. This bit halts DART translation requests from being granted. Used for debug. 0 No halt requests 1 Halt requests	R/W	0b0
17:18	CurrFreq Sel	Current frequency select. This field indicates the current frequency mode that PI is in (full, half, or quarter) due to a power tuning command. This is a read only value. Changing this value can only occur with a power tuning command. 00 Full 01 Half 10 Quarter	R	0b00
19:28	Unused	This field is not writable and will read all 0’s.	R	0x000
29	ApiEn	API interface enable. After reset the PI interface is disabled so that queue parameters can be updated. After the parameters are updated, this bit is enabled to start PI transactions. The queue parameters affected are in the APIWdbCfg and APIMemRdCfg registers. This bit should not be disabled (0) by a write. 0 Disabled 1 Enabled	R/W	0b0
30	ApiPortSel	API port select. This bit remaps ProcessID to PI Module. The remapping maps “abcd” to “abdc” for MTag bits [0:3]. 0 No Mapping 1 Mapping	R/W	0b0
31	EiEncdDis	PI encode disable. This bit enables PI encoding. 0 Encoding 1 No Encoding	R/W	0b0

12.9.2 API I/O Pending Queue Configuration Register (APIIOPnd)

This register defines the values associated with the I/O pending queues.

Reset Value	0x88880000
Offset	0xF8030010
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:3	SizPcRPndQ [0:3]	Size of PCIe Read Pending Queue. Minimum: 1, Maximum: 8, Default: 8	R/W	0b1000
4:7	SizPcWPndQ [0:3]	Size of PCIe Write Pending Queue. Minimum: 1, Maximum: 8, Default: 8	R/W	0b1000
8:11	SizHtRPndQ [0:3]	Size of HT Read Pending Queue. Minimum: 1, Maximum: 8, Default: 8	R/W	0b1000
12:15	SizHtWPndQ [0:3]	Size of HT Write Pending Queue. Minimum: 1, Maximum: 8, Default: 8	R/W	0b1000
16:31	Unused	This field is not writable and will read all 0’s	R	0x0000

12.9.3 API Command Arbitration Register (APICmdArb)

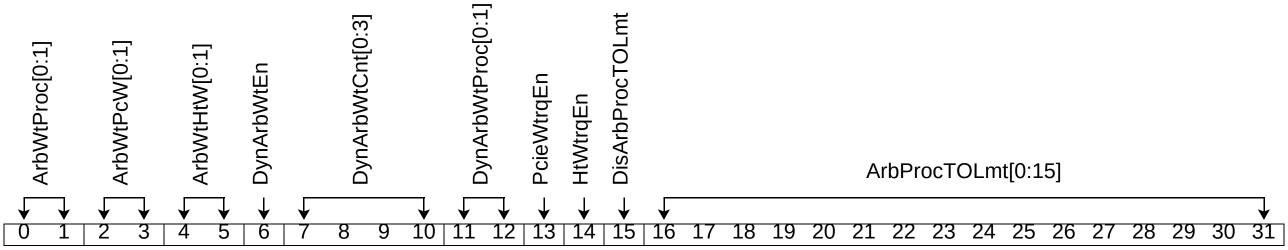
This register specifies the weights for the round-robin arbitration. Each agent can have a weight assigned such that agent retains the priority for that many requests of the arbiter once it has priority. A weight of zero indicates that this requester does not participate in the round-robin selection (it has low priority) and is only serviced when the other requesters are idle.

This arbitration is temporarily overridden when a HT or PCIe request encounters a retry (and DynArbWtEn is active). This is called dynamic arbitration. When this occurs, the arbitration weight for processor requests is temporarily set to the value specified by DynArbWtProc. This weight stays in effect until the number of requests specified by DynArbWtCnt has been granted to the processor or the processor command request queue is empty. When this occurs the processor arbitration weight returns to ArbWtProc.

Reset Value 0x57180200

Offset 0xF8030020

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
0:1	ArbWt Proc[0:1]	Arbitration weight for processor commands Minimum: 0, Maximum: 3, Default: 1 0 Low priority n n requests	R/W	0b01
2:3	ArbWt PcW[0:1]	Arbitration weight for PCIe commands. Minimum: 0, Maximum: 3, Default: 1 0 Low priority n n requests	R/W	0b01
4:5	ArbWt HtW[0:1]	Arbitration weight for HT commands. Minimum: 0, Maximum: 3, Default: 1 0 Low priority n n requests	R/W	0b01
6	DynArb WtEn	Enables the dynamic arbitration weight feature described above. 0 Disabled 1 Enabled	R/W	0b1
7:10	DynArbWt Cnt[0:3]	Dynamic arbitration weight count The number of processor requests that need to be satisfied during dynamic arbitration before returning to ArbWtProc.	R/W	0b1000
11:12	DynArbWt Proc[0:1]	The temporary weight for processor requests during dynamic arbitration. Minimum: 1, Maximum: 3, Default: 3	R/W	0b11

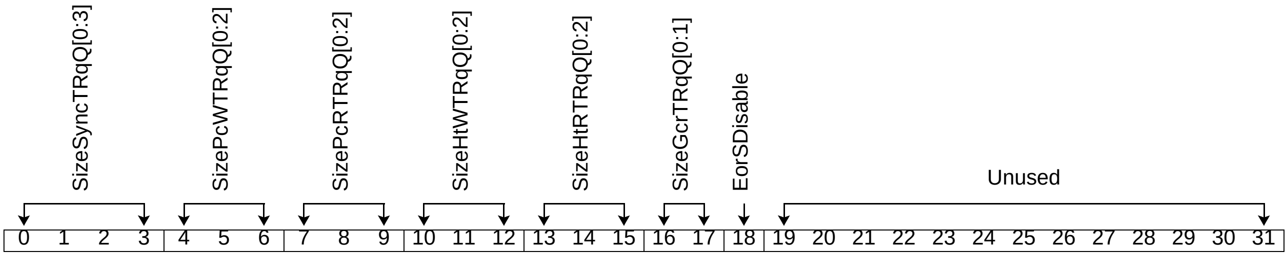
Bits	Field Name	Description	Access	Reset
13	Pcie WtrqEn	PCIe waiting to reflect queue (Wtrq) enable This bit enables the feature of allowing more than PCIe read to be in the Waiting to Reflect Queue. 0 Disabled 1 Enabled	R/W	0b0
14	Ht WtrqEn	HT Wtrq enable This bit enables the feature of allowing more than HT read to be in the waiting to reflect queue. 0 Disabled 1 Enabled	R/W	0b0
15	DisArbProc-TOLmt	Disable Arb Proc timeout limit This bit disables the processor timeout (snoop watchdog timer). 0 Enabled 1 Disabled	R/W	0b0
16:31	ArbProc TOLmt[0:15]	Arb Proc timeout limit Initial value for Arb Proc Timeout Counter (snoop watchdog timer). Whenever the processor is granted arbitration, this counter is reset to this initial value, otherwise it decrements. If the timer reaches zero then the highest priority command to I/O that is waiting to be reflected is reflected and forced to retry.	R/W	0x0200

12.9.4 API Target Request Queue Configuration Register (APITRqCfg)

These values are used to configure the target and synchronization request queues for nonmemory accesses.

If any of these values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value	0x89248000
Offset	0xF8030030
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:3	SizeSync TRqQ[0:3]	Number of entries in Sync Target Request queue. Minimum: 4, Maximum: 8, Default: 8	R/W	0b1000
4:6	SizePcW TRqQ [0:2]	Number of entries in PCIe Write Target Request queue. Minimum: 2, Maximum: 4, Default: 4 Must be greater than or equal to the sum of MinGuarProcPcWTRqQ and MinGuarIOPcWTRqQ. See *Section 12.9.4 API Target Request Queue Configuration Register (APITRqCfg)* on page 414.	R/W	0b100
7:9	SizePcR TRqQ [0:2]	Number of entries in PCIe Read Target Request queue. Minimum: 2, Maximum: 4, Default: 4 Must be greater than or equal to the sum of MinGuarProcPcRTRqQ and MinGuarIOPcRTRqQ. See *Section 12.9.4 API Target Request Queue Configuration Register (APITRqCfg)* on page 414.	R/W	0b100
10:12	SizeHtW TRqQ [0:2]	Number of entries in HT Write Target Request queue. Minimum: 2, Maximum: 4, Default: 4 Must be greater than or equal to the sum of MinGuarProcHtWTRqQ and MinGuarIOHtWTRqQ. See *Section 12.9.4 API Target Request Queue Configuration Register (APITRqCfg)* on page 414.	R/W	0b100
13:15	SizeHtR TRqQ [0:2]	Number of entries in HT Read Target Request queue. Minimum: 2, Maximum: 4, Default: 4 Must be greater than or equal to the sum of MinGuarProcHtRTRqQ and MinGuarIOHtRTRqQ. See *Section 12.9.4 API Target Request Queue Configuration Register (APITRqCfg)* on page 414.	R/W	0b100
16:17	SizeGcr TRqQ[0:1]	Number of entries in Gcr Target Request queue. Minimum: 1, Maximum: 2, Default: 2	R/W	0b10

Bits	Field Name	Description	Access	Reset
18	EorS Disable	EorS disable Disables the EorS signal on the target interfaces. 0 Enabled 1 Disabled	R/W	0b0
19:31	Unused	This field is not writable and will read all 0’s.	R	0x0000

12.9.5 API Target Response Queue Configuration Register (APITRspCfg)

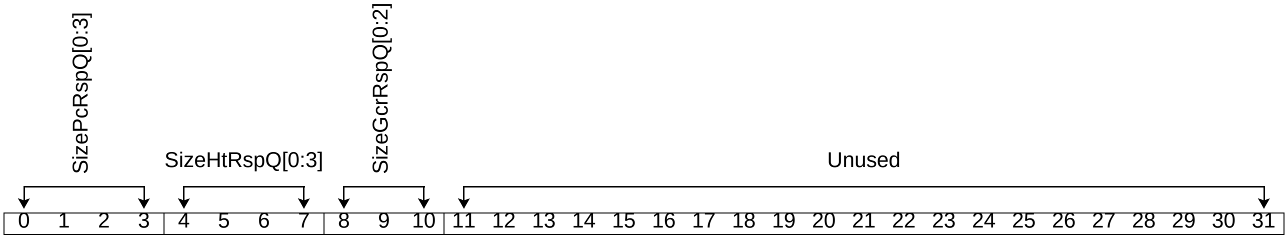
These values are used to configure the target response queues for nonmemory accesses.

If any of these values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value 0x88800000

Offset 0xF8030040

Access Type Read/Write, Read Only



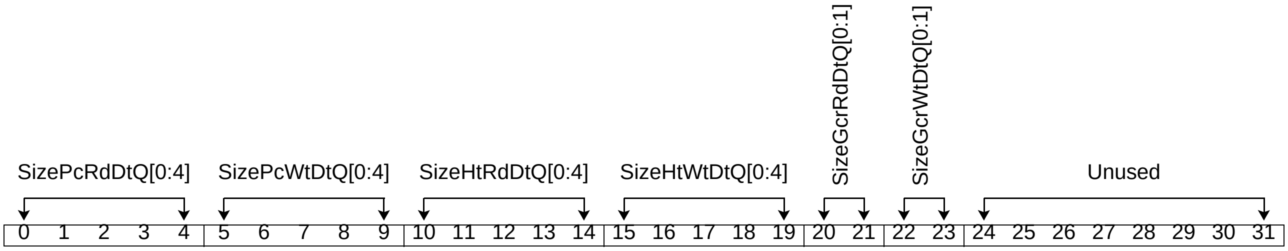
Bits	Field Name	Description	Access	Reset
0:3	SizePcRspQ [0:3]	Number of entries in PCIe Response queue. Minimum: 4, Maximum: 8, Default: 8	R/W	0b1000
4:7	SizeHtRspQ [0:3]	Number of entries in HT Response queue. Minimum: 4, Maximum: 8, Default: 8	R/W	0b1000
8:10	SizeGcrRspQ [0:2]	Number of entries in GCR Response queue. Minimum: 2, Maximum: 4, Default: 4	R/W	0b100
11:31	Unused	This field is not writable and will read all 0’s	R	0x000000

12.9.6 API Target Data Queue Configuration Register (APIDtQCfg)

These values are used to configure the target data queues for nonmemory accesses.

If any of these values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value	0x7BDEDA00
Offset	0xF8030050
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:4	SizePc RdDtQ [0:4]	Number of entries in PCIe Read Data queue. Minimum: 8, Maximum: 16, Default: 15	R/W	0b01111
5:9	SizePc WtDtQ [0:4]	Number of entries in PCIe Write Data queue. Minimum: 8, Maximum: 15, Default: 15	R/W	0b01111
10:14	SizeHt RdDtQ [0:4]	Number of entries in HT Read Data queue. Minimum: 8, Maximum: 16, Default: 15	R/W	0b01111
15:19	SizeHt WtDtQ [0:4]	Number of entries in HT Write Data queue. Minimum: 8, Maximum: 15, Default: 15	R/W	0b01111
20:21	SizeGcr RdDtQ [0:1]	Number of entries in GCR Read Data queue. Minimum: 1, Maximum: 2, Default: 2	R/W	0b10
22:23	SizeGcr WtDtQ [0:1]	Number of entries in GCR Write Data queue. Minimum: 1, Maximum: 2, Default: 2	R/W	0b10
24:31	Unused	This field is not writable and will read all 0’s.	R	0x00

12.9.7 API Write Data Buffer (WDB) Configuration Register (APIWdbCfg)

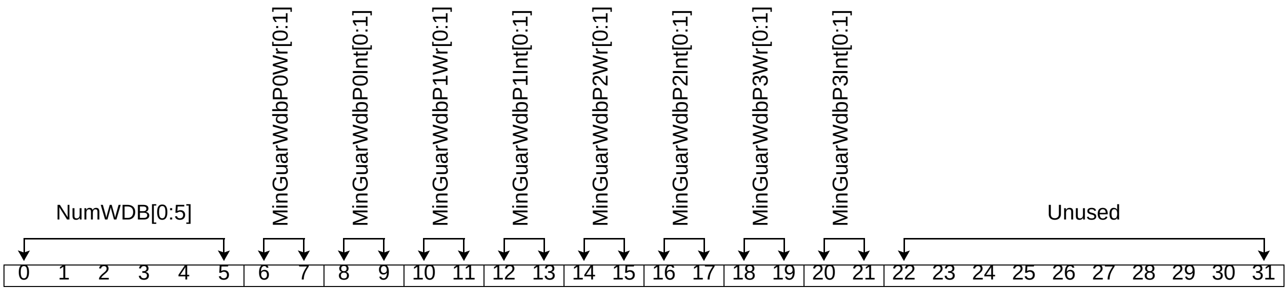
These values are used to configure the size of the API WDB and how many entries are guaranteed for writes and interventions per processor. This register should only be updated when ApiEn is disabled (see *Section 12.9.1 API Proc Command Slot Configuration Register (APIProcCmd)*).

If any of these values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429)

Reset Value 0x82AAA800

Offset 0xF8030060

Access Type Read/Write, Read Only



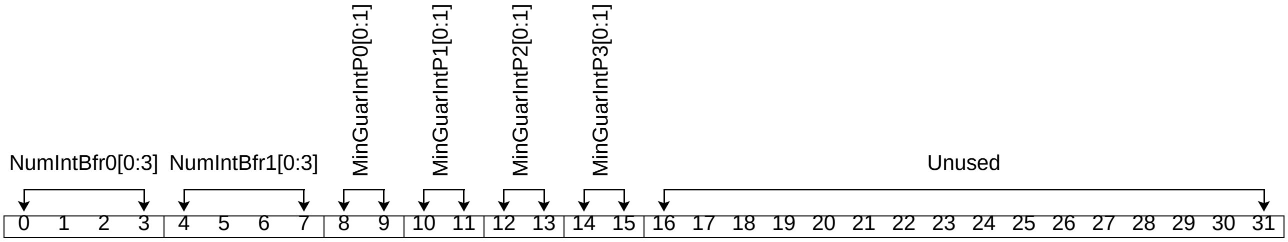
Bits	Field Name	Description	Access	Reset
0:5	NumWDB[0:5]	Number of 128-byte PI WDB entries. Minimum: 12, Maximum: 32, Default: 32 Must be greater than or equal to the sum of the MinGuarWdbP[0,1,2,3]Wr and MinGuarWdbInt[0,1,2,3] values.	R/W	0b100000
6:7	MinGuarWdbP0Wr [0:1]	Minimum number of PI WDB entries guaranteed for Proc 0 Writes. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
8:9	MinGuarWdbP0Int [0:1]	Minimum number of PI WDB entries guaranteed for Proc 0 Interventions. Minimum: 1, Maximum: 3, Default: 2	R/W	0b10
10:11	MinGuarWdbP1Wr [0:1]	Minimum number of PI WDB entries guaranteed for Proc 1 Writes. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
12:13	MinGuarWdbP1Int [0:1]	Minimum number of PI WDB entries guaranteed for Proc 1 Interventions. Minimum: 1, Maximum: 3, Default: 2	R/W	0b10
14:15	MinGuarWdbP2Wr [0:1]	Minimum number of PI WDB entries guaranteed for Proc 2 Writes. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
16:17	MinGuarWdbP2Int [0:1]	Minimum number of PI WDB entries guaranteed for Proc 2 Interventions. Minimum: 1, Maximum: 3, Default: 2	R/W	0b10
18:19	MinGuarWdbP3Wr [0:1]	Minimum number of PI WDB entries guaranteed for Proc 3 Writes. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
20:21	MinGuarWdbP3Int [0:1]	Minimum number of PI WDB entries guaranteed for Proc 3 Interventions. Minimum: 1, Maximum: 3, Default: 2	R/W	0b10
22:31	Unused	This field is not writable and will read all 0’s.	R	0x000

12.9.8 API Intervention Buffer Configuration Register (APIIntCfg)

These values are used to configure the size of the API intervention buffer and how many entries are guaranteed for per processor.

If any of these values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value	0xFFAA0000
Offset	0xF8030070
Access Type	Read/Write, Read Only



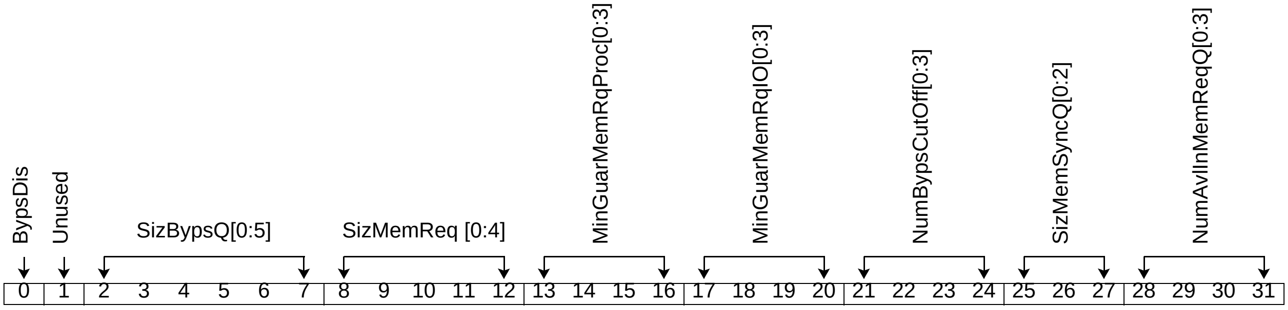
Bits	Field Name	Description	Access	Reset
0:3	NumIntBfr0 [0:3]	Number of Intervention Buffers for Port 0. Minimum: 1, Maximum: 15, Default: 15 Must be greater than or equal to the sum of the MinGuarIntP[0,1] values.	R/W	0b1111
4:7	NumIntBfr1 [0:3]	Number of Intervention Buffers for Port 1. Minimum: 1, Maximum: 15, Default: 15 Must be greater than or equal to the sum of the MinGuarIntP[2,3] values.	R/W	0b1111
8:9	MinGuarIntP0 [0:1]	Minimum number of Intervention Buffers guaranteed for Proc 0. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
10:11	MinGuarIntP1 [0:1]	Minimum number of Intervention Buffers guaranteed for Proc 1. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
12:13	MinGuarIntP2 [0:1]	Minimum number of Intervention Buffers guaranteed for Proc 2. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
14:15	MinGuarIntP3 [0:1]	Minimum number of Intervention Buffers guaranteed for Proc 3. Minimum: 2, Maximum: 3, Default: 2	R/W	0b10
16:31	Unused	This field is not writable and will read all 0’s.	R	0x0000

12.9.9 API Memory Request Configuration Register (APIMemReqCfg)

These values are used to configure the parameters associated with memory requests.

If any of the “Size” or “Guar” values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value	0x2082234C
Offset	0xF8030080
Access Type	Read/Write



Bits	Field Name	Description	Access	Reset
0	BypsDis	Bypass Disable Specifies whether SnoopBypass is disabled. SnoopBypass is the feature that allows a memory request to be initiated before a command has been snooped by the processors. 0 Enabled 1 Disabled	R/W	0b0
1	Unused	This field is not writable and will read all 0’s.	R/W	0b0
2:7	SizBypsQ[0:5]	Size of Bypass Queue. For now this should always be 32 and not modified.	R/W	0b100000
8:12	SizMemReq [0:4]	Size of Memory Request Queue. Minimum: MinGuarMemRqProc + MinGuarMemRqIO + 2 Maximum: 16 Default: 16	R/W	0b10000
13:16	MinGuarMem RqProc[0:3]	Minimum number of Memory Request Queue entries guaranteed for Processor requests. Minimum: 1 Maximum: SizMemReq - MinGuarMemRqIO – 2 Default: 4	R/W	0b0100
17:20	MinGuarMem RqIO[0:3]	Minimum number of Memory Request Queue entries guaranteed for I/O requests. Minimum: 1 Maximum: SizMemReq - MinGuarMemRqProc - 2 Default: 4	R/W	0b0100

Bits	Field Name	Description	Access	Reset
21:24	NumBypsCutOff [0:3]	Number Bypass CutOff. Number of entries in Memory Request Queue after which bypass is blocked. Minimum: 1 Maximum: SizMemReq - MinGuarMemRqProc – MinGuarMemRqIO - 2 Default: 6	R/W	0b0110
25:27	SizMemSyncQ [0:2]	Size of Sync Queue to DDR2. For now this should always be 4 and not modified.	R/W	0b100
28:31	NumAvlIn MemReqQ[0:3]	Number Avail Mark in MemReqQ. This field specifies the number of Memory Request Queue entries that need to be available to make it safe for the Snoop Pipe to not have to check for resources. This field is only valid when the SafeQCnt feature is enabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429) Minimum: MinGuarMemRqIO + MinGuarMemRqProc + 4 Maximum: SizMemReq Default: 12	R/W	0b1100

12.9.10 API Memory Read Configuration Register (APIMemRdCfg)

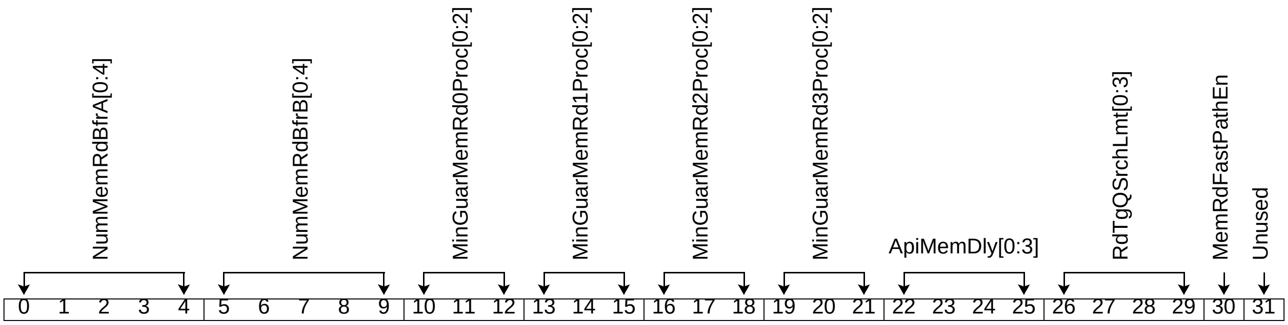
These values are used to configure the parameters associated with memory read requests and read data.

This number of buffer fields and guaranteed values (bits 0:21) should only be updated when ApiEn is disabled (see *Section 12.9.1 API Proc Command Slot Configuration Register (APIProcCmd)* on page 409). If any of the “Number” or “Guar” values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value 0x841B6D62

Offset 0xF8030090

Access Type Read/Write



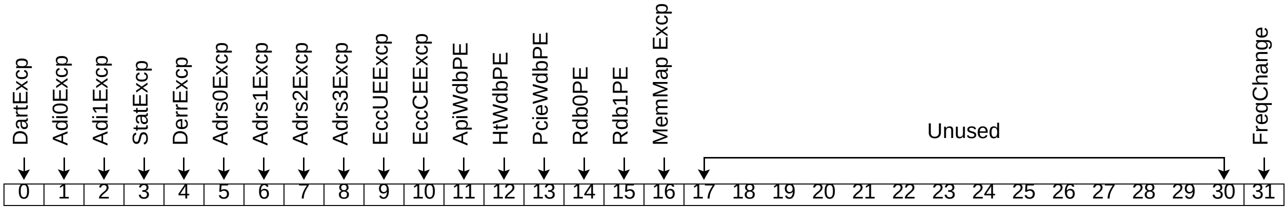
Bits	Field Name	Description	Access	Reset
0:4	NumMem RdBfrA [0:4]	Number of Read Buffers for Port 0. Minimum: MinGuarMemRd0Proc. + MinGuarMemRd1Proc. Maximum: 16, Default: 16	R/W	0b10000
5:9	NumMem RdBfrB [0:4]	Number of Read Buffers for Port 1. Minimum: MinGuarMemRd2Proc. + MinGuarMemRd3Proc. Maximum: 16, Default: 16	R/W	0b10000
10:12	MinGuar MemRd0Proc [0:2]	Minimum number of Memory Read buffers guaranteed for Processor 0. Minimum: 3, Maximum: 7, Default: 3	R/W	0b010
13:15	MinGuar MemRd1Proc [0:2]	Minimum number of Memory Read buffers guaranteed for Processor 1. Minimum: 3, Maximum: 7, Default: 3	R/W	0b010
16:18	MinGuar MemRd2Proc [0:2]	Minimum number of Memory Read buffers guaranteed for Processor 2. Minimum: 3, Maximum: 7, Default: 3	R/W	0b010
19:21	MinGuar MemRd3Proc [0:2]	Minimum number of Memory Read buffers guaranteed for Processor 3. Minimum: 3, Maximum: 7, Default: 3	R/W	0b010
22:25	ApiMemDly [0:3]	PI Mem Delay Delay in DDR2 cycles from the TgV to Read Data on the DDR2 to PI Read Data Interface. If this value is changed, then the ApiRdTgDelay field in the “Memory Bus Configuration Register 2” on page 474 (in PROGDDR) needs to be updated.	R/W	0b0101

Bits	Field Name	Description	Access	Reset
26:29	RdTgQ SrchLmt [0:3]	Read Tag Queue Search Limit This field specifies how many entries into the Tag Q should be searched looking for requests that have Snoop Status available, skipping over bypass requests that do not have Snoop Status available.	R/W	0b1000
30	MemRd FastPathEn	Memory Read Data Fast Path Enable Enables Memory Read Fast Path that bypasses the Memory Read Data Buffers. Even if this bit is enabled, the FastPath feature is disabled by hardware if the PI block is not running in “Full” frequency Power tuning mode. See CurrFreqSel field in *Section 12.9.1 API Proc Command Slot Configuration Register (APIProcCmd)* on page 409. 0: Fast Path disabled 1: Fast Path enabled **Warning**: api_clk frequency must be greater than ddr_clk frequency for FastPath to be enabled.	R/W	0b1
31	Unused	This field is not writable and will read all 0’s	R/W	0b0

12.9.11 API Exception Register (APIExcp)

Exceptions detected in API Block. Any bit on that is enabled through APIMASK0 register activates an PI Exception (pin PI_CSTP). Any bit on that is enabled through APIMASK1 register activates a ChipFault (pin CHP_FAULT_N).

Reset Value	0x60000000
Offset	0xF80300A0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	DartExcp	DART Translation Exception	*R/W	0b0
1	Adi0Excp	ADI0 Exception Handshake error on ADI bus for PI port 0.	*R/W	0b0
2	Adi1Excp	ADI1 Exception Handshake error on ADI bus for PI port 1.	*R/W	0b0
3	StatExcp	Snoop Status Exception Incorrect or unexpected Accumulated Snoop Status received.	*R/W	0b0
4	DerrExcp	Data Error Exception Data Error Exception due to receiving a TEA.	*R/W	0b0
5	Adrs0Excp	Addressing exception from Processor 0.	*R/W	0b0
6	Adrs1Excp	Addressing exception from Processor 1.	*R/W	0b0
7	Adrs2Excp	Addressing exception from Processor 2.	*R/W	0b0
8	Adrs3Excp	Addressing exception from Processor 3.	*R/W	0b0
9	EccUEExcp	ECC Uncorrectable Error Exception. This exception is from DDR2 so that it can send a ChipFault and CheckStop.	*R/W	0b0
10	EccCEExcp	ECC Correctable Error Exception. This exception is from DDR2 so that it can send a ChipFault and CheckStop.	*R/W	0b0
11	ApiWdbPE	PI WDB Parity Error Exception.	*R/W	0b0
12	HtWdbPE	HT WDB Parity Error Exception.	*R/W	0b0
13	PcieWdbPE	PCIe WDB Parity Error Exception.	*R/W	0b0
14	Rdb0PE	RDB0 Parity Error Exception.	*R/W	0b0

Note: *These bits clear when read.

Bits	Field Name	Description	Access	Reset
15	Rdb1PE	RDB1 Parity Error Exception.	*R/W	0b0
16	MemMap Excp	Memory Mapping Exception. This exception is from DDR2 so that it can send a ChipFault and CheckStop.	*R/W	
17:30	Unused	This field is not writable and will read all 0’s	R	0x0000
31	FreqChange	Frequency Change Pending Indicates whether a Frequency change operation is pending. 0 Not pending 1 Pending	R/W	
Note: *These bits clear when read.				

12.9.12 API Exception Mask 0 Register (APIMask0)

These bits enable the Exceptions in APIEXCP. Any enabled exception bit causes an API exception, which causes a checkstop (pin PI_CSTP).

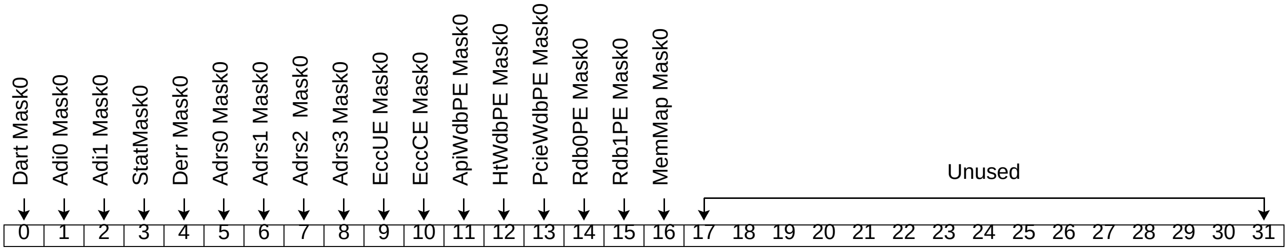
Encoding is:

```
0: disabled
1: enabled
```

Reset Value 0x00000000

Offset 0xF80300B0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	Dart Mask0	Enables DART Translation Exception	R/W	0b0
1	Adi0 Mask0	Enables ADI0 Exception	R/W	0b0
2	Adi1 Mask0	Enables ADI1 Exception	R/W	0b0
3	StatMask0	Enables Snoop Status Exception	R/W	0b0
4	Derr Mask0	Enables Data Error Exception	R/W	0b0
5	Adrs0 Mask0	Enables Addressing Exception from Processor 0	R/W	0b0
6	Adrs1 Mask0	Enables Addressing Exception from Processor 1	R/W	0b0

Bits	Field Name	Description	Access	Reset
7	Adrs2 Mask0	Enables Addressing Exception from Processor 2	R/W	0b0
8	Adrs3 Mask0	Enables Addressing Exception from Processor 3	R/W	0b0
9	EccUE Mask0	Enables ECC Uncorrectable Error Exception	R/W	0b0
10	EccCE Mask0	Enables ECC Correctable Error Exception	R/W	0b0
11	ApiWdbPE Mask0	Enables PI WDB Parity Error Exception	R/W	0b0
12	HtWdbPE Mask0	Enables HT WDB Parity Error Exception	R/W	0b0
13	PcieWdbPE Mask0	Enables PCIe WDB Parity Error Exception	R/W	0b0
14	Rdb0PE Mask0	Enables RDB0 Parity Error Exception	R/W	0b0
15	Rdb1PE Mask0	Enables RDB1 Parity Error Exception	R/W	0b0
16	MemMap Mask0	Enables Memory Mapping Exception	R/W	0b0
17:31	Unused	This field is not writable and will read all 0’s	R	0x0000

12.9.13 API Exception Mask 1 Register (APIMask1)

These bits enable the exceptions in APIEXCP. Any enabled exception bit causes a ChipFault (pin CHP_FAULT_N).

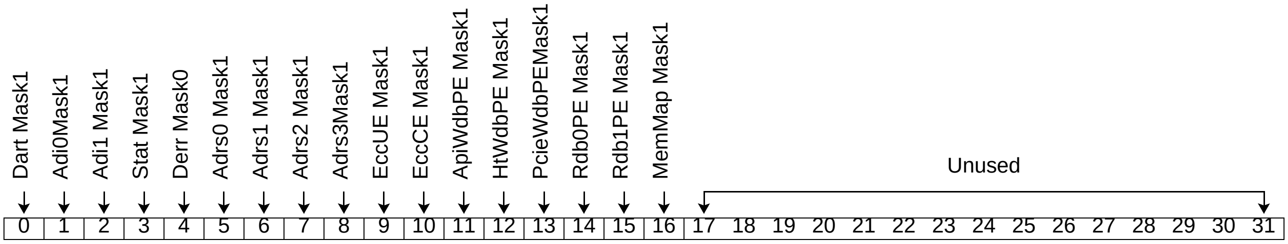
Encoding is:

```
0: disabled
1: enabled
```

Reset Value 0x00000000

Offset 0xF80300C0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	Dart Mask1	Enables DART Translation Exception	R/W	0b0
1	Adi0Mask1	Enables ADI0 Exception	R/W	0b0
2	Adi1 Mask1	Enables ADI1 Exception	R/W	0b0
3	Stat Mask1	Enables Snoop Status Exception	R/W	0b0
4	Derr Mask0	Enables Data Error Exception	R/W	0b0
5	Adrs0 Mask1	Enables Addressing Exception from Processor 0	R/W	0b0
6	Adrs1 Mask1	Enables Addressing Exception from Processor 1	R/W	0b0
7	Adrs2 Mask1	Enables Addressing Exception from Processor 2	R/W	0b0
8	Adrs3Mask1	Enables Addressing Exception from Processor 3	R/W	0b0
9	EccUE Mask1	Enables ECC Uncorrectable Error Exception.	R/W	0b0
10	EccCE Mask1	Enables ECC Correctable Error Exception.	R/W	0b0
11	ApiWdbPE Mask1	Enables PI WDB Parity Error Exception.	R/W	0b0
12	HtWdbPE Mask1	Enables HT WDB Parity Error Exception.	R/W	0b0
13	PcieWdbPEMask1	Enables PCIe WDB Parity Error Exception.	R/W	0b0
14	Rdb0PE Mask1	Enables RDB0 Parity Error Exception.	R/W	0b0
15	Rdb1PE Mask1	Enables RDB1 Parity Error Exception.	R/W	0b0
16	MemMap Mask1	Enables Memory Mapping Exception.	R/W	0b0

Bits	Field Name	Description	Access	Reset
17:31	Unused	This field is not writable and will read all 0's.	R	0x0000

12.9.14 API Target Request Queues Guarantees Register (APITRqGuar)

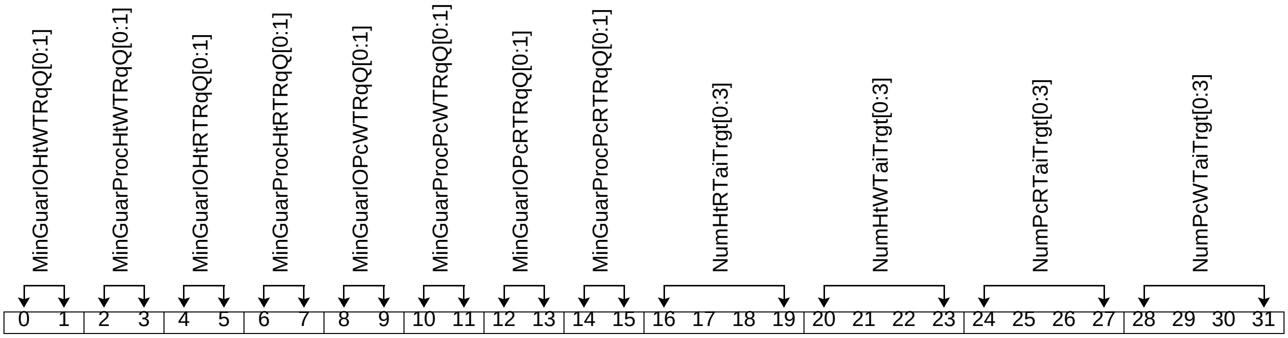
These values are used to configure the parameters associated with nonmemory target request queue guarantee values.

If any of the "Guar" values are changed from their default value, then SafeQCnt needs to be disabled. (See *Section 12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)* on page 429.)

Reset Value 0x55554444

Offset 0xF80300D0

Access Type Read/Write, Read Only



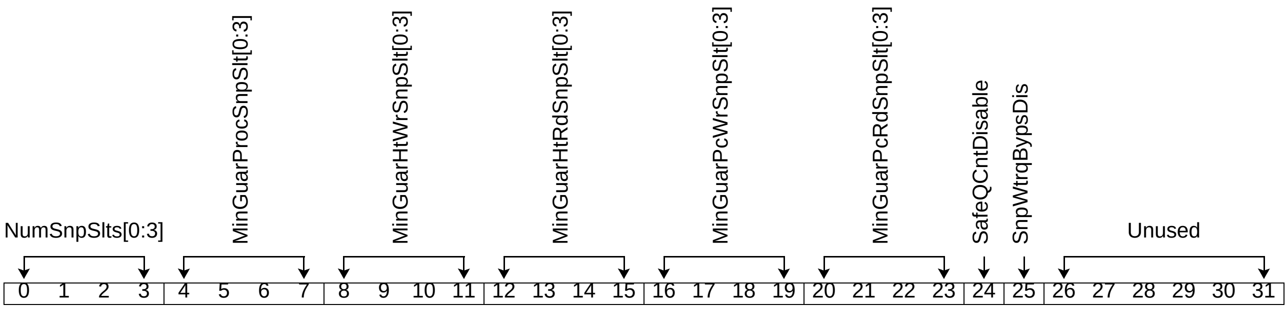
Bits	Field Name	Description	Access	Reset
0:1	MinGuar IOHtWTRqQ [0:1]	Minimum guarantee of available spaces for I/O writes to the HT Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizeHtWTRqQ - MinGuarProcHtWTRqQ	R/W	0b01
2:3	MinGuar ProcHtWTRqQ [0:1]	Minimum guarantee of available spaces for Proc writes to the HT Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizeHtWTRqQ - MinGuarIOHtWTRqQ	R/W	0b01
4:5	MinGuar IOHtRTRqQ [0:1]	Minimum guarantee of available spaces for I/O reads to the HT Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizeHtRTRqQ - MinGuarProcHtRTRqQ	R/W	0b01
6:7	MinGuar ProcHtRTRqQ [0:1]	Minimum guarantee of available spaces for Proc reads to the HT Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizeHtRTRqQ - MinGuarIOHtRTRqQ	R/W	0b01
8:9	MinGuar IOPcWTRqQ [0:1]	Minimum guarantee of available spaces for I/O writes to the PCIe Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizePcWTRqQ - MinGuarProcPcWTRqQ	R/W	0b01

Bits	Field Name	Description	Access	Reset
10:11	MinGuar ProcPcWTRqQ [0:1]	Minimum guarantee of available spaces for Proc writes to the PCIe Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizePcWTRqQ – MinGuarIOPcWTRqQ	R/W	0b01
12:13	MinGuar IOPcRTRqQ [0:1]	Minimum guarantee of available spaces for I/O reads to the PCIe Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizePcRTRqQ – MinGuarProcPcRTRqQ	R/W	0b01
14:15	MinGuar ProcPcRTRqQ [0:1]	Minimum guarantee of available spaces for Proc reads to the PCIe Target Request Queue. Minimum: 1, Maximum: 3, Default:1 Must be less than or equal to SizePcRTRqQ - MinGuarIOPcRTRqQ	R/W	0b01
16:19	NumHtR TaiTrgt [0:3]	The number of entries in the HT Read TAI Target Queue in the HT Unit. Minimum: 1, Maximum: 4, Default:4	R/W	0b0100
20:23	NumHtW TaiTrgt [0:3]	The number of entries in the HT Write TAI Target Queue in the HT Unit. Minimum: 1, Maximum: 4, Default:4	R/W	0b0100
24:27	NumPcR TaiTrgt [0:3]	The number of entries in the PCIe Read TAI Target Queue in the PCIe Unit. Minimum: 1, Maximum: 4, Default:4	R/W	0b0100
28:31	NumPcW TaiTrgt [0:3]	The number of entries in the PCIe Write TAI Target Queue in the PCIe Unit. Minimum: 1, Maximum: 4, Default: 4	R/W	0b0100

12.9.15 API Snoop Slot Configuration Register (APISnpSltCfg)

These values are used to configure the parameters associated with configuring the snoop slots.

Reset Value	0xC4111100
Offset	0xF80300E0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:3	NumSnpSlts [0:3]	Number of available Snoop Slots. Minimum: 6, Maximum: 12, Default: 12 Must be greater than or equal to the sum of MinGuarProcSnpSlt, MinGuarHtWrSnpSlt, MinGuarHtRdSnpSlt, MinGuarPcWrSnpSlt, and MinGuarPcRdSnpSlt.	R/W	0b1100
4:7	MinGuar ProcSnpSlt [0:3]	Minimum guarantee of Snoop Slots for Proc Ops. Minimum: 2, Maximum: 8, Default: 4 See NumSnpSlts constraints.	R/W	0b0100
8:11	MinGuar HtWrSnpSlt [0:3]	Minimum guarantee of Snoop Slots for HT Writes. Minimum: 1, Maximum: 8, Default: 1 See NumSnpSlts constraints.	R/W	0b0001
12:15	MinGuar HtRdSnpSlt [0:3]	Minimum guarantee of Snoop Slots for HT Reads. Minimum: 1, Maximum: 8, Default: 1 See NumSnpSlts constraints.	R/W	0b0001
16:19	MinGuar PcWrSnpSlt [0:3]	Minimum guarantee of Snoop Slots for Pcie Writes. Minimum: 1, Maximum: 8, Default: 1 See NumSnpSlts constraints.	R/W	0b0001
20:23	MinGuar PcRdSnpSlt [0:3]	Minimum guarantee of Snoop Slots for Pcie Reads. Minimum: 1, Maximum: 8, Default: 1 See NumSnpSlts constraints.	R/W	0b0001
24	SafeQCnt Disable	Safe Queue Count Disable Disables the feature of reflecting a command quickly without checking resources because all of the resources are at a “safe” level. If any of the “Size”, “Number”, or “Guar” values of the PI queues and buffers change from their default value, then SafeQCnt must be disabled. 0 Enabled 1 Disabled	R/W	0b0

Bits	Field Name	Description	Access	Reset
25	SnpWtrq BypsDis	Snoop Wtrq Bypass Disable Disables the feature of bypassing the Waiting-to-Reflect-queue when the snoop slots are empty. 0: enabled 1: disabled	R/W	0b0
26:31	Unused	This field is not writable and will read all 0’s	R	0x00

12.9.16 API Bus Configuration Registers

These fields specify the API Bus parameters.

Note: An additional constraint for the hardware is that the PAAMWIN value must be greater than the SNOOPLAT value. Normally the PAAMWIN value is set to SNOOPLAT + SNOOPACCC + 6.

12.9.16.1 PI PAAM Window (APIPaamWin)

PAAMWIN specifies the number of beats (PI bus clocks) that have to occur between reflecting successive commands that target the same 128-byte block address.

Every time a command is reflected, a window (PAAM Window) is generated that is PAAMWIN cycles wide. There can many PAAM Windows active at various stages of time.

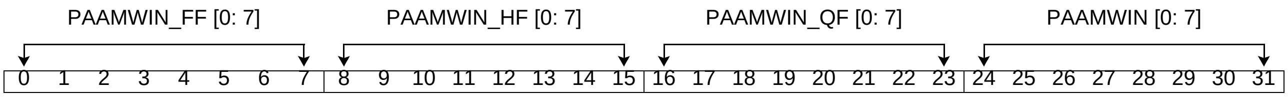
When a new command enters the Snoop Slots, a qualification is made whether this new command should check the PAAM Windows. If this command is from an HT or PCIe source and its destination is HT or PCIe, or if it is a power tuning command, then the check is not made. All other commands check the PAAM Windows.

If the block address of the new command matches the address of the PAAM Window command, or the P Bit of the new command is 0 and the P bit of the PAAM Window command is 0, then the new command waits until the PAAM Window has completed.

Reset Value 0x12121212

Offset 0xF8030100

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	PAAMWIN_FF [0: 7]	PAAM Window for full frequency. Minimum: 10, Maximum: 52, Initial: 18.	R/W	0x12
8:15	PAAMWIN_HF [0: 7]	PAAM Window for Half frequency. Minimum: 10, Maximum: 52, Initial: 18.	R/W	0x12
16:23	PAAMWIN_QF [0: 7]	PAAM Window for Quarter frequency. Minimum: 10, Maximum: 52, Initial: 18.	R/W	0x12

Bits	Field Name	Description	Access	Reset
24:31	PAAMWIN [0: 7]	Current PAAMWIN Read only value of the current PAAMWIN selected by CurFreqSel.	R	0x12

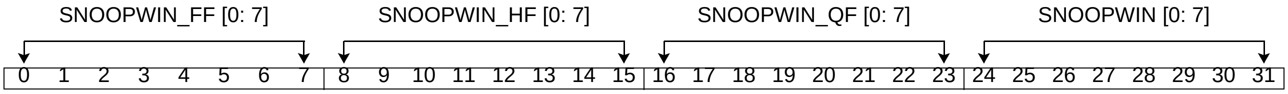
12.9.16.2 API Snoop Window (APISnoopWin)

SNOOPWIN specifies the number of beats (PI bus clocks) that have to occur between reflecting any successive commands. The commands might be under additional constraints as specified by PAAMWIN and IOSNOOPWIN.

Reset Value 0x04040404

Offset 0xF8030110

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	SNOOPWIN_FF [0: 7]	Snoop Window for Full frequency. Minimum: 4, Max 30, Initial: 4.	R/W	0x04
8:15	SNOOPWIN_HF [0: 7]	Snoop Window for Half frequency. Minimum: 4, Max 30, Initial: 4.	R/W	0x04
16:23	SNOOPWIN_QF [0: 7]	Snoop Window for Quarter frequency. Minimum: 4, Max 30, Initial: 4.	R/W	0x04
24:31	SNOOPWIN [0: 7]	Current SNOOPWIN Read only value of the current SNOOPWIN selected by CurFreqSel.	R	0x04

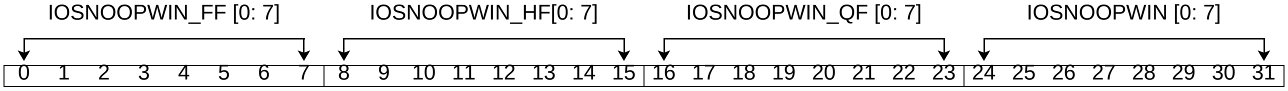
12.9.16.3 I/O Snoop Window (APIIOSnoopWin)

The IOSNOOPWIN specifies a constraint (in addition to PAAMWIN and SNOOPWIN) on the number of beats (PI bus clocks) that have to occur between reflecting HT and PCIe originated commands. IOSNOOPWIN beats have to occur since the last HT command before reflecting the next HT Command and since the last PCIe command before reflecting the next PCIe command.

Reset Value 0x08080808

Offset 0xF8030120

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	IOSNOOPWIN_FF [0: 7]	I/O Snoop Window for Full frequency. Minimum: 8, Max 30, Initial: 8.	R/W	0x08
8:15	IOSNOOPWIN_HF[0: 7]	I/O Snoop Window for Half frequency. Minimum: 8, Max 30, Initial: 8.	R/W	0x08
16:23	IOSNOOPWIN_QF [0: 7]	I/O Snoop Window for Quarter frequency. Minimum: 8, Max 30, Initial: 8.	R/W	0x08
24:31	IOSNOOPWIN [0: 7]	Current IOSNOOPWIN Read only value of the current IOSNOOPWIN selected by CurFreqSel.	R	0x08

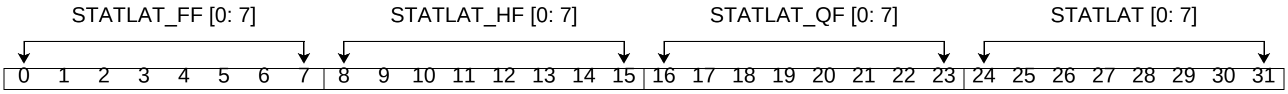
12.9.16.4 API Handshake Status Latency (APIStatLat)

STATLAT specifies the number of beats (PI bus clocks) that occur between reflecting a command or returning read data and receiving the Handshake Status.

Reset Value 0x14141414

Offset 0xF8030130

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	STATLAT_FF [0: 7]	Handshake Status Latency for Full frequency. Minimum: 14, Max 40, Initial: 20.	R/W	0x14
8:15	STATLAT_HF [0: 7]	Handshake Status Latency for Half frequency. Minimum: 14, Max 40, Initial: 20.	R/W	0x14
16:23	STATLAT_QF [0: 7]	Handshake Status Latency for Quarter frequency. Minimum: 14, Max 40, Initial: 20.	R/W	0x14
24:31	STATLAT [0: 7]	Current STATLAT Read only value of the current STATLAT selected by CurFreqSel.	R	0x14

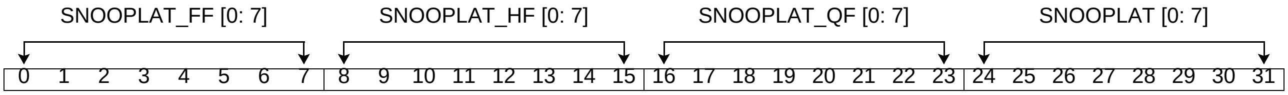
12.9.16.5 API Snoop Latency Values (APISnoopLat)

SNOOPLAT specifies the number of beats (PI bus clocks) that occur between reflecting a command and receiving the Snoop Status.

Reset Value 0x0C0C0C0C

Offset 0xF8030140

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	SNOOPLAT_FF [0: 7]	Snoop Status Latency for Full frequency. Minimum: 8, Max 40, Initial: 12.	R/W	0x0b
8:15	SNOOPLAT_HF [0: 7]	Snoop Status Latency for Half frequency. Minimum: 8, Max 40, Initial: 12.	R/W	0x0b
16:23	SNOOPLAT_QF [0: 7]	Snoop Status Latency for Quarter frequency. Minimum: 8, Max 40, Initial: 12.	R/W	0x0b
24:31	SNOOPLAT [0: 7]	Current SNOOPLAT. Read only value of the current SNOOPLAT selected by CurFreqSel.	R	0x0b

12.9.17 PSRO Register (PSRO)

Reset Value x‘00000000’

Address x‘F80301C0’

Access Type Read

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Reserved																PSRO									Reserved						

Bits	Field Name	Description	Access	Reset
0:15	Reserved		Read	0x0000
16:24	PSRO	Nine-bit hexadecimal value representing PRSO. For additional information, see the *CPC945 Datasheet.*	Read	0b000000000
25:31	Reserved		Read	0b0000000

12.9.18 PI System Command Registers

The PI System Command Registers provide a mechanism for accessing any memory-mapped location in the system through the SBus Interface. A request is written to the SysCmdCntl0 and SysCmdCntl1 registers. These registers have similar attributes to PI Bus requests. Writing to SysCmdCntl0 initiates the request.

The request participates in the Snoop reflection protocol on the PI bus and returns an Accumulated Snoop Status in the SysCmdStat register. If the status indicates a retry, the request will be reissued the number of times specified in the NumRetry field in that register. The number of times the request was retried is indicated in the RetryCnt field. If the request comes back with a nonretry status or the RetryCnt exceeds the NumRetry value then the StatDone bit is set. A NumRetry value of zero indicates that the request should be reissued until nonretry status is received (RetryCnt is ignored). The RetryCnt will stay at its maximum value if it is reached.

On a write operation the data that is placed in the SysCmdData registers is written to the requested destination when the status indicated is nonretry. When the nonretry status is seen, StatDone is set and the write request is sent. When the data operation has completed the DataDone bit is set. If the RetryCnt exceeds the NumRetry value, the StatDone is set and DataAbort is set indicating that the operation has completed without the data being written.

On a read operation the StatDone bit is set if the RetryCnt exceeds the NumRetry value, the status returned is Intervention, or a valid (Null or Shared) response is received. Only with a Null or Shared response does the read request continue to its destination. When the data is returned from the Read request, it is placed in SysData registers and the DataDone bit is set completing the operation. If the Read request is not sent, and a StatDone condition occurs, the DataAbort bit is set indicating the operation has completed.

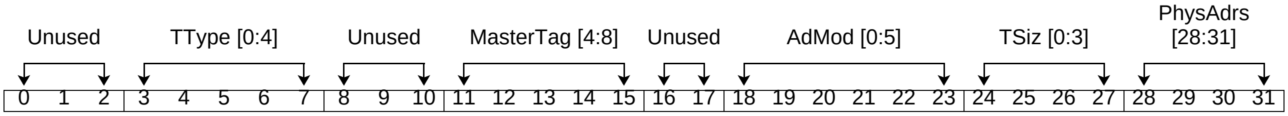
12.9.18.1 System Command Control [0:1] Registers (SysCmdCntl[0:1])

Reset Value 0x0E0F0040 (SysCmdCntl0)
0xF8030210 (SysCmdCntl1)

Address 0xF8030200 (SysCmdCntl0)
0xF8030210 (SysCmdCntl1)

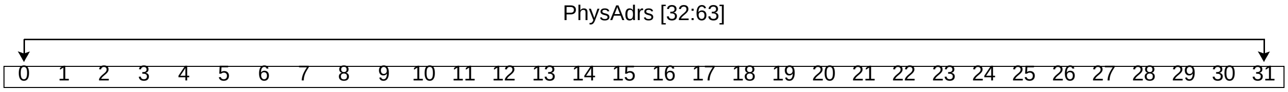
Access Type Read/Write, Read Only

SysCmdCntl0



Bits	Field Name	Description	Access	Reset
0:2	Unused	This field is not writable and will read all 0’s.	R	0b000
3:7	TType [0:4]	Transfer Type.	R/W	0b00000
8:10	Unused	This field is not writable and will read all 0’s.	R	0b000
11:15	MasterTag [4:8]	Master Tag. 0xF is used for Master Tag bits [0:3].	R/W	0b00000
16:17	Unused	This field is not writable and will read all 0’s.	R	0b00
18:23	AdMod [0:5]	Address Modifiers.	R/W	0b000000
24:27	TSiz [0:3]	Transfer Size. Only sizes of 16 bytes or less are allowed.	R/W	0b1001
28:31	PhysAdrs [28:31]	Physical Address. Spans registers 0xF8030200 and 0xF8030210.	R/W	0x0_0000_0000

SysCmdCntl1



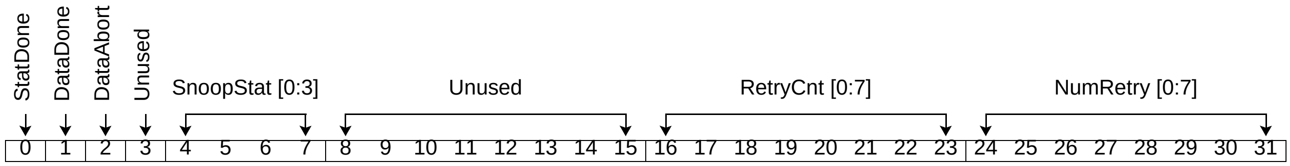
Bits	Field Name	Description	Access	Reset
0:31	PhysAdrs [32:63]	Physical Address Spans registers 0xF8030200 and 0xF8030210	R/W	0x0_0000_0000

12.9.18.2 System Command Status Register (SysCmdStat)

Reset Value 0xC0000000

Address 0xF8030220

Access Type Read/Write, Read Only



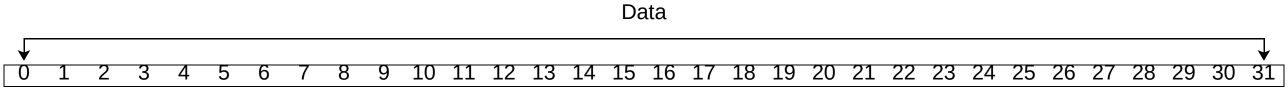
Bits	Field Name	Description	Access	Reset
0	StatDone	Status Done Nonretry has been received and is indicated in the SnoopStat field or the RetryCnt has reached the NumRetry value.	R/W	0b0
1	DataDone	Data Done Data has been written for a write request or data has returned on a read request.	R/W	0b0
2	DataAbort	Data Abort The operation has completed without data being sent or received due to a StatDone condition.	R/W	0b0
3	Unused	This field is not writable and will read all 0’s	R	0b0
4:7	SnoopStat [0:3]	Snoop Status Snoop status when StatDone condition has occurred.	R/W	0b0000
8:15	Unused	This field is not writable and will read all 0’s	R	0b0
16:23	RetryCnt [0:7]	Current retry count. Incremented when a status of retry is returned. When this value exceeds the NumRetry value, the operation completes.	R/W	0x00
24:31	NumRetry [0:7]	Number of Retries Limit Indicates how many times the request should be retried before giving up and setting StatDone.	R/W	0x00

12.9.18.3 System Command Data0 Register (SysCmdDt0)

Reset Value N/A

Address 0xF8030240

Access Type



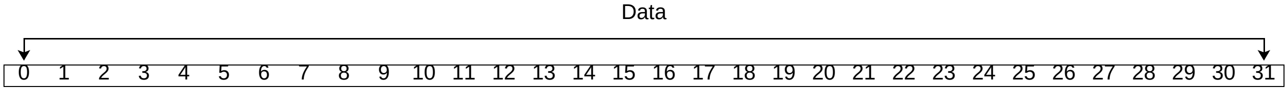
Bits	Field Name	Description	Access	Reset
0:31	Data			

12.9.18.4 System Command Data1 Register (SysCmdDt1)

Reset Value N/A

Address 0xF8030250

Access Type



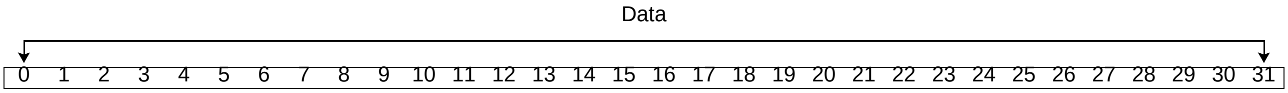
Bits	Field Name	Description	Access	Reset
0:31	Data			

12.9.18.5 System Command Data2 Register (SysCmdDt2)

Reset Value N/A

Address x‘F8030260’

Access Type



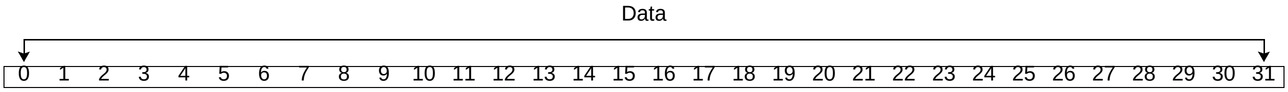
Bits	Field Name	Description	Access	Reset
0:31	Data			

12.9.18.6 System Command Data3 Register (SysCmdDt3)

Reset Value N/A

Address x‘F8030270’

Access Type



Bits	Field Name	Description	Access	Reset
0:31	Data			

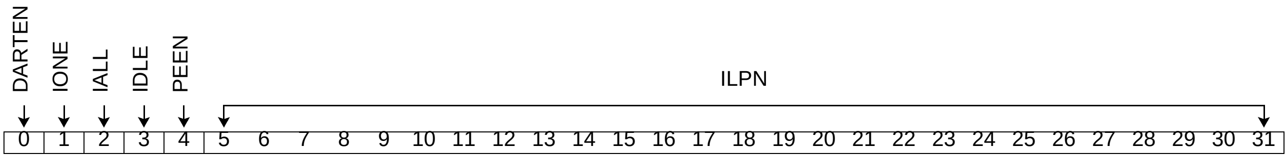
12.9.19 DART Control Register (DARTCNTL)

The DART Control Register is a 32-bit memory-mapped location that enables the DART for translation and controls the invalidating of entries in the DART TLB. The DARTEN bit enables the DART for mapping. If this bit is zero no mapping occurs. The DART TLB entries can be invalidated individually by the IONE bit or entirely by the IALL bit. For individual invalidates the ILPN field specifies the Logical Page Address (27 bits, [25:51]) associated with the DART TLB entry that is to be invalidated. Writing a one to the IALL bit invalidates the entire DART TLB. This bit returns to zero when the invalidation is complete. Writing a one to the IONE bit invalidates the one entry that matches (hits) with the ILPN. This bit returns to zero when the invalidation is complete. The IDLE bit indicates that the DART is not currently processing a miss or a read or write request. The PEEn bit enables DART RAM parity checking.

Reset Value 0x08000000

Address 0xF8033000

Access Type Read/Write, Read Only

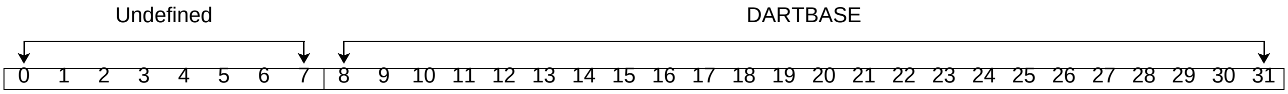


Bits	Field Name	Description	Access	Reset
0	DARTEN	DART Enable (0:disabled)	R/W	0b0
1	IONE	Invalidate one DART TLB entry (using ILPN)	R/W	0b0
2	IALL	Invalidate all DART TLB entries	R/W	0b0
3	IDLE	DART is idle.	R	0b0
4	PEEN	Parity checking is enabled.	R/W	0b1
5:31	ILPN	27-bit logical page address for invalidating one TLB entry.	R/W	undefined

12.9.20 DART Base Register (DARTBASE)

The DART is located in Memory at a 4K-byte-aligned address. The 24-bit DART Base Register (DARTBASE) defines the location of the DART in Memory (24-bit address concatenated with 12 bits of zeroes).

Reset Value	0x00000000
Address	0xF8033010
Access Type	Read/Write

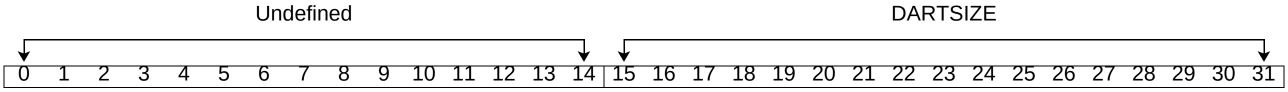


Bits	Field Name	Description	Access	Reset
0:7	Undefined	This field is not writable and will read all 0’s	R/W	0x00
8:31	DARTBASE	Base Address of DART (4K–byte Alignment)	R/W	undefined

12.9.21 DART Size Register (DARTSIZE)

The 17-bit DART Size Register (DARTSIZE) defines how big the DART is in 4K-byte pages. A value of zero indicates the maximum size (128K pages).

Reset Value	0x00000000
Address	0xF8033020
Access Type	Read/Write



Bits	Field Name	Description	Access	Reset
0:14	Undefined	This field is not writable and will read all 0’s	R/W	0x0000
15:31	DARTSIZE	Size of DART in 4K-Byte Pages (Max 128K pages, all zeroes). 27-bit LPN determines 128M pages to be mapped. Each page of the DART maps 1024 pages so the maximum size of DART is 128K pages to map 128M pages.	R/W	undefined

12.9.22 DART Exception Status Register (DARTEXCP)

The DARTEXCP register is used when an exception occurs and contains information pertaining to the address that caused the exception. Addresses that access outside of the DART Table cause an Out-Bounds-Exception. Addresses that access a DART Table entry that have a "Valid" bit of zero cause an Entry Exception. Addresses from Read operations that access a Valid DART Table entry that have a Read bit of one cause a Read Protection Exception. Addresses from Write operations that access a Valid DART Table entry that have a Write bit of one cause a Write Protection Exception. When mapping is disallowed, an access beyond the range of physical memory causes an Address Exception (XAD). See Memory Mapping Table above. When an XAD occurs the LPN contains bits 25:51 of the address causing the exception. A Parity Error in the DART TLB causes an XPE. A DART TLB Parity Error only occurs when no other exception occurs and the TLB Data Array is accessed.

There are 6 different kinds of DART exception:

1. XWE - DART Write Protection Exception
   - when a write request hits a page that is write protected, this exception flag will be set.
   - the write request will continue as normal to DDR2 interface but the actual "write" to memory will be disabled.
   - DDR2 will return with a TgEV (instead of TgV) to complete this write request.
   - the exception address and the agent that causes the exception will be logged.
   - the exception flag will be clear with a read to the DART Exception Register.
2. XRE - DART Read Protection Exception
   - when a read request hits a page that is read protected, this exception flag will be set.
   - the read request will continue as normal to DDR2 interface but the actual "read" to memory will be disabled.
   - DDR2 will return with a TgEV (instead of TgV) to complete this read request.
   - the exception address and the agent that causes the exception will be logged.
   - the exception flag will be clear with a read to the DART Exception Register.
3. XEE - DART Entry Exception
   - when a read/write request hits a page that is not VALID in the DART memory, this exception flag will be set.
   - the corresponding DART TLB entry will be forced to VALID in order for this request to complete.
   - the request will continue as normal to DDR2 interface but the actual "request" to memory will be disabled.
   - DDR2 will return with a TgEV (instead of TgV) to complete this request.
   - the exception address and the agent that causes the exception will be logged.
   - the exception flag will be clear with a read to the DART Exception Register.
4. XPE - DART TLB Parity Error
   - when a DART TLB read that detected a parity error, this exception flag will be set.
   - the request will continue as normal to DDR2 interface but the actual "request" to memory will be disabled.
   - DDR2 will return with a TgEV (instead of TgV) to complete this request.
   - the exception address and the agent that causes the exception will be logged.
   - the exception flag will be clear with a read to the DART Exception Register.
5. XBE - DART Out-of-Bounds Exception (FATAL)
   - when a request hits a page that is outside the DART Table, this exception flag will be set.
   - the "DART Miss" request will still be sent and whatever data returned by the system, will be used for translation and protection check.
   - since unpredictable data is returned, this exception is FATAL to the system.

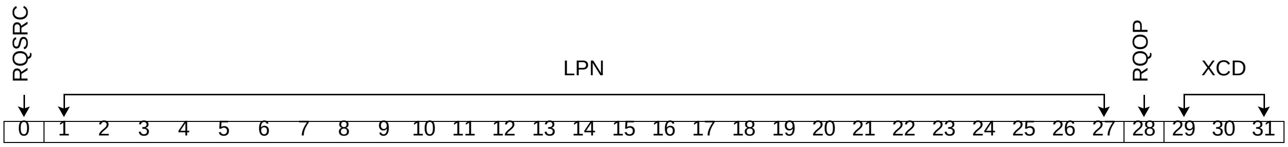
- the exception address and the agent that causes the exception will be logged.
- the exception flag will be clear with a read to the DART Exception Register.

6. XAE - DART Addressing Exception (FATAL)
   - when a request hits a page that is beyond the range of physical memory, this exception flag will be set.
   - the “DART Miss” request will still be sent and whatever data returned by the system, will be used for translation and protection check.
   - since unpredictable data is returned, this exception is FATAL to the system.
   - the exception address and the agent that causes the exception will be logged.
   - the exception flag will be clear with a read to the DART Exception Register.

Reset Value 0xFFE77DDF

Address 0xF8033030

Access Type Read Only



Bits	Field Name	Description	Access	Reset
0	RQSRC	Request Source. 0 PCIe 1 HT	R	undefined
1:27	LPN	27-bit Logical Address of Exception [25:51]	R	undefined
28	RQOP	Requesting Operation 0 Read 1 Write	R	undefined
29:31	XCD	3-bit DART Exception Code (Initial value undefined): 000 **XBE** DART Out-of-Bounds Exception 001 **XEE** DART Entry Exception 010 **XRE** DART Read Protection Exception 011 **XWE** DART Write Protection Exception 100 **XAD** Addressing Exception 101 **XPE** DART TLB Parity Error 110 Undefined 111 Undefined	R	undefined

12.9.23 Entry in DART TLB Tag Array Register (DARTTAG)

The DART TLB can be accessed by direct read and write operations and by an invalidate operation. The following shows the system address needed for read and write operations to the Tag Array.

Bits 13:19 of the 32 bit DMA address generated by HyperTransport or PCI0 are used to address the DART TLB Tag Array Register. An address for the correct Tag Array Address Register is formed by eight bits [20:27] arranged as:

```
wwtttttt, where
"ww" represents the way (correct set of a set-associative TLB), and
"tttttt" is the set of the desired entry.
```

The desired Tag Array Register is accessed using the address: 0x F 8 0 3 4 wwtt tttt 0000.

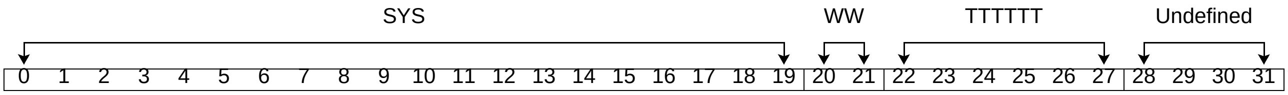
See *Section 3.4 DMA Address Relocation Table (DART)* on page 53 for details.

Note: The eight bits used to determine the correct Tag Array Address Register shown above are for a 32 bit address space. For a programmer using the PPC970xx, a 64-bit address space is desired. The correct bit locations can be found by adding 32 to produce [53:59].

Reset Value

Address 0xF8034000-0xF8034FF0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:19	SYS	System defined address	R/W	0x00000
20:21	WW	2-bit tag way number	R/W	0b00
22:27	TTTTTT	6-bit tag index format of TLB tag address	R/W	0b000000
28:31	Undefined	This field will read all 0’s	R	0x0000

12.9.24 Entry in DART TLB Data Array Register (DARTDATA)

The DART TLB can be accessed by direct read and write operations and by an invalidate operation. The following shows the system address needed for read and write operations to the Data Array.

Bits 11:19 of the 32-bit DMA address generated by HyperTransport or PCI0 are used to address the DART TLB Data Array Register. A Data Array Address is formed by 9 bits [19:27]. The desired Data Array Register is accessed using the address:

```
0x F 8 0 3 10ww qqtt tttt 0000
  "tttttt" is the set of the desired entry.
  "ww" represents the way within the set (correct set of a set-associative
  TLB), and
  "qq" is the page.
```

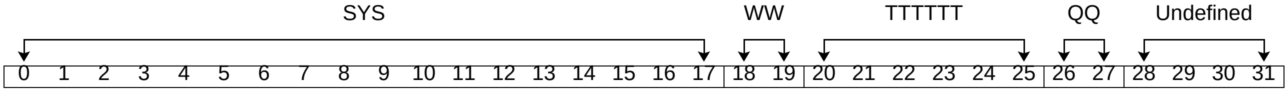
See *Section 3.4 DMA Address Relocation Table (DART)* on page 53 for details.

Note: The seven bits used to determine the correct Tag Array Address Register shown above are for a 32-bit address space. For a programmer using the PPC970xx, a 64-bit address space is desired. The correct bit locations can be found by adding 32 to produce [51:59].

Reset Value

Address xF8038000-xF803BFF0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:17	SYS	System defined address	R/W	0x00000
18:19	WW	2-bit tag way number	R/W	0b00
20:25	TTTTTT	6-bit tag index format of TLB tag address	R/W	0b000000
26:27	QQ	2-bit entry index within quadword	R/W	0b00
28:31	Undefined	This field will read all 0’s	R	0b0000

12.10 Memory Control Registers

The memory controller block contains the following registers to program the memory timing parameters, physical bank address boundaries, memory types, refresh rate, interface configuration, and other controls. The following table lists all the memory controller register addresses. Functional details of each register follow.

Table 12-17. Memory Controller Register Address Space

System Bus Address	Register Name	Description	Page
0XF8002030	RASTimer0	RAS Command Timer0 Register	449
0XF8002040	RASTimer1	RAS Command Timer1 Register	449
0XF8002050	CASTimer0	CAS Command Timer0 Register	449
0XF8002060	CASTimer1	CAS Command Timer1 Register	449
0XF8002070	MemRfshCntl	Memory Refresh Control Register (refresh period)	454
0XF80020B0	MemProgCntl	Programming of the DDR2 SDRAM memory	456
0XF80020C0	MRSRegCntl	Settings in the Mode Register on the SDRAMs	457
0XF80020D0	EMRSRegCntl	Settings in the Extended Mode Register on the SDRAMs	457
0XF80020E0- 0XF80020F0	MemMapExcp[0:1]	Contains info when an addressing exception occurs during memory mapping	460
0XF8002100- 0XF80021F0	MemInitReg[0:15]	Memory Initialization Registers [0:15]	462
0XF8002200- 0XF8002230	DmCnfg[0:3]	DIMM [0:3] Configuration Registers	463
0XF8002280	MemArbWt	Specifies weights in the memory arbiter	464
0XF8002290	UsrCnfg	User Configuration: CS mode, bank mode	465
0XF80022A0	MemRdQCnfg	Memory read request queue configuration	467
0XF8002270	MemWrQCnfg	Memory write request queue configuration	468
0XF80022B0	MemQArb	Memory reorderQ arbitration configuration	470
0XF80022C0	MemRWArb	Memory R/W arbitration configuration	471
0XF80022D0	MemBusConfig	Specifies external timing delays to read data	472
0XF80022E0	MemBusConfig2	Additional timing delays to read data	474
0XF80023A0	ODTCntl	ODT Control Register	475
0XF8002400	MSCR	Memory Scrub Control Register	476
0XF8002410	MSRSR	Memory Scrub Range Start Register	477
0XF8002420	MSRER	Memory Scrub Range End Register	478
0XF8002430	MSPR	Memory Scrub Pattern Register	478
0XF8002440	MCCR	Memory Check Control Register	479

Table 12-17. Memory Controller Register Address Space (Continued)

System Bus Address	Register Name	Description	Page
0XF8002460- 0XF8002470	MEAR[0:1]	Memory Error Address Register [0:1]	480
0XF8002480	MESR	Memory Error Syndrome Register	482
0XF8002500	MemModeCntl	Memory Mode Control Register	483
0xF8002880	MemPhyModeCntl	Memory PHY Mode Control	485
0xF80029A0	IOPadCntl	I/O Pad Control bits	487
0xF8002800- 0xF8002830	ByteWrClkDelayC0B00- ByteWrClkDelayC0B03	Write Strobe Control Registers	489
0xF8002900- 0xF8002930	ByteWrClkDelayC1B04- ByteWrClkDelayC1B07	Write Strobe Control Registers	489
0xF8002980	ByteWrClkDelayC1B16	Write Strobe Control Registers	489
0xF8002A00- 0xF8002A30	ByteWrClkDelayC2B08- ByteWrClkDelayC2B11	Write Strobe Control Registers	489
0xF8002B00- 0xF8002B30	ByteWrClkDelayC3B12- ByteWrClkDelayC3B15	Write Strobe Control Registers	489
0xF8002B80	ByteWrClkDelayC3B17	Write Strobe Control Registers	489
0xF8002840- 0xF8002870	ReadStrobeDelayC0B00- ReadStrobeDelayC0B03	Read Data Strobe Control Registers	491
0xF8002940- 0xF8002970	ReadStrobeDelayC1B04- ReadStrobeDelayC1B07	Read Data Strobe Control Registers	491
0xF8002990	ReadStrobeDelayC1B16	Read Data Strobe Control Registers	491
0xF8002A40- 0xF8002A70	ReadStrobeDelayC2B08- ReadStrobeDelayC2B11	Read Data Strobe Control Registers	491
0xF8002B40- 0xF8002B70	ReadStrobeDelayC3B12- ReadStrobeDelayC3B15	Read Data Strobe Control Registers	491
0xF8002B90	ReadStrobeDelayC3B17	Read Data Strobe Control Registers	491
0xF8002890	CKDelayL	CK Control Lower Register	493
0xF80028A0	CKDelayU	CK Control Upper Register	493
0xF80028D0	RstLdEnVerniersC0	ResetLdEn Vernier Control Register C0	494
0xF80029D0	RstLdEnVerniersC1	ResetLdEn Vernier Control Register C1	494
0xF8002AD0	RstLdEnVerniersC2	ResetLdEn Vernier Control Register C2	494
0xF8002BD0	RstLdEnVerniersC3	ResetLdEn Vernier Control Register C3	494
0xF80028B0	ExtMuxVernier0	ExtMux Vernier Control Register 0	494
0xF80028C0	ExtMuxVernier1	ExtMux Vernier Control Register 1	494
0xF80028F0	CalCntlDlyMeasC0	Calibration Control and Delay Measurement Register C0	496
0xF80029F0	CalCntlDlyMeasC1	Calibration Control and Delay Measurement Register C1	496

Table 12-17. Memory Controller Register Address Space (Continued)

System Bus Address	Register Name	Description	Page
0xF8002AF0	CalCntlDlyMeasC2	Calibration Control and Delay Measurement Register C2	496
0xF8002BF0	CalCntlDlyMeasC3	Calibration Control and Delay Measurement Register C3	496
0xF80029B0	CalConf0	Calibration Configuration 0 Register	499
0xF80029C0	CalConf1	Calibration Configuration 1 Register	499
0xF80028E0	CalRsltC0	Calibration Read Margin Result Register C0	502
0xF80029E0	CalRsltC1	Calibration Read Margin Result Register C1	502
0xF8002AE0	CalRsltC2	Calibration Read Margin Result Register C2	502
0xF8002BE0	CalRsltC3	Calibration Read Margin Result Register C3	502

12.10.1 Memory Timing Parameter Registers

The Memory Timing Parameter Registers contain values that are used in the Page Table Timers and Reorder Queue State Timers. There are 4 registers of Timer values. The RAS Command Timer [0:1] Registers (RASTimer0, RASTimer1) contain values associated with Activates and Precharges. The CAS Command Timer [0:1] Registers (CASTimer0, CASTimer1) contain values associated with read and writes, like data multiplexer switching time effects.

The values put into these registers are functions of the DDR2 JEDEC specification or vendor specification values.

- CL is the programmed CAS Latency of the memories; usually 3, 4 or 5.
- WL = Write Latency is defined by JEDEC as RL -1, where RL (Read Latency) is defined by JEDEC as AL + CL. Since AL = 0 for the CPC945 (*“EMRS Settings” on page 153*), WL = CL -1.
- BL = Burst Length = 4 for the 128-bit configuration and 8 for the 64-bit configurations.
- Values starting with “t” are found in the JEDEC or memory vendor specifications, usually in terms of ns. There values must be converted to tCK clock cycles. Example: 533MHz SDRAM has tRAS = 45ns. The CK going to memory is 266MHz which has a period tCK = 3.75ns. To satisfy the tRAS requirement at the memory chip, the controller must generate tRAS = 45ns <==> (45ns/3.75ns/clock) = 12 clocks. So “12” is the value for tRAS in the CPC945 RASTimer0 registers.
- For the CAS Timer0 and CAS Timer1 registers, the terms RRMux, WRMux, WWMux and RWMux provide time, in (in tCK clocks), for the external data bus to be “turned around” (to allow one device to stop driving in time before another device starts driving the bus). These values have a minimum value of 1 clock. For boards with external muxes, the value might need to be increased to account for the switching time of the multiplexer.
- The equations for the register settings take into account an internal 2 clock delay required in accessing the programmed value. For example, the minimum Activate to Precharge delay at the SDRAM is the JEDEC specified parameter tRAS. The equation for TiAtP, Activate to Precharge, in RASTimer0 gives a programming value of tRAS - 2, reflecting the fact that 2 additional cycles will be added to the programmed value.

For the above example of tRAS = 12 cycles the value programmed for TiAtP in RASTimer0 should be 10 (= 12 - 2).

- Many values are a function of the type of memory installed. If the installed memories support different timing values among themselves, the most conservative value must be used. Fox example, if one of the DRAMS has a slower tRCD then the slower value should be used. tRCD is one of several values that is different depending on the “speed grade” of the SDRAM. Note that some parameters can be a function of other attributes, such as the chip size (in bits or banks) or row size. For example, tRDD is one value for SDRAMs with 1K pages and a different value for SDRAMs with 2K pages. Again, for a mixture of values the most conservative value must be used.
- The memory controller generates timings on clock period boundaries; calculated values must therefore be rounded up to the nearest clock. tRDD (1K pages) is 7.5ns. For 533MHz DIMMs (tCK = 3.75ns) the exactly 2 clocks will meet this specification. But for 400 MHz DIMMs (tCK = 5ns), one clock is not enough (5ns), and 2 clocks generates more timing than needed (10ns). The rounded up value of 2 clocks must be used.
- Similarly, “RND” in the equation indicates “Round-up to nearest integer cycle count”.
- See *Section 7.15 Timing Parameters* on page 175 for additional discussion of these timing parameters, and *Section 7.15.3 Timing Parameter Examples* on page 176 for example calculations.

12.10.1.1 RAS Command Timer0 Register (RASTimer0)

Reset Value 0x72566380

Offset 0xF8002030

Access Type Read/Write, Read Only

TiAtP[0:4]	TiRtP[0:4]	TiWtP[0:4]	TiPtA[0:4]	TiPAtA[0:4]	Unused
0 1 2 3 4	5 6 7 8 9	10 11 12 13 14	15 16 17 18 19	20 21 22 23 24	25 26 27 28 29 30 31

Bits	Field Name	Description	Access	Reset
0:4	TiAtP[0:4]	Timer Init Activate to Precharge Time Calculation: tRAS -2	R/W	0b01110
5:9	TiRtP[0:4]	Timer Init Read to Precharge Time Calculation: (BL/2-2) + tRTP -2	R/W	0b01001
10:14	TiWtP[0:4]	Timer Init Write to Precharge Time Calculation: WL + BL/2 + tWR -2	R/W	0b01011
15:19	TiPtA[0:4]	Timer Init Precharge to Activate Time Calculation: tRP -2	R/W	0b00110
20:24	TiPAtA[0:4]	Timer Init Precharge All to Activate Time Calculation: tRP -2 or tRP - 1, if 8 Bank devices	R/W	0b00111
25:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.1.2 RAS Command Timer1 Register (RASTimer1)

Reset Value 0x39CAB120

Offset 0xF8002040

Access Type Read/Write, Read Only

TiRAPtA[0:4]					TiWAPtA[0:4]					TiAtARk[0:4]					TiAtABk[0:4]					TiAtRW[0:4]					TiAtARkWin[0:4]					Unused	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:4	TiRAPtA[0:4]	Timer Init Read Auto Precharge to Activate Time Calculation: (BL/2) – 2 + RND (tRTP + tRP) - 2	R/W	0b00111
5:9]	TiWAPtA[0:4]	Timer Init Write Auto Precharge to Activate Time Calculation: CL + (BL/2) – 1 + RND (tWR + tRP) - 2	R/W	0b00111
10:14	TiAtARk[0:4]	Timer Init Activate to Activate Time within Rank to different Bank. Calculation: tRRD - 2	R/W	0b00101
15:19	TiAtABk[0:4]	Timer Init Activate to Activate Time to same Bank. Calculation: tRC - 2	R/W	0b01011
20:24	TiAtRW[0:4]	Timer Init Activate to Read/Write Time. Calculation: tRCD - 2	R/W	0b00010
25:29	TiAtARkWin[0:4]	Timer Activate Window: Designates the window where 4 activates are allowed for 8 bank devices. Calculation: 4 * tRRD	R/W	0b01000
30:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.1.3 CAS Command Timer0 Register (CASTimer0)

Reset Value 0x00445084

Offset 0xF8002050

Access Type Read/Write, Read Only

TiRtRRk[0:4]	TiRtRDm[0:4]	TiRtRSy[0:4]	TiWtRRk[0:4]	TiWtRDm[0:4]	TiWtRSy[0:4]	Unused
0 1 2 3 4	5 6 7 8 9	10 11 12 13 14	15 16 17 18 19	20 21 22 23 24	25 26 27 28 29	30 31

Bits	Field Name	Description	Access	Reset
0:4	TiRtRRk[0:4]	Timer Init Read to Read Time within the same Rank. Encoding: BL/2 - 2	R/W	0b00000
5:9	TiRtRDm[0:4]	Timer Init Read to Read Time to the same DIMM, different Rank. Encoding: BL/2 - 1	R/W	0b00001
10:14	TiRtRSy[0:4]	Timer Init Read to Read Time to different DIMM. Encoding: BL/2 + RRMux - 2 RRMux is time to switch Data Multiplexer for Read-Read	R/W	0b00010
15:19	TiWtRRk[0:4]	Timer Init Write to Read Time within the same Rank. Encoding: (CL-1) + BL/2 + tWTR - 2	R/W	0b00101
20:24	TiWtRDm[0:4]	Timer Init Write to Read Time to the same DIMM, different Rank. Encoding: BL/2 - 1	R/W	0b00001
25:29	TiWtRSy[0:4]	Timer Init Write to Read Time to different DIMM. Encoding: BL/2 +WRMux - 2 WRMux is time to switch Data Multiplexer for Write-Read	R/W	0b00001
30:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.1.4 CAS Command Timer1 Register (CASTimer1)

Reset Value 0x0044210C

Offset 0xF8002060

Access Type Read/Write, Read Only

TiWtWRk[0:4]	TiWtWDm[0:4]	TiWtWSy[0:4]	TiRtWRk[0:4]	TiRtWDm[0:4]	TiRtWSy[0:4]	Unused
0 1 2 3 4	5 6 7 8 9	10 11 12 13 14	15 16 17 18 19	20 21 22 23 24	25 26 27 28 29	30 31

Bits	Field Name	Description	Access	Reset
0:4	TiWtWRk[0:4]	Timer Init Write to Write Time within the same Rank. Encoding: BL/2 - 2	R/W	0b00000
5:9	TiWtWDm[0:4]	Timer Init Write to Write Time to the same DIMM, different Rank. Encoding: BL/2 - 1	R/W	0b00001
10:14	TiWtWSy[0:4]	Timer Init Write to Write Time to different DIMM. Encoding: BL/2 + WWMux - 2 WWMux is time to switch Data Multiplexer for Write - Write	R/W	0b00010
15:19	TiRtWRk[0:4]	Timer Init Read to Write Time within the same Rank. Encoding: TRTW = BL/2	R/W	0b00010
20:24	TiRtWDm[0:4]	Timer Init Read to Write Time to the same DIMM, different Rank. Encoding: BL/2	R/W	0b00010
25:29	TiRtWSy[0:4]	Timer Init Read to Write Time to different DIMM. Encoding: BL/2 + RWMux - 1 RWMux is time to switch Data Multiplexer for Read - Write	R/W	0b00011
30:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.2 Memory Refresh Control Register (MemRfshCntl)

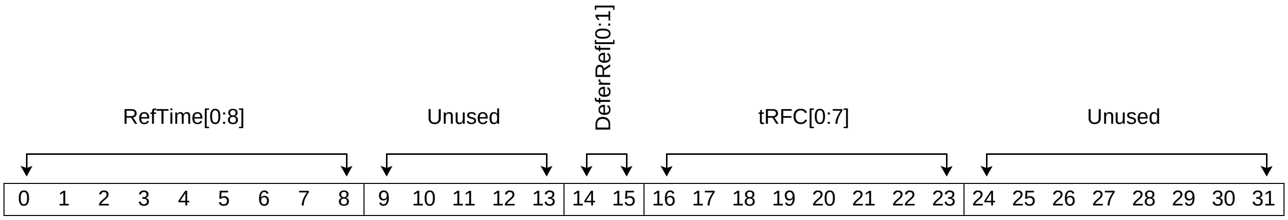
This register programs the refresh parameters. RefTime defines the frequency of refresh cycles. A 13-bit refresh counter continuously counts MemClk cycles. When the high order 9 bits of the refresh counter matches the RefTime value, a refresh operation is initiated and the refresh counter is reset to zero. This gives the ability to program the following Refresh Periods.

MemClk Frequency	MemClk Period	Refresh Period Minimum RefTime 0b0_0000_0001 (16 cycles)	Refresh Period Maximum RefTime 0x1_1111_1111 (16 x 511 = 8176 cycles)
200	5 ns	80 ns	40.88 µs
267	3.75 ns	60 ns	30.66 µs
333	3 ns	48 ns	24.53 µs

Reset Value 0x04001800

Offset 0xF8002070

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:8	RefTime[0:8]	The RefTime value is compared to the high order 9 bits of the 13-bit Refresh Counter to determine the Refresh Period as defined above.	R/W	0b00001000
9:13	Unused	Writes have no effect; reads are undefined	R	undefined
14:15	DeferRef[0:1]	The DeferRef field specifies how many refreshes can be deferred before the Refresh is forced. NOTE: If scrub is enabled, a scrub request will be presented to the request arbiter at the time the refresh interval counter kicks off, not later when a deferred refresh actually happens. Also, the update of the 1/2-bit time measurement in the DDR2 PHY happens when the refresh happens, so if the refresh is deferred so is the 1/2-bit time update. 0x00 No deferred. 0x01 1 deferred 0x10 2 deferred 0x11 3 deferred	R/W	0x00
16:23	tRFC[0:7]	The Refresh Cycle Time is the minimum number of MemClks between a Refresh command to an Activate or another Refresh command within the same rank. 0x00 2 cycles 0x01 3 cycles .... 0xFF 257 clocks	R/W	0x18

Bits	Field Name	Description	Access	Reset
24:31	Unused	Writes have no effect; reads are undefined	R	undefined

RefTime example: JEDEC specification = 7.8µs/refresh. 400 MHz DIMMs, CK = 200 MHz, tCK = 5ns. 7.8 µs/5ns = 1560 cycles, = 0x618 cycles. Drop low 4 bits => 0x610 cycles. 13-bit value adjusted to register field boundary => MemRfshCntl[0:11] = 0x308.

The tRFC value in the MemRfshCntl register is a cycle count. Similar to the cycle counts in the RAS and CAS Timer registers, the value must be rounded up to satisfy the tRFC value in ns specified by JEDEC. The tRFC value varies with chip size (larger chips have larger values). Just as with the RAS and CAS Timing registers the most conservative value must be used.

Also note that the computed value must compensate for an internal 2 cycle delay, the same as the RAS and CAS Timer values. For example, the programmed value = the computed value - 2.

tRFC example: tCK = 5 ns for 400 MHz chips. Largest chip size = 1 Gb => tRFC = 127.5 ns (from JEDEC). 127.5 ns/5 ns = 25.5. Round up to 26. Subtract 2, the result is 24. The value programmed into MemRfshCntl bits 16:23 = 24 = 0x18.

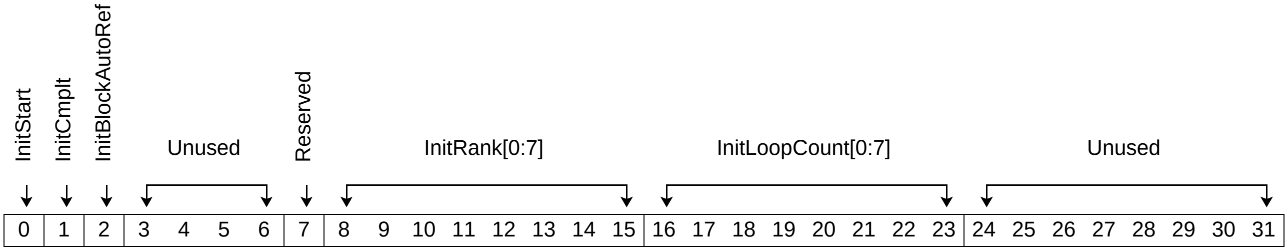
12.10.3 Memory Programming Control Register (MemProgCntl)

This register is used to start, control, and detect the completion of the initialization sequence. See *Section 12.10.6 Memory Initialization Registers [0:15] (MemInitReg[0:15])* on page 462.

Reset Value 0x00000000

Offset 0xF80020B0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	InitStart	Start initialization. When the StartInit bit is written from low (0) to high (1) the Initialization Sequence starts. This bit automatically clears to zero when the full initialization sequence completes, as indicated by the InitCmplt bit.	R/W	0b0
1	InitCmplt	The Initialization Complete bit is set by hardware when the full initialization sequence (as specified in the InitRegs) has been executed and the loop counter has gone to zero. This bit will automatically clear when a new initialization sequence is started with the InitStart bit.	R/W	0b0
2	InitBlockAutoRef	This bit controls the setting of the arefEnable bit when an initialization sequence completes. When active, auto-refresh cycles are prevented (blocked). When the bit is inactive, the auto-refresh function is enabled when the initialization sequence is finished. 0 Auto-refresh Enabled 1 Block Auto-refresh	R/W	0b0
3:6	Unused	Writes have no effect; reads are undefined.	R	undefined
7	Reserved	This bit must be set to 0.	R/W	0b0
8:15	InitRank[0:7]	Initialization Rank Enables. The InitRank field specifies which ranks are to undergo the Initialization Sequence. If the InitRank is 0h00 then the DmEn and SS bits determine which ranks are enabled. (See *Section 12.10.7 DIMM Configuration Registers* on page 463) 0x00 use DmCnfg Anything else use InitRank bit.	R/W	0h00
16:23	InitLoopCount[0:7]	The InitLoopCount specifies how many times the initialization sequence is executed. A Count of zero indicates that the sequence runs indefinitely. 0x00 Infinite Anything else use InitLoopCount value.	R/W	0h00
24:31	Unused	Writes have no effect; reads are undefined.		

12.10.4 Mode Register Set (MRS) Register (MRSRegCntl) and Extended Mode Register Set Register (EMRSRegCntl)

The MRSRegCntl Register contains a copy of the fields that are sent to the SDRAMs during initialization time. These values are defined in the JEDEC DDR2 SDRAM Spec. The A[x] designation indicates the Address signals that are driven to the external memories when the MRS is loaded during the SDRAM initialization sequence.

Note: The MRS register has no functional use. If programmed with a copy of the actual MRS contents that are written to the memories, software routines can read this register as a reference.

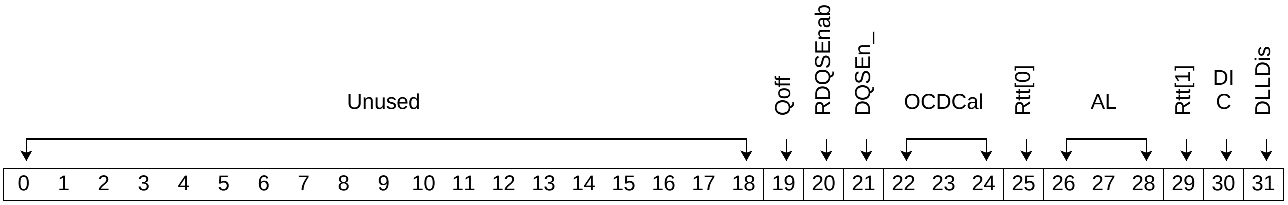
Reset Value	0x0000064A
Offset	0xF80020C0 (MRSRegCntl) 0xF80020D0 (EMRSRegCntl)
Access Type	Read/Write, Read Only

Unused	PD	WR[0:2]	DLL	TM	CL[0:2]	BT	BL [0:2]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18	19	20 21 22	23	24	25 26 27	28	29 30 31

Bits	Field Name	Description	Access	Reset
0:18	Unused	Writes have no effect; reads are undefined	R	undefined
19	PD	This bit specifies whether the active power down exit time is tXARD (fast exit) or tXARDS (slow exit). 0 Fast Exit 1 Slow Exit	R/W	0b0
20:22	WR[0:2]	Auto-Pre-Charge Write Recovery time. 001 2 cycles 010 3 cycles 011 4 cycles 100 5 cycles 101 6 cycles All others reserved	R/W	0b011
23	DLL	This field specifies DLL Reset. If the “Reset DLL” mode is used, the next command should return the part to normal operation. 0 No DLL reset 1 DLL reset	R/W	0b0
24	TM	This field specifies whether Test Mode is active. 0 Normal 1 Test mode	R/W	0b0
25:27	CL[0:2]	Programs the CAS latency required by the SDRAM modules. 011 3 bus clocks 100 4 bus clocks 101 5 bus clocks All others reserved	R/W	0b100

Bits	Field Name	Description	Access	Reset
28	BT	This field programs the burst type. Interleaved burst mode must be used in CPC945. 0 Sequential 1 Interleaved	R/W	0b1
29:31	BL[0:2]	This field programs the burst length. Burst length of 8 is only used for 64BitCfg or 64BitBus. 010 4 011 8 All others are reserved	R/W	0b010

The EMRSRegCntl Register does have a functional use if OCD calibration is performed. If the optional OCD Calibration is not performed then this EMRS register copy has no functional use. See *Section 7.11 MRS Register* on page 154 for more discussion.



Bits	Field Name	Description	Access	Reset
0:18	Unused	Writes have no effect; reads are undefined	R	undefined
19	QoffA[12]	Output buffer disable. This is an optional DDR2 feature 0 Output buffer enabled 1 Output buffer disabled	R/W	0b0
20	RDQSEnabA[11]	If RDQS is enabled, the DM function is disabled. RDQS is active for reads and don’t care for writes 0 Disable 1 Enable **Note:** RDQS is not supported. Always set bit 20 to ‘0’.	R/W	0b0
21	DQSEn_A[10]	DQS enable. Note that this is active low. 0 Enable 1 Disable	R/W	0b0
22:24	OCDCalA[9:7]	Off-chip driver (OCD) calibration program 000 OCD mode exit; maintain setting 001 Drive(1) 010 Drive(0) 100 Adjust mode 111 OCD calibration default	R/W	0b000
25	Rtt[0]A[6]	Rtt (nominal), [A6, A2], Specifies ODT (on-die termination) disable and strength. See bit 29. 00 ODT disabled 01 75 Ω 10 150 Ω 11 Reserved	R/W	0b00

Bits	Field Name	Description	Access	Reset
26:28	AL A[5:3]	Additive latency (AL) 000 0 cycles (default) 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles 101-111 Reserved	R/W	0b000
29	Rtt[1] A[2]	Rtt (nominal), [A6, A2], Specifies ODT (on-die termination) disable and strength. See bit 25. 00 ODT disabled 01 75 Ω 10 150 Ω 11 Reserved	R/W	0b00
30	DIC A[1]	Driver impedance control. This field programs the strength of the DRAM output driver impedance. 0 Normal (100% driver size) 1 Weak (60% driver size)	R/W	0b0
31	DLLDis A[0]	DLL disable. This field defines whether the DLL mode (DelayLock Loop) is used. When enabled, aligns the strobe signal to data (normal Operation). When enabled, data strobes cannot be used. 0 Enable delay lock loop 1 Disable delay lock loop	R/W	0b0

12.10.5 Memory Mapping Exception Registers

The 2 Memory Mapping Exception Registers contain information about a memory request that caused an addressing exception while trying to map to a rank, bank, and so on. The main cause of this exception is probably accessing beyond the bounds of memory. The MemMapExcpAd contains address bits 28:59 of the request while the MemMapExcpCtl contain control information like TSiz, RW, Tag, RID, etc.

12.10.5.1 Memory Mapping Exception Address Register (MemMapExcpAd)

Reset Value 0x00000000

Offset 0xF80020E0

Access Type Read Only

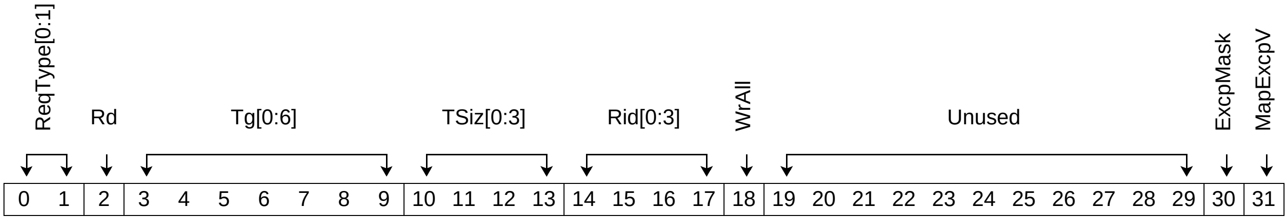
Ad

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:31	Ad	Contains bits 28:59 of the memory request that caused the addressing exception.	R	undefined

12.10.5.2 MemMapExcpCtl Register (MemMapExcpCtl)

Reset Value	0x00000000
Offset	0xF80020F0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:1	ReqType[0:1]	Request Type. Indicates whether this request is coherent, noncoherent (PCIe only) or a scrub request 00 Coherent 01 Non-coherent 10 Scrub	R/W	0b00
2	Rd	Indicates if this request is a read or write. 0 Write 1 Read	R/W	0b0
3:9	Tg[0:6]	Tag. The Tg is the 7-bit Tg presented with the requests. For writes, this Tag specifies the WDB location. For reads, this Tag is returned with the read data.	R/W	0b0000000
10:13	TSiz[0:3]	Transaction Size. Indicates the size in bytes of the transaction	R/W	0b0
14:17	Rid[0:3]	Requestor ID. Indicates the source of the request, either Proc[0:3] or Debug, HT, or PCIe. 00nn Proc[nn] 0100 PCIe 0101 HT 1110 Scrub 1111 Debug	R/W	0b0000
18	WrAll	Write All. On a write request, the WrAll signal indicates that all the Write Enables are active that are specified in the TSiz.	R/W	0b0
19:29	Unused	Writes have no effect; reads are undefined	R	undefined
30	ExcpMask	The Exception Mask turns on or off the generation of an interrupt when an address Exception occurs. 0 Interrupt Enabled 1 Interrupt Disabled	R/W	0b0
31	MapExcpV	The Map Exception Valid bit (when set) indicates that an Exception has occurred. This bit generates an interrupt to the processor when the Exception Mask is 0. *When the MemMapExcpCtl Register is read, this bit clears (0). Warning: Writing a one to this bit will cause an Exception. 0 No Exception 1 Exception occurred	R*/W	0b0

12.10.6 Memory Initialization Registers [0:15] (MemInitReg[0:15])

There are sixteen 32-bit Memory Initialization Registers. The fields in these registers control the command and address signals to memory during the initialization sequence. The following is the format of each register.

Reset Value 0x00000000

Offset 0xF8002100-0xF80021F0

Access Type Read/Write

En	RAS	CAS	WE	Delay								Loop	BA [2:0]			A [15:0]															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0	En	This command is enabled. 0 No Command 1 Valid Command	R/W	0b0
1	RAS	RAS signal to DRAM. Active low	R/W	0b0
2	CAS	CAS signal to DRAM. Active low	R/W	0b0
3	WE	WE signal to DRAM. Active low	R/W	0b0
4:11	Delay	Number of MemClks to delay after issuing this command 0 2 cycles 1 3 cycles … 255 257 cycles	R/W	0x00
12	Loop	This field indicates whether the sequence should start over from MemInitReg0. When active the sequence continues to loop for the number of times specified in MemProgCntl or turning this bit off cause the loop to terminate. 0 Looping disabled 1 Looping enabled	R/W	0b0
13:15	BA [2:0]	Bank address signals to DRAM.	R/W	0b000
16:31	A [15:0]	Memory address signals to DRAM.	R/W	0x0000

The initialization sequence is initiated by setting the InitStart bit in the MemProgCntl register defined above. This initiates a sequence starting with MemInitReg0. If the En bit (Enable) in MemInitReg0 is active (1), then the MemInitReg0 defines a Command that is sent to the memory interface with the specified state of the RAS, CAS, WE, BA, and MA signals. Each enabled Rank (See MemProgCntl register) is sent the Command on consecutive cycles.

While the Command is being issued, a Delay counter is counting off the number of cycles (MemClks) specified in the Delay field. Once that command has been sent to the enabled Ranks, the sequencer waits the number of cycles specified in the Delay counter before sending the next command. For example if there are 8 enabled Ranks and the Delay value is 2 (4 cycles), the delay is hidden under the CS cycles. If there are 2 Ranks and the delay is 3 (5 cycles) then there will be 2 CS cycles followed by 3 noop cycles.

The next Command comes from MemInitReg1. If this En bit is active then another command is sent using the fields from MemIntReg1. A Delay also occurs before proceeding to the next MemInitReg. This continues until an En bit is reached that is 0. When this happens the sequence is terminated and the InitCmplt bit in the MemProgCntl reg is set.

If a MemInitReg is executed with its Loop bit set, then the sequence starts over with MemInitreg0. The sequence continues until an En is found with a 0, or repeats when a Loop bit of 1 is found. Also, the looping continues for the number of times specified in the InitLoopCount field of the MemProgCntl register.

12.10.7 DIMM Configuration Registers

These registers specify the size and types of DIMMs installed in the system and are used by the Memory Bank Mapping to determine how the 36-bit physical address maps into Rank, Bank, Row, and Column addresses.

There are a total of four DIMM configuration registers. Each register designates the size and type of a single DIMM and whether that DIMM is enabled. These registers are loaded based upon the types and configuration of DIMMs installed. See *Section 7.14.10 DIMM Configuration Algorithm* on page 163 for an general method of programming these registers, and *Section 7.14.11 DIMM Configuration Examples* on page 165 for example values.

Table 12-18. DIMM Configuration Registers

DIMM Configuration Register	Reset Value	Offset	Access Type
Dm0Cnfg	08081001	0xF8002200	Read/Write, Read Only
Dm1Cnfg	08081041	0xF8002210	
Dm2Cnfg	08081101	0xF8002220	
Dm3Cnfg	08081141	0xF8002230	

Add2G[28:32]					Unused			Sub2G[28:32]					Unused			MemMd[0:3]				Strategies[28:36]									Unused	SS	DmEn
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:4	Add2G[28:32]	High order address bits of the additional 2 GB memory space that is part of this rank due to the 2 GB I/O hole	R/W	0b0000_1
5:7	Unused	Writes have no effect; reads are undefined	R	undefined
8:12	Sub2G[28:32]	High order address bits of the 2 GB memory space that should be removed from the address spaces defined by the StartAdrs and MemMd due to the 2 GB I/O hole	R/W	0b0000_1
13:15	Unused	Writes have no effect; reads are undefined	R	undefined
16:19	MemMd[0:3]	Memory mode. Specifies one of the 12 types (size, chip width) of DIMMs for this rank. See *Table 7-6 Memory Modes* on page 157.	R/W	0b0001

Bits	Field Name	Description	Access	Reset
20:28	Strategies [28:36]	Specifies the starting physical address for this DIMM of memory. This address along with the memory size specified in the MemMd determines the address space for this DIMM. If this is part of a DIMM group that is greater than 2 GB then the Add2G and Sub2G fields provide corrections to this address space.	R/W	Dm0: 0b0000_0000_0 Dm1: 0b0000_0100_0 Dm2: 0b0001_0000_0 Dm3: 0b0001_0100_0
29	Unused	Writes have no effect; reads are undefined	R	undefined
30	SS	Single Sided. Specifies whether this DIMM is single sided (contains only 1 rank) or double sided (contains 2 ranks). 0 Double 1 Single	R/W	0b0
31	DmEn	DIMM Enable. Specifies whether this DIMM is enabled. Used in bank mapping and memory initialization sequencing. 0 Disabled 1 Enabled	R/W	0b1

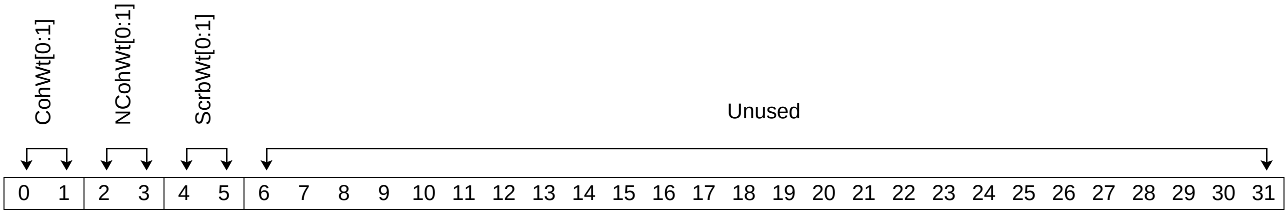
12.10.8 Memory Arbiter Weight Register (MemArbWt)

This register specifies the weights for the round-robin arbitration. Each agent can have a weight assigned such that the agent retains the priority for that many requests of the arbiter once it has priority. A weight of zero indicates that this requester does not participate in the round-robin selection (it has low priority) and is only serviced when the other requesters are idle.

Reset Value 0x54000000

Offset 0xF8002280

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:1	CohWt[0:1]	Coherent weight. Specifies the arbiter weight for coherent requests. 00 Low priority 01 1 request 10 2 requests 11 3 requests	R/W	0b01
2:3	NCohWt[0:1]	Noncoherent weight. Specifies the arbiter weight for noncoherent requests. 00 Low priority 01 1 request 10 2 requests 11 3 requests	R/W	0b01

Bits	Field Name	Description	Access	Reset
4:5	ScrbWt[0:1]	Scrub weight. Specifies the arbiter weight for scrub requests. If the weight is zero, then the scrub requests are the lowest priority and do not participate in the round-robin arbitration. 00 Low priority 01 1 request 10 2 requests 11 3 requests	R/W	0b01
6:31	Unused	Writes have no effect; reads are undefined.	R	undefined

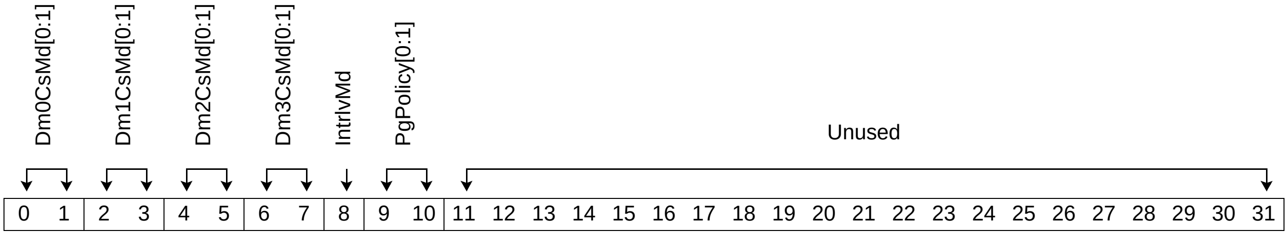
12.10.9 Memory User Configuration Register (UsrCnfg)

This register indicates the chip select mode, interleave mode, and page strategy mode.

Reset Value 0x00000000

Offset 0xF8002290

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:1	Dm0CsMd[0:1]	Specifies the type of ChipSelect grouping for DIMM 0. Encoding: See *Section 7.14.8 Chip Select Mode* on page 158.	R/W	0b00
2:3	Dm1CsMd[0:1]	Specifies the type of ChipSelect grouping for DIMM 1.	R/W	0b00
4:5	Dm2CsMd[0:1]	Specifies the type of ChipSelect grouping for DIMM 2.	R/W	0b00
6:7	Dm3CsMd[0:1]	Specifies the type of ChipSelect grouping for DIMM 3.	R/W	0b00
8	IntrlvMd	Specifies whether the bank interleave mode is based on a DRAM page or the L2 cache line size. 0 DRAM page 1 L2 cache line	R/W	0b0

Bits	Field Name	Description	Access	Reset
9:10	PgPolicy[0:1]	Specifies the type of open/close page policy in effect. Usually open policy causes a page to be left open after a data command unless there are no other data commands to that page from the same queue and there are other requests to a different page and same bank/rank from the same queue. Usually closed policy causes a page to be closed after the data command unless there are requests still pending to the same page from the same queue. 0 Usually open 1 Usually closed 2 Leave open 3 Leave closed	R/W	0b00
11:31	Unused	Writes have no effect; reads are undefined	R	undefined

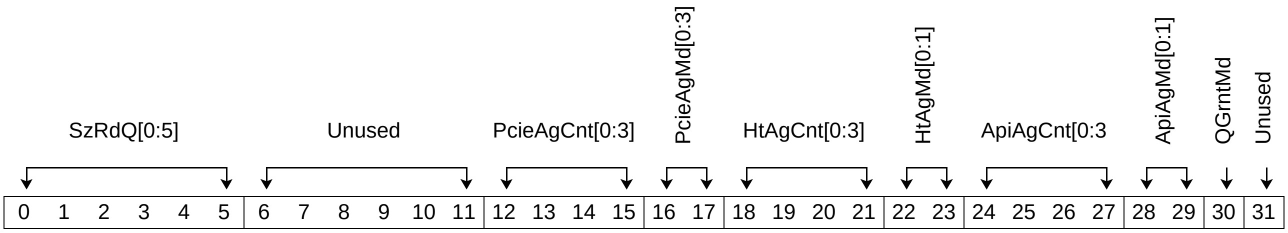
12.10.10 Memory Read Request Queue Configuration Register (MemRdQCnfg)

This register specifies the parameters controlling the Read Reorder Queue.

Reset Value 0x20020820

Offset 0xF80022A0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:5	SzRdQ[0:5]	Size of Read Reorder Queue. Specifies the number of entries in the Read Reorder Queue. Minimum: 1, Maximum: 8, Default: 8	R/W	0b001000
6:11	Unused	Writes have no effect; reads are undefined.	R	undefined
12:15	PcieAgCnt [0:3]	Specifies the number of cycles (MemClks) a PCIe read request can wait before it is aged and whether aging is enabled. Counting does not start until a transaction hits the bottom of the queue. The value changes depending on PcieAgMd. 0000-1110 See PcieAgMd 1111 PCIe Aging Disabled	R/W	0b0010
16:17	PcieAgMd[0:3]	Specifies how the PcieAgCnt is to be interpreted. 0 Use PcieAgCnt 1 Use PcieAgCnt * 4 2 Use PcieAgCnt * 16 3 Use PcieAgCnt * 64	R/W	0b00
18:21	HtAgCnt[0:3]	Specifies the number of cycles (MemClks) a HT read request could wait before it is aged and whether aging is enabled. Counting does not start until a transaction hits the bottom of the queue. The value changes depending on HtAgMd. 0000-1110 See HtAgMd 1111 HT Aging Disabled	R/W	0b0010
22:23	HtAgMd[0:1]	HT Read Aging Mode. Specifies how the HtAgCnt is to be interpreted. 0 Use HtAgCnt 1 Use HtAgCnt * 4 2 Use HtAgCnt * 16 3 Use HtAgCnt * 64	R/W	0b00
24:27	ApiAgCnt[0:3	PI Read Aging Count. Specifies the number of cycles (MemClks) an PI (proc) read request can wait before it is aged and whether aging is enabled. Counting does not start until a transaction hits the bottom of the queue. The value changes depending on ApiAgMd. 0000-1110 See ApiAgMd 1111 PI Aging Disabled	R/W	0b0010

Bits	Field Name	Description	Access	Reset
28:29	ApiAgMd[0:1]	PI Read Aging Mode. Specifies how the ApiAgCnt is to be interpreted. 0 Use ApiAgCnt 1 Use ApiAgCnt * 4 2 Use ApiAgCnt * 16 3 Use ApiAgCnt * 64	R/W	0b00
30	QGrntMd	Queue Grant Mode. When this bit is set, no new requests are allowed into the Read Reorder Queue (RdQ) or the Write Reorder Queue (WrQ) if either the RdQ or the WrQ are full. With this bit set, fastpath disabled and the RdQ and WrQ size set to one, the memory controller will execute requests in the same order that was seen on the request interface. 0 Normal 1 Special	R/W	0b0
31	Unused	Writes have no effect; reads are undefined.	R	undefined

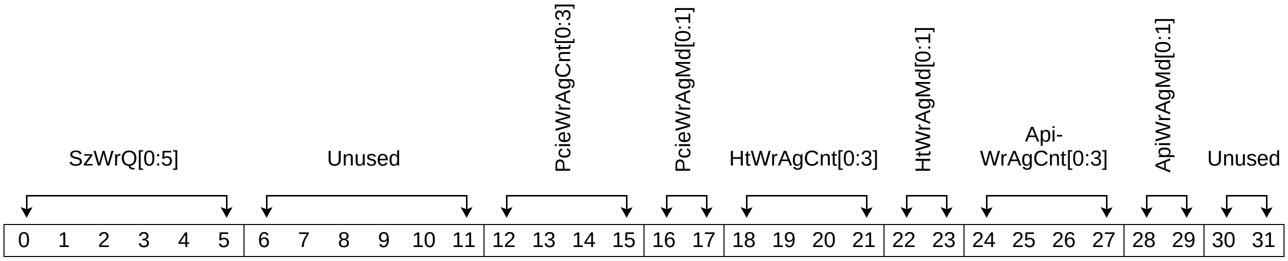
12.10.11 Memory Write Request Queue Configuration Register (MemWrtQCnfg)

This register specifies the parameters controlling the Write Reorder Queue.

Reset Value 0x40041040

Offset 0xF8002270

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:5	SzWrQ[0:5]	Size of Write Reorder Queue. Specifies the number of entries on the Write Reorder Queue. Minimum: 1, Maximum: 16, Default: 16	R/W	0b010000
6:11	Unused	Writes have no effect; reads are undefined	R	undefined
12:15	PcieWr AgCnt[0:3]	PCIe Write Aging Count. Specifies the number of write requests a PCIe write request can wait before it is aged and whether aging is enabled. Counting does not start until a transaction hits the bottom of the queue. The value changes depending on PcieWtAgMd. 0000-1110 See PcieWtAgMd 1111 PCIe aging disabled	R/W	0b0100

Bits	Field Name	Description	Access	Reset
16:17	PcieWr AgMd[0:1]	PCIe write aging mode. Specifies how the PcieWtAgCnt is to be interpreted. 0 Use PcieWtAgCnt 1 Use PcieWtAgCnt × 4 2 Use PcieWtAgCnt × 16 3 Use PcieWtAgCnt × 64	R/W	0b00
18:21	HtWr AgCnt[0:3]	HT write aging count. Specifies the number of write requests a HT write request can wait before it is aged and whether aging is enabled. Counting does not start until a transaction hits the bottom of the queue. The value changes depending on HtWtAgMd. 0000-1110 See HtWtAgMd 1111 HT aging disabled	R/W	0b0100
22:23	HtWr AgMd[0:1]	HT Write Aging Mode. Specifies how to interpret HtWtAgCnt. 0 Use HtWtAgCnt 1 Use HtWtAgCnt × 4 2 Use HtWtAgCnt × 16 3 Use HtWtAgCnt × 64	R/W	0b00
24:27	ApiWr AgCnt[0:3]	PI write aging count. Specifies the number of write requests an PI (proc) write request can wait before it is aged and whether aging is enabled. Counting does not start until a transaction hits the bottom of the queue. The value changes depending on ApiWtAgMd. 0000-1110 See ApiWtAgMd 1111 PI aging disabled	R/W	0b0100
28:29	ApiWr AgMd[0:1]	PI write aging mode. Specifies how the ApiWtAgCnt is to be interpreted. 0 Use ApiWtAgCnt 1 Use ApiWtAgCnt × 4 2 Use ApiWtAgCnt × 16 3 Use ApiWtAgCnt × 64	R/W	0b00
30:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.12 Memory Reorder Queue Arbitration Register (MemQArb)

This register specifies the parameters controlling the read and write ReOrderQ Arbitration modes.

Reset Value 0x00000000

Offset 0xF80022B0

Access Type Read/Write, Read Only

RdQCVEn[0:5]						RdQRVEn[0:5]						WrQCVEn[0:5]						WrQRVEn[0:5]						Unused							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:5	RdQCVEn [0:5]	Specifies the number of entries in the read reorder queue (RdReoQ) that participate in CV arbitration. 0 All entries n Number of entries	R/W	0b000000
6:11	RdQRVEn [0:5]	Specifies the number of entries in the read reorder queue (RdReoQ) that participate in RV Arbitration. 0 All entries n Number of entries	R/W	0b000000
12:17	WrQCVEn [0:5]	Specifies the number of entries in the write reorder queue (WrReoQ) that participate in CV Arbitration. 0 All entries n Number of entries	R/W	0b000000
18:23	WrQRVEn [0:5]	Specifies the number of entries in the write reorder queue (WrReoQ) that participate in RV Arbitration. 0 All entries n Number of entries	R/W	0b000000
24:31	Unused	Writes have no effect; reads are undefined	R	undefined

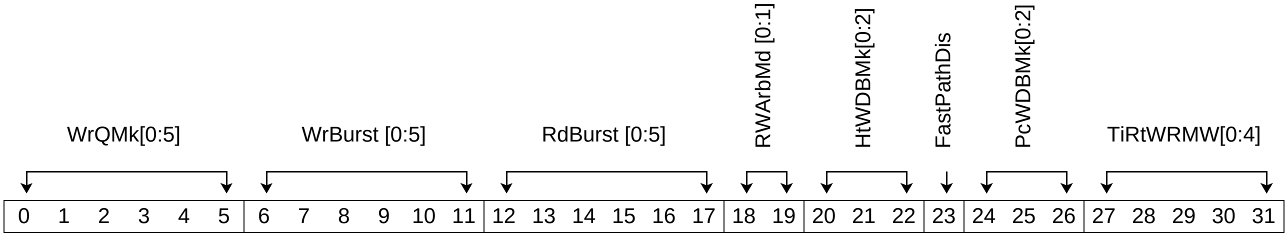
12.10.13 Memory R/W Arbitration Register (MemRWArb)

This register specifies the parameters controlling the read and write arbitration modes.

Reset Value 0x30413CC0

Offset 0xF80022C0

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
0:5	WrQMk[0:5]	Write queue high water mark. Specifies the high water mark in the WrReoQ.	R/W	0b001100
6:11	WrBurst [0:5]	Write burst. Specifies the minimum number of write requests issued before switching priority to RdQ. Only counts for switches caused by high watermarks. 0 No entries n n entries Maximum: All entries	R/W	0b000100
12:17	RdBurst [0:5]	Read burst. Specifies the minimum number of read requests issued before switching priority to WrQ. 0 No entries n n entries Maximum: All entries	R/W	0b000100
18:19	RWArbMd [0:1]	Read/write arbitration mode. RWArbMd is a 2-bit field that designates allowing requests from nonpriority queue when giving priority to the current queue. When in a current queue state, only requests from that queue can be processed. The RWArbMd overrules the priority and allows CAS commands to execute if they are ready and there is not a CAS command ready for the current queue. This only applies for CAS commands. RWArbMd[0] = 1: Writes are allowed in read queue state. RWArbMd[1] = 1: Reads are allowed in write queue state.	R/W	0b11
20:22	HtWDBMk[0:2]	Specifies the high water mark for the HT write data buffer (WDB). 0 8 entries. n n entries.	R/W	0b110
23	FastPathDis	Disables the fast bypass path around the read reorder queue. 0 Enable 1 Disable	R/W	0b0
24:26	PcWDBMk[0:2]	High water mark for the PCIe write data buffer (WDB). 0 8 entries n n entries	R/W	0b110

Bits	Field Name	Description	Access	Reset
27:31	TiRtWRMW[0:4]	Read to write timer for read-modify-write (RMW). Specifies the timer delay to be loaded when transitioning from read to write for read-modify-write operations. Calculation: CEIL((RdMacDel + 1)/2) + 6 – WL. Because of the latency in loading a timer, 2 should be subtracted from the “calculation” value.	R/W	0b00000

12.10.14 Memory Bus Configuration Register (MemBusConfig)

This register specifies parameters for the timing of data bus for reads and writes.

Reset Value 0x00000008

Offset 0xF80022D0

Access Type Read/Write

RdMacDel[0:3]	ResMuxDel[0:3]	RdExtMuxDly[0:3]	WrExtMuxDly[0:3]	WdbRqDly[0:3]	StartLdEnOn	RdOEOnDly[0:1]	RdOEOffDly[0:3]	Reserved	Ad2Cyc	64BitCfg	64BitBus
0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20	21 22 23	24 25 26 27	28	29	30	31

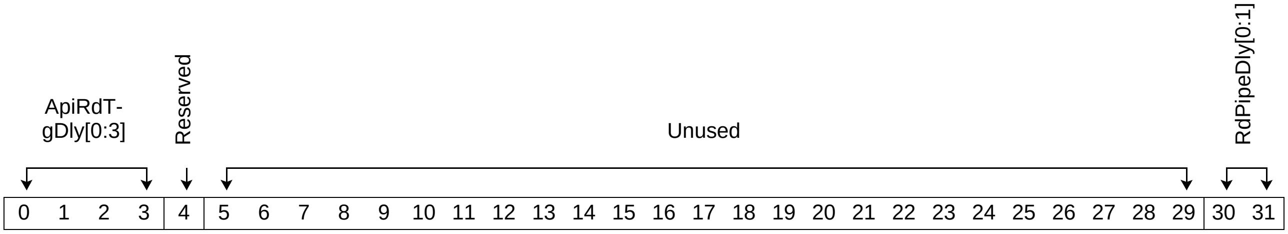
Bits	Field Name	Description	Access	Reset
0:3	RdMacDel[0:3]	Read macro delay. RdMacDel sets the delay (ddr_clks) from the read command to the cycle in which the ddr_phy data capture registers begin to be transferred to the ddr_clk domain. There is a multi-ddr_clk cycle window in which to begin this transfer and the exact cycle is dependent on the round-trip board and system logic delays in the memory interface. See *Section 12.10.15 Memory Bus Configuration Register 2* on page 474. 0b0000 = 2 0b0001 = 3 0b0010 = 4 … 0b1111 = 17	R/W	0b0000
4:7	ResMuxDel[0:3]	Res multiplexer delay. ResMuxDel sets the delay (ddr_clks) from the read command to the cycle in which the ddr_phy data capture registers begin to sample the DQ data signal with the DQS strobe input (delayed). There is a multi-ddr_clk cycle window where this delay must fall and it includes the read DQS preamble time. Round-trip board delays, system logic delays and byte-lane skews contribute to the position of this window as well as its width and the proper value is typically found through educated guess and test methods. Additional delay verniers are available in the MiscVernierC0-3 registers. See *Section 12.10.30 Reset LdEn Offset Delay Registers (RstLdEnVerniersCn)* on page 494. 0b0000 = 0 0b0001 = 1 0b0010 = 2 0b0011 = 3 … 0b1111 = 15	R/W	0b0000

Bits	Field Name	Description	Access	Reset
8:11	RdExtMuxDly [0:3]	Read Multiplexer Enable Delay. Sets the delay (ddr_clks) from a Read command to the cycle in which the external multiplexer controls change to their new value. Additional delay verniers are available in the MiscVernierC0-3 registers. See *Section 12.10.15 Memory Bus Configuration Register 2* on page 474. 0b0000 = 0 0b0001 = 1 … 0b1111 = 15	R/W	0b0000
12:15	WrExtMuxDly [0:3]	Write Multiplexer Enable Delay. Sets the delay (ddr_clks) from a Write command to the cycle in which the external multiplexer controls change to their new value. Additional delay verniers are available in the MiscVernierC0-3 registers. 0b0000 = 0 0b0001 = 1 … 0b1111 = 15	R/W	0b0000
16:19	WdbRqDly[0:3]	WDB request delay. Specifies the number of ddr_clk cycles to wait after sending a Write command to request write data from the WDB. Set to 2 × (CL + n_register_clk_dlys - 4) + n ddr_cycles of CK delay to SDRAM. For example, CL = 4, registered DIMMs, negligible wiring delay, WdbRqDly = 2 × (4 + 1 - 4) + 0 = 2. Note that unregistered DIMMs with CL = 3 and negligible board delay is not supported because this would require a negative RdbRqDly. 0b0000 = 0 0b0001 = 1 … 0b1111 = 15	R/W	0b0000
20	StartLdEnOn	StartLdEnOn. Override of the DQS glitch filter logic in the DDR_PHY read data capture module.	R/W	0b0
21:23	RdOEOnDly[0:1]	Read OE On Delay. Specifies how many ddr_clk cycles the DQ/DQS OE remains “off” following the completion of a Read Command. Typically set to zero. See *Section 12.10.15 Memory Bus Configuration Register 2* on page 474.	R/W	0b00
24:27	RdOEOffDly[0:3]	Read OE off delay. Specifies the delay in ddr_clk cycles from the CAS read Command to the DQ/DQS OE-turning “off”. Typically set to (2 × (CL + n_register_delays)) – 4 + n ddr_cycles of CK delay.	R/W	0b0000
28	Reserved	This bit must be set to 0.	R/W	0b1
29	Ad2Cyc	Specifies using 2-cycle addressing. 2-cycle addressing places the memory command on the memory bus for two memory cycles instead of one. 0 1 cycle 1 2 cycle	R/W	0b0
30	64BitCfg	This bit specifies whether the DIMMs are in a 64-bit configuration. Data is sent to the DIMMs 64 bits at a time instead of 128 bits. 0 128 bit 1 64 bit	R/W	0b0
31	64BitBus	This bit specifies whether the external interface has a 64-bit data bus. A 128-bit bus requires paired DIMMs to work. 0 128-bit bus 1 64-bit bus	R/W	0b0

12.10.15 Memory Bus Configuration Register 2

This register specifies additional parameters for the timing of data bus for read.

Reset Value	0x00000000
Offset	0xF80022E0
Access Type	Read/Write, Read Only

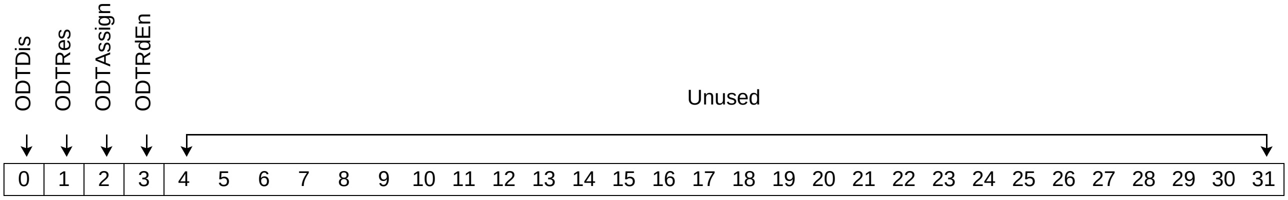


Bits	Field Name	Description	Access	Reset
0:3	ApiRdTgDly [0:3]	ApiRdTgDly sets the delay (ddr_clks) from the read command to the cycle in which the TgV is sent to the PI unit. Initially this value should be set the same as RdMacDel. Reducing this delay causes the TgV to be seen earlier giving the PI unit more time to set up for the data. Whenever this delay is reduced, the ApiMemDly value in the PI register, APIMemRdCfg, needs to be increased by an equal amount. 0b0000 = 2 0b0001 = 3 0b0010 = 4 … 0b1111 = 17	R/W	0b0000
4	Reserved	This bit must be set to 0.	R/W	0
5:29	Unused	Writes have no effect; reads are undefined.	R	undefined
30:31	RdPipeDly [0:1]	RdPipeDly specifies the additional delay in ddr_clks that is added to the read parameters to move the read data window into a better programmable range. This additional delay is added to the following fields: ApiRdTgDly, RdMacDel, ResMuxDel, RdExtMuxDly, and RdOEOffDly. (See *Section 12.10.14 Memory Bus Configuration Register (MemBusConfig)* on page 472) 0b00 0 delay 0b01 4 cycle delay 0b10 8 cycle delay 0b11 12 cycle delay	R/W	0b00

12.10.16 ODT Control Register (ODTCntl)

This register specifies the controls for ODT.

Reset Value	0x00000000
Offset	0xF80023A0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	ODTDis	ODT disable. Defines whether ODT is disabled. 0 Enabled 1 Disabled	R/W	0b0
1	ODTRes	ODT resolution mode When ODT is enabled, this bit must be set to 1. Although the default is 0, the bit must be set to 1 for ODT to function correctly. When set to 1, ODT is assumed to be routed per DIMM so: ODT[0] = 1 enables DIMM 0, ODT[1] = 1 enables DIMM 1, ODT[2] = 1 enables DIMM 2, etc. Assuming the following DIMM/rank numbering: rank 0,1 DIMM 0 DIMM 4 rank 2,3 DIMM 1 DIMM 5 rank 4,5 DIMM 2 DIMM 6 rank 6,7 DIMM 3 DIMM 7	R/W	0b0
2	ODTAssign	ODT assignment mode Defines whether ODT is using direct mode (for multiplexer systems) or indirect mode (for multidrop systems). In direct mode, the DIMM being accessed is the one for which termination is turned on. In indirect mode, termination is turned on for the DIMM that is not being accessed. 0 Direct 1 Indirect	R/W	0b0
3	ODTRdEn	ODT Read Enable. Selects whether ODT is driven for reads as well as writes, or for writes only. 0 Writes only 1 Both	R/W	0b0
4:31	Unused	Writes have no effect; reads are undefined.	R	undefined

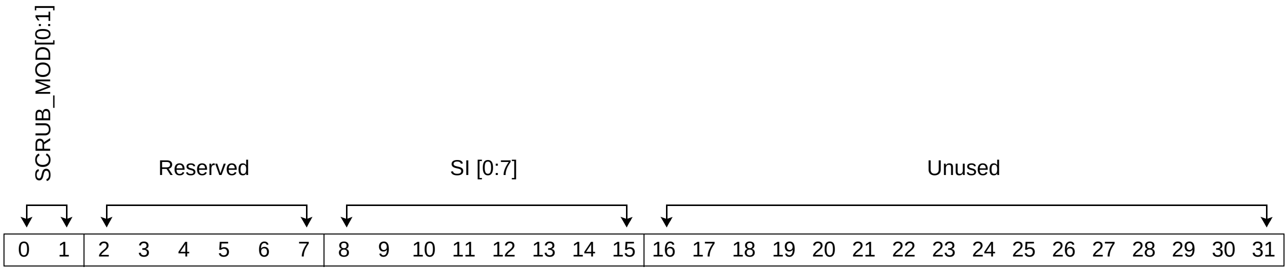
12.10.17 Memory Scrub Control Register (MSCR)

This register is used in conjunction with the Memory Scrub Pattern Register (MSPR), Memory Scrub Range Start Register (MSRSR), Memory Scrub Range End Register (MSRER) to perform main memory fill, initialization, test, and background soft error scrub. The scrub logic automatically skips over the third and fourth GB of address space (if the MPRSR and MSRER are set to include a portion or all of this space) that is reserved for I/O.

Reset Value 0x00000000

Offset 0xF8002400

Access Type Read/Write, Read Only

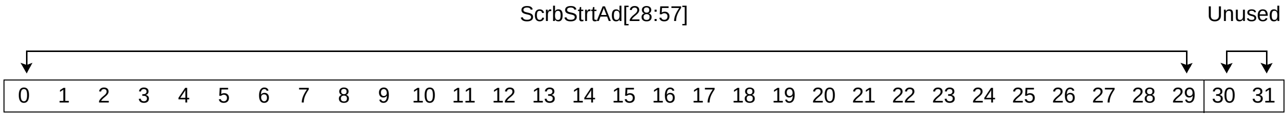


Bits	Field Name	Description	Access	Reset
0:1	SCRUB_MOD [0:1]	Scrub mode This field defines the mode of operation for the memory scrub control logic. **Background**: Scrub read and write one successive 64-byte block every interval. This mode will remain set indefinitely, or until re-programmed by system software. **Immediate**: Scrub read and write one successive 64-byte block continuously (without regard to the background interval). This mode will be reset after one complete pass through memory. **Immediate with Fill**: Scrub write (without read) successive 64-byte blocks continuously (without regard to the background interval). Write data is derived from the Memory Scrub Pattern Register (MSPR), where the 4-byte register content is repeated (modulo 4 byte) across the 16-byte memory data path before data is sent through the ECC generator circuits. This mode will be reset after one complete pass through memory. 00 Off. No scrub activity. 01 Background 10 Immediate 11 Immediate with fill	R/W	0b00
2:7	Reserved	Reserved	R/W	0x00

Bits	Field Name	Description	Access	Reset
8:15	SI [0:7]	Background scrub interval. This field defines the period at which the scrub controller will scrub 64-byte block while operating in background scrub mode. The scrub controller maintains an internal decrementer, which is clocked at the refresh rate, defined by the Memory Refresh Period Register. When the internal decrementer reaches zero, a request is issued to scrub the next successive 64-byte memory block, and the decrementer is reloaded with the scrub interval value. 0x00 Every refresh interval 0x01 Every other refresh interval 0x02 Every 3rd refresh interval ….. 0xFF Every 256th refresh interval	R/W	0h00
16:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.18 Memory Scrub Range Start Register (MSRSR)

Reset Value	0x00000000
Offset	0xF8002410
Access Type	Read/Write, Read Only



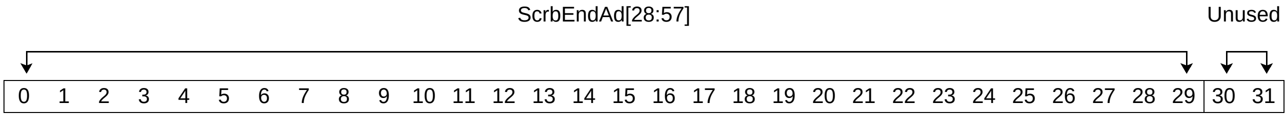
Bits	Field Name	Description	Access	Reset
0:29	ScrbStrtAd[28:57]	This field defines the 64-byte scrub operation start address, within a 64 GB address range. The value in this register represents the upper 30 address bits of a 36-bit address. For example, if this register is set to 0x0000_00EC, then the scrub starts with physical address 0x0_0000_0EC0.	R/W	b0
30:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.19 Memory Scrub Range End Register (MSRER)

Reset Value 0x00000000

Offset 0xF8002420

Access Type Read/Write, Read Only



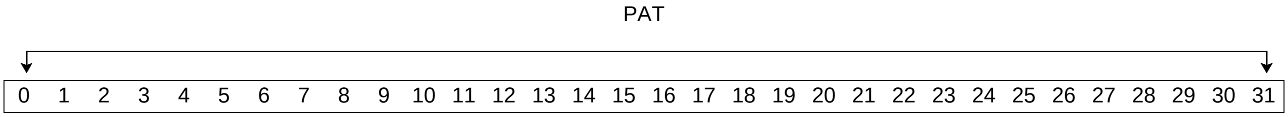
Bits	Field Name	Description	Access	Reset
0:29	Scrb EndAd [28:57]	This field defines the 64-byte scrub operation end address, within a 64 GB address range. For example, if this register is set to 0x0000_00EC, then the scrub ends with physical address 0x0_0000_0EFF. If this address is less than the start address defined in the MSRSR, then no scrub operation will occur. If the addresses are equal, a scrub range of 64-byte is defined. For example, if MSRSR = MSRER = 0x0000_00A8 then the range of addresses that are scrubbed are 0x0_0000_0A80 – 0x0_0000_0ABF.	R/W	0b0
30:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.20 Memory Scrub Pattern Register (MSPR)

Reset Value 0x00000000

Offset 0xF8002430

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
0:31	PAT	Scrub pattern. This field defines the bit patterns required by scrub modes defined in Memory Scrub Control Register (MSCR).	R/W	0b00

12.10.21 Memory Check Control Register (MCCR)

This register is used to enable and test memory data error detection and correction (EDC) circuits. Memory address and ECC syndrome for detected data errors are logged in the Memory Error Syndrome and Address Registers (MESR, MEAR).

Reset Value 0x00000000

Offset 0xF8002440

Access Type Read/Write

ECC_EN	ECC_AP_DIS	Reserved		ByteLaneECCSub	EI_EN	ECC_UE_MASK	ECC_CE_MASK	Reserved								EI_PAT[0:15]															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0	ECC_EN	Specifies whether the ECC check is enabled which checks all memory read data. If disabled, ECC check bit data strobe signal pins (DDR_DQSP[0:1]), are high Z, and ECC check bit signal pins (DDR_DQP[0:15]), are driven low. 0 ECC disabled 1 ECC enabled	R/W	0b0
1	ECC_AP_DIS	Specifies whether the ECC address parity generation and checking functions normally or generates special uncorrectable errors. 0 AP enabled (normal) 1 AP disabled (special)	R/W	0b0
2:3	Reserved	Bits are r/w, but not connected to logic.	R/W	0b00
4	ByteLane ECCSub	Controls whether operating in normal mode where the 16-bit ECC code is placed on byte-lanes 16 and 17 or in calibrate mode where the 16-bit ECC code is distributed on the least significant bit (LSb) of each byte-lane, overriding the original data that was written or expected in those locations. This bit is only used to assist in memory timing calibration of systems that do not have ECC DIMMs. 0 Normal 1 Calibrate	R/W	0b0
5	EI_EN	Specifies whether the 16-bit pattern in EI_PAT is selected instead of the 16-bit ECC check bit generator output during memory outputs. 0 Select generator 1 Select EI_PAT	R/W	0b0
6	ECC_UE_ MASK	Specifies whether a ECC_UE bit state is propagated outside this register except during read operations. 0 Propagate UE 1 Mask off UE	R/W	0b0

Bits	Field Name	Description	Access	Reset
7	ECC_CE_ MASK	Specifies whether a ECC_CE bit state is propagated outside this register except during read operations. 0 Propagate CE 1 Mask off CE	R/W	0b0
8:15	Reserved	Reserved. Bits are r/w, but not connected to logic.	R/W	0x00
16:31	EI_PAT[0:15]	This field defines the pattern used for the ECC check bits during a memory write operation when ECC check bit injection is enabled.	R/W	0b0

12.10.22 Memory Error Address Registers

This pair of registers is loaded with the memory address signals upon detection of an error while reading the memory. These registers are updated simultaneously with the Memory Error Syndrome Register (MESR), as these registers exist as a logical set. Updates occur according to the Error Code priority defined in the MESR.

12.10.22.1 MEAR0 Register (MEAR0)

Reset Value 0x00000000

Offset 0xF8002460

Access Type Read/Write

RK [0:2]			COL [10:0]											BK [2:0]			ROW [14:0]														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:2	RK [0:2]	Rank Address. Encoded 0:7 chip select interface.	R/W	0b000
3:13	COL [10:0]	SDRAM Column Address	R/W	0b0
14:16	BK [2:0]	SDRAM Bank Address	R/W	0b0
17:31	ROW [14:0]	SDRAM Row Address	R/W	0b0

12.10.22.2 MEAR1 Register (MEAR1)

Reset Value 0x00000000

Offset 0xF8002470

Access Type Read/Write, Read Only

UECnt[0:7]								CECnt[0:7]								Unused									BCNT[0:3]				RK [0:2]		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:7	UECnt[0:7]	This field counts the number of uncorrectable errors (UE) detected.	R*/W	0x00
8:15	CECnt[0:7]	This field counts the number of correctable errors (CE) detected.	R*/W	0x00
16:24	Unused	Writes have no effect; reads are undefined.	R	undefined
25:28	BCNT[0:3]	This field contains the burst count for the memory read access with which an ECC error was detected.	R/W	0b0000
29:31	RK [0:2]	Rank address. Encoded 0:7 chip select interface.	R/W	0b000

Note: *Resets to zero when read and holds at its maximum value, 0xFF.

12.10.23 Memory Error Syndrome Register (MESR)

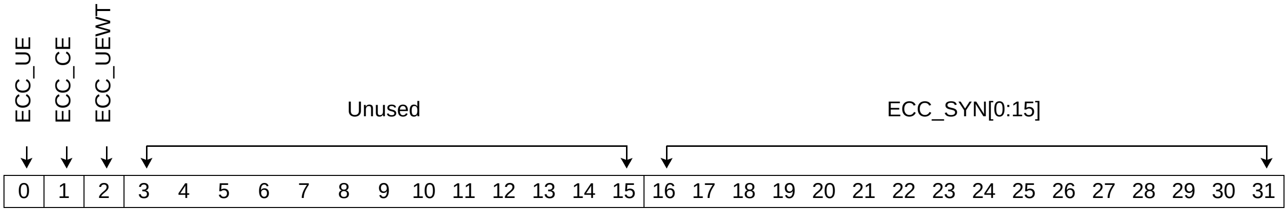
This register is loaded with the ECC syndromes upon detection of an error while reading the physical memory via scrub or normal memory access. This register is updated simultaneously with the Memory Error Address Registers (MEAR1, MEAR2), as these registers exist as a logical set.

The MEAR and the MESR.ECC_SYN are captured for the first correctable error and no uncorrectable error has been logged (MESR.ECC_CE = 0 and MESR.ECC_UE = 0), or the first uncorrectable error (MESR.ECC_UE = 0). This implies that the uncorrectable error is a higher priority over a correctable error, where the MESR.ECC_SYN and MEAR1 and MEAR2 state always pertains to the uncorrectable error when MESR.ECC_UE = 1 and MESR.ECC_CE = 1.

Reset Value 0x00000000

Offset 0xF8002480

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	ECC_UE	ECC uncorrectable error. This bit is set when the ECC checker detects an uncorrectable error during a valid data read access to the memory. This bit is cleared when read.	R*/W	0b0
1	ECC_CE	ECC correctable error. This bit is set when the ECC checker detects a correctable error during a valid data read access to the memory. This bit is cleared when read.	R*/W	0b0
2	ECC_UEWT	ECC uncorrectable error write. This bit is set when an uncorrectable error is detected in the read data that is modified during a read-modify-write (RMW) memory operation. This state implies that the erroneous data was changed after the error was detected during the initial RMW. This bit is not set if a write operation is performed on the erroneous data, after the initial RMW. This bit is cleared when read.	R*/W	0b0
3:15	Unused	Writes have no effect; reads are undefined.	R	undefined
16:31	ECC_SYN [0:15]	ECC syndrome for the full 16 bytes of the SDRAM interface (DDR_DQ[0:127]).	R/W	0b0
Note: *The error bits are cleared when read.				

12.10.24 Memory Mode Control Register (MemModeCntl)

This register contains various timing and power saving modes of the memory interface.

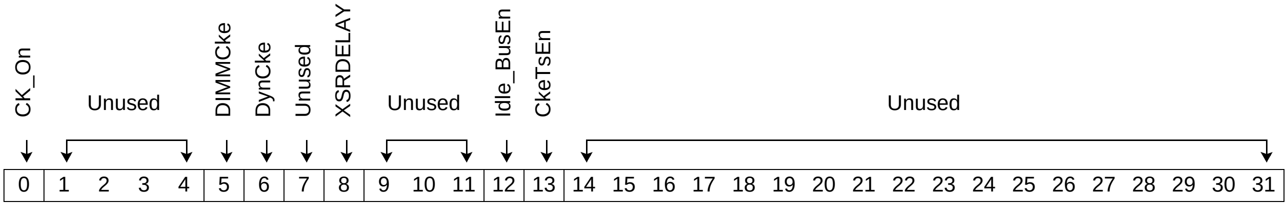
The dynamic CKE power down mode is implemented to be able to switch the DDR2 devices into a low power state while the memory bus is idle, even for short periods. The clock is still running during the power down state. The Clock Enable signals, CKE [0:7], are de-asserted while the respective banks are idle when this mode is enabled. The CKE signals do not toggle during normal operation when this mode is disabled.

The self-refresh power down mode is used by the external power manager to power down the memory subsystem while preserving the memory content. In this mode, the clocks to the memory devices could be halted.

Reset Value 0x00000000

Offset 0xF8002500

Access Type Read/Write, Read Only, Write Only



Bits	Field Name	Description	Access	Reset
0	CK_On	Turns on clocks to the DIMMs. Must be set prior to the setting of the InitStart bit (see MemProgCtl) which turns on CKE. CK_On resets to zero (off) when entering sleep. Once CK_On is one (on) only a reset or sleep can reset it. 0 CK Off 1 CK On	W	0b0
1:4	Unused	Writes have no effect; reads are undefined.	R	undefined
5	DIMMCke	Specifies whether CKEs are on a per DIMM or per Rank basis. If DynCKe is enabled and ODT is enabled, then DIMMCke must be set equal to ‘1’. Otherwise, this bit should be set equal to ‘0’. 0 DIMM Mode 1 Rank Mode	R/W	0b0
6	DynCke	When this mode is enabled, all Clock Enables (CKEs) are brought low whenever the request and data queues stay empty for 12 memory clocks, and selectively brought high as requests to their respective banks/DIMMs are detected. 0 Disabled 1 Enabled	R/W	0b0
7	Unused	Writes have no effect; reads are undefined.	R	undefined

Bits	Field Name	Description	Access	Reset
8	XSR DELAY	This bit delays the exit from self-refresh (sleep mode only). When the bit is set it allows extra time for the clock to wake up and become stable. The memory controller will not exit self-refresh (raise CKE) while this bit is set in the event that the clock is not yet stable. This bit can be set before entering self-refresh and then cleared once the clock is stable. **Note:** The use of this bit is mandatory. ddr_clk will tristate during sleep and this bit must be used to guarantee that memory does not exit self-refresh until the ddr_clk has resumed clocking. 0 Exits self-refresh normally 1 Will not exit self-refresh	R/W	0b0
9:11	Unused	Writes have no effect; reads are undefined.	R	undefined
12	Idle_ BusEn	This determines whether output enable remains asserted when bus is idle or output enable is deasserted and mode control termination turned off when bus is idle and no requests are in the queues. 0 Asserted 1 De-asserted	R/W	0b0
13	CkeTsEn	Specifies whether CKE remains deasserted or tristates during sleep. 0 De-asserted 1 Tri-state	R/W	0b0
14:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.25 Mem PHY Mode Control Register (MemPhyModeCntl)

This register contains the delay adjusts for the address and command buses – DDR_MA, DDR_RAS, DDR_CAS, DDR_WE, DDR_CKE, DDR_CS, and DDR_ODT. The default setting is 0b00, which specifies a minimum delay through the delay adjust pipeline. Incrementing the DelayAdjust field by one causes the relevant bus to be registered one additional ddr_clk.

The DDR_CKE bus is split into two 4-bit wide lower and upper segments. Normally these 2 fields will have the same value.

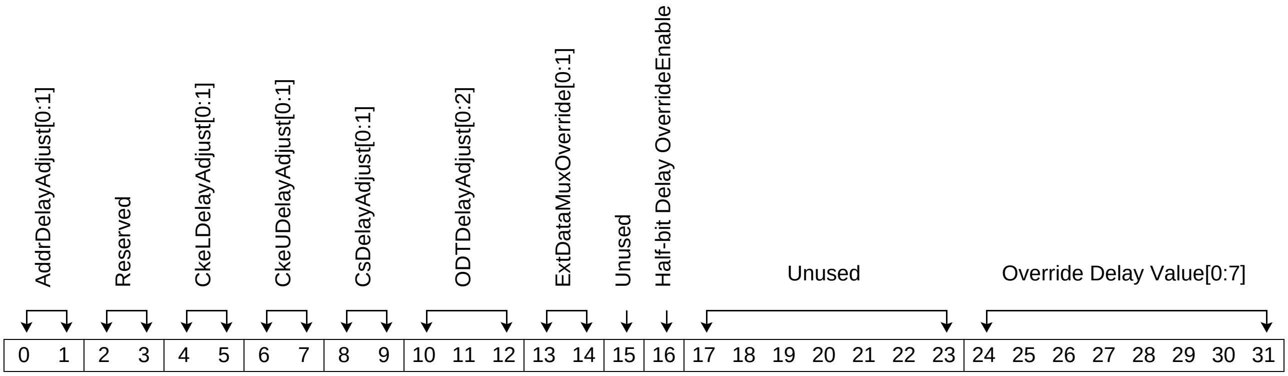
This register also contains a control field to override the automatic selection of the verniers used for controlling the optional external data muxes.

The override bit and override value are used to control the 1/2 bit time offset used by the DQS write and read verniers. By default the 1/2 Bit Time Offset is measured by a hardware unit and sent to the DQS verniers. If the override bit is set, the override value in bits 24:31 is used instead.

Reset Value 0x00000000

Offset 0xF8002880

Access Type Read/Write, Read Only



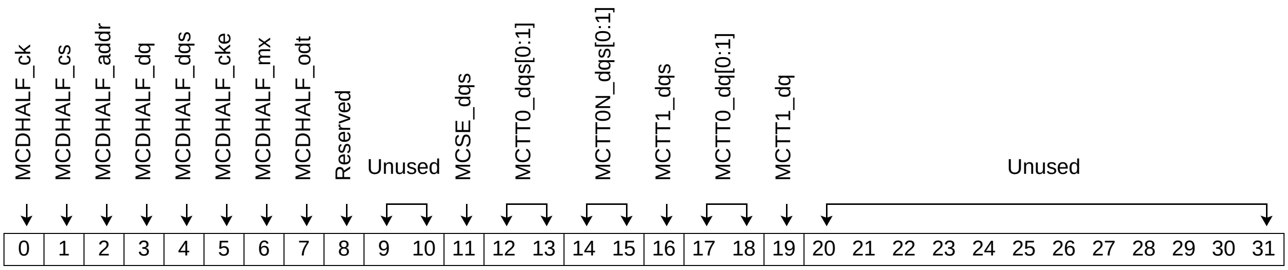
Bits	Field Name	Description	Access	Reset
0:1	AddrDelayAdjust [0:1]	This specifies a timing adjustment for ADDR. The ADDR could be directly driven to the pad or registered through 1, 2, or 3 ddr_clk delay stages and then sent to the output. 0 Minimum delay 1 Registered 1 ddr_clk cycle 2 Registered 2 cycles 3 Registered 3 cycles	R/W	0b00
2:3	Reserved	These bits must be set to 00.	R/W	0b00
4:5	CkeLDelayAdjust [0:1]	This specifies a timing adjustment for CSKE[0:3]. The signals could be directly driven to the pad or registered through 1, 2, or 3 ddr_clk delay stages and then sent to the output. 0 Minimum delay 1 Registered 1 ddr_clk cycle 2 Registered 2 cycles 3 Registered 3 cycles	R/W	0b00

Bits	Field Name	Description	Access	Reset
6:7	CkeUDelayAdjust [0:1]	This specifies a timing adjustment for CSKE[4:7]. The signals could be directly driven to the pad or registered through 1, 2, or 3 ddr_clk delay stages and then sent to the output. 0 Minimum delay 1 Registered 1 ddr_clk cycle 2 Registered 2 cycles 3 Registered 3 cycles	R/W	0b00
8:9	CsDelayAdjust[0:1]	This specifies a timing adjustment for CS. The signals could be directly driven to the pad or registered through 1, 2, or 3 ddr_clk delay stages and then sent to the output. 0 Minimum delay 1 Registered 1 ddr_clk cycle 2 Registered 2 cycles 3 Registered 3 cycles	R/W	0b00
10:12	ODTDelayAdjust [0:2]	This specifies a timing adjustment for ODT. The signals could be directly driven to the pad or registered through 1, 2, or up to 7 additional ddr_clk delay stages and then sent to the output. Outside of the phy, Read ODT is driven a fixed one mem_clk later than this setting, which is the delay for write ODT. 0 Minimum delay 1 Registered 1 ddr_clk cycle 2 Registered 2 cycles … 7 Registered 7 cycles	R/W	0b000
13:14	ExtDataMux Override[0:1]	This field specifies which of two verniers, read vernier and write vernier is used to delay the external multiplexer selects. Setting this to a nonzero value overrides the internal (to the DDR2 controller) read v/s write switching of the verniers. That is the default operation is to let the DDR2 controller pick the appropriate vernier, based on the command (read or write). (See *Section 12.10.31 External Data Multiplexer Delay Registers (ExtMuxVernier)* on page 495) 00 Default, automatic switching for read versus write 01 Use read vernier delay for all operations (read, write) 10 Use write vernier delay for all operations (read, write) 11 Unused	R/W	0b00
15	Unused	Writes have no effect; reads are undefined.	R	undefined
16	Half-bit Delay OverrideEnable	This specifies that the half-bit delay offset be overridden with the appropriate override value. 0 No override 1 Override	R/W	0b0
17:23	Unused	Writes have no effect; reads are undefined.	R	undefined
24:31	Override Delay Value[0:7]	The override delay setting for a half-bit time	R/W	0x00

12.10.26 I/O Pad Control Register (IOPadCntl)

These are the mode control bits for the I/O pads. Two types of driver/receivers are used: a three state driver/receiver and a differential driver/receiver. Control bits below are for both unless stated otherwise.

Reset Value	0x00000000
Offset	0xF80029A0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	MCDHALF_ck	Mode control to specify output impedance for clock pads 0 SSTL 18 1 Half Strength	R/W	0b0
1	MCDHALF_cs	Mode control to specify output impedance for CS pads 0 SSTL 18 1 Half strength	R/W	0b0
2	MCDHALF_addr	Mode control to specify output impedance for Address/RAS/CAS/WE pads 0 SSTL 18 1 Half strength	R/W	0b0
3	MCDHALF_dq	Mode control to specify output impedance for data pads 0 SSTL 18 1 Half strength	R/W	0b0
4	MCDHALF_dqs	Mode control to specify output impedance for DQS pads 0 SSTL 18 1 Half Strength	R/W	0b0
5	MCDHALF_cke	Mode control to specify output impedance for CKE pads 0 SSTL 18 1 Half strength	R/W	0b0
6	MCDHALF_mx	Mode control to specify output impedance for the external data multiplexer select pads 0 SSTL 18 1 Half strength	R/W	0b0
7	MCDHALF_odt	Mode control to specify output impedance for ODT pads 0 SSTL 18 1 Half strength	R/W	0b0
8	Reserved	This bit must be set to 0	R/W	0b0
9:10	Unused	Writes have no effect; reads are undefined.	R	undefined

Bits	Field Name	Description	Access	Reset
11	MCSE_dqs	Mode control to specify single-ended or differential driver/receiver strobe pad configuration 0 Differential 1 Single ended	R/W	0b0
12:13	MCTT0_dqs[0:1]	Selects whether termination for the bidirectional strobe pad is determined from DQSEnable (termination is on for reads and off for writes) or is always on or is always off. MCTT0 controls the termination at the PAD output pin. 0x Termination always off 10 Termination based on dsqEn (OE) On for reads, off for writes 11 Termination always on	R/W	0b00
14:15	MCTT0N_dqs[0:1]	Selects whether termination for the bidirectional strobe pad is determined from DQSEnable (termination is on for reads and off for writes) or is always on or is always off. MCTT0N controls the termination at the PADN output pin. 0x Termination always off 10 Termination based on dsqEn (OE) On for reads, off for writes 11 Termination always on	R/W	0b00
16	MCTT1_dqs	Selects the termination impedance setting for the bidirectional strobe pad 0 75 Ω 1 150 Ω	R/W	0b0
17:18	MCTT0_dq[0:1]	Selects whether termination for the DQ (data) pads is determined from DQSEnable (termination is on for reads and off for writes) or is always on or is always off 0x Termination always off 10 Termination based on dsqEn (OE) On for reads, off for writes 11 Termination always on	R/W	0b00
19	MCTT1_dq	Selects the termination impedance setting for the DQ data pads 0 75 Ω 1 150 Ω	R/W	0b0
20:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.27 Write Strobe Control Registers (ByteWrClkDelay)

The write clock delay is adjusted through the use of two delay stacks or chains, each spanning up to roughly a half-bit time and this delayed write clock drives the output registers for data (DQ and OE). A single register, WrClkOffset, is used to program the two delay stacks, each delay stage gets half the value. The write strobe is adjusted by an additional delay to center the strobe in the data window. There are two separate such delays in each byte lane, one for each of the two strobes. The additional strobe delay adjust is specified as a delta offset, added or subtracted from the half-bit time delay value. The lower and upper strobe offsets (DeltaL and DeltaU respectively) drive the APD and APN pins on the strobe pads, and can be independently adjusted for ×4 mode of operation. In ×8 mode, it is recommended that both the offsets be programmed to the same value for consistency but this is not required for correct operation. Only the DeltaL offset is referred to for ×8 mode.

Reset Value	
Offset	0xF8002800 (ByteWrClkDelayC0B00) 0xF8002810 (ByteWrClkDelayC0B01) 0xF8002820 (ByteWrClkDelayC0B02) 0xF8002830 (ByteWrClkDelayC0B03) 0xF8002900 (ByteWrClkDelayC1B04) 0xF8002910 (ByteWrClkDelayC1B05) 0xF8002920 (ByteWrClkDelayC1B06) 0xF8002930 (ByteWrClkDelayC1B07) 0xF8002980 (ByteWrClkDelayC1B16) 0xF8002A00 (ByteWrClkDelayC2B08) 0xF8002A10 (ByteWrClkDelayC2B09) 0xF8002A20 (ByteWrClkDelayC2B10) 0xF8002A30 (ByteWrClkDelayC2B11) 0xF8002B00 (ByteWrClkDelayC3B12) 0xF8002B10 (ByteWrClkDelayC3B13) 0xF8002B20 (ByteWrClkDelayC3B14) 0xF8002B30 (ByteWrClkDelayC3B15) 0xF8002B80 (ByteWrClkDelayC3B17)
Access Type	Read/Write, Read Only

WrClkOffset[0:7]								Unused								WrClkOffsetDeltaL[0:7]								WrClkOffsetDeltaU[0:7]							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:7	WrClkOffset[0:7]	Write clock delay offset Dual delay chain spans a full bit time (with setup and hold margin). The two delay stacks each get half the value programmed into this field. 8-bit unsigned magnitude	R/W	0x00
8:15	Unused	Writes have no effect; reads are undefined.	R	undefined
16:23	WrClkOffsetDeltaL[0:7]	Write clock delay lower strobe delta offset 1-bit sign, 7-bit magnitude for half-bit +/- offset, with (negative) saturation clamping (LDQS)	R/W	0x00
24:31	WrClkOffsetDeltaU[0:7]	Write clock delay upper strobe delta offset 1-bit sign, 7-bit magnitude for half-bit +/- offset, with (negative) saturation clamping (UDQS)	R/W	0x00

12.10.28 Read Data Strobe Control Registers (ReadStrobeDelay)

The read strobe is passed through delay chains to derive the rising (R) and falling (F) edge clocks that feed the DQ data FIFO. Separate delay adjusts are used for each edge and also for the two strobes in each byte lane ((L)ower and (U)pper). In the ×4 mode of operation, the lower and upper strobe offsets (DeltaLR and DeltaLF as a pair and DeltaUR and DeltaUF as a pair) are driven by the ZPD and ZPN pins on the strobe pads respectively, and can be independently programmed. In ×8 mode, the ZDF pad output is fed to two both delay offset chains and hence, the same value must be programmed into both fields. For example, in ×8 mode, DeltaLR should always be set equal to DeltaUR, and DeltaLF should always be set equal to DeltaUF.

Reset Value	0x00000000
Offset	0xF8002840 (ReadStrobeDelayC0B00) 0xF8002850 (ReadStrobeDelayC0B01) 0xF8002860 (ReadStrobeDelayC0B02) 0xF8002870 (ReadStrobeDelayC0B03) 0xF8002940 (ReadStrobeDelayC1B04) 0xF8002950 (ReadStrobeDelayC1B05) 0xF8002960 (ReadStrobeDelayC1B06) 0xF8002970 (ReadStrobeDelayC1B07) 0xF8002990 (ReadStrobeDelayC1B16) 0xF8002A40 (ReadStrobeDelayC2B08) 0xF8002A50 (ReadStrobeDelayC2B09) 0xF8002A60 (ReadStrobeDelayC2B10) 0xF8002A70 (ReadStrobeDelayC2B11) 0xF8002B40 (ReadStrobeDelayC3B12) 0xF8002B50 (ReadStrobeDelayC3B13) 0xF8002B60 (ReadStrobeDelayC3B14) 0xF8002B70 (ReadStrobeDelayC3B15) 0xF8002B90 (ReadStrobeDelayC3B17)
Access Type	Read/Write

RdStrOffsetDeltaLR[0:7]	RdStrOffsetDeltaLF[0:7]	RdStrOffsetDeltaUR[0:7]	RdStrOffsetDeltaUF[0:7]
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

Bits	Field Name	Description	Access	Reset
0:7	RdStrOffsetDeltaLR[0:7]	Read lower strobe delta offset rising edge 1-bit sign, 7-bit magnitude for half-bit +/- offset, with (negative) saturation clamping	R/W	0x00
8:15	RdStrOffsetDeltaLF[0:7]	Read lower strobe delta offset falling edge 1-bit sign, 7-bit magnitude for half-bit +/- offset, with (negative) saturation clamping x	R/W	0x00
16:23	RdStrOffsetDeltaUR[0:7]	Read upper strobe delta offset rising edge 1-bit sign, 7-bit magnitude for half-bit +/- offset, with (negative) saturation clamping	R/W	0x00
24:31	RdStrOffsetDeltaUF[0:7]	Read upper strobe delta offset falling edge 1-bit sign, 7-bit magnitude for half-bit +/- offset, with (negative) saturation clamping	R/W	0x00

12.10.29 CK Control Registers (CKDelay)

The 1x clock is generated through the use of three delay stacks in series. The first two stacks each span up to roughly a half-bit time and the cumulative delay through the pair, is typically set to match the write DQ/DQS delay adjusts. A single register, CKDelayOffset, is used to program the two delay stacks. Each delay stage gets half the value. The third and final delay chain is used to push the signal out by an additional delta offset.

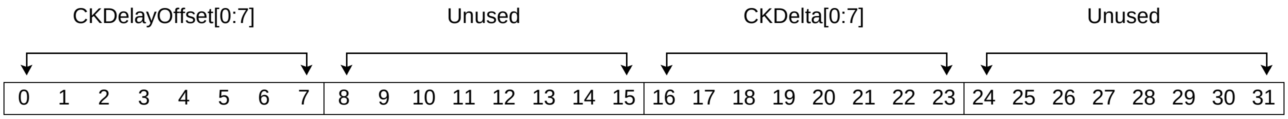
There is a special override control on the delta offset. When the delta offset field is all zeros (that is, its default value, coming out of hardware reset), the override logic will force a value of 0x20 onto the delta offset control. This is meant to guarantee that the 1× clk has sufficient hold time margin with respect to the address and command bits. This constant override value of 0x20 corresponds to approximately 800 ps to 1600 ps of hold margin, depending on the process corner, and it provides sufficient hold and setup margins for DDR2 266/400/533 MHz operation.

When a nonzero value is programmed into the delta offset field, the override logic is implicitly disabled and the user value is passed through, unmodified.

Reset Value 0x00000000

Offset 0xF8002890
0xF80028A0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	CKDelayOffset [0:7]	1× Clk (CK) delay stack offset. Dual delay chain spans a full bit time (with setup and hold margin). The two delay stacks each get half the value programmed into this field. Encoding: 8-bit unsigned magnitude.	R/W	0x00
8:15	Unused	Writes have no effect; reads are undefined.	R	undefined
16:23	CKDelta [0:7]	1× Clk (CK) delta offset. 8-bit unsigned pure offset delay (for example, no half-cycle adjustment). See notes above, on the hardware override control.	R/W	0x00
24:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.30 Reset LdEn Offset Delay Registers (RstLdEnVerniersCn)

The Reset LdEn offset delay register is used to delay tune the reset control to the FIFO load pointers. There is a dual stack, each delay spanning up to roughly a half-bit time. This delay offset is shared by two bytes in each cluster (or group) of four byte-lanes (and one delay offset is shared by up to three byte-lanes in the ECC clusters). The cluster grouping of the various byte lanes is described in more detail later, in the calibration configuration register section.

Reset Value	0x00000000
Offset	0xF80028D0 (RstLdEnVerniersC0) 0xF80029D0 (RstLdEnVerniersC1) 0xF8002AD0 (RstLdEnVerniersC2) 0xF8002BD0 (RstLdEnVerniersC3)
Access Type	Read/Write, Read Only

ResetLdEnOffset[0:7]								ResetLdEnOffset[0:7]								Unused															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:7	ResetLdEn Offset[0:7]	Single delay stack only, shared across two byte lanes (three for the ECC cluster). Dual delay chain spans a full bit time (with setup and hold margin). The two delay stacks each get half the value programmed into this field. Encoding: 8-bit unsigned	R/W	0h00
8:15	ResetLdEn Offset[0:7]	Single delay stack only, shared across two byte lanes. Dual delay chain spans a full bit time (with setup and hold margin). The two delay stacks each get half the value programmed into this field. Encoding: 8-bit unsigned	R/W	0h00
16:31	Unused	Writes have no effect; reads are undefined.	R	0h00

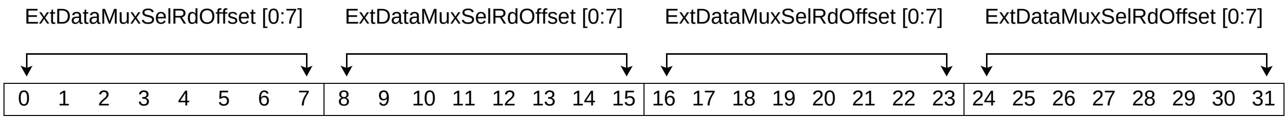
12.10.31 External Data Multiplexer Delay Registers (ExtMuxVernier)

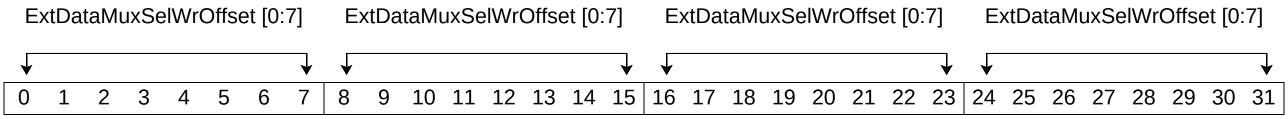
The external data multiplexer selects are delay tuned by programming two separate registers, one for reads and one for writes. The hardware controller automatically switches to the correct vernier based on read versus write commands. See *Section 12.10.25 Mem PHY Mode Control Register (MemPhyModeCntl)* on page 485 for a description of how this default behavior can be overridden statically. There are four sets or pairs of multiplexer selects, each pair has two bits that drive the 4:1/1:4 multiplexer / demultiplexer. Hence a total of eight verniers, four read adjusts and four write adjusts, are provided – bits 0:7 of register F80028B0 control bits [0:1] of the external multiplexer select bus for read operations, bits 8:15 control selects [2:3] and so on.

Reset Value 0x00000000

Offset 0xF80028B0 (ExtMuxVernier0)
0xF80028C0 (ExtMuxVernier1)

Access Type Read/Write


ExtDataMuxSelRdOffset [0:7]
ExtDataMuxSelRdOffset [0:7]
ExtDataMuxSelRdOffset [0:7]
ExtDataMuxSelRdOffset [0:7]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31


ExtDataMuxSelWrOffset [0:7]
ExtDataMuxSelWrOffset [0:7]
ExtDataMuxSelWrOffset [0:7]
ExtDataMuxSelWrOffset [0:7]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Bits	Field Name	Description	Access	Reset
0:7, 8:15, 16:23, 24:31	ExtDataMuxSel RdOffset [0:7]	External data multiplexer read offset. Single delay stack only, in each cluster, for the two bit encoded selects. 8-bit unsigned 0:7 Mux Sel[0:1] 8:15 Mux Sel[2:3] 16:23 Mux Sel[4:5] 24:31 Mux Sel[6:7]	R/W	0x00
0:7, 8:15, 16:23, 24:31	ExtDataMuxSel WrOffset [0:7]	External data multiplexer write offset. Single delay stack only, in each cluster, for the two bit encoded selects. 8-bit unsigned 0:7 Mux Sel[0:1] 8:15 Mux Sel[2:3] 16:23 Mux Sel[4:5] 24:31 Mux Sel[6:7]	R/W	0x00

12.10.32 Calibration Control and Delay Measurement Registers

The DDR2 PHY contains a calibration unit which might optionally be used to assist in determining values to be programmed into the controls for capturing read data. The calibration unit has 2 modes of operation: single-step (manual) and autocalibration.

As discussed in *Section 7.22.2 Clusters* on page 214, the PHY is divided into 4 clusters:

Cluster 0 : Byte lanes 0, 1, 2 and 3

Cluster 1 : Byte lanes 4, 5, 6, 7. Also byte lane 16 for ECC.

Cluster 2 : Byte lanes 8, 9, 10 and 11

Cluster 3: Byte lanes 12, 13, 14, 15. Also byte lane 17 for ECC.

Each cluster has its own calibration unit. In general all four clusters will be used in parallel to perform a given calibration or measurement.

In single-step mode bit 0, Reset Cal Registers (Control and Delay Measurement Registers, this section) is used to initiate a calibration step. The results are made available in the Calibration Read Margin Result registers (*Section 12.10.34* on page 502).

In autocalibration mode bit 1, Start Autocalibration State Machine (Control and Delay Measurement registers, this section) is used to initiate a series of calibration steps. Bit 3, autocalibration done, indicates when the FSM (finite state machine) has completed the series. In autocalibration mode a timing vernier will be adjusted between steps. The final value of this vernier (the “delay measurement”) is made available in bits 8:15, CalFSMSettingResult.

The CalFSMSettingResult also encodes two special conditions – the value 0xFF signifies that the delay offset setting exceeded the user-programmed limit (see *Section 12.10.33 Calibration Configuration Registers* on page 499) and the autocalibration measurement was hence terminated. The value 0xFE signals that an internal time-out error was encountered. The timeout is implicitly set to (2^{31}) - 1 ddr_clk cycles, if an insufficient numbers of read operations were encountered in that time interval, the autocalibration is terminated.

The Reset bit (single-step mode) and the Start bit (autocalibration mode) are written to a ‘1’ by the programmer to initiate the action and they get automatically cleared to a ‘0’ by the hardware later, no programmer update required. The Done bit (autocalibration mode) is set to a ‘1’ by the autocalibration state machine to indicate that the result is ready (or an error condition occurred) and it must be cleared by the programmer before subsequent runs.

The Calibration Configuration 0 and 1 Registers (*Section 12.10.33* on page 499) specify the calibration options and are common to all four clusters.

12.10.32.1 Half Bit Time Measurement Results

As discussed in *“1/2 Bit Time Averager” on page 216* and shown in *Figure 7-8* on page 218, the DDR2 PHY contains a 1/2 bit time measurement unit in Cluster 0. The outputs of this unit are shared with 2 calibration registers.

The full and half-bit time measurement results can be read in bits 16:31 of the CalCntlDlyMeasC0 Register (this section).

Half-bit time status bits can be read in bits 29:31 of the Calibration Read Margin Result C0 Register (*Section 12.10.34 Calibration Read Margin Result Registers* on page 502).

Calibration Control and Delay Measurement Registers

Register	Reset Value	Offset	Access Type
CalCntlDlyMeasC0	0x00008843	0xF80028F0	Read/Write, Read Only
CalCntlDlyMeasC1	0x00000000	0xF80029F0	
CalCntlDlyMeasC2	0x00000000	0xF8002AF0	
CalCntlDlyMeasC3	0x00000000	0xF8002BF0	

CntlDlyMeasC0 Register

Cal Control	Cal FSM Setting Result	Full Bit Time Measurement[0:7]	Half bit time average[0:7]
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

CntlDlyMeasC1, CntlDlyMeasC2, CntlDlyMeasC3 Registers

Cal Control	Cal FSM Setting Result	Unused	Unused
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

Bits	Field Name	Description	Access	Reset
0:7	Cal Controll	Control and status byte for calibration unit. In single-step mode, the Reset Cal bit is used to place all calibration monitor registers or latches into a known state and ready for subsequent measurement(s). In autocalibration mode, the Start bit is used to initiate an autocalibration measurement cycle, the Done bit indicates the completion of the cycle. The Reset and the Start bits are written to a ‘1’ by the programmer to initiate the action and they get automatically cleared to a ‘0’ by the hardware later. The Done bit is set to a ‘1’ by the autocalibration state machine to indicate that the result is ready or an error condition occurred. The done bit must be cleared by the user before subsequent runs. (*) Writing a ‘1’ into one or more bits of this field triggers an internal pulse generator that will implicitly clear the bit(s) to zeros. Hence, reading this field back subsequently returns zeros. Writing all zeros to this field will clear all bits. Encoding: [0] Reset calibration registers [1] Start autocalibration state machine [2] Unused [3] Autocalibration done 1 Done 0 In-progress (or not started) [4:7] Unused	R/W*	0x00
8:15	CalFSMSetting Result	The result of the autocalibration cycle (the “delay measurement”), written by the hardware upon completion, as indicated by the Done bit. There are different modes of operation and the result signifies various different margins, as configured by the autocalibration config 0 register (*Section 12.10.33* on page 499). (*) Writing a ‘1’ into one or more bits of this field triggers an internal pulse generator that implicitly clears the bit(s) to zeros. Hence, reading this field back subsequently returns zeros. Writing all zeros to this field clears all bits. Encoding: [0:7]: Value read is the measurement expressed in increments of the delay chain. Special (Error) Values: 0xFF Delay setting limit exceeded, autocalibration terminated 0xFE Read timeout, autocalibration terminated	R/W*	0x00
16:23	Full Bit Time Measurement [0:7]	Full bit time measurement from the on-chip cycle time calibration register, value read is the bit time or ddr_clk, expressed in increments of the delay chain. The measurement is expressed in increments of the delay chain.	R	0x00
24:31	Half bit time average[0:7]	Half bit time measurement from the on-chip cycle time calibration register, value read is the 1/2 bit time or ddr_clk, expressed in increments of the delay chain and averaged over several measurements. The measurement is expressed in increments of the delay chain.	R	0x00

12.10.33 Calibration Configuration Registers

The two calibration configuration registers, CalConf0 (0xF80029B0) and CalConf1 (0xF80029C0) are common to all 4 clusters.

The Calibration Configuration 0 Register contains settings for the read margin measurement and calibration unit. This register must be programmed for any calibration type.

The Calibration Configuration 1 Register contains a vernier control setting used by the calibration unload monitor. It needs to be programmed only for the unload offset measurement mode, when running the calibration unit in single-step mode.

The calibration unit can be run in one of two modes: single-step or autocalibration.

In single-step mode, the user programs the various delay offsets. The pass/fail data, that is, Calibration Read Margin Result Registers are then reset to a known, initial state by writing a ‘1’ to the ‘Reset Cal Registers’ bit in the each of the Calibration Status and Delay Measurement Registers. Next, several read commands are issued and the Calibration Read Margin Result Registers continuously accumulate the margin measurement pass/fail results and can be read at any time. Note that the ‘Reset Cal Registers’ bit can also be written to a ‘1’ at any time to restart the measurement.

In autocalibration mode, a state machine automatically steps through a range of delay offsets for a specified test mode and it monitors the calibration result to derive pass and fail status. Based on the result, it continually adjusts the delay settings until it locks onto the final result. This is saved in the Calibration Control and Delay Measurement Register in the CalFSMSetting field, one for each cluster. The state machines share configuration settings, but each cluster has its own independent control and status. Note that the state machines cannot be interrupted without an explicit reset.

The auto mode is initiated by writing a ‘1’ to the ‘Start auto cal’ bit in the control register, the state machine indicates completion or an error condition by writing a ‘1’ to the ‘Done’ bit in the same register.

The autocalibration logic compares the load result (one nibble per byte lane) or the unload result (single bit per byte lane) against the four bit Pass Result field in the Calibration Configuration 0 Register, using the 4-bit pass mask, to determine if the current measurement qualifies as a pass or a fail.

Since each cluster has a minimum of four byte lanes, the byte lane select field is used to pick one of the lanes for auto-calibration. There are four different measurement modes:

- Read Strobe Eye Width (rising hHalf)
- Read Strobe Eye Width (falling half)
- Reset Load Enable Margin (read preamble)
- Unload Clock Offset

The first three modes above correspond to the transfer of data bits, read from the memory DIMMs into a data FIFO in the memory controller and the fourth mode checks the unloading of the data FIFO, as it is transferred from the external read strobe domain to the internal (controller) clock domain. Since the data FIFO has four entries, the load calibration monitors store four bits for each byte lane and a single bit for the unload result, in each byte lane. (See *Section 12.10.34 Calibration Read Margin Result Registers* on page 502.)

In Unload Clock Offset measurement mode Calibration Configuration 1 supplies the unload clock offset value.

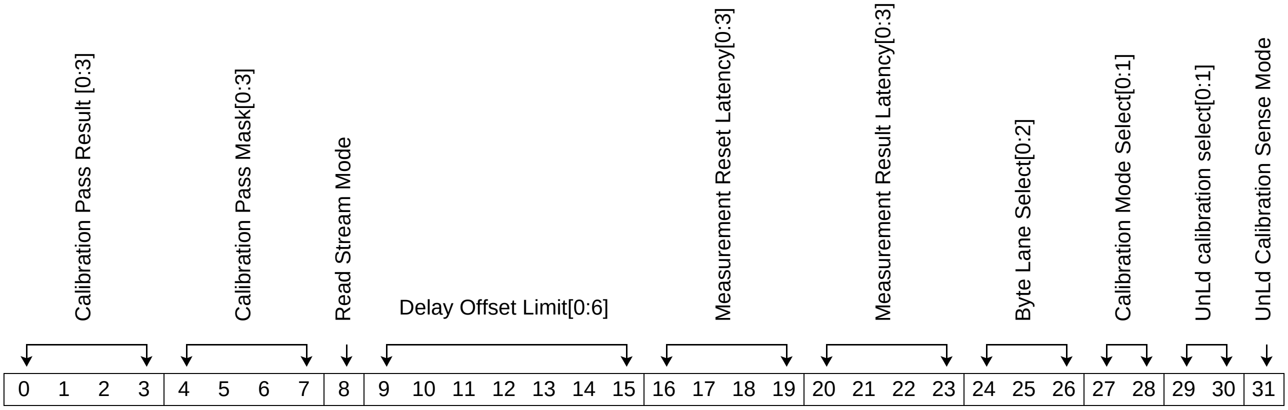
In single-fstep mode, the byte lane select and the mode select are irrelevant. The programmer simply reads the measurement result from the register, and the result is available across all the byte lanes in the cluster.

The eighteen byte lanes (for a total of 144 data bits, big-endian layout) are distributed across four clusters in this manner:

- Cluster 0 : Byte lanes 0, 1, 2, and 3
- Cluster 1 : Byte lanes 4, 5, 6, 7, and 16 (ECC)
- Cluster 2 : Byte lanes 8, 9, 10, and 11
- Cluster 3 : Byte lanes 12, 13, 14, 15, and 17 (ECC)

Thus, a byte lane select of 0bb001 picks byte lanes 1, 5, 9 and 13 in the four clusters.

Reset Value	0x00000000
Offset	0xF80029B0
Access Type	Read/Write


Calibration Pass Result [0:3]
Calibration Pass Mask[0:3]
Read Stream Mode
Delay Offset Limit[0:6]
Measurement Reset Latency[0:3]
Measurement Result Latency[0:3]
Byte Lane Select[0:2]
Calibration Mode Select[0:1]
UnLd calibration select[0:1]
UnLd Calibration Sense Mode
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

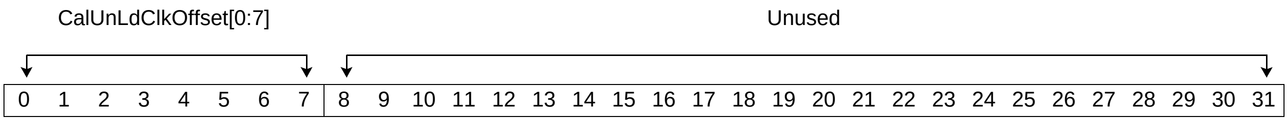
Bits	Field Name	Description	Access	Reset
0:3	Calibration Pass Result [0:3]	This is the expected load monitor result (a nibble per byte lane), which qualifies as a pass. For unload calibration, only bit[0] is used.	R/W	0x0
4:7	Calibration Pass Mask[0:3]	A 4-bit mask field is used to determine which of the four bits in the pass result field are checked to determine the pass/fail status. For unload calibration, only bit[0] is used. 1: Sample this bit 0: Ignore (or mask) this bit.	R/W	0x0
8	Read Stream Mode	This specifies whether the read commands issued for calibration are in a continuous stream, with minimum delay or are pulsed (for example, spaced apart by several cycles). 0: Pulsed Mode 1: Continuous Mode	R/W	0b0
9:15	Delay Offset Limit [0:6]	This specifies the upper bound on the delay offset driven from the autocalibration state machine, as it attempts to get to the final result. The limit is in delay chain increments and is added or subtracted on top of the full bit time setting. 7-bit sign magnitude, in delay chain increments.	R/W	0x00

Bits	Field Name	Description	Access	Reset
16:19	Measurement Reset Latency[0:3]	This specifies, in bit clock (ddr_clk) increments, how long the calibration state machine waits after a read command is issued before the result monitors are reset to a known state (that is, at the start of each measurement) to determine pass/fail status. In the hardware, the reset latency is implicitly set to match the controller ResMuxDel. This field is a sign-magnitude delta offset which can be used to increment or decrement over the base delay - ResMuxDel. 4-bit sign magnitude, bit 0 is the sign bit, bits 1:3 are the magnitude. Actual Reset Latency = Controller ResMuxDel +/- Offset	R/W	0x0
20:23	Measurement Result Latency[0:3]	This specifies, in bit clock (ddr_clks) increments, how long the calibration state machine waits after a read command is issued and before the result monitors are read to determine pass/fail status. The delay is implicitly incremented by the Measurement Reset Latency Field. This field is a positive-only delta offset. Actual Result Latency = Actual Reset Latency + Offset 4-bit magnitude only	R/W	0x0
24:26	Byte Lane Select [0:2]	Selects byte lane, in each cluster, for autocalibration measurements. Only valid for 0 - 4. n Cluster Byte Lane n	R/W	0b000
27:28	Calibration Mode Select[0:1]	Specifies the mode for autocalibration. 00 Read Strb (rising) 01 Read Strb (falling) 10 ResetLdEn 11 Unload clock offset	R/W	0b00
29:30	UnLd calibration select[0:1]	Specifies which of the four latches in the read FIFO is sampled by the UnLd calibration logic. n Read FIFO bit n	R/W	0b00
31	UnLd Calibration Sense Mode	Specifies the Sense mode ONLY for the UnLd calibration monitor register. Late mode measures the latest arriving signal transition, early mode measures the earliest arriving signal transition. For the read strobe and resetLdEn modes, the sense mode is implicitly set to early. Changing this bit has no effect in the strobe and resetLdEn modes. 0 Late mode 1 Early mode	R/W	0b0

Reset Value 0x00000000

Offset 0xF80029C0

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:7	CalUnLdClkOffset [0:7]	The clock insertion delay of the calibration monitor register used for data unload margin testing	R/W	0x00
8:31	Unused	Writes have no effect; reads are undefined.	R	undefined

12.10.34 Calibration Read Margin Result Registers

There are four different measurement or calibration modes:

- Read Strobe Eye Width / Margin (Rising Half)
- Read Strobe Eye Width / Margin (Falling Half)
- Reset Load Enable Margin (Read Preamble)
- Unload Clock Offset

The first three modes above correspond to the transfer of data bits, read from the memory dimms to a data FIFO in the memory controller and the fourth mode checks the unloading of the read data in the memory controller, as it is transferred from the external read strobe domain to the internal controller clock domain.

The calibration can be done in two modes – single-stepping and auto-calibration, each with its own start bit in register(s) 0xF8002*F0 as described previously. Note how each cluster has its own control in the 0xF8002*F0 register(s). The setting of the 'done' flag in the same register signals the end of auto-calibration.

The unload result is the measurement from one sticky latch on the unload side of the read data FIFO, one per byte lane. The load result is the measurement from four sticky latches on the load side of the read data FIFO, a nibble per byte lane. The 'stickiness' refers to the mechanism by which the pass or fail status is accumulated over a sequence of several read operations. The clusters 0 through 3 contain the following byte lanes:

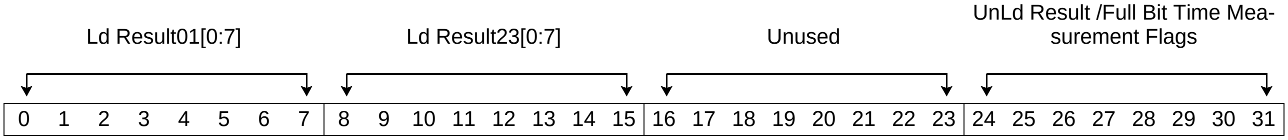
- Cluster 0 : Byte lanes 0, 1, 2 and 3
- Cluster 1 : Byte lanes 4, 5, 6, 7 and 16 (ECC)
- Cluster 2 : Byte lanes 8, 9, 10 and 11
- Cluster 3: Byte lanes 12, 13, 14, 15 and 17 (ECC)

The above mapping is to be used when decoding the byte lane references in the following description. For instance, a reference to byte lane 0 in the table below would correspond to byte lane 0 in cluster 0, byte lane 4 in cluster 1, byte lane 8 in cluster 2 and byte lane 12 in cluster 3. Similarly, byte lane 4 in the following description refers to one of the ECC byte lanes, 16 or 17.

As discussed in *Section 12.10.32.1 Half Bit Time Measurement Results* on page 497 the CalRsltC0 Register is shared: bits 29:31 contain the status bits from the Half Bit Time Measurement unit.

Reset Value	0x00000000
Offset	0xF80028E0 (CalRsltC0) 0xF80029E0 (CalRsltC1) 0xF8002AE0 (CalRsltC2) 0xF8002BE0 (CalRsltC3)
Access Type	Read/Write, Read Only

CalRsltC0 Register



CalRsltC2 Register

Ld Result01[0:7]	Ld Result23[0:7]	Unused	UnLd Result
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

CalRsltC1 and CalRsltC3 Registers

Ld Result01[0:7]	Ld Result23[0:7]	LdResultECC	UnLd ResultECC
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

Bits	Field Name	Description	Access	Reset
0:7	Ld Result01[0:7]	Calibration Load results from byte lanes 0 and 1 in each cluster. [0:3] Load result from cluster byte lane 0 [4:7] Load result from cluster byte lane 1	R/W*	0x0
8:15	Ld Result23[0:7]	Calibration Load results from byte lanes 2 and 3 in each cluster. [0:3] Load result from cluster byte lane 2 [4:7] Load result from cluster byte lane 3	R/W*	0x0
0:7	UnLd Result[0:7]	Calibration Unload result. [0] Unload result from cluster byte lane 0 [1] Unload result from cluster byte lane 1 [2] Unload result from cluster byte lane 2 [3] Unload result from cluster byte lane 3 [4:7] Unused	R/W*	0x0
0:7	UnLd ResultEcc[0:7]	Calibration Unload result for ECC cluster [0] Unload result from cluster byte lane 0 [1] Unload result from cluster byte lane 1 [2] Unload result from cluster byte lane 2 [3] Unload result from cluster byte lane 3 [4] Unload result from cluster byte lane 4 (ECC) [5:7] Unused	R/W*	0x0
16:23	Ld ResultEcc[0:7]	Calibration Load results from byte lanes 4 (ECC) in each cluster. [0:3] Load result from cluster byte lane 4 (ECC) [4:7] Unused	R/W*	0x0
16:23	Unused	Writes have no effect; reads are undefined.	R	undefined
24:31	UnLd Result / Full Bit Meas Flags [0:7]	Calibration unload result (cluster) and Full Bit Time Measurement status flags [0] Unload result from cluster byte lane 0 [1] Unload result from cluster byte lane 1 [2] Unload result from cluster byte lane 2 [3] Unload result from cluster byte lane 3 [4] Unused [5] (C0 Only) Measurement done 1 Done 0 Not done [6] (C0 Only) Overflow (bit time too long) [7] (C0 Only) Underflow (bit time too short)	R/W*	0x00

(*) Writing a ‘1’ into one or more bits of this register triggers an internal pulse generator that will implicitly clear the bit(s) to zeros. Hence, reading this field back will subsequently return zeroes. Writing all zeros to this register will clear all bits. (This does not apply to C0 bits 29:31, the Half Bit Time Measurement unit status bits.)

12.11 PCI Express Registers

PCI Express registers are categorized in three groups:

- Configuration Registers defined by the PCI Express Specifications. Refer to *Section 12.11.1* , below.
- Expansion Registers unique to CPC945’s PCIe implementation. Refer to *Section 12.11.2* , below.
- General Control Registers for configuration control of the PCIe interface as implemented in CPC945. (See *Section 12.11.3 PCI Express GCR Registers* on page 584.)

Note: The register reset values as described in this section are shown as read through the I^{2}C slave interface port and are in big-endian format. This is byte reversed from what might be expected as the register descriptions are in Little Endian format, in compliance with the PCI specification.

12.11.1 PCIe Configuration Registers

The CPC945 PCIe Configuration Register space consists of four groups of registers as shown in *Table 12-19*. These register groups are defined in the PCIe Specification as PCI-compatible configuration space registers, PCI Power Management Capability register structure, PCIe capability register structure and Advanced Error reporting structure. If an optional extended capability is not described in the following sections, that capability is not supported in the CPC945.

Table 12-19. PCIe Configuration Registers Blocks

Offset	Block	Description	Page
0x000-0x03F	PCI 2.3	PCI Configuration Space Header (PCI)	506
0x040-0x047	Power Management (PM)	PCI Power Management Capability Structure (PM)	520
0x048-0x06B	PCI Express Capability	PCI Express Capability Structure (EC)	524
0x100-0x137	Advanced Error Reporting	Advanced Error Reporting Extended Capability Structure (AER)	538

The CPC945 uses a Type-1 PCI Header. The header fields and offsets are shown below.

Table 12-20. PCI Configuration Space Header

Register Name (Mnemonic)	Address DWord Offset	Access Mode	See Page
PCI - Vendor ID Register	0x00	Read Only	507
PCI - Device ID Register	0x00	Read Only	507
PCI - Command Register	0x04	Mixed	508
PCI - Status Register	0x04	Mixed	508
PCI - Revision ID Register	0x08	Read Only	509
PCI - Class Code Register	0x08	Read Only	509
PCI - Cache Line Size Register	0x0C	Read/Write	510
PCI - Master Latency Timer	0x0C	Read Only	510
PCI - Header Type Register	0x0C	Read Only	510
PCI - BIST Register	0x0C	Read Only	510
PCI - Base Address Register 0 (BAR0)	0x10	Mixed	510
PCI - Base Address Register 1 (BAR1)	0x14	Mixed	511
PCI - Bridge Primary Bus Number	0x18 (Type-1)	Read/Write	511
PCI - Bridge Secondary Bus Number	0x18 (Type-1)	Read/Write	511
PCI - Bridge Subordinate Bus Number	0x18 (Type-1)	Read/Write	511
PCI - Bridge Secondary Latency Timer	0x18 (Type-1)	Read Only	511
PCI - Bridge I/O Base Register	0x1C (Type-1)	Mixed	512
PCI - Bridge I/O Limit Register	0x1C (Type-1)	Mixed	512
PCI - Bridge Secondary Status Register	0x1C (Type-1)	Mixed	512
PCI - Bridge Memory Base Register	0x20 (Type-1)	Mixed	514
PCI - Bridge Memory Limit Register	0x20 (Type-1)	Mixed	514
PCI - Bridge Prefetchable Memory Base Register	0x24 (Type-1)	Mixed	515
PCI - Bridge Prefetchable Memory Limit Register	0x24 (Type-1)	Mixed	515
PCI - Bridge Prefetchable Base Upper 32 Bits Register	0x28 (Type-1)	Mixed	516
PCI - Bridge Prefetchable Limit Upper 32 Bits Register	0x2C (Type-1)	Mixed	516
PCI - Bridge I/O Base Upper 16Bits	0x30 (Type-1)	Mixed	512
PCI - Bridge I/O Limit Upper 16Bits	0x30 (Type-1)	Mixed	512
PCI - Capabilities Pointer Register	0x34	Read Only	517
PCI - Bridge Expansion ROM Base Address Register	0x38 (Type-1)	Mixed	518
PCI - Interrupt Line Register	0x3C	Read/Write	519
PCI - Interrupt Pin Register	0x3C	Read Only	519
PCI - Bridge Control Register (BCR)	0x3C (Type-1)	Mixed	519

12.11.1.1 PCI 2.3 Configuration Space Header

PCI – Vendor ID and Device ID Registers

The PCI vendor ID and device ID registers are read-only registers, used to communicate device identification information to the software operating system.

Reset Value 0x6B105B00

Address x0F1F00000

Access Type Read Only

Device_ID_Register																Vendor_ID_Register															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:16	Device_ID_Register	Register bit I/O signals: SYS_PCI00_DEVICE_ID	R	0x005B
15:0	Vendor_ID_Register	Register bit I/O signals: SYS_PCI00_VENDOR_ID	R	0x106B

PCI – Command and Status Registers

The PCI Command and Status registers are used to communicate device status and control information between the hardware and software.

Reset Value 0x0010 0007 (Note that this default is little Endian.)

Address Offset 0x04

Access Type Mixed

SR15_DetErr	SR14_DetErr	SR13_RecdMstrAbort	SR12_RecdTargtAbort	SR11_SignldTgtAbortr	Reserved	SR08_MstrDataParityErr	Reserved	SR04_ICapList	SR03_IntStatus	Reserved	CR15_11	CR10_IntDisable	Reserved	CR8_SERRenable	Reserved	CR6_PERRenable	Reserved	CR2_BusMstrEnable	CR1_MemSpEnable	CR0_IoSpEnable
31	30	29	28	27	26 25	24	23 22 21	20	19	18 17 16	15 14 13 12 11	10	9	8	7	6	5 4 3	2	1	0

Bits	Field Name	Description	Access	Reset
31	Status Reg 15 - Detected Parity Error	PCIe received poisoned TLP, primary bus for TYPE1 devices Field register bit I/O signals: AL_PCI04_REC_POISONED_TLP	R	0b0
30	Status Reg 14 - Signaled System Error	Set when PCIe devices send fatal or nonfatal Messages and the SERR Enable bit in the Command Register (PCI04[8]) is set. Field register bit I/O signals: CFG_PCI04_SERR	R	0b0
29	Status Reg 13 - Received Master Abort	This bit is always forced to zero Field register bit I/O signals: AL_PCI04_CMPL_UR	R	0b0
28	Status Reg 12 - Received Target Abort	PCIe received completion abort primary bus for TYPE1 devices Field register bit I/O signals: AL_PCI04_REC_CMPL_ABORT	R	0b0
27	Status Reg 11 - Signaled Target Abort	Field register bit I/O signals: AL_PCI04_SIG_CMPL_ABORT	R	0b0
26:25	Reserved	Reserved	R	0b00
24	Status Reg 8 - Master Data Parity Error	Never get a PCIe poisoned TLP from the primary Register bit I/O signals: AL_PCI04_SIG_POISONED_TLP	R	0b0
23:21	Reserved	Reserved	R	0b000
20	Status Reg 4 - Capabilities List	Always equals ‘1’ for PCIe	R	0b1
19	Status Reg 3 - Interrupt Status	Indicates INTx pending in PCIe. Register bit I/O signals: AL_PCI04_INTX_STATUS	R/W	0b0
18:16	Reserved	Reserved	R	0b000
15:11	Command Regs 15:11	Reserved	R	0b0000
10	Command Reg 10 - Interrupt Disable	Controls INTx Messages in PCIe. Register bit I/O signals: CFG_PCI04_INTX_DISABLE	R/W	0b0

Bits	Field Name	Description	Access	Reset
9	Reserved	Reserved	R	0b0
8	Command Reg 8 - System Error (SERR) Enable	Register bit I/O signals: CFG_PCI04_SERR_ENABLE	R	0b0
7	Reserved	Reserved	R	0b0
6	Command Reg 6 - Parity Error (PERR) Enable	Register bit I/O signals: PERR_ENABLE	R/W	0b0
3:5	Reserved	Reserved	R	0b00
2	Command Reg 2 - Bus Master Enable	Register bit I/O signals: CFG_PC104_BUSMASTER_ENABLE	R/W	0b1
1	Command Reg 1 - Memory Space Enable	Register bit I/O signals: CFG_PC104_MEM_ENABLE	R/W	0b1
0	Command Reg 0 - I/O Space Enable	Register bit I/O signals: CFG_PC104_IO_ENABLE	R/W	0b1

PCI – Revision ID and Class Code Registers

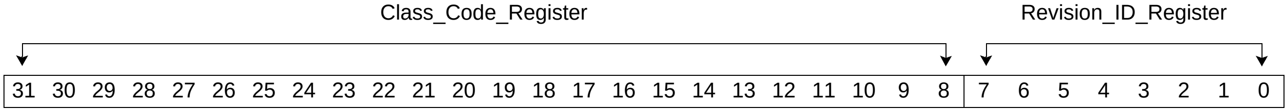
The PCI revision ID and class code are read-only registers used to communicate additional device identification information to the software operating system.

The class code register is used to identify the generic function of the device and/or register level programming interface.

Reset Value 0x06040000

Address Offset 0x08

Access Type Read Only



Bits	Field Name	Description	Access	Reset
31:8	Class Code Register	Register bit I/O signals: SYS_PCI08_CLASS_CODE [23:0]	R	0x060400
7:0	Revision_ID_Register	Register bit I/O signals: SYS_PCI08_REVISION_ID [7:0]	R	0x00

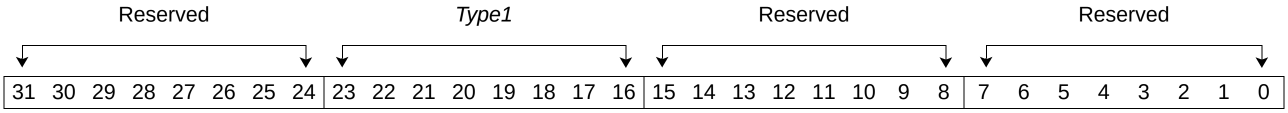
PCI – Header Type

The Header Type Register is read only and is used by the software to determine the register memory map of the remaining PCI Legacy header registers (0x00=TYPE0, 0x01=TYPE1). The CPC945 uses a TYPE1 header. (Note that the default for this register is big-endian.)

Reset Value 0x00000100

Address Offset 0x0C

Access Type Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31:24	Reserved	Reserved	R	0x00
23:16	Type1	Register bit I/O signals: HDRTYPE	R	0x01
15:8	Reserved	Reserved	R	0x00
7:0	Reserved	Reserved	R/W	0x00

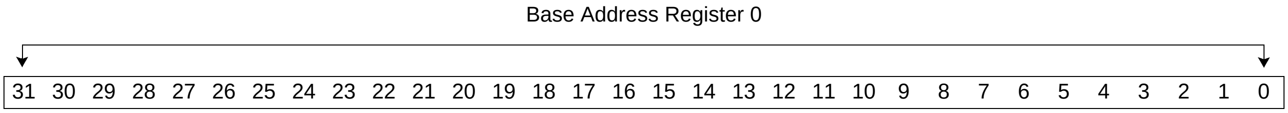
PCI – Base Address Register #0 (BAR0)

The BAR registers must exist in the configuration space memory map (required), but are not used in the CPC945 implementation.

Reset Value 0x00000000

Address Offset 0x10

Access Type Read Only



Bits	Field Name	Description	Access	Reset
31:0	Base Address Register 0	Reserved. Not used in CPC945 implementation.	R	0x00000000

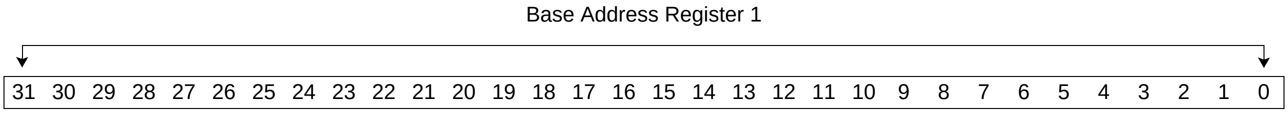
PCI – Base Address Register #1 (BAR1)

The BAR1 is not used in the CPC945 implementation.

Reset Value 0x00000000

Address Offset 0x14

Access Type Read Only



Bits	Field Name	Description	Access	Reset
31:0	Base Address Register 1	Reserved. Not used in CPC945 implementation.	R	0x00000000

PCI – Primary, Secondary, and Subordinate Bus Number Registers, and Secondary Latency Timer Register

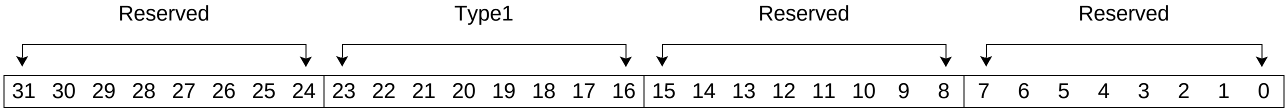
The Primary Bus Number, Secondary Bus Number, and Subordinate Bus Number Registers are required for all TYPE-1 devices such as the CPC945. These registers are implemented as read/write bits for software bus number assignment in the PCI bus hierarchy. The hardware uses the value programmed in these registers for decoding, and handling of PCI configuration space transactions.

The secondary latency timer does not apply to PCI Express devices and is implemented as an 8-bit read-only zero register.

Reset Value 0xF0F1F100

Address Offset 0x18

Access Type Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31:24	Secondary Latency Timer [7:0]	Reserved	R	0x00
23:16	Subordinate Bus Number [7:0]	Type 1 Register bit I/O signals: CFG_PCI18_SUBORDINATE_BUS	R/W	0xF1
15:8	Secondary Bus Number [7:0]	Reserved Register bit I/O signals: CFG_PCI18_SECONDARY_BUS	R	0xF1
7:0	Primary Bus Number [7:0]	Reserved Register bit I/O signals: CFG_PCI18_PRIMARY_BUS	R/W	0xF0

PCI – I/O Base, I/O Limit, and Secondary Status Registers

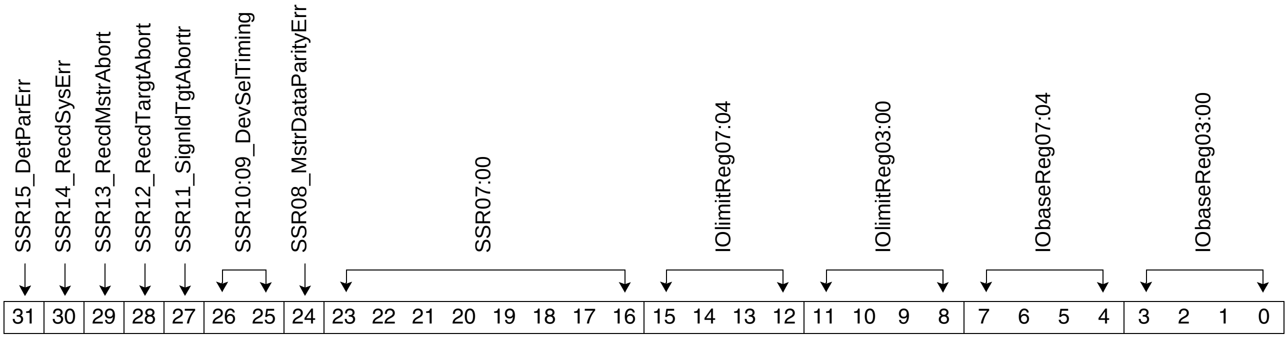
The I/O Base and I/O Limit Registers are used to control I/O transaction forwarding in a TYPE-1 device.

The Secondary Status Register is similar in function and bit definition to the Device Status Register (configuration space address 0x004), however, the Secondary Status Register reflects the status of the secondary bus while the device status register reflects status of the primary bus.

Reset Value 0x0010 0007

Address Offset 0x1C

Access Type Read, Read/Write



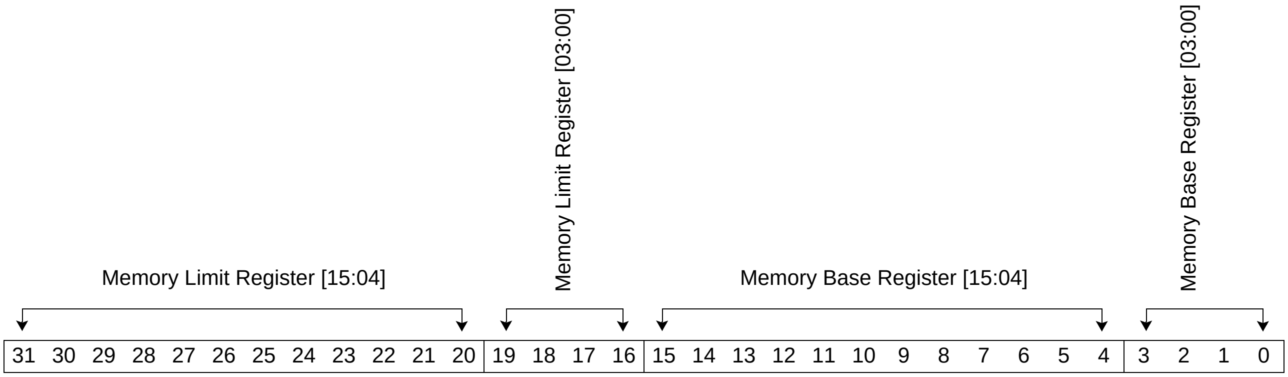
Bits	Field Name	Description	Access	Reset
31	Secondary Status Register [15] - Detected Parity Error	PCIe received a poisoned TLP on the TYPE1 device secondary bus AL_PCI1C_REC_POISONED_TLP	R/W	0b0
30	Secondary Status Register [14] - Received System Error	PCIe received nonfatal/fatal error message on the TYPE1 device secondary bus AL_PCI1C_NONFATAL_ERROR_MSG AL_PCI1C_FATAL_ERROR_MSG	R/W	0b0
29	Secondary Status Register [13] - Received Master Abort	PCIe received completion with unsupported request on the TYPE1 Device Secondary Bus AL_PCI1C_CMPL_UR	R/W	0b0
28	Secondary Status Register [12] - Received Target Abort	PCI Express received completion abort on the TYPE1 device secondary bus AL_PCI1C_REC_CMPL_ABORT	R/W	0b0
27	Secondary Status Register [11] - Signaled Target Abort	PCIe signaled completion abort on the TYPE1 device secondary bus AL_PCI1C_SIG_CMPL_ABORT	R/W	0b0
26:25	Secondary Status Register [10:9] - DevSEL Timing		R	0b00
24	Secondary Status Register [8] - Master Data Parity Error	Poisoned TLP, on the TYPE1 device secondary bus AL_PCI1C_SIG_POISONED_TLP	R/W	0b0
23:16	Secondary Status Register [7:0]		R	0x00
15:12	I/O Limit Register [7:4]	PCIEXCFG_IO_BASE_LIMIT_32 CFG_PCI1C_IO_LIMIT [7:4]	R/W	0x1
11:8	I/O Limit Register [3:0]	PCIEXCFG_IO_BASE_LIMIT_32	R	0x1

Bits	Field Name	Description	Access	Reset
7:4	I/O Base Register [7:4]	CFG_PCI1C_IO_BASE [7:4] PCIEXCFG_IO_BASE_LIMIT_32 Defines RW bits	R/W	0x0
3:0	I/O Base Register [3:0]	PCIEXCFG_IO_BASE_LIMIT_32 Defines register = 4’b0001	R	0x1

PCI – Memory Base and Memory Limit Registers

The Memory Base and Memory Limit Registers define the memory mapped address range used to control memory transaction forwarding in a TYPE-1 device. The lower four bits of both registers are always read only zero.

Reset Value	0x00000000
Address Offset	0x20
Access Type	Read Only, Read/Write


Memory Limit Register [15:04]
Memory Limit Register [03:00]
Memory Base Register [15:04]
Memory Base Register [03:00]
31 30 29 28 27 26 25 24 23 22 21 20
19 18 17 16
15 14 13 12 11 10 9 8 7 6 5 4
3 2 1 0

Bits	Field Name	Description	Access	Reset
31:20	Memory Limit Register [15:04]	Register bit I/O signals: CFG_PCI20_MEM_LIMIT [15:4]	R/W	0x000
19:16	Memory Limit Register [03:00]		R	0x0
15:4	Memory Base Register [15:04]	Register bit I/O signals: CFG_PCI20_MEM_BASE [15:4]	R/W	0x000
3:0	Memory Base Register [03:00]		R	0x0

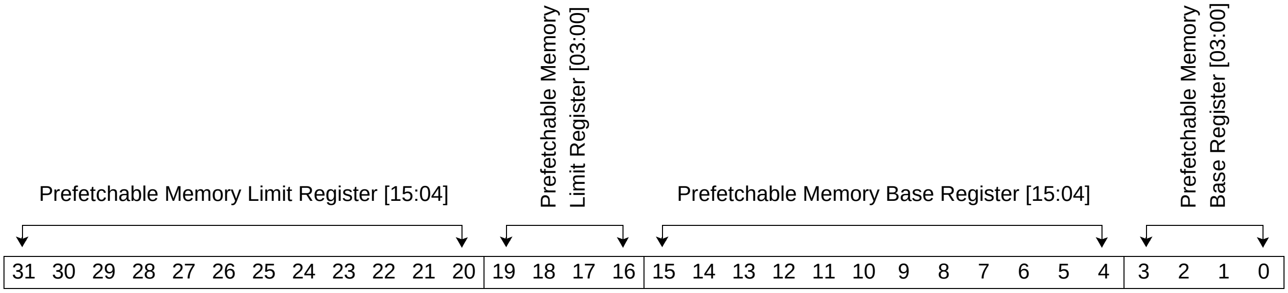
PCI – Prefetchable Memory Base and Prefetchable Memory Limit Registers

The Prefetchable Memory Base and Prefetchable Memory Limit Registers are used to control memory transaction forwarding in a TYPE-1 device. The lower four bits of the prefetchable memory base and prefetchable memory limit registers are always read-only bits, with the same value in each register (describes either 32-bit or 64-bit memory prefetch addressing).

Reset Value 0x10001000

Address Offset 0x24

Access Type Read Only, Read/Write


Prefetchable Memory Limit Register [15:04]
Prefetchable Memory Limit Register [03:00]
Prefetchable Memory Base Register [15:04]
Prefetchable Memory Base Register [03:00]
31 30 29 28 27 26 25 24 23 22 21 20 | 19 18 17 16 | 15 14 13 12 11 10 9 8 7 6 5 4 | 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:20	Prefetchable Memory Limit Register [15:04]	Register bit I/O signals: CFG_PCI20_PREFETCH_LIMIT [15:4]	R/W	0x000
19:16	Prefetchable Memory Limit Register [03:00]		R	0x1
15:4	Prefetchable Memory Base Register [15:04]	Register bit I/O signals: CFG_PCI24_PREFETCH_BASE [15:4]	R/W	0x000
3:0	Prefetchable Memory Base Register [03:00]		R	0x1

PCI – Prefetchable Memory Base Upper 32 Bits Register

The Prefetchable Memory Base Upper 32 Bits Register is used (with the Prefetchable Memory Limit Upper 32 Bits Register) to control 64-bit memory transaction forwarding in a TYPE-1 device.

Reset Value	0x00000000
Address Offset	0x28
Access Type	Read/Write

Prefetchable Mem Base Upper 32 Bits Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:0	Prefetchable Memory Base Upper-32 bits Register	Register bit I/O signals: CFG_PCI28_PREFETCH_BASE_UPPER [31:0]	R/W	0x00000000

PCI – Prefetchable Memory Limit Upper 32 Bits Register

The prefetchable memory limit upper 32 bits register is used (with the prefetchable memory base upper 32 bits register) to control 64-bit memory transaction forwarding in a TYPE-1 device.

Reset Value	0x00000000
Address Offset	0x2C
Access Type	Read/Write

Prefetchable Mem Limit Upper 32 Bits Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:0	Prefetchable Memory Limit Upper-32 bits Register	Register bit I/O signals: CFG_PCI28_PREFETCH_LIMIT_UPPER [31:0]	R/W	0x00000000

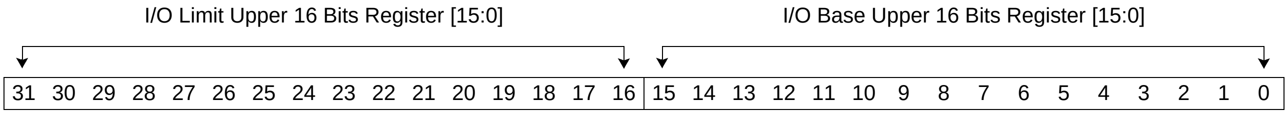
PCI – I/O Base Upper 16 Bits and I/O Limit Upper 16 Bits Registers (Required for TYPE1)

The I/O Base Upper 16 Bits and I/O Limit Upper 16 Bits Registers are used to control I/O transaction forwarding in a TYPE-1 device.

Reset Value 0x00000000

Address Offset 0x30

Access Type Read/Write



Bits	Field Name	Description	Access	Reset
31:16	I/O Limit Upper 16 Bits Register [15:0]	Register bit I/O signals: CFG_PCI30_IO_LIMIT_UPPER [15:0]	R/W	0x0000
15:0	I/O Base Upper 16 Bits Register [15:0]	Register bit I/O signals: CFG_PCI30_IO_BASE_UPPER [15:0]	R	0x0000

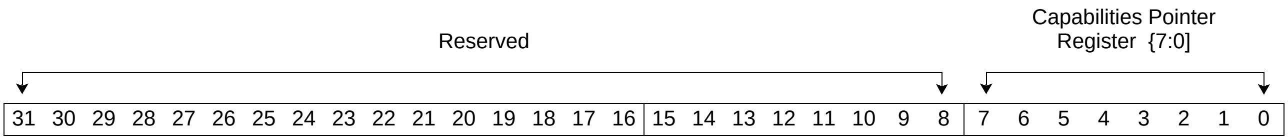
PCI – Capabilities Pointer Register

The PCI Capabilities Pointer Register is required for all PCI Express devices. This byte wide register contains the linked list pointer to the first PCI capabilities structure.

Reset Value 0x40000000

Address Offset 0x34

Access Type Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31:8	Reserved		R	0x000000
7:0	Capabilities Pointer Register [7:0]	Register bit I/O signals: SYS_PCI34_CAP_PTR [7:0]	R/W	0x40

PCI – Expansion ROM Base Address Register (Required for TYPE1)

The PCI Expansion ROM Base Address Register (BAR) is not used in the CPC945.

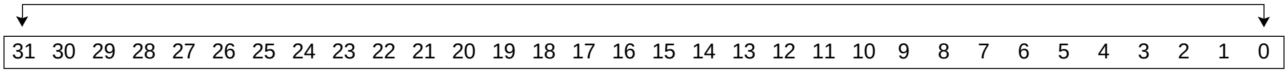
The Expansion ROM Base Address Register exists in the configuration space memory map, but is implemented with 32 read-only zero bits to indicate that no expansion ROM is present.

Reset Value 0x00000000

Address Offset 0x38

Access Type Read

Expansion ROM Base Address Register



Bits	Field Name	Description	Access	Reset
31:0	Expansion ROM Base Address Register	Register bit I/O signals: CFG_PCIXX_ROM_BAR [31:0]	R	0x00000000

PCI – Interrupt Line, Interrupt Pin, Bridge Control Registers

The byte wide PCI Interrupt Line and Interrupt Pin Registers are the first two bytes located at configuration space address 0x3C.

The Interrupt Line Register is an 8-bit read/write register that is used by the system software to save the system interrupt routing number associated with the device. This register has no effect on the PCI device hardware functionality.

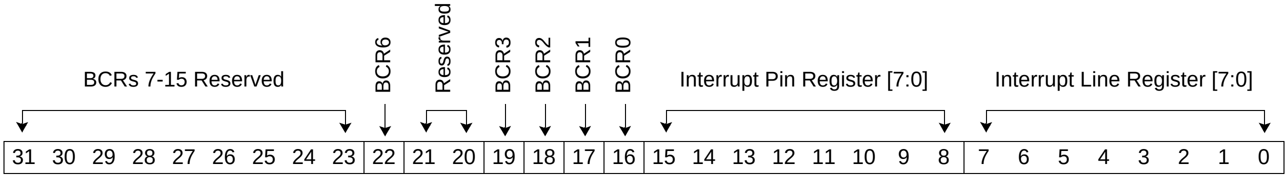
The interrupt pin in this register is an 8-bit read-only register that defines interrupt pins implementation. The CPC945 does not have an interrupt pin, so these bits are all zero.

The Bridge Control Register (BCR) provides extensions to the device command register (configuration space address 0x004). The Bridge Control Register provides many of the same controls for the secondary interface that are provided be the command register for the primary interface.

Reset Value 0x00000000

Address Offset 0x3C

Access Type Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31:23	Bridge Control Registers [7] through [15] Reserved	Reserved	R	0x00
22	Bridge Control Register [6] (Secondary Bus Reset)	Register bit I/O signals: CFG_PCI3C_BCR_SB_RESET	R/W	0b0
21:20	Reserved	Reserved	R	0b00
19	Bridge Control Register [3]	Register bit I/O signals: CFG_PCI3C_BCR_VGA_ENABLE	R/W	0b0
18	Bridge Control Register [2]	Register bit I/O signals: CFG_PCI3C_BCR_ISA_ENABLE	R/W	0b0
17	Bridge Control Register [1]	Register bit I/O signals: CFG_PCI3C_BCR_SERR_ENABLE	R/W	0b0
16	Bridge Control Register [0] - Parity Error Response Enable	Register bit I/O signals: CFG_PCI3C_BCR_PERR_RESP_ENABLE	R/W	0b0
15:8	Interrupt Pin Register [7:0]	Register bit I/O signals: SYS_PCI3C_INTERRUPT_PIN [7:0] - I/O Pins if NO define	R	0x00
8:0	Interrupt Line Register [7:0]		R/W	0x00

12.11.1.2 PCI Power Management Capability Structure

The PCI power management (PM) capability structure contains eight bytes of registers (two DWords of configuration space). The PM registers are used for communicating and controlling the power management capabilities of the CPC945 all PCI Express root complex.

Table 12-21. PCI Power Management Capability Structure

Register Name (Mnemonic)	Address DWord Offset	Access Mode	See Page
PM - Capability ID Register	0x40	Read Only	521
PM - Next PTR Register	0x40	Read Only	521
PM - Power Management Capabilities Register (PMC)	0x40	Read Only	521
PM - Power Management Control and Status Register (PMCSR)	0x44	Mixed	523
PM - Power Management Control/Status Bridge Support Extension Register (PMCSR_BSE) - Reserved in CPC945	0x44	Read Only	523
PM - Data Register - Reserved in CPC945	0x44	Read Only	523

PM – Power Management Capability ID, Next PTR, and Capabilities Registers

The first two bytes of the PCI capability structure contains the:

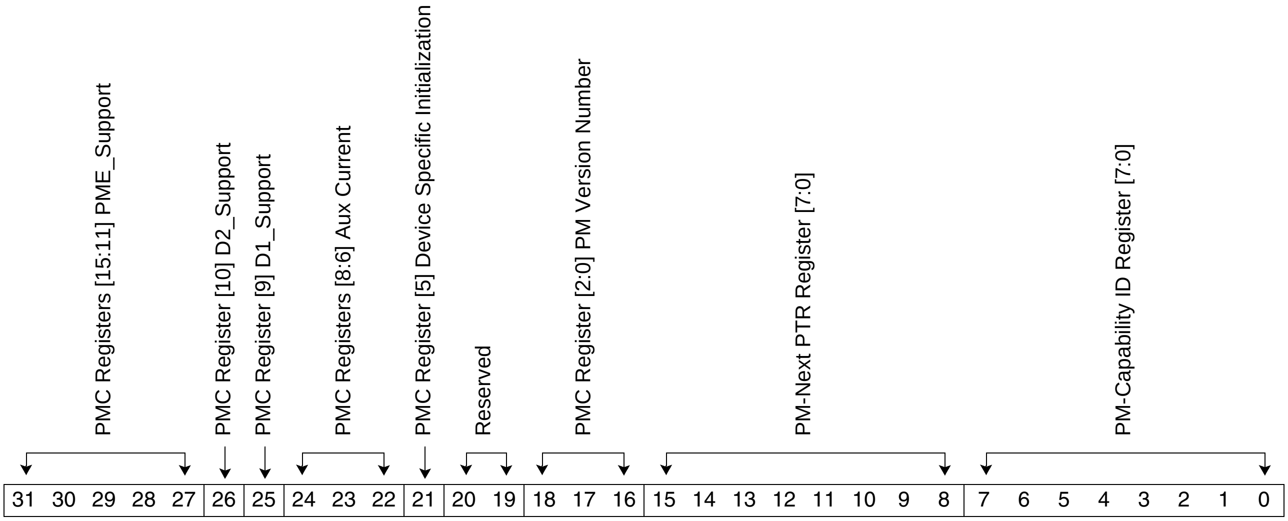
- Read-only capability ID (a unique code that identifies the type, size, and format of the capability structure)
- Next pointer register (which points to the next capability structure in the linked list, or all zeros to terminate the capability structure linked list)

The Power Management Capability (PMC) Register is the two-byte, read-only register that is used to communicate the hardware supported power management capabilities of the device to the software operating system.

Reset Value 0x0148446F

Address Offset 0x40

Access Type Read, Read/Write


PMC Registers [15:11] PME_Support
PMC Register [10] D2_Support
PMC Register [9] D1_Support
PMC Registers [8:6] Aux Current
PMC Register [5] Device Specific Initialization
Reserved
PMC Register [2:0] PM Version Number
PM-Next PTR Register [7:0]
PM-Capability ID Register [7:0]
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:27	PM Capabilities Registers [15:11]	Register bit I/O signals: PM_PME_Support	R	0b01111
26	PM Capabilities Register [10]	Register bit I/O signals: PM_D2_Support	R	0b1
25	PM Capabilities Register [9]	Register bit I/O signals: PM_D1_Support	R	0b1
24:22	PM Capabilities Registers [8:6]	Register bit I/O signals: PM_AUX_CURRENT	R	0b000
21	PM Capabilities Register [5]	Register bit I/O signals: PM_DSI	R	0b1
20:19	Reserved	Bit [20] is RsvdP; reserved and preserved. Reserved for future R/W implementations. Registers are read-only and must return 0 when read. Software must preserve the value read for writes to bits. Bit [19] is not applicable to PCIe and hardwired to ‘0’.	R	0b00
18:16	PM Capabilities Registers [2:0]	PM Version Number	R	0b010

Bits	Field Name	Description	Access	Reset
15:8	Next PTR Register	Register bit I/O signals: PM_NEXT_CAP_PTR	R	0x48
7:0	PM Capability ID Register	Power Management Capability	R	0x01

PM – PM Control/Status (PMCSR), PM Control/Status Bridge Support Extensions (PMCSR_BSE), and PM Data Registers

The 16-bit Power Management Control/Status (PMCSR) Register is used to communicate power management control and status information between the hardware and software.

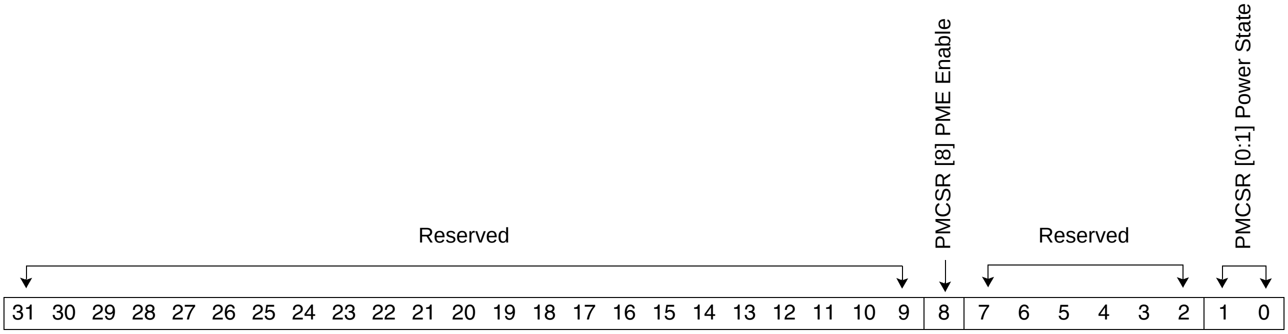
The 8-bit Power Management Control/Status Bridge Support Extensions (PMCSR_BSE) Register is implemented as read only zero for all PCI Express devices.

The 8-bit Power Management Data Register is not implemented in the CPC945.

Reset Value 0x00800000

Address Offset 0x44

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:9	Reserved	Bits Description 31:24 Reserved, unused 23:22 Not applicable to PCIe, wired to ‘0’ 21:16 RsvdP, per the PCI Express specification	R	0b0000000000000000000000
8	PM - PM Control/Status (PMCSR) [8] PME Enable	SYS_EC08_AUXPOWER_AVAIL is inverted, then AND’ed with DL_PGRESET to block the Cold Reset function for PCI Express Devices that implement VAUX support. See the PCI Express 1.0a specification and PCI Bus Power Management Interface 1.1 specification for special case Sticky Bit behavior related to VAUX and PME Support Register bit I/O signals: • CFG_PM04_PME_ENABLE • SYS_EC08_AUXPOWER_AVAIL	R/W	0b1
7:2	Reserved	RsvdP, per the PCI Express specification	R	0b000000
1:0	PM - PM Control/Status (PMCSR) [1:0] Power State	Register bit I/O signals: CFG_PM04_POWER_STATE[1:0]	R/W	0b0

12.11.1.3 PCI Express Capability Structure

The PCI Express capability (EC) structure is as nine DWords (rootport devices) of configuration space. The following section describes the EC registers used for communicating and controlling the PCI Express capabilities.

Table 12-22. PCI Express Configuration Space

Register Name (Mnemonic)	Address DWord Offset	Access Mode	Page
EC - Capability ID Register	0x48	Read Only	525
EC - Next PTR Register	0x48	Read Only	525
EC - PCI Express Capabilities Register	0x48	Read Only	525
EC - Device Capabilities Register	0x4C	Read Only	526
EC - Device Control Register	0x50	Mixed	526
EC - Device Status Register	0x50	Mixed	526
EC - Link Capabilities Register	0x54	Read Only	530
EC - Link Control Register	0x58	Mixed	531
EC - Link Status Register	0x58	Mixed	531
EC - Slot Capabilities Register	0x5C	Read Only	533
EC - Slot Control Register	0x60	Mixed	534
EC - Slot Status Register	0x60	Mixed	534
EC - Root Control Register	0x60	Mixed	536
EC - Root Status Register	0x68	Mixed	537

EC – PCI Express Capability ID, Next PTR, and Capabilities Registers

The first two bytes of the PCI capability structures contain the:

- Read-only capability ID
- Next Pointer Register (which points to the next capability structure in the linked list, or all zeros to terminate the capability structure linked list)

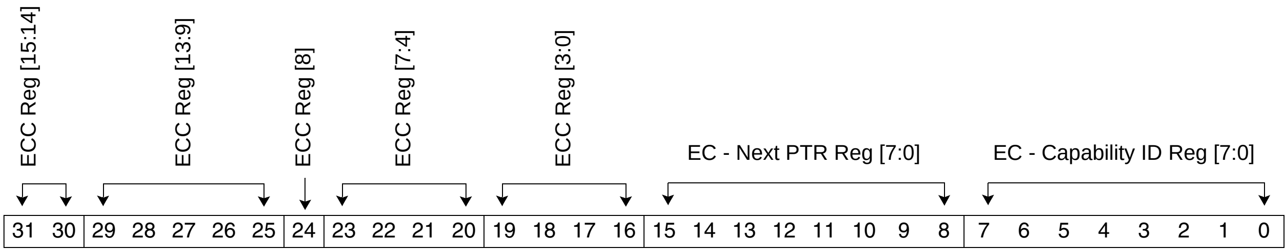
The PCI Express capability (EC) structure always has the PCI-SIG assigned value of 0x10 for this 8-bit read-only capability ID register.

The PCI Express Capabilities Register is the 16-bit read-only register that describes the PCI Express device hardware type and basic capabilities to the software.

Reset Value 0x00000000

Address Offset 0x48

Access Type Read, Read/Write

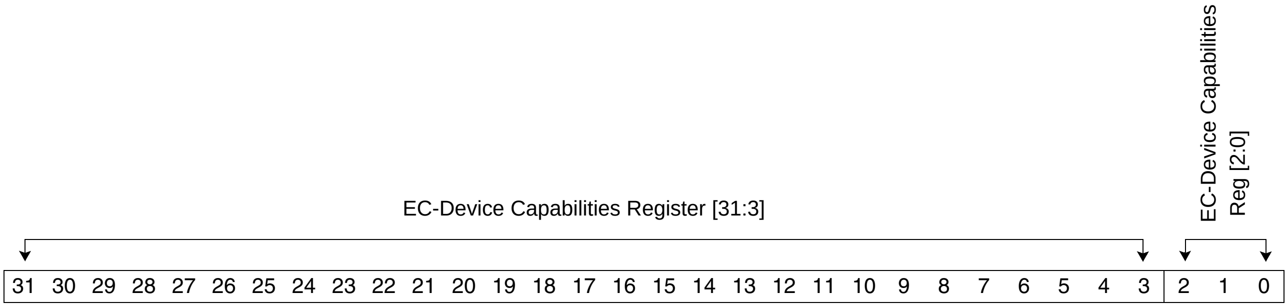


Bits	Field Name	Description	Access	Reset
31:30	EC- Capabilities Register [15:14]	RsvdP, per the PCIe Specification.	R	0b00
29:25	EC - Capabilities Register [13:9] Interrupt Message Number	The CPC945 implementation does not support multiple MSIs Register bit I/O signals: AL_EC00_INTR_MSG_NUM [4:0]	R/W	0b00000
24	EC - Capabilities Register [8] - Slot Implemented	Register bit I/O signals: SYS_EC00_SLOT	R	0b1
23:20	EC - Capabilities Register [7:4] - Device/Port Type	SYS_EC00_PORTTPE[3:0]	R/W	0x1
19:16	EC - Capabilities Register [3:0] - PCI Express Capability Version	PCI Express Capability Version as defined in the PCIe Specification 1.0a.	R	0b0001
15:8	EC - Next PTR Register [7:0]	PCIEXCFG_EC_NEXT_CAP_PTR xxh - defines the register value	R	0x00
7:0	EC - Capability ID Register [7:0]		R	0x10

EC – Device Capabilities Register

The Device Capabilities Register is the 32-bit read-only register that describes the PCI Express device specific capabilities in detail to the software.

Reset Value	0x00000000
Address Offset	0x4C
Access Type	Read



Bits	Field Name	Description	Access	Reset
31:3	EC-Device Capabilities Register [31:3]	Reserved, unused.	R	0b0000000000 000000000000 0000000
2:0	EC- Device Capabilities Register [2:0] Maximum Payload Size Supported	Maximum payload size supported is128 bytes	R	0b000

EC – Device Control and Device Status Registers

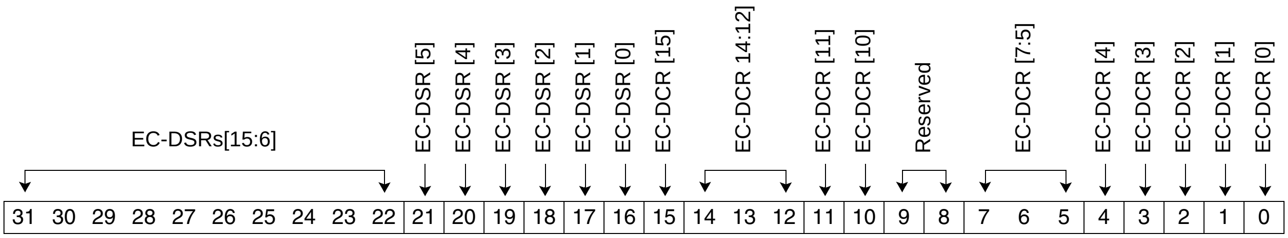
The Device Control Register is the 16-bit read/write register used by the software to control the PCI Express device specific parameters.

The Device Status Register is the 16-bit read/write register that provides PCI Express device specific information to the software.

Reset Value 0x01100000

Address Offset 0x50

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:22	EC - DSR [15:6]	RsvdZ. Per the PCI Express Specification, reserved and zero: Reserved for future RW1C implementations; software must use 0 for writes to bits.	R	0b0000000000
21	EC - DSR [5] Transactions Pending	Register bit I/O signals: AL_EC08_TRANS_PENDING	R	0b0
20	EC - DSR [4] AUX Power Detected	Reserved	R	0b0
19	EC - DSR [3] Unsupported Request Detected	Register bit I/O signals: AL_EC08_UR	R/W	0b0
18	EC - DSR [2] Fatal Error Detected	Register bit I/O signals: DL_EC08_DLLPE AL_PCI04_REC_POISONED_TLP AL_PCI1C_REC_POISONED_TLP TL_EC08_FCPE AL_EC08_CMPL_TIMEOUT AL_PCI04_SIG_CMPL_ABORT AL_PCI1C_SIG_CMPL_ABORT AL_EC08_UNEXPECTED_CMPL AL_EC08_REC_OVERFLOW AL_EC08_MALFORMED_TLP AL_AER04_ECRC_ERROR AL_EC08_UR	R/W	0b0

Bits	Field Name	Description	Access	Reset
17	EC - DSR [1] Nonfatal Error Detected	Register bit I/O signals: DL_EC08_DLLPE AL_PCI04_REC_POISONED_TLP AL_PCI1C_REC_POISONED_TLP TL_EC08_FCPE AL_EC08_CMPL_TIMEOUT AL_PCI04_SIG_CMPL_ABORT AL_PCI1C_SIG_CMPL_ABORT AL_EC08_UNEXPECTED_CMPL AL_EC08_REC_OVERFLOW AL_EC08_MALFORMED_TLP AL_AER04_ECRC_ERROR AL_EC08_UR	R/W	0b0
16	EC - Device Status Register [0] Correctable Error Detected	Register bit I/O signals: DL_EC08_RECEIVERERROR DL_EC08_BADTLP DL_EC08_BADDLLP DL_EC08_REPLAYROLLOVER DL_EC08_REPLAYTIMEOUT	R/W	0b0
15	EC - DCR [15]	RsvdP.	R/	0b0
14:12	EC - DCR [14:12] Max_Read_Request_Size	Register bit I/O signals: CFG_EC08_MAX_READ_REQ_SIZE [2:0] CPC945 does not support and will not generate a Read Request larger than 128 bytes. In accordance with the PCI Express specification, this field is implemented as Read Only (R) with a value of 0b000.	R	0x000
11	EC - DCR [11] Enable No Snoop	Register bit I/O signals: CFG_EC08_NOSNOOP_ENABLE	R/W	0b1
10	EC - DCR [10] Auxiliary (AUX) Power PM Enable	Register bit I/O signals: CFG_EC08_AUXPOWER_PM_ENABLE SYS_EC08_AUXPOWER_AVAIL	R	0b0
9	EC - DCR [9] Phantom Function Enable	Not supported in CPC945 Register bit I/O signals: CFG_EC08_PHANTOM_FUNC_ENABLE	R	0b0
8	EC - DCR[8] Extended Tag Field Enable	Not supported in CPC945 Register bit I/O signals: CFG_EC08_EXT_TAG_ENABLE	R	0b0
7:5	EC - DCR [7:5] Max_Payload_Size=128 bytes	Register bit I/O signals: CFG_EC08_MAX_PAYLOAD_SIZE [2:0]	R/W	0b000
4	EC - DCR [4] Enable Relaxed Ordering	Register bit I/O signals: CFG_EC08_RELAXED_ORDERING	R/W	0b1
3	EC - DCR [3] Unsupported Request Reporting Enable		R/W	0b0
2	EC - DCR [2] Fatal Error Reporting EnableDCR	Register bit I/O signals: CFG_EC08_FATAL_ERROR	R/W	0b0
1	EC - DCR [1] Nonfatal Error Reporting Enable	Register bit I/O signals: CFG_EC08_NONFATAL_ERROR	R/W	0b0

Bits	Field Name	Description	Access	Reset
0	EC - DCR [0] Correctable Error Reporting Enable	Register bit I/O signals: CFG_EC08_CORR_ERROR	R/W	0b0

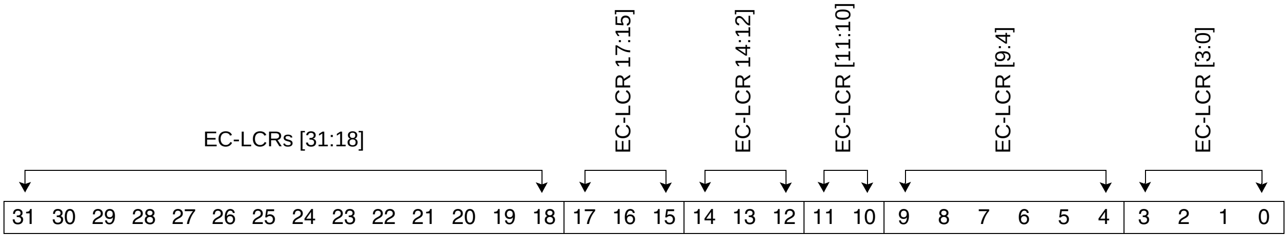
EC – Link Capabilities Register

The Link Capabilities Register is the 32-bit read-only register that describes the PCI Express link specific capabilities to the software.

Reset Value 0x0000 AD01

Address Offset 0x054

Access Type Read



Bits	Field Name	Description	Access	Reset
31:18	EC - Link Capabilities Registers [31:18]	31:24 Read only, always '0', always port 0 34:18 RsvdP	R	0b00000000 000000
17:15	EC - Link Capabilities Register [17:15] L1 Exit Latency		R	0b001
14:12	EC - Link Capabilities Register [14:12] L0 Exit Latency		R	0b010
11:10	EC - Link Capabilities Register [11:10] Active State Link PM Support		R	0b11
9:4	EC - Link Capabilities Register [9:4] Max_Link_Width	Register bit I/O signals: SYS_EC0C_MAXLINKWIDTH	R	0b010000
3:0	EC - Link Capabilities Register [3:0] Max_Link_Speed		R	0x1

EC – Link Control and Link Status Registers

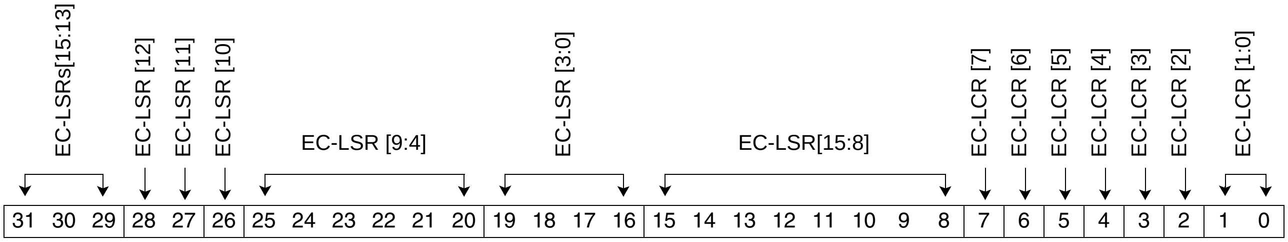
The Link Control Register is the 16-bit read/write register used by the software to control the PCI Express link specific parameters.

The Link Status Register is the 16-bit read-only register that provides PCI Express link specific information to the software.

Reset Value 0x10010000

Address Offset 0x58

Access Type Read, Read/Write



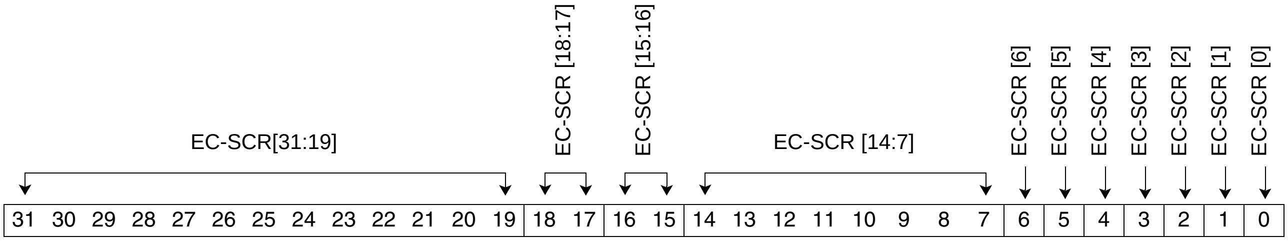
Bits	Field Name	Description	Access	Reset
31:29	EC - Link Status Register [15:13] Reserved Bits	RsvdZ. Per the PCI Express Specification.	R	0b000
28	EC - Link Status Register [12] Slot Clock Configuration	Register bit I/O signals: SYS_EC10_SLOTCLOCK	R	0b1
27	EC - Link Status Register [11] Link Training	Register bit I/O signals: DL_EC10_LINKTRAINING	R	0b0
26	EC - Link Status Register [10] Training Error	Reserved. This bit field can not be used to detect the presence of training errors and always returns '0' when read	R	0b0
25:20	EC - Link Status Register [9:4] Negotiated Link Width	Register bit I/O signals: DL_EC10_NEGLINKWIDTH [5:0]	R	0b000000
19:16	EC - Link Status Register [3:0] Link Speed	Link speed	R	0x1
15:8	EC - Link Control Register [15:8]	RsvdP, per the PCI Express specification: reserved and preserved: reserved for future read/write implementations; software must preserve the value read for writes to bits.	R	0x00
7	EC - Link Control Register [7] Extended Synch	Register bit I/O signals: CFG_EC10_EXTENDEDSYNCH	R/W	0b0
6	EC - Link Control Register [6] Common Clock Configuration	Register bit I/O signals: CFG_EC10_COMMONCLOCK	R/W	0b0
5	EC - Link Control Register [5] Retrain Link	Register bit I/O signals: CFG_EC10_RETRAINLINK	R/W	0b0

Bits	Field Name	Description	Access	Reset
4	EC - Link Control Register [4] Link Disable	Register bit I/O signals: CFG_EC10_LINKDISABLE	R/W	0b0
3	EC - Link Control Register [3] Read Completion Boundary (RCB)	Boundary = 128bytes Register bit I/O signals: CFG_EC10_RCB128	R/W	0b0
2	EC - Link Control Register [2]	RsvdP, per the PCI Express specification	R	0b0
1:0	EC - Link Control Register [1:0] Active State Link PM Control	Register bit I/O signals: CFG_EC10_LINKPMCONTROL [1:0] SYS_EC10_LINKPMINIT [1:0]	R/W	0b00

EC – Slot Capabilities Register

The Slot Capabilities Register is the 32-bit read-only register that describes the PCI Express slot specific capabilities to the software.

Reset Value	0x00000000
Address Offset	0x5C
Access Type	Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:19	EC - Slot Capabilities Register [31:19] Physical Slot Number	Register Bit I/O Signals: SYS_EC14_SLOT_NUMBER [12:0]	R	0b00000000 00000
18:17	EC - Slot Capabilities Register [18:17]	RsvdP, per the PCI Express specification.	R	0b00
16:15	EC - Slot Capabilities Register [16:15] Slot Power Limit Scale	Register Bit I/O Signals: SYS_EC14_PWR_LIMIT_SCALE [1:0] CFG_EC14_SLOT_PWR_SENDMSG	R	0b00
14:7	EC - Slot Capabilities Register [14:7] Slot Power Limit Value	The CPC945 does not provide slot power. Register Bit I/O Signals: SYS_EC14_PWR_LIMIT_VALUE [7:0] CFG_EC14_SLOT_PWR_SENDMSG T	R	0x00
6	EC - Slot Capabilities Register [6] Hot-Plug Capable	Register Bit I/O Signals: SYS_EC14_HOTPLUG_CAPABLE	R	0b0
5	EC - Slot Capabilities Register [5] Hot-Plug Surprise	Register Bit I/O Signals: SYS_EC14_HOTPLUG_SURPRISE	R	0b0
4	EC - Slot Capabilities Register [4] Power Indicator Present	Register Bit I/O Signals: SYS_EC14_PWR_IND_PRESENT	R	0b0
3	EC - Slot Capabilities Register [3] Attention Indicator Present	Register Bit I/O Signals: SYS_EC14_ATN_IND_PRESENT	R	0b0
2	EC - Slot Capabilities Register [2] MRL Sensor Present	Register Bit I/O Signals: SYS_EC14_MRL_PRESENT	R	0b0
1	EC - Slot Capabilities Register [1] Power Controller Present	Register Bit I/O Signals: SYS_EC14_PWR_CTL_PRESENT	R	0b0

Bits	Field Name	Description	Access	Reset
0	EC - Slot Capabilities Register [0] Attention Button Present	Register Bit I/O Signals: SYS_EC14_ATN_BTN_PRESENT	R	0b0

EC – Slot Control and Slot Status Registers

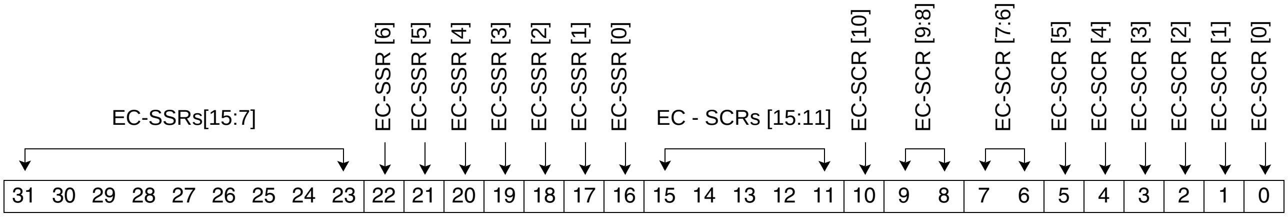
The Slot Control Register is the 16-bit read/write register used by the software to control the PCI Express slot specific parameters.

The Slot Status Register is the 16-bit read/write register that provides PCI Express slot specific information to the software.

Reset Value 0x0000000

Address Offset 0x60

Access Type Read, Read/Write



Bitss	Field Name	Description	Access	Reset
31:23	EC - Slot Status Register [15:7]	RsvdZ, per the PCI Express specification	R	0b0
22	EC - Slot Status Register [6] Presence Detect State	Register bit I/O signals: SYS_EC18_SLOT_DETECT_STATE	R	0b0
21	EC - Slot Status Register [5] MRL Sensor State	Register bit I/O signals: SYS_EC18_MRL_SENSOR_STATE	R	0b0
20	EC - Slot Status Register [4] Command Completed	Register bit I/O signals: SYS_EC18_PWR_CTL_CMD_CMPLT CFG_EC18_SLOT_INTR	R/W	0b0
19	EC - Slot Status Register [3] Presence Detect Changed	Register bit I/O signals: SYS_EC18_SLOT_DETECT_CHANGED CFG_EC18_SLOT_WAKE	R/W	0b0
18	EC - Slot Status Register [2] MRL Sensor Changed	Register bit I/O signals: SYS_EC18_MRL_SENSOR_CHANGED CFG_EC18_SLOT_WAKE	R/W	0b0
17	EC - Slot Status Register [1] Power Fault Detected	Register bit I/O signals: SYS_EC18_PWR_FAULT_DETECTED CFG_EC18_SLOT_WAKE	R/W	0b0
16	EC - Slot Status Register [0] Attention Button Pressed	Register bit I/O signals: SYS_EC18_ATN_BTN_PRESSED CFG_EC18_SLOT_WAKE	R/W	0b0

Bitss	Field Name	Description	Access	Reset
15:11	EC - Slot Control Register [15:11]	RsvdP, per the PCI Express specification	R	0b1
10	EC - Slot Control Register [10] Power Controller Control	Register bit I/O signals: CFG_EC18_PWR_CTL_CONTROL	R/W	0x000
9:8	EC - Slot Control Register [9:8] Power Indicator Control	Register bit I/O signals: CFG_EC18_PWR_IND_CONTROL [1:0] CFG_EC18_PWR_IND_SENDMSG	R/W	0b0
7:6	EC - Slot Control Register [7:6] Attention Indicator Control	Register bit I/O signals: CFG_EC18_ATN_IND_CONTROL [1:0] CFG_EC18_ATN_IND_SENDMSG	R/W	0b00
5	EC - Slot Control Register [5] Hot-Plug Interrupt Enable	Register bit I/O signals: CFG_EC18_SLOT_INTR	R/W	0b0
4	EC - Slot Control Register [4] Command Completed Interrupt Enable	Register bit I/O signals: CFG_EC18_SLOT_INTR	R/W	0b0
3	EC - Slot Control Register [3] Presence Detect Changed Enable	Register bit I/O signals: CFG_EC18_SLOT_WAKE	R/W	0b0
2	EC - Slot Control Register [2] MRL Sensor Changed Enable	Register bit I/O signals: CFG_EC18_SLOT_WAKE	R/W	0b0
1	EC - Slot Control Register [1] Power Fault Detected Enable	Register bit I/O signals: CFG_EC18_SLOT_WAKE	R/W	0b0
0	EC - Slot Control Register [0] Attention Button Pressed Enable	Register bit I/O signals: CFG_EC18_SLOT_WAKE	R/W	0b0

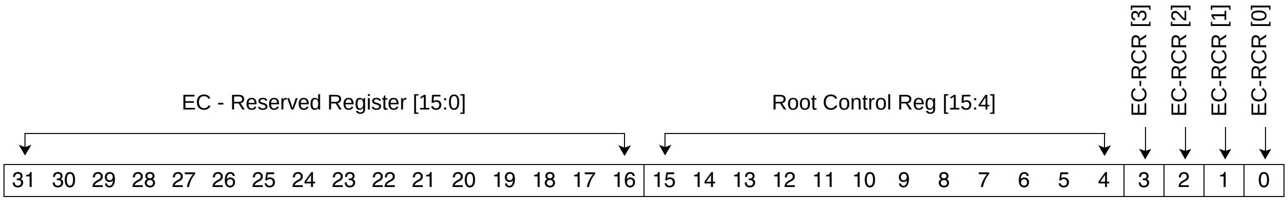
EC – Root Control Register

The 16-bit read/write Root Control Register is used by the system software to control PCI Express root complex specific parameters. The remaining 16-bits of this Express capability structure DWord address is defined as reserved for this version of the PCI Express specification (read-only zero).

Reset Value 0x0000000

Address Offset 0x64

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:16	EC - Reserved Register [15:0]	Reserved	R	0x0000
15:4	EC - Root Control Register [15:4]	RsvdP, per the PCI Express specification	R	0x000
3	EC - Root Control Register [3] PME Interrupt Enable	Register Bit I/O Signals: CFG_EC1C_PME_INTR_EN	R/W	0b0
2	EC - Root Control Register [2] System Error on Fatal Error Enable	Register Bit I/O Signals: CFG_EC1C_FATAL_SERR_EN	R/W	0b0
1	EC - Root Control Register [1] System Error on Nonfatal Error Enable	Register Bit I/O Signals: CFG_EC1C_NONFATAL_SERR_EN	R/W	0b0
0	EC - Root Control Register [0] System Error on Correctable Error Enable	Register Bit I/O Signals: CFG_EC1C_CORR_SERR_EN	R/W	0b0

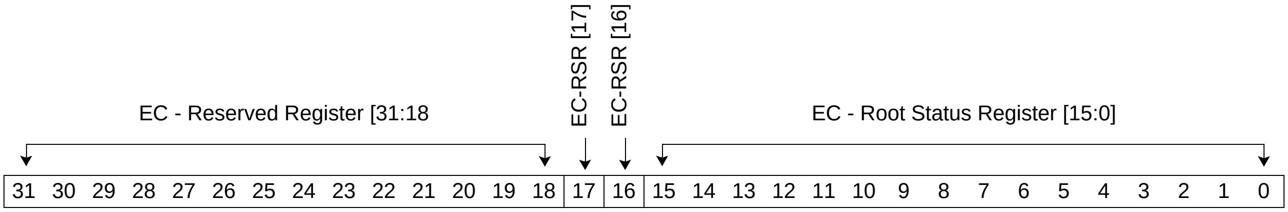
EC – Root Status Register

The 32-bit read/write Root Status Register provides PCI Express root complex specific information to the software.

Reset Value 0x00000000

Address Offset 0x68

Access Type Read Only, Read/Write



Bits	Field Name	Description	Access	Reset
31:18	EC - Reserved Register [31:18]	Reserved	R	0b00000000000000
17	EC - Root Status Register [17] PME Pending	Register Bit I/O Signals: AL_EC20_PME_PENDING	R	0b0
16	EC - Root Status Register [16]	Register Bit I/O Signals: AL_EC20_PME_STATUS CFG_EC20_PME_STATUS_CLR PME Status: - Set with Pulse on Input Pin - Write1 to Clear will pulse Output Pin	R/W	0b0
15:0	EC - Root Status Register [15:0] PME Requestor ID	Register Bit I/O Signals: AL_EC20_PME_REQUESTOR_ID	R	0x00

12.11.1.4 Advanced Error Reporting Extended Capability Structure

The PCI Express advanced error reporting (AER) extended capability structure is defined for the CPC945, but does not support end-to-end CRC (ECRC). It is 14 DWords in length and is used for communicating and controlling the optional advanced error reporting capabilities of the CPC945.

Register Name (Mnemonic)	Address DWord Offset	Access Mode	Relevant PCI Specifications	See Page
AER - PCI Express Enhanced Capability Header (ID, version, next PTR) Registers	0x100	Read Only	PCI Express Base Specification v1.0a	539
AER - Uncorrectable Error Status Register	0x104	Mixed	PCI Express Base Specification v1.0a	540
AER - Uncorrectable Error Mask Register	0x108	Mixed	PCI Express Base Specification v1.0a	541
AER - Uncorrectable Error Severity Register	0x10C	Mixed	PCI Express Base Specification v1.0a	542
AER - Correctable Error Status Register	0x110	Mixed	PCI Express Base Specification v1.0a	543
AER - Correctable Error Mask Register	0x114	Mixed	PCI Express Base Specification v1.0a	544
AER - Capabilities and Control Register	0x118	Mixed	PCI Express Base Specification v1.0a	545
AER - Header Log Register #1	0x11C	Read Only	PCI Express Base Specification v1.0a	547
AER - Header Log Register #2	0x120	Read Only	PCI Express Base Specification v1.0a	547
AER - Header Log Register #3	0x124	Read Only	PCI Express Base Specification v1.0a	547
AER - Header Log Register #4	0x128	Read Only	PCI Express Base Specification v1.0a	547
AER - Root Error Command Register (only for rootports)	0x12C	Mixed	PCI Express Base Specification v1.0a	548
AER - Root Error Status Register (only for rootports)	0x130	Mixed	PCI Express Base Specification v1.0a	549
AER - Error Source Identification Register (only for rootports)	0x134	Read Only	PCI Express Base Specification v1.0a	550

AER – Advanced Error Reporting Enhanced Capability Header Registers: ID, Version, and Next PTR

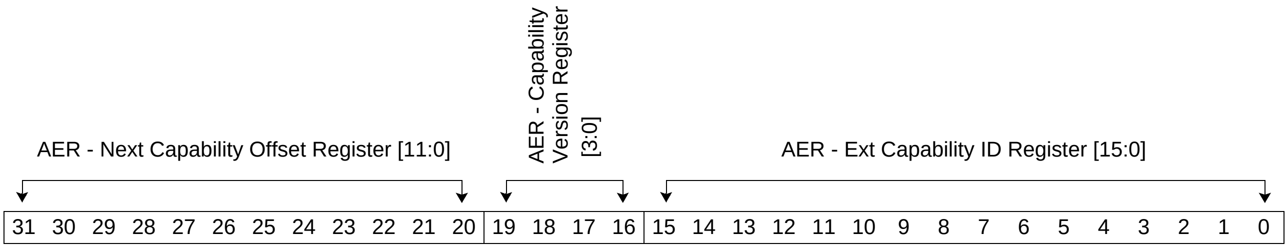
The first DWord contains the read-only enhanced capability header. The enhanced capability header contains a:

- PCI Express extended capability ID (a unique PCI-SIG defined ID code that identifies the type, size, and format of the extended capability)
- Capability version (PCI-SIG defined version of the capability structure)
- Next capability offset register (which points to the next extended capability structure in the linked list, or all zeros to terminate the extended capability structure linked list)

The AER extended capability structure always has the PCI-SIG assigned value of 0x0001 for the 16-bit read-only extended capability ID register.

The AER extended capability structure always has the PCI-SIG assigned value of 0x1 for the 4-bit read-only capability version register.

Reset Value	0x00010001
Address Offset	0x100
Access Type	Read Only


AER - Next Capability Offset Register [11:0]
AER - Capability Version Register [3:0]
AER - Ext Capability ID Register [15:0]
31 30 29 28 27 26 25 24 23 22 21 20
19 18 17 16
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:20	AER - Next Capability Offset Register [11:0]		R	0x000
19:16	AER - Capability Version Register [3:0]		R	0x1
15:0	AER - Ext Capability ID Register [15:0]		R	0x0001

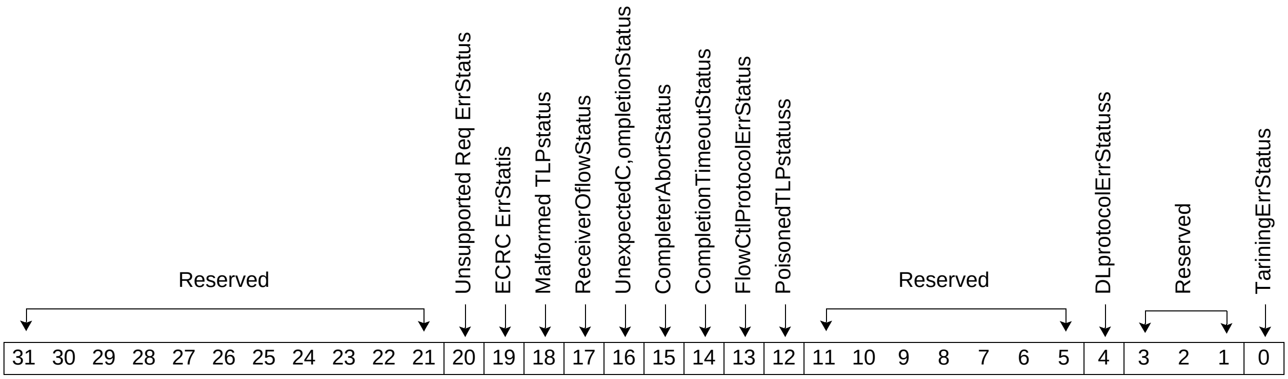
AER – Uncorrectable Error Status Register

The 32-bit read/write one-to-clear/sticky Uncorrectable Error Status Register reports error status of the individual uncorrectable error sources (nonfatal and fatal) on a PCI Express device.

Reset Value 0x00000000

Address Offset 0x104

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:21	Reserved	RsvdZ, per the PCI Express specification	R	0b00000000000
20	AER - Unsupported Request Error Status	Register Bit I/O Signals: AL_EC08_UR	R/W	0b0
19	AER - ECRC Error Status	ECRC is not supported in CPC945. Register Bit I/O Signals: AL_AER04_ECRC_ERROR SYS_AER18_ECRC_CHECK_AVAIL	R/W	0b0
18	AER - Malformed TLP Status	Register Bit I/O Signals: AL_EC08_MALFORMED_TLP	R/W	0b0
17	AER - Receiver Overflow Status	Register Bit I/O Signals: AL_EC08_REC_OVERFLOW	R/W	0b0
16	AER - Unexpected Completion Status	Register Bit I/O Signals: AL_EC08_UNEXPECTED_CMPL	R/W	0b0
15	AER - Completer Abort Status	Register Bit I/O Signals: AL_PCI04_SIG_CMPL_ABORT	R/W	0b0
14	AER - Completion Timeout Status	Register Bit I/O Signals: AL_EC08_CMPL_TIMEOUT	R/W	0b0
13	AER - Flow Control Protocol Error Status	Register Bit I/O Signals: TL_EC08_FCPE	R/W	0b0
12	AER - Poisoned TLP Status	Register Bit I/O Signals: AL_PCI04_REC_POISONED_TLP AL_PCI1C_REC_POISONED_TLP	R	0b0
11:5	Reserved	RsvdZ, per the PCI Express specification	R	0b0000000
4	AER - Data Link Protocol Error Status	Register Bit I/O Signals: DL_EC08_DLLPE	R/W	0b0

Bits	Field Name	Description	Access	Reset
3:1	Reserved	RsvdZ, per the PCI Express specification	R	0b000
0	AER - Training Error Status		R	0b0

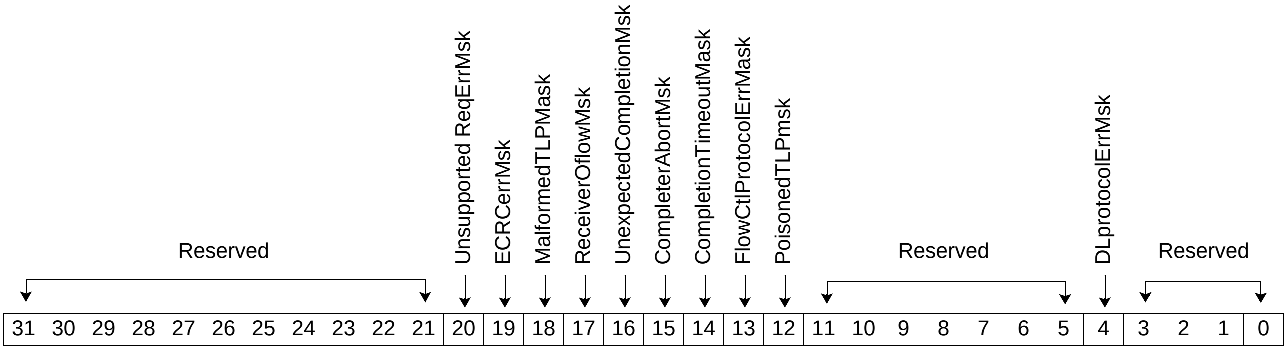
AER – Uncorrectable Error Mask Register

The 32-bit read/write/sticky Uncorrectable Error Mask Register controls reporting of the individual uncorrectable error sources (nonfatal and fatal) of a PCI Express device to the PCI Express root complex using the PCI Express defined error messages.

Reset Value 0x00000000

Address Offset 0x108

Access Type Read, Read/Write



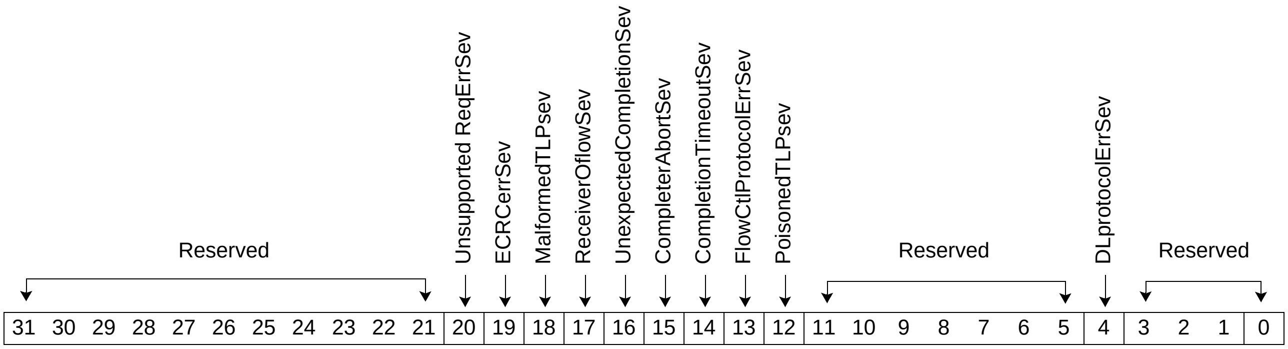
Bits	Field Name	Description	Access	Reset
31:21	AER - Reserved Bits	RsvdP, per the PCI Express specification	R	0b00000000000
20	AER - Unsupported Request Error Mask		R/W	0b0
19	AER - ECRC Error Mask	Not available in CPC945. Register bit I/O Signals: SYS_AER18_ECRC_CHECK_AVAIL	R	0b0
18	AER - Malformed TLP Mask		R/W	0b0
17	AER - Receiver Overflow Mask		R/W	0b0
16	AER - Unexpected Completion Mask		R/W	0b0
15	AER - Completer Abort Mask		R/W	0b0
14	AER - Completion Timeout Mask		R/W	0b0
13	AER - Flow Control Protocol Error Mask		R/W	0b0
12	AER - Poisoned TLP Mask		R/W	0b0
11:5	AER - Reserved Bits	RsvdP, per the PCI Express specification	R	0b0000000

Bits	Field Name	Description	Access	Reset
4	AER - Data Link Protocol Error Mask		R/W	0b0
3:1	AER - Reserved Bits	RsvdP, per the PCI Express specification	R	0b000
0	AER - Training Error Mask		R	0b0

AER – Uncorrectable Error Severity Register

The 32-bit read/write/sticky Uncorrectable Error Severity Register controls whether individual uncorrectable errors are reported as nonfatal or fatal. Individual errors are reported as fatal when the severity bit is set equal to ‘1’, nonfatal when the severity bit is set equal to ‘0’.

Reset Value	0x00062010
Address Offset	0x10C
Access Type	Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:21	AER - Reserved Bits	RsvdP, per the PCI Express specification	R	0b00000000000
20	AER - Unsupported Request Error Severity		R/W	0b0
19	AER - ECRC Error Severity	Not available in CPC945.	R	0b0
18	AER - Malformed TLP Severity		R/W	0b1
17	AER - Receiver Overflow Severity		R/W	0b1
16	AER - Unexpected Completion Severity		R/W	0b0
15	AER - Completer Abort Severity		R/W	0b0
14	AER - Completion Timeout Severity		R/W	0b0
13	AER - Flow Control Protocol Error Severity		R/W	0b1

Bits	Field Name	Description	Access	Reset
12	AER - Poisoned TLP Severity		R/W	0b0
11:5	AER - Reserved Bits	RsvdP, per the PCI Express specification	R	0b0000000
4	AER - Data Link Protocol Error Severity		R/W	0b1
3:1	AER - Reserved Bits	RsvdP, per the PCI Express specification	R	0b000
0	AER - Training Error Severity		R	0b0

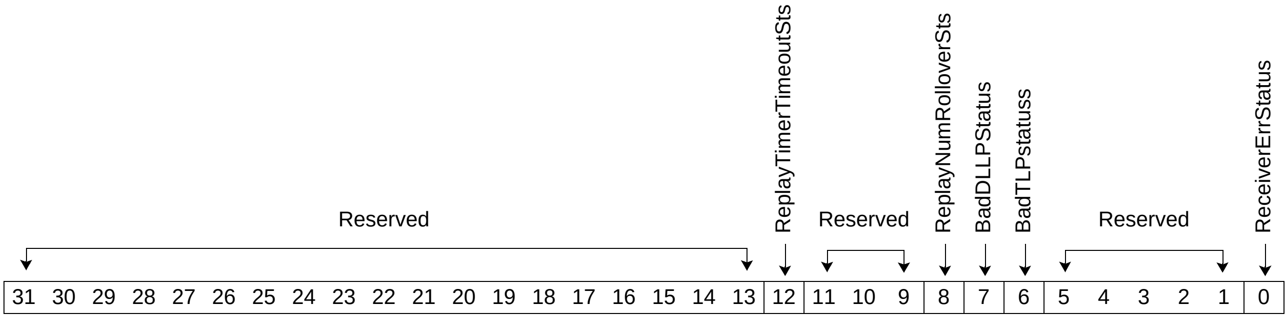
AER – Correctable Error Status Register

The 32-bit read/write one to clear/sticky Correctable Error Status Register reports error status of the individual correctable error sources on a PCI Express device.

Reset Value 0x00000000

Address Offset 0x110

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:13	AER - Reserved Bits	RsvdZ, per the PCI Express specification	R	0b0000000000000000000
12	AER - Replay Timer Timeout Status	Register Bit I/O Signals: DL_EC08_REPLAYTIMEOUT	R/W	0b0
11:9	AER - Reserved Bits	RsvdZ, per the PCI Express specification	R	0b000
8	AER - REPLAY_NUM Rollover Status	Register Bit I/O Signals: DL_EC08_REPLAYROLLOVER	R/W	0b0
7	AER - Bad DLLP Status	Register Bit I/O Signals: DL_EC08_BADDLLP	R/W	0b0
6	AER - Bad TLP Status	Register Bit I/O Signals: DL_EC08_BADTLP	R/W	0b0
5:1	AER - Reserved Bits	RsvdZ, per the PCI Express specification	R	0b00000
0	AER - Receiver Error Status	Register Bit I/O Signals: DL_EC08_RECEIVERERROR	R/W	0b0

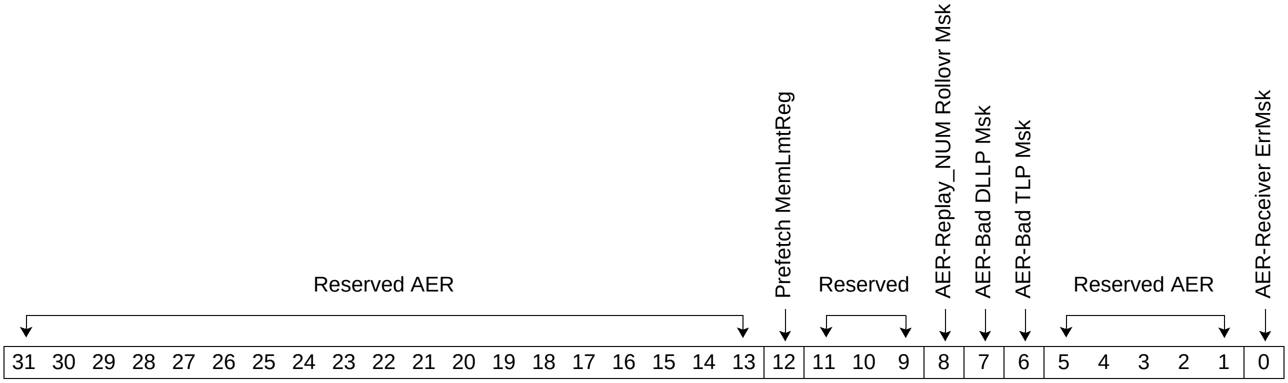
AER – Correctable Error Mask Register

The 32-bit read/write/sticky Correctable Error Mask Register controls reporting of the individual correctable error sources of a PCI Express device to the PCI Express root complex using the PCI Express defined error messages.

Reset Value 0x00000000

Address Offset 0x114

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
31:13	Reserved	RsvdP, per the PCI Express specification	R	0x00000
12	Prefetchable Memory Limit Register [03:00]		R/W	0b0
11:9	Reserved	RsvdP, per the PCI Express specification	R	0b000
8	AER-Replay_NUM Rollovr Msk		R/W	0b0
7	AER-Bad DLLP Msk		R/W	0b0
6	AER-Bad TLP Msk		R/W	0b0
5:1	Reserved	RsvdP, per the PCI Express specification	R	0b00000
0	AER - Receiver Error Mask		R/W	0b0

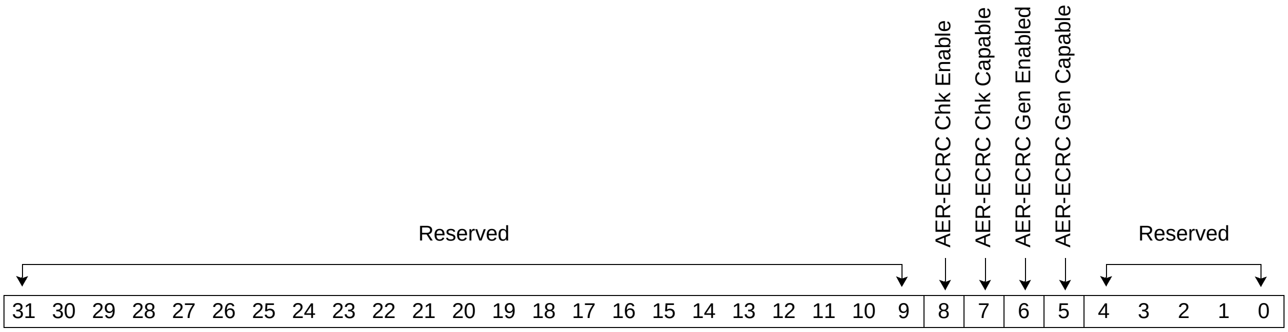
AER – Capabilities and Control Register

The AER Capabilities and Control Register contains the First Error Pointer Register and also controls and reports the state of the ECRC generation capabilities of the PCI Express device.

The First Error Pointer Register identifies the bit position of the first error reported in the uncorrectable error status register. If multiple uncorrectable errors occur on the same PCLK250 clock cycle, the PCIEXCFG core uses the following error priority for setting the first error pointer register value (listed from lowest to highest priority):

DL_EC08_DLLPE
TL_EC08_FCPE
AL_EC08_CMPL_TIMEOUT
AL_EC08_REC_OVERFLOW
AL_EC08_UR
AL_PCI04_SIG_CMPL_ABORT
AL_PCI1C_SIG_CMPL_ABORT
AL_EC08_UNEXPECTED_CMPL
AL_PCI04_REC_POISONED_TLP
AL_PCI1C_REC_POISONED_TLP
AL_EC08_MALFORMED_TLP
AL_AER04_ECRC_ERROR

Reset Value	0x00000000
Address Offset	0x118
Access Type	Read



Bits	Field Name	Description	Access	Reset
31:9	Reserved	RsvdP, per the PCI Express specification	R	0b00000000000 000000000000
8	AER-ECRC Check Enable	Not supported in the CPC945, always reads as ‘0’	R	0x0
7	AER-ECRC Check Capable	Not supported in the CPC945, always reads as ‘0’	R	0x0
6	AER-ECRC Generation Enable	Not supported in the CPC945, always reads as ‘0’	R	0x0
5	AER-ECRC Generation Capable	Not supported in the CPC945, always reads as ‘0’	R	0x0

Bits	Field Name	Description	Access	Reset
4:0	AER-First Error Pointer Register	Points to the bit position of the error in uncorrectable status register at offset 0x104. Register bit I/O signals: DL_EC08_DLLPE TL_EC08_FCPE AL_EC08_CMPL_TIMEOUT AL_EC08_REC_OVERFLOW AL_EC08_UR AL_PCI04_SIG_CMPL_ABORT AL_PCI1C_SIG_CMPL_ABORT AL_EC08_UNEXPECTED_CMPL AL_PCI04_REC_POISONED_TLP AL_PCI1C_REC_POISONED_TLP AL_EC08_MALFORMED_TLP AL_AER04_ECRC_ERROR	R	0b00000

AER – Header Log Registers: 1 through 4

The AER Header Log Registers capture the TLP header of the packet corresponding to the first loggable nonmasked uncorrectable error.

The TLP header is captured in the AER Header Log Registers (1–4) such that:

- Header Log Register #1 (AER1C) contains the TLP header bytes 0–3
  - Where TLP header byte 3 is the least significant byte of the AER header log register
- Header Log Register #2 (AER20) contains the TLP header bytes 4–7
  - Where TLP header byte 7 is the least significant byte of the AER header log register
- Header Log Register #3 (AER24) contains the TLP header bytes 8–11
  - Where TLP header byte 11 is the least significant byte of the AER header log register
- Header Log Register #4 (AER28) contains the TLP header bytes 12–15
  - Where TLP header byte 15 is the least significant byte of the AER header log register
  - This value in this register is undefined for when 12-byte TLP headers are logged

Reset Value 0x000001000

Address Offset Header Log Register 1 of 4: 0x11C
Header Log Register 2 of 4: 0x120
Header Log Register 3 of 4: 0x124
Header Log Register 4 of 4: 0x128

Access Type Read/Write

Bits	Register	Description	Access	Reset
31:0	AER - (AER1C) Header Log Register #1	TLP header bytes 0:3 Register Bit I/O Signals: AL_AER1C_HEADER_LOG[31:0]	R/W	0x00000000
31:0	AER - (AER20) Header Log Register #2	TLP header bytes 4:7 Register Bit I/O Signals: AL_AER1C_HEADER_LOG[63:31]	R/W	0x00000000
31:0	AER - (AER24) Header Log Register #3	TLP header bytes 8:11 Register Bit I/O Signals: AL_AER1C_HEADER_LOG[95:64]	R/W	0x00000000
31:0	AER - (AER28) Header Log Register #4	TLP header bytes 12:15 Register Bit I/O Signals: AL_AER1C_HEADER_LOG[127:96]	R/W	0x00000000

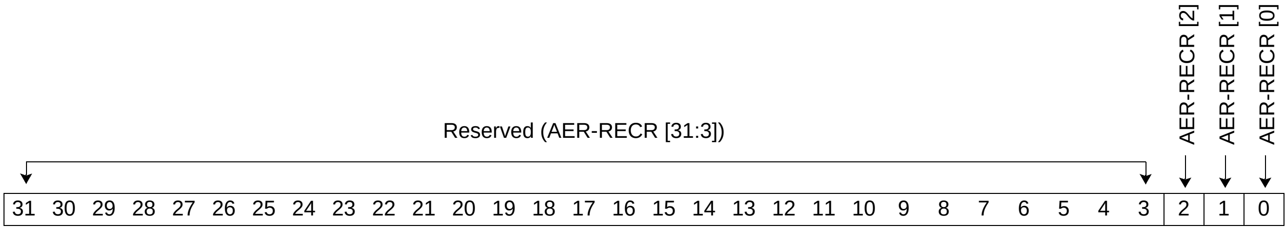
AER – Root Error Control Register

The AER Root Error Control Register allows additional control of the root complex response to correctable, nonfatal, and fatal error messages. This register enables and disables AER interrupt generation for each error type.

Reset Value 0x0000 0000

Address Offset 0x012C

Access Type Read, Read/Write

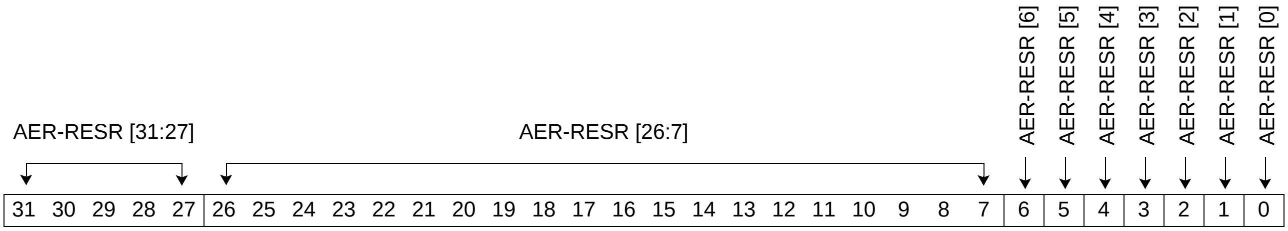


Bits	Field Name	Description	Access	Reset
31:3	Reserved. AER - Root Error Control Registers [31:3]	RsvdP, per the PCI Express specification	R	0x0000000
2	AER - Root Error Control Register [2]	Fatal Error Reporting Enable. Register Bit I/O Signals: CFG_AER2C_FATAL_REPORT_EN	R/W	0b0
1	AER - Root Error Control Register [1]	Nonfatal Error Reporting Enable. Register Bit I/O Signals: CFG_AER2C_NONFATAL_REPORT_EN	R/W	0b0
0	AER - Root Error Control Register [0]	Correctable Error Reporting Enable. Register Bit I/O Signals: CFG_AER2C_CORR_REPORT_EN	R/W	0b0

AER – Root Error Status Register

The AER Root Error Status Register reports the status of all errors (correctable, nonfatal, fatal) in the PCI Express hierarchy of the root port device (both internal rootport errors and upstream error messages). It also contains the AER message signaled interrupt number associated with the AER reporting interrupt.

Reset Value	0x00000000
Address Offset	0x130
Access Type	Mixed



Bits	Field Name	Description	Access	Reset
31:27	AER - Root Error Status Register [31:27]	Advanced Error Interrupt Message Number AL_AER30_MSI_NUMBER. Note: The CPC945 does not support multiple MSIs.	R	0x0
26:7	AER - Root Error Status Register [26:7]	RsvdZ, per the PCI Express specification	R	0x00000
6	AER - Root Error Status Register [6]	Fatal Error Received. Register Bit I/O Signals: CFG_EC08_FATAL_ERROR AL_PCI1C_FATAL_ERROR_MSG	R/W	0b0
5	AER - Root Error Status Register [5]	Nonfatal Error Received. Register Bit I/O Signals: CFG_EC08_NONFATAL_ERROR AL_PCI1C_NONFATAL_ERROR_MSG	R/W	0b0
4	AER - Root Error Status Register [4]	First Uncorrectable Fatal. Register Bit I/O Signals: CFG_EC08_FATAL_ERROR AL_PCI1C_FATAL_ERROR_MSG	R/W	0b0
3	AER - Root Error Status Register [3]	Multiple ERR_FATAL/NONFATAL Received. Register Bit I/O Signals: CFG_EC08_FATAL_ERROR CFG_EC08_NONFATAL_ERROR AL_PCI1C_FATAL_ERROR_MSG AL_PCI1C_NONFATAL_ERROR_MSG	R/W	0b0
2	AER - Root Error Status Register [2]	ERR_FATAL/NONFATAL Received. Register Bit I/O Signals: CFG_EC08_FATAL_ERROR CFG_EC08_NONFATAL_ERROR AL_PCI1C_FATAL_ERROR_MSG AL_PCI1C_NONFATAL_ERROR_MSG	R/W	0b0
1	AER - Root Error Status Register [1]	Multiple ERR_CORR Received. Register Bit I/O Signals: CFG_EC08_CORR_ERROR AL_AER30_CORR_ERROR_MSG	R/W	0b0

Bits	Field Name	Description	Access	Reset
0	AER - Root Error Status Register [0]	ERR_CORR Received. Register Bit I/O Signals: CFG_EC08_CORR_ERROR AL_AER30_CORR_ERROR_MSG	R/W	0b0

AER – Error Source Identification Register

The AER Error Source Identification Register reports the source (requestor ID) of the first correctable and uncorrectable (nonfatal and fatal) errors reported in the AER root status register. There are two sets of source ID signals feeding this register based on the origination of the error message:

- Source ID for internal rootport generated errors (AL_AER34_SOURCE_ID)
- Source ID from upstream error messages (AL_AER34_SOURCE_ID_MSG)

The PCIEXCFG core always gives the upstream message errors priority over internal rootport errors when both errors occur on the same PCLK250 cycle (the assumption is that the upstream message based errors must have occurred earlier in time).

Reset Value 0x00000000

Address Offset 0x134

Access Type Read Only

AER-ERR_FATAL/ NONFATAL_SOURCE_IDENTIFICATION_REGISTER	AER-ERR_CORR_SOURCE_IDENTIFICATION_REGISTER
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	Field Name	Description	Access	Reset
31:16	AER-ERR_FATAL/ NONFATAL_SOURCE_IDENTIFICATION_REGISTER	Register bit I/O signals: AL_AER34_SOURCE_ID AL_AER34_SOURCE_ID_MSG Note that the PCIEXCFG core always gives upstream message errors priority over internal rootport errors.	R	0x0000
15:0	AER-ERR_CORR_SOURCE_IDENTIFICATION_REGISTER	Register bit I/O signals: AL_AER34_SOURCE_ID AL_AER34_SOURCE_ID_MSG Note that the PCIEXCFG core always gives upstream message errors priority over internal rootport errors.	R	0x0000

12.11.2 XBus PCI Express Configuration Registers

The XBus registers are essentially an extension of the IBM configuration registers, but are unique to the CPC945. The majority, though not all, of these registers are used to program various parts of the PCI Express Application Layer, providing a means to tweak performance. The rest of the registers record errors or unsupported transactions. A complete list of the XBus configuration registers along with their corresponding addresses is shown in *Table 12-23*. Remember that accessing these registers through I^{2}C, SBUS or a Limited Direct Configuration Access requires a different address. Each individual register description is present in the following subsections. For each register there are two addresses listed. The first address is for the classic configuration access method and the second address is for the SBUS or I^{2}C.

There are 3 types of XBus registers: read only, read/write, and protected. Read only registers can be read from but not written to, while read/write registers can be both read from and written to. Protected registers can always be read from, but must be unlocked before being written to. For information on unlocking protected registers, see *“Unlock Protected Register” on page 575*.

Table 12-23. XBus PCIe Configuration Registers

Address	Register Name	Type	Initial Value	Page
0xF0BF_F0F0	32-bit Legacy Interrupt Control	Read Only, Read/Write	0x0000_000F	553
0xF0BF_F0F4	32-bit Link Integrity Interrupt Control	Read/Write	0x0000_000F	554
0xF0BF_F0F8	32-bit Link Down Interrupt Control	Read/Write	0x0000_010F	555
0xF0BF_F0FC	32-bit PCI Express 0 Address Mask	Read/Write	0x0000_0003	556
0xF0BF_FF00	32-bit Memory Read Completion Time-Out	Protected	0x0026_25A0	558
0xF0BF_FF04	32-bit I/O Completion Time-Out	Protected	0x0026_25A0	558
0xF0BF_FF08	32-bit Configuration Completion Time-Out	Protected	0x0026_25A0	559
0xF0BF_FF0C	32-bit Local Completion Time-Out	Protected	0x0000_30D4	559
0xF0BF_FF10	5-bit Maximum Advertised Posted Credits	Read Only	0x10	560
0xF0BF_FF14	5-bit Maximum Advertised Nonposted Credits	Read Only	0x10	560
0xF0BF_FF18	4-bit Number Of Reserved Posted Credits	Protected	0x0	562
0xF0BF_FF1C	4-bit Number Of Reserved Nonposted Credits	Protected	0x0	563
0xF0BF_FF20	5-bit Maximum Available Tags	Protected	0x00	564
0xF0BF_FF24	4-bit Completion Arbiter Priority	Read/Write	0x0	565
0xF0BF_FF28	8-bit Version Number	Read Only	0x00	566
0xF0BF_FF2C	1-bit L1 Power Mode Request Response	Protected	0x0	567
0xF0BF_FF30	32-bit Interrupt Filter (UNUSED)	Read/Write	0x0000_0000	568
0xF0BF_FF34	11-bit Last NAK’d Write Address	Read Only	0x000	568
0xF0BF_FF38	32-bit Transmission Error Count	Read Only	0x0000_0000	569
0xF0BF_FF3C	1-bit Dispatch Read Mode	Protected	0x0	569
0xF0BF_FF40	1-bit No Snoop Request Mode	Protected	0x0	570

Table 12-23. XBus PCIe Configuration Registers (Continued)

Address	Register Name	Type	Initial Value	Page
0xF0BF_FF44	1-bit Direct Access Mode	Protected	0x0	571
0xF0BF_FF48	32-bit L23 Message Time-Out	Protected	0xFFFF_FFFF	572
0xF0BF_FF4C	14-bit Invalid Transaction	Read Only	0x0000	572
0xF0BF_FF50	1-bit Configuration 4 or 8	Protected	0x0	574
0xF0BF_FF54	1-bit Unlock Protected	Read/Write	0x0	575
0xF0BF_FF58	2-bit Coherent Memory Write Tag Delay (UNUSED)	Protected	0x3	576
0xF0BF_FF5C	1-bit Block Transactions During Configuration Reads	Protected	0x1	577
0xF0BF_FF60	32-bit CRC Error Count	Read Only	0x0000_0000	578
0xF0BF_FF64	1-bit Unsupported Request or Completer Abort	Protected	0x0	578
0xF0BF_FF68	1-bit Enable TEA on Unsupported Request Completion	Protected	0x0	579
0xF0BF_FF6C	1-bit Enable TEA on Completer Abort Completion	Protected	0x0	580
0xF0BF_FF70	1-bit Enable TEA on Configuration Retry Time-Out	Protected	0x0	581
0xF0BF_FF74	1-bit Enable TEA on Completion Time-Out	Protected	0x0	582
0xF0BF_FF78	1-bit Set PCIE04 Received Completer Abort on Completer Abort	Protected	0x0	583

12.11.2.1 Legacy Interrupt Control Register

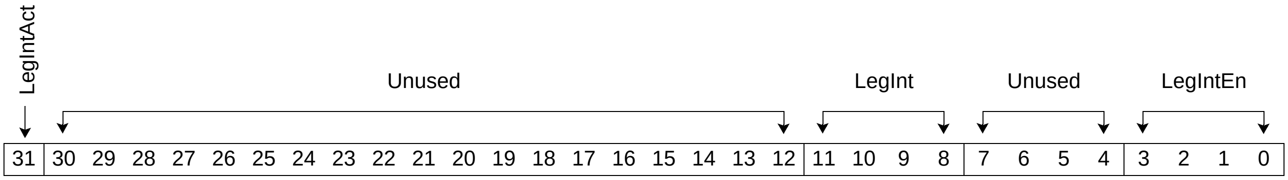
The Legacy Interrupt Control Register is a 32-bit register providing support for legacy PCI interrupts. Though the register is listed as having 32 bits, this is not exactly true for several reasons. First of all, only 9 bits are actually used. Secondly, of the 9 bits, only four are register bits. The other five are tied to various signals.

Extended Register Number 0b0000

Register Number 0b111100

Access Type Read, Read/Write

Address 0xF0BFF0F0, 0xF80903C0

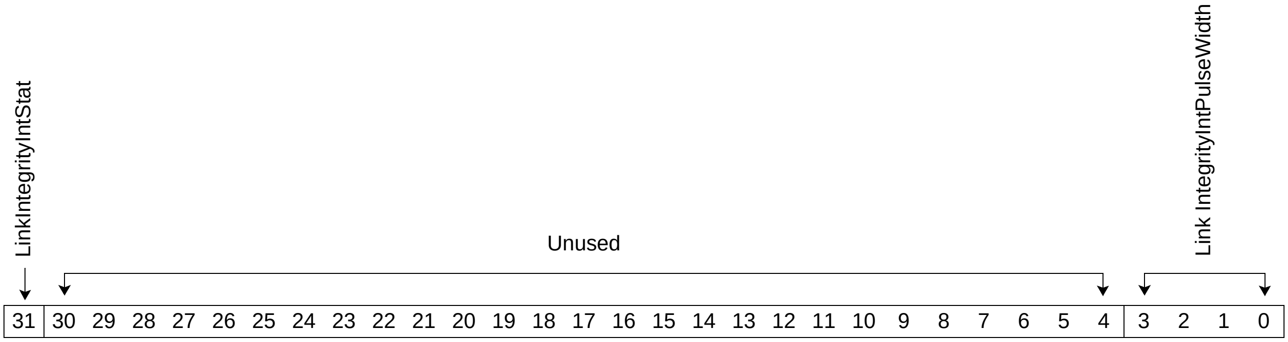


Bits	Field Name	Description	Access	Reset
31	Legacy Interrupt Active	This bit is tied to a signal that indicates when an interrupt is detected. This signal is created by ANDing each interrupt signal with its corresponding enable bit, and then ORing the results together.	R	0b0
30:12	Unused	Returns 0 on read.	R	0x00000
11:8	Legacy Interrupt	These bits are tied to the actual interrupt signals A (bit 8), B (bit 9), C (bit 10), and D (bit 11).	R	0x0
7:4	Unused	Returns 0 on read.	R	0x0
3:0	Legacy Interrupt Enable	Enable bits for interrupts A (bit 0), B (bit 1), C (bit 2), and D (bit 3). Any assertion of an interrupt signal when its corresponding enable bit is not set is ignored.	R/W	0xF

12.11.2.2 Link Integrity Interrupt Control Register

The Link Integrity Interrupt Control Register is a 32-bit register used to set characteristics of and to clear the Link Integrity Interrupt signal. Thought the register is listed as having 32 bits, only 5 bits ([3:0] and [31]) actually exist.

Extended Register Number	0b0000
Register Number	0b111101
Access Type	Read/Write
Address	0xF0BFF0F4, 0xF80903D0



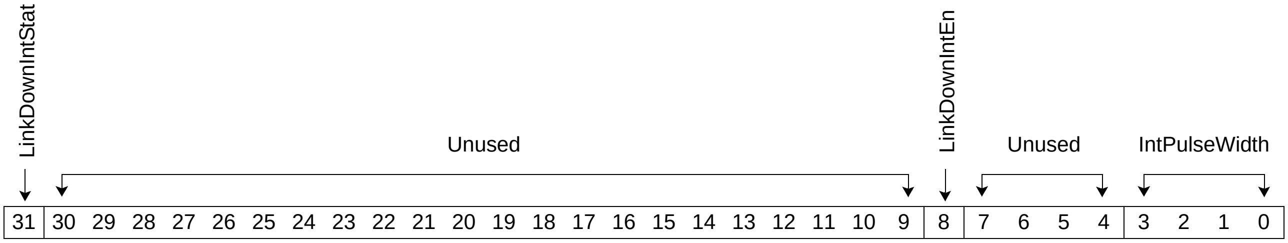
Bits	Field Name	Description	Access	Reset
31	LinkIntegrityIntStat	This bit serves two purposes, to check (via a read) and to clear (using a write) the Link Integrity interrupt. A read returns the value of the Link Integrity Interrupt signal itself, while writing 0b1 results in a 1 PCI Express clock cycle wide pulse of the Clear Link Integrity Interrupt signal. Note that writing 0b0 to this bit has no effect.	R/W	0b0
30:4	Unused	Returns 0 on a read.	R	0x0000000
3:0	Link IntegrityIntPulseWidth	Indicates the width (in PCI Express clock cycles) of both the Link Integrity Interrupt signal and the Link Down Interrupt signal. A value of 0b0001 corresponds to a width of 1 PCI Express clock cycle, 0b0010 corresponds to a width of 2 PCI Express clock cycles, etc. A value of 0b0000 indicates that, instead of being pulsed, the Link Integrity Interrupt signal is set until the interrupt is cleared.	R/W	0xF

12.11.2.3 Link Down Interrupt Control Register

The Link Down Interrupt Control Register is a 32-bit register used to set characteristics of, to enable, and to clear the Link Down Interrupt signal. Though this register is listed as having 32-bits, only 6 bits ([3:0], [8] and [31]) actually exist.

.

Extended Register Number	0b0000
Register Number	0b111110
Access Type	Read/Write
Address	0xF0BFF0F8, 0xF80903E0



Bits	Field Name	Description	Access	Reset
31	LinkDownIntStat	This bit serves two purposes, to check (via a read) and to clear (using a write) the Link Down interrupt. A read of bit 31 returns the values of the Link Down Interrupt signal itself, while writing 0b1 to bit 31 results in a 1 PCI Express clock cycle wide pulse of the Clear Link Down Interrupt signal. **Note:** Writing 0b0 to this bit has no effect.	R/W	0b0
30:9	Unused	Returns 0 on a read.	R	0x000000
8	LinkDownIntEn	This is the enable bit for the Link Down interrupt. If this bit is not set, the Link Down Interrupt signal is ignored.	R/W	0b1
7:4	Unused	Returns 0 on a read.	R	0x0
3:0	IntPulseWidth	Set the interrupt pulse width for PCIe interrupts 0x0 Infinite. Level sensitive, must be cleared manually. 0x1-0xF Number of 250 MHz clock cycles an interrupt signal is asserted.	R/W	0xF

12.11.2.4 PCI Express 0 Address Mask Register

The PCI Express 0 Address Mask Register is a 32-bit register used to indicate the address range within the first 4 GB of system memory designated for PCI Express transactions. This is done by using two fields: the Coarse Address Select field and the Fine Address Select field.

Extended Register Number	0b0000
Register Number	0b111111
Access Type	Read/Write
Address	0xF0BFF0FC, 0xF80903F0

PCIEAddrMaskCAS																PCIEAddrMaskFAS															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:16	Coarse Address Select	Each bit in this field represents a 256 MB region within the first 4 GB of system memory (*Table 12-24*). If any bit within this field is set, its corresponding region in system memory is designated as PCI Express memory space. Note that bits 23:16 must always be 0 as this region of memory is permanently assigned to system RAM. Also note that bit 31 must always be 0, as this region of memory is controlled by the Fine Address Select field. See *Table 12-24*.	R/W	0x0000
15:0	Fine Address Select	Each bit in this field represents a 16 MB region within the last 256 MB of the first 4 GB of system memory (*Table 12-25* on page 557). If any bit within this field is set, its corresponding region in system memory is designated as PCI Express memory space. Note that bits 1:0 are always set. See *Table 12-25* on page 557.	R/W	0x0003

Table 12-24. Coarse Address Select Field

Bit	Starting Address	Ending Address
16	0x00000000	0x0FFFFFFF
17	0x10000000	0x1FFFFFFF
18	0x20000000	0x2FFFFFFF
19	0x30000000	0x3FFFFFFF
20	0x40000000	0x4FFFFFFF
21	0x50000000	0x5FFFFFFF
22	0x60000000	0x6FFFFFFF
23	0x70000000	0x7FFFFFFF
24	0x80000000	0x8FFFFFFF
25	0x90000000	0x9FFFFFFF
26	0xA0000000	0xAFFFFFFF
27	0xB0000000	0xBFFFFFFF

Table 12-24. Coarse Address Select Field (Continued)

Bit	Starting Address	Ending Address
28	0xC0000000	0xCFFFFFFF
29	0xD0000000	0xDFFFFFFF
30	0xE0000000	0xEFFFFFFF
31	0xF0000000	0xFFFFFFFF

Table 12-25. Fine Address Select Field

Bit	Starting Address	Ending Address
0	0xF0000000	0xF0FFFFFF
1	0xF1000000	0xF1FFFFFF
2	0xF2000000	0xF2FFFFFF
3	0xF3000000	0xF3FFFFFF
4	0xF4000000	0xF4FFFFFF
5	0xF5000000	0xF5FFFFFF
6	0xF6000000	0xF6FFFFFF
7	0xF7000000	0xF7FFFFFF
8	0xF8000000	0xF8FFFFFF
9	0xF9000000	0xF9FFFFFF
10	0xFA000000	0xFAFFFFFF
11	0xFB000000	0xFBFFFFFF
12	0xFC000000	0xFCFFFFFF
13	0xFD000000	0xFDFFFFFF
14	0xFE000000	0xFEFFFFFF
15	0xFF000000	0xFFFFFFFF

12.11.2.5 Memory Read Completion Time-Out Register

Extended Register Number	0b1111
Register Number	0b000000
Access Type	Protected
Address	0xF0BFFF00, 0xF8093C00

MemRdCmpltnTimeOut

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:0	MemRdCmpltnTimeOut	Sets the maximum amount of time (in PCI Express clock cycles) that a read to memory is given to complete. A value of 0x00010000 corresponds to 2^{16} PCI Express clock cycles, 0x00100000 corresponds to 2^{20} PCI Express clock cycles, etc. A value of 0x00000000 results in all memory reads immediately timing out.	Protected	0x002625A0

12.11.2.6 I/O Completion Time-Out Register

Extended Register Number	0b1111
Register Number	0b000001
Access Type	Protected
Address	0xF0BFFF04, 0xF8093C10

IOCmpltnTimeOut

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:0	IOCmpltnTimeOut	Sets the maximum amount of time (in PCI Express clock cycles) that an I/O transaction is given to complete. A value of 0x00010000 corresponds to 2^{16} PCI Express clock cycles, 0x00100000 corresponds to 2^{20} PCI Express clock cycles, etc. A value of 0x00000000 results in all I/O transactions immediately timing out.	Protected	0x002625A0

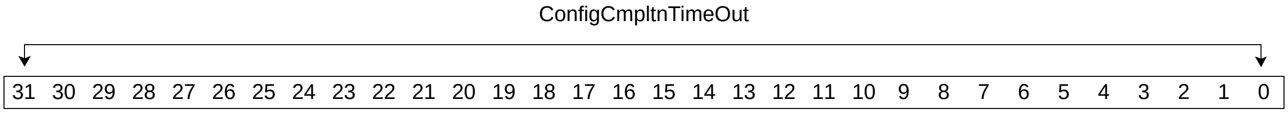
12.11.2.7 Configuration Completion Time-Out Register

Extended Register Number 0b1111

Register Number 0b000010

Access Type Protected

Address 0xF0BFFF08, 0xF8093C20



Bits	Field Name	Description	Access	Reset
31:0	ConfigCmpltnTimeOut	Sets the maximum amount of time (in PCI Express clock cycles) that a configuration transaction is given to complete. A value of 0x00010000 corresponds to 2^{16} PCI Express clock cycles, 0x00100000 corresponds to 2^{20} PCI Express clock cycles, etc. A value of 0x00000000 results in all configuration transactions immediately timing out.	Protected	0x002625A0

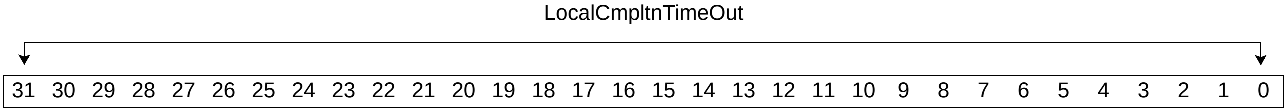
12.11.2.8 Local Completion Time-Out Register

Extended Register Number 0b1111

Register Number 0b000011

Access Type Protected

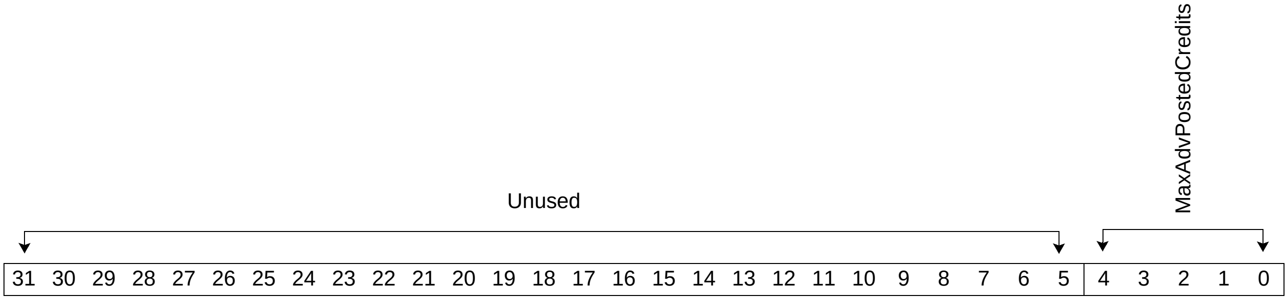
Address 0xF0BFFF0C, 0xF8093C30



Bits	Field Name	Description	Access	Reset
31:0	LocalCmpltnTimeOut	Sets the maximum amount of time (in PCI Express clock cycles) that a local transaction is given to complete. A value of 0x00010000 corresponds to 2^{16} PCI Express clock cycles, 0x00100000 corresponds to 2^{20} PCI Express clock cycles, etc. A value of 0x00000000 results in all local transactions immediately timing out.	Protected	0x000030D4

12.11.2.9 Maximum Advertised Posted Credits Register

Extended Register Number	0b1111
Register Number	0b000100
Access Type	Read Only
Address	0xF0BFFF10, 0xF8093C40



Bits	Field Name	Description	Access	Reset
31:5	Unused	Returns 0 on a read.	R	0x0000000
4:0	MaxAdvPostedCredits	Determines the maximum number of posted credits advertised over the PCI Express link.	R	0b10000

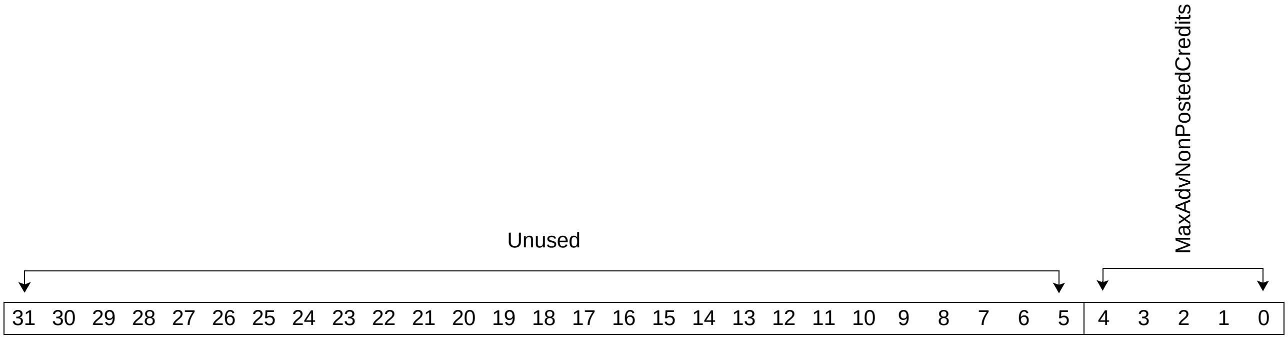
12.11.2.10 Maximum Advertised Nonposted Credits Register

Extended Register Number 0b1111

Register Number 0b000101

Access Type Read Only

Address 0xF0BFFF14, 0xF8093C50



Bits	Field Name	Description	Access	Reset
31:5	Unused	Returns 0 on a read.	R	0x0000000
4:0	MaxAdvNonPostedCredits	Determines the maximum number of nonposted credits advertised over the PCI Express link.	R	0b10000

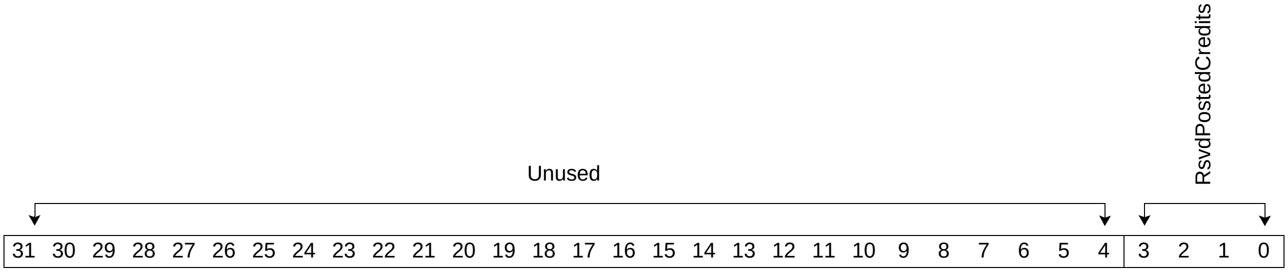
12.11.2.11 Number of Reserved Posted Credits Register

Extended Register Number 0b1111

Register Number 0b000110

Access Type Protected

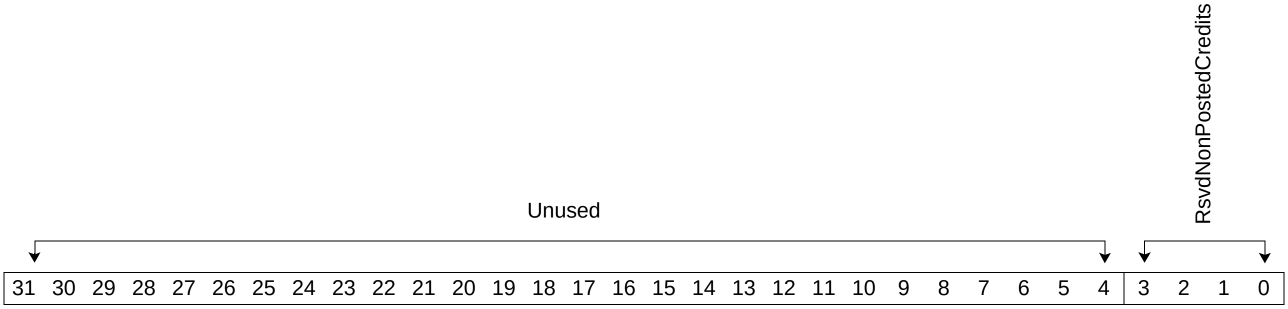
Address 0xF0BFFF18, 0xF8093C60



Bits	Field Name	Description	Access	Reset
31:4	Unused	Returns 0 on a read.	R	0x0000000
3:0	RsvdPostedCredits	Varies the number of posted credits the credit manager holds in reserve. The actual number of advertised posted credits is calculated by subtracting the value of this register from the value of the Maximum Advertised Posted Credits Register. Anywhere from 0 (0b0000) to 15 (0b1111) credits can be held in reserve.	Protected	0x0

12.11.2.12 Number of Reserved Nonposted Credits Register

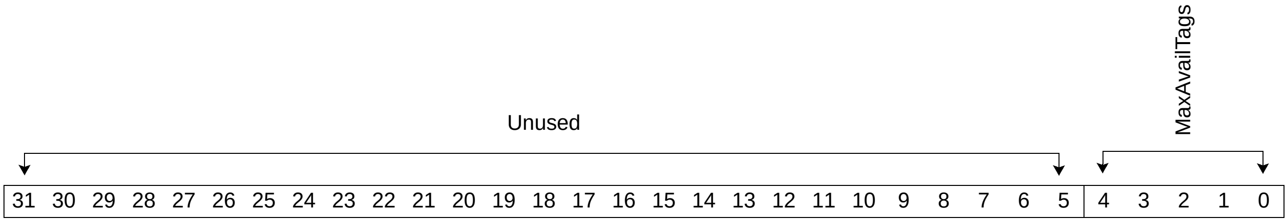
Extended Register Number	0b1111
Register Number	0b000111
Access Type	Protected
Address	0xF0BFFF1C, 0xF8093C70



Bits	Field Name	Description	Access	Reset
31:4	Unused	Returns 0 on a read.	R	0x0000000
3:0	RsvdNonPostedCredits	Varies the number of nonposted credits the credit manager holds in reserve. The actual number of advertised nonposted credits is calculated by subtracting the value of this register from the value of the Maximum Advertised Nonposted Credits Register. Anywhere from 0 (0b0000) to 15 (0b1111) credits can be held in reserve.	Protected	0x0

12.11.2.13 Maximum Available Tags Register

Extended Register Number	0b1111
Register Number	0b001000
Access Type	Protected
Address	0xF0BFFF20, 0xF8093C80



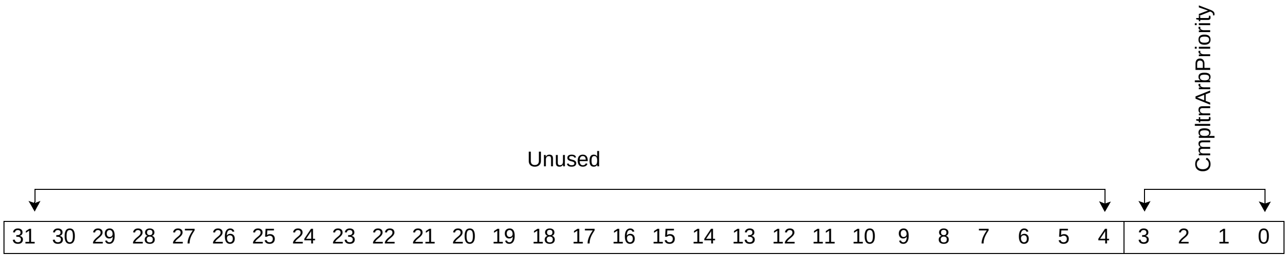
Bits	Field Name	Description	Access	Reset
31:5	Unused	Returns 0 on a read.	R	0x0000000
4:0	MaxAvailTags	Sets the maximum number of 16-byte entries in the Completion Data buffer located in the Inbound Completion Unit. A value of 0b00001 corresponds to 1 entry, 0b00010 corresponds to 2 entries, etc. A value of 0b00000 corresponds to the maximum number of entries, 32.	Protected	0x0

12.11.2.14 Completion Arbiter Priority Register

The Completion Arbiter Priority Register is a 4-bit register used to allow the priority of the four outbound completion interfaces to be increased. These interfaces are Registers (bit 3), HyperTransport (bit 2), Coherent Memory (bit 1), and Noncoherent Memory (bit 0). Writing 0b1 to one of these bits promotes that interface to high priority.

Arbitration between the various interfaces is done in a two-tiered round-robin system. The lower tier performs round-robin arbitration on all four interfaces, while the upper tier performs round-robin arbitration on only those interfaces that have been promoted to high priority. After one complete round-robin circuit has been made of the upper tier, a single round-robin step is performed on the lower tier, followed by another complete round-robin circuit of the upper tier, etc. For a more detailed description of the arbitration procedure, see Section *7. DDR2 Memory Controller on page 129*.

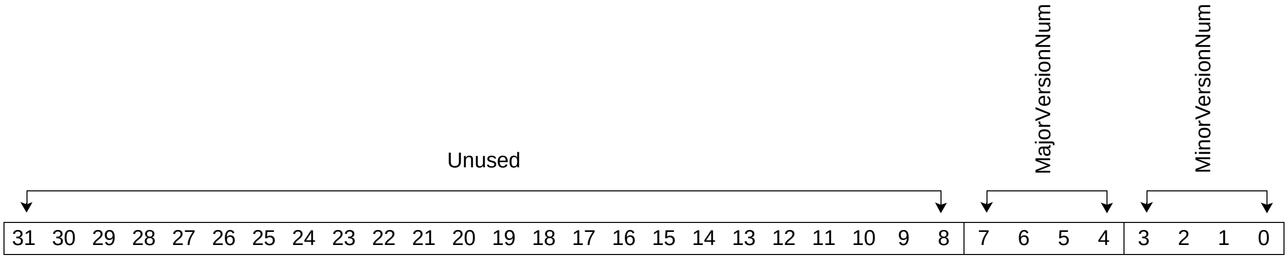
Extended Register Number	0b1111
Register Number	0b001001
Access Type	Read/Write
Address	0xF0BFFF24, 0xF8093C90



Bits	Field Name	Description	Access	Reset
31:4	Unused	Returns 0 on a read.	R	0x0000000
3:0	CmpltnArbPriority	Sets priority of the four outbound completion interfaces.	R/W	0x0

12.11.2.15 Version Number Register

Extended Register Number	0b1111
Register Number	0b001010
Access Type	Read Only
Address	0xF0BFFF28, 0xF8093CA0



Bits	Field Name	Description	Access	Reset
31:8	Unused	Returns 0 on a read.	R	0x000000
7:4	MajorVersionNum	CPC945 PCIe root complex major version number.	R	0x0
3:0	MinorVersionNum	CPC945 PCIe root complex minor version number.	R	0x0

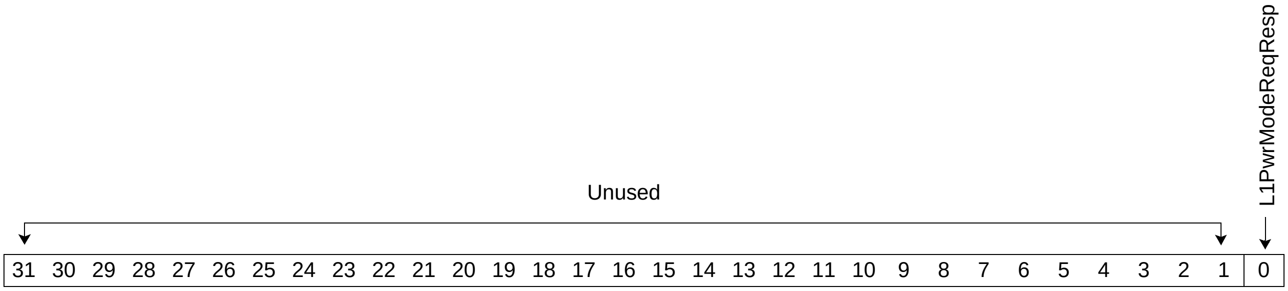
12.11.2.16 L1 Power Mode Request Response Register

Extended Register Number 0b1111

Register Number 0b001011

Access Type Protected

Address 0xF0BFFF2C, 0xF8093CB0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on a read.	R	0x00000000
0	L1PwrModeReqResp	Sets the way in which the Outbound Transaction Layer Interface responds to a request to enter the L1 power state (ASL1). A value of 0b1 causes the request to be ACK'd at the next transaction boundary, while a value of 0b0 causes the request to be ACK'd at the next transaction boundary only if another transaction isn't pending. If another transaction is pending, the request is NAK'd.	Protected	0x0

12.11.2.17 Interrupt Filter Register (UNUSED)

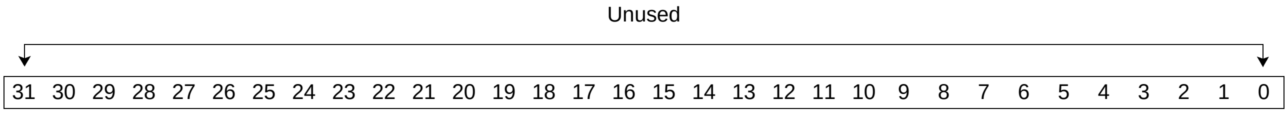
The Interrupt Filter Register is not used.

Extended Register Number 0b1111

Register Number 0b001100

Access Type Read/Write

Address 0xF0BFFF30, 0xF8093CC0



Bits	Field Name	Description	Access	Reset
31:0	Unused	Returns 0 on a read.	R/W	0x00000000

12.11.2.18 Last NAK'd Write Address Register

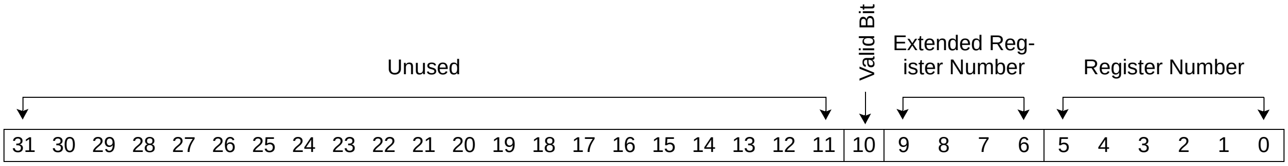
Keeps track of the most recent unused address in configuration space (from 0xF0BFF000 to 0xF0BFFFFC) targeted by a write. The address, if valid, is stored as the extended register number (bits 9:6) and the register number (bits 5:0).

Extended Register Number 0b1111

Register Number 0b001101

Access Type Read Only

Address 0xF0BFFF34, 0xF8093CD0



Bits	Field Name	Description	Access	Reset
31:11	Unused	Returns 0 on a read.	R	0x000000
10	Valid Bit	Valid bit. Indicates whether (0b1) or not (0b0) the address stored in the rest of the register corresponds to a NAK'd write. After the first NAK'd write occurs, the valid bit is always 0b1.	R	0b0
9:6	Extended Register Number	Extended register number of the address if valid.	R	0x0
5:0	Register Number	Register number of the address if valid.	R	0b000000

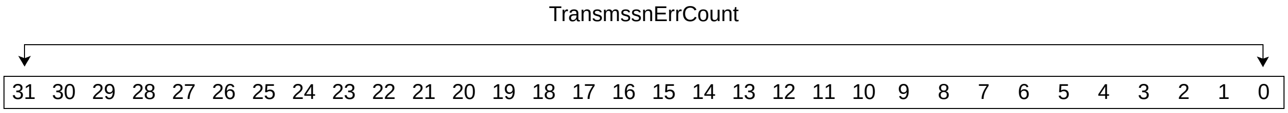
12.11.2.19 Transmission Error Count Register

Extended Register Number 0b1111

Register Number 0b001110

Access Type Read Only

Address 0xF0BFFF38, 0xF8093CE0



Bits	Field Name	Description	Access	Reset
31:0	TransmssnErrCount	Tracks the number of transmission errors that have occurred. Any write to this register, regardless of the value written, resets the count to 0.	R	0x00000000

12.11.2.20 Dispatch Read Mode Register

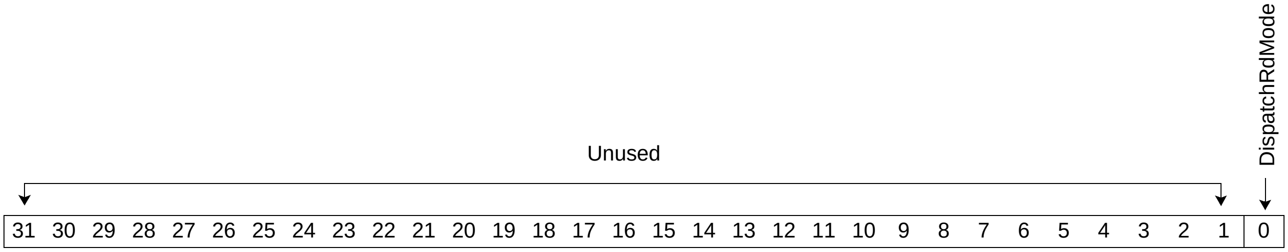
The Dispatch Read Mode Register is a 1-bit register that is used to set when an inbound write is considered issued. It is necessary to know when a given write is considered issued because any reads being blocked by a write might not be released until after the write has been issued.

Extended Register Number 0b1111

Register Number 0b001111

Access Type Protected

Address 0xF0BFFF3C, 0xF8093CF0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on a read.	R	0x00000000
0	DispatchRdMode	Determines when an inbound write is considered issued. Writing 0b1 to this register results in an inbound write being considered issued only after both the command and the data have been dispatched. Writing 0b0 to this register results in an inbound write being considered issued after only the command has been dispatched.	Protected	0b0

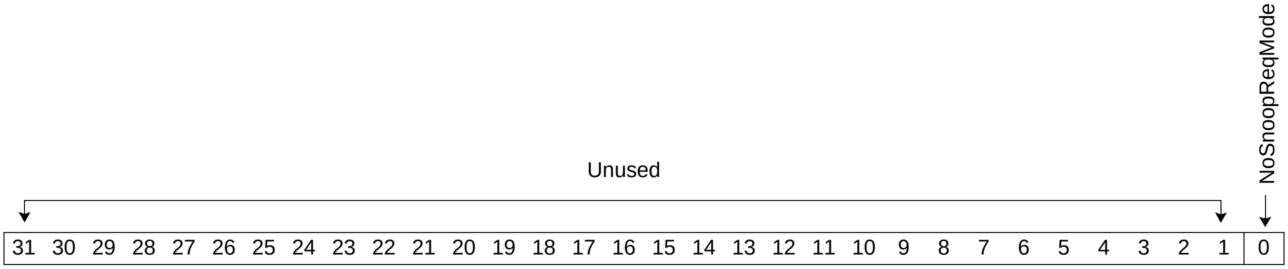
12.11.2.21 No Snoop Request Mode Register

Extended Register Number 0b1111

Register Number 0b010000

Access Type Protected

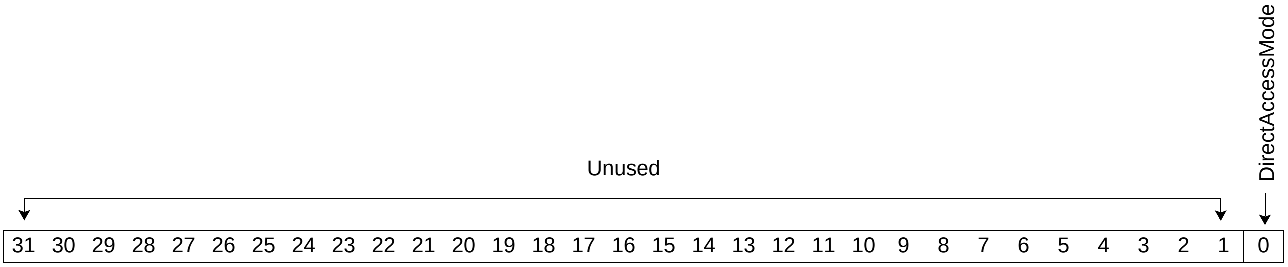
Address 0xF0BFFF40, 0xF8093D00



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on a read.	R	0x00000000
0	No Snoop Request Mode	Determines whether or not the No Snoop bit in packet headers should be ignored. Writing a 0b1 to the register causes the No Snoop bit to be ignored, resulting in every inbound transaction being snooped. Writing a 0b0 to the register results in transactions marked as No Snoop not being snooped.	Protected	0b0

12.11.2.22 Direct Access Mode Register

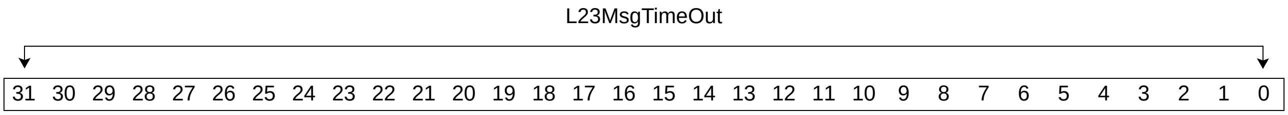
Extended Register Number	0b1111
Register Number	0b010001
Access Type	Protected
Address	0xF0BFFF44, 0xF8093D10



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on a read.	R	0x00000000
0	DirectAccessMode	Activates direct access mode for registers in configuration memory space. Writing 0b1 to this location allows type 0 configuration transactions to be performed by directly reading or writing configuration registers, as opposed to using the Configuration Address Register and the virtual Configuration Data Register to perform traditional type 0 configuration transactions. Writing a 0b0 to this location disables direct configuration transactions.	Protected	0b0

12.11.2.23 L23 Message Time-Out Register

Extended Register Number	0b1111
Register Number	0b010010
Access Type	Protected
Address	0xF0BFFF48, 0xF8093D20



Bits	Field Name	Description	Access	Reset
31:0	L23MsgTimeOut	Sets the maximum amount of time (in PCI Express clock cycles) from when the Power Manager Turn Off message is issued that the Outbound Transaction Layer Interface waits for the message to be ACK'd. A value of 0x00010000 corresponds to 2^{16} PCI Express clock cycles, 0x00100000 corresponds to 2^{20} PCI Express clock cycles, etc. If the allotted time expires before an ACK is received, a fatal error has occurred.	Protected	0xFFFFFFFF

12.11.2.24 Invalid Transaction Register

Extended Register Number	0b1111
Register Number	0b010011
Access Type	Read Only
Address	0xF0BFFF4C, 0xF8093D30

The Invalid Transaction Register is a 14-bit register that keeps track of the most recent write and read transactions that have been deemed invalid. A transaction is deemed invalid if, for any reason, it must be master aborted. It is important to note that not all invalid transactions are unexpected. While this register is technically read only, writing any value to this register causes it to be cleared. *Table 12-26* lists the invalid transfer types. Note that bit [0] will never be set, as there is no mechanism in the Outbound Posted Unit to flag this event.

Table 12-26. Invalid Transaction Types

Invalid Type	Read Bit	Write Bit
Unrecognized traditional configuration type.	[12]	[5]
Type 0 traditional configuration transaction has a nonzero Cycle bit.	[11]	[4]
Type 0 traditional configuration transaction has a nonzero Function Number.	[10]	[3]
Type 1 configuration transaction fails the outbound as type 0 or 1 checks.	[9]	[2]
I/O transaction without I/O transactions enabled.	[8]	[1]
Memory transaction without memory transactions enabled.	[7]	[0] (N/A)

Unused																		Read Valid Bit	Invalid Read Type						Write Valid Bit	Invalid Write Type					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
31:14	Unused	Returns 0 on a read.	R	0x00000
13	Read Valid Bit	Valid bit for the most recent invalid read transaction. A value of 0b1 indicates that an invalid read has occurred	R	0b0
12:7	Invalid Read Type	If the Read Valid Bit is high, a value of 0b1 in any of the bits indicates the type of invalid read. See *Table 12-26 Invalid Transaction Types*.	R	0b000000
6	Write Valid Bit	Valid bit for the most recent invalid write transaction. A value of 0b1 indicates that an invalid write has occurred.	R	0b0
5:0	Invalid Write Type	If the Write Valid bit is high, a value of 0b1 in any of the bits indicates the type of invalid write. (Bit 0 is never set. See *Table 12-26 Invalid Transaction Types*.)	R	0b000000

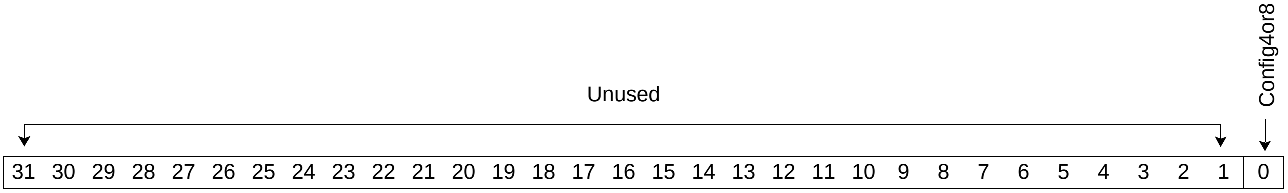
12.11.2.25 Configuration 4 Or 8 Register

Extended Register Number 0b1111

Register Number 0b010100

Access Type Protected

Address 0xF0BFFF50, 0xF8093D40

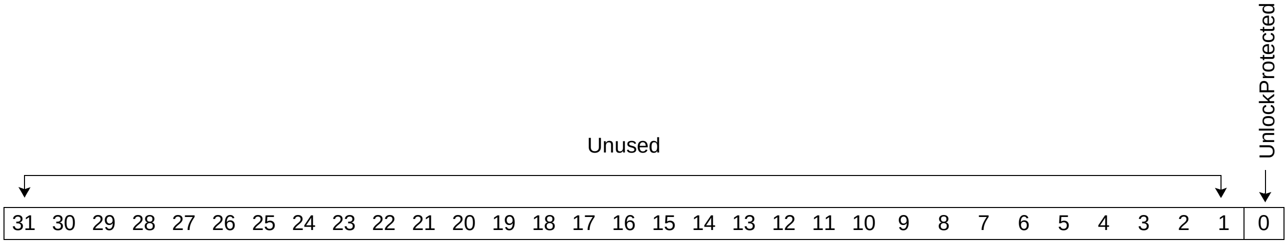


Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	Config4or8	Sets the number and size of transactions accepted across the PI interface. A value of 0b0 indicates that the maximum size of transactions is 64 bytes and that the maximum number allowed is 8, while a value of 0b1 indicates that the maximum size of transactions is 128-bytes and that the maximum number allowed is 4. In this implementation, the only valid value of this register is 0b0.	Protected	0b0

12.11.2.26 Unlock Protected Register

The Unlock Protected Register is a 1-bit register used to allow registers designated as protected to be written. Note that any attempt at writing a register, regardless of whether the register actually exists or is protected, relocks all protected registers by causing 0b0 to be written to the Unlock Protected Register. In other words, each write to a protected register must be preceded by a write of 0b1 to this register.

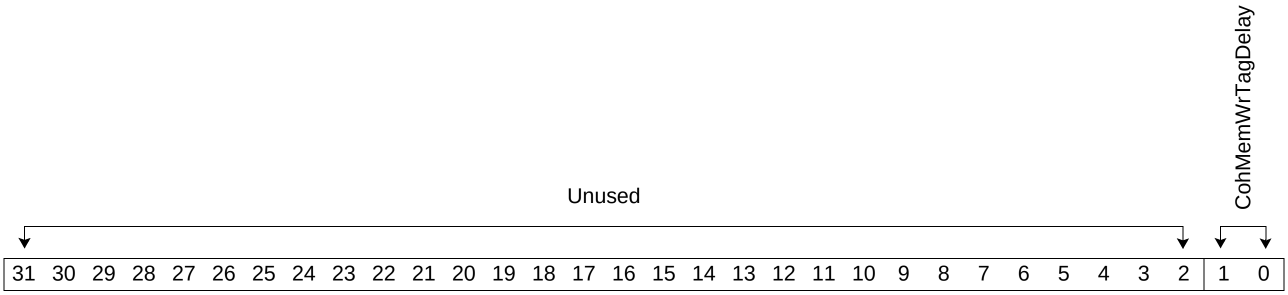
Extended Register Number	0b1111
Register Number	0b010101
Access Type	Read/Write
Address	0xF0BFFF54, 0xF8093D50



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	UnlockProtected	Allows registers designated as protected to be written. Writing 0b1 to this register “unlocks” all protected registers, allowing them to be written. Any subsequent write to any XBus Configuration Register causes this bit to revert to 0b0.	R/W	0b0

12.11.2.27 Coherent Memory Write Tag Delay Register (UNUSED)

Extended Register Number	0b1111
Register Number	0b010110
Access Type	Protected
Address	0xF0BFFF58, 0xF8093D60

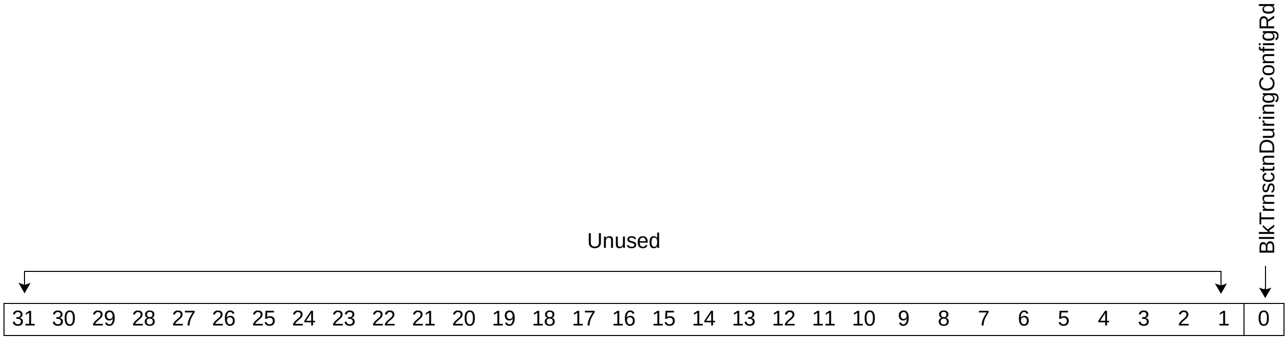


Bits	Field Name	Description	Access	Reset
31:2	Unused	Returns 0 on read.	R	0x00000000
1:0	CohMemWrTagDelay	Sets the number of PCI Express clock cycles the Inbound Write Data Buffer Unit must wait after receiving a request from the Inbound Request Unit before ACK'ing and supplying the write data buffer tag. This delay can range from 0 (0b00) to 3 (0b11) clock cycles.	Protected	0b0

12.11.2.28 Block Transactions During Configuration Reads Register

The Block Transactions During Configuration Reads Register is a 1-bit register used to set whether or not any additional transactions might be issued before a configuration read has completed. By definition, only one configuration write might be outstanding at a time. Since any outstanding configuration transaction must be buffered in case they need to be retried, limiting the number of outstanding configuration transactions reduces the required buffer space.

Extended Register Number	0b1111
Register Number	0b010111
Access Type	Protected
Address	0xF0BFFF5C, 0xF8093D70



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	BlkTrnsctnDuringConfigRd	Writing a 0b0 to this register allows additional transactions to be issued, while writing a 0b1 prevents additional transactions from being issued until the outstanding configuration read completes.	Protected	0b0

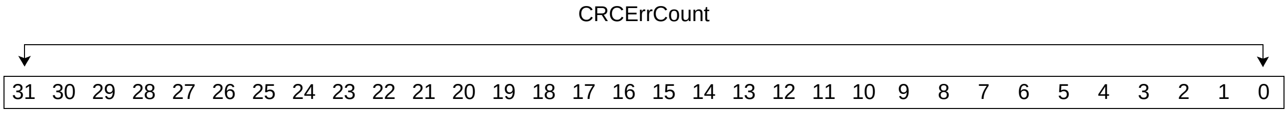
12.11.2.29 CRC Error Count Register

Extended Register Number 0b1111

Register Number 0b011000

Access Type Read Only

Address 0xF0BFFF60, 0xF8093D80



Bits	Field Name	Description	Access	Reset
31:0	CRCErrCount	Tracks the number of CRC errors that have occurred. Any write to this register, regardless of the value written, resets the count to 0.	R	0x00000000

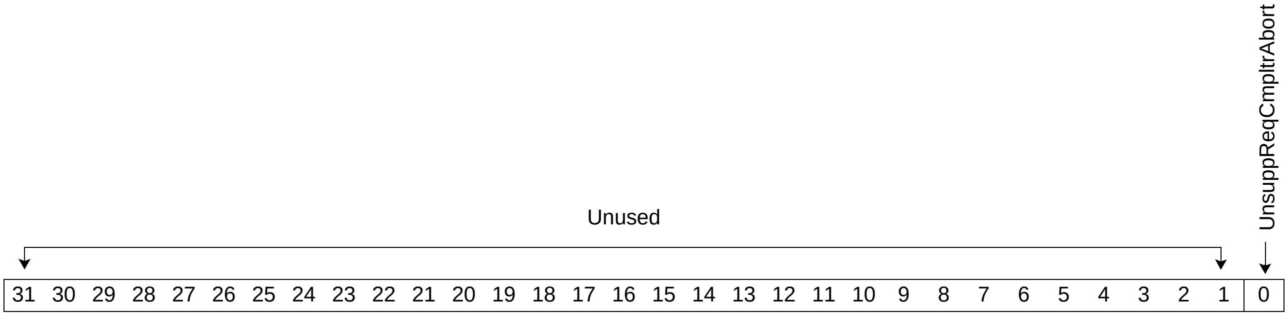
12.11.2.30 Unsupported Request Or Completer Abort Register

Extended Register Number 0b1111

Register Number 0b011001

Access Type Protected

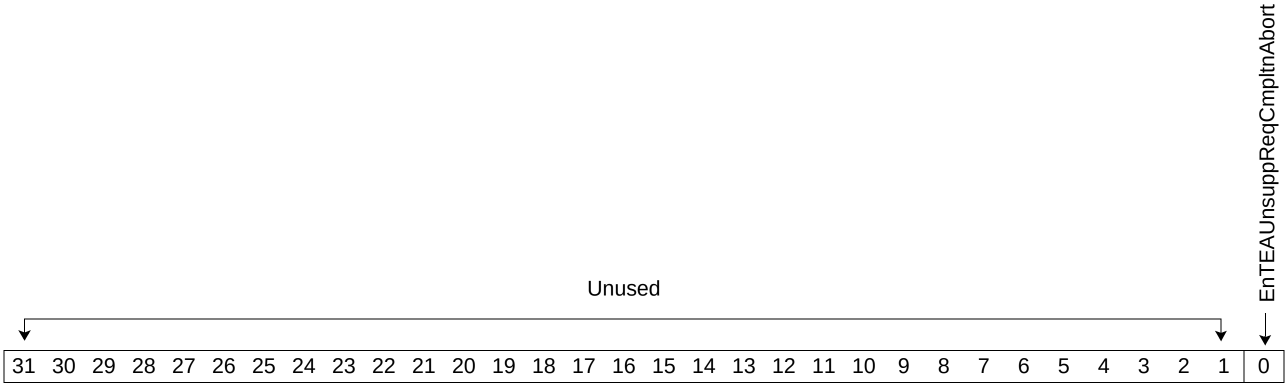
Address 0xF0BFFF64, 0xF8093D90



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	UnsuppReqCmpltrAbort	Sets the method of responding to an errant inbound TLP. The methods of response are either unsupported request (for a value of 0b0) or completer abort (for a value of 0b1).	Protected	0b0

12.11.2.31 Enable Transaction Error Acknowledge On Unsupported Request Completion Register

Extended Register Number	0b1111
Register Number	0b011010
Access Type	Protected
Address	0xF0BFFF68, 0xF8093DA0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	EnTEAUnsuppReq CmpltnAbort	Sets the way an Unsupported Request is completed on PI. A value of 0b0 results in the completion being indicated by the assertion of the Transaction Acknowledge signal, while a value of 0b1 results in the completion being indicated by the assertion of the Transaction Error Acknowledge signal.	Protected	0b0

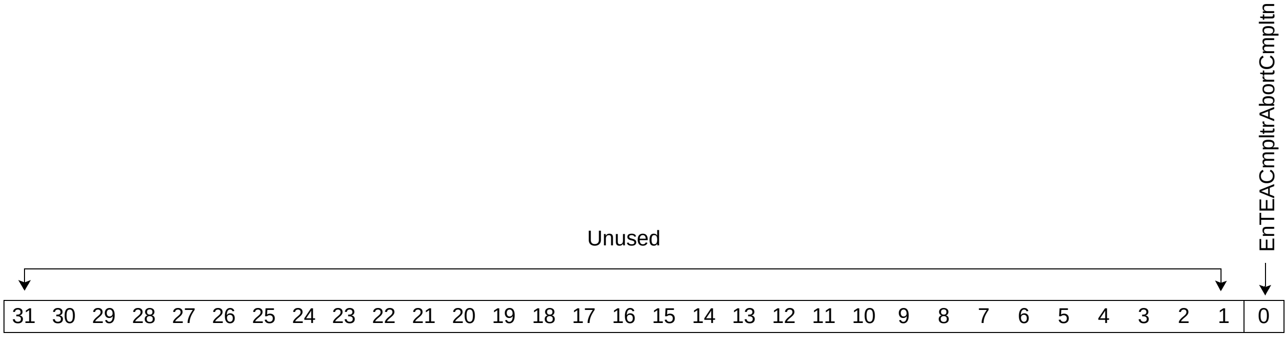
12.11.2.32 Enable Transaction Error Acknowledge On Completer Abort Completion Register

Extended Register Number 0b1111

Register Number 0b011011

Access Type Protected

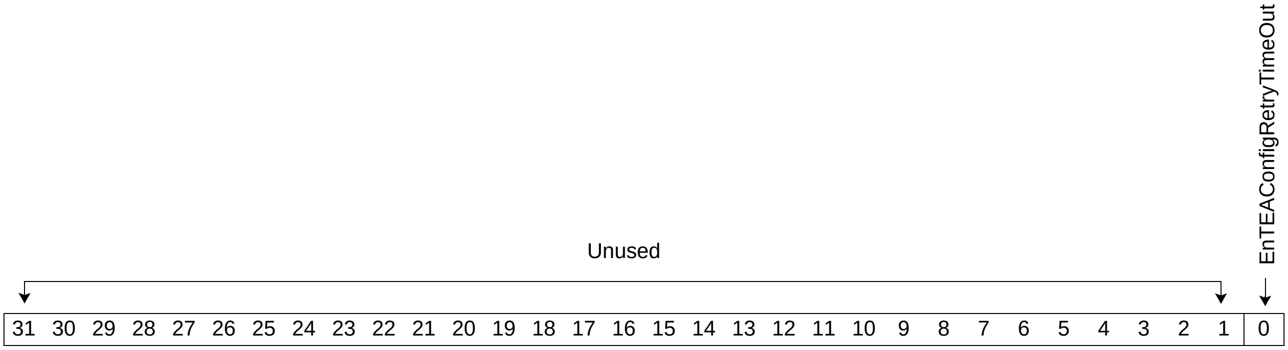
Address 0xF0BFFF6C, 0xF8093DB0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	EnTEACmpltrAbortCmpltn	Sets the way a completer abort is completed on PI. A value of 0b0 results in the completion being indicated by the assertion of the Transaction Acknowledge signal, while a value of 0b1 results in the completion being indicated by the assertion of the Transaction Error Acknowledge signal.	Protected	0b0

12.11.2.33 Enable Transaction Error Acknowledge On Configuration Retry Time-Out Register

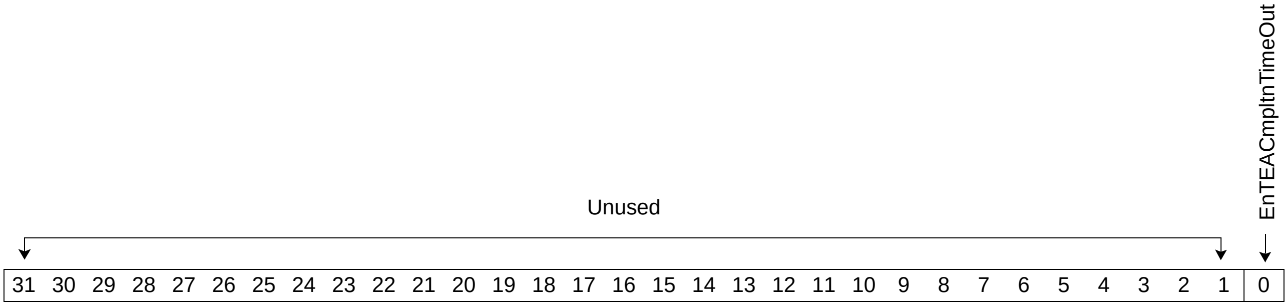
Extended Register Number	0b1111
Register Number	0b011100
Access Type	Protected
Address	0xF0BFFF70, 0xF8093DC0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	EnTEAConfigRetryTimeOut	Sets the way an configuration retry which has timed out is completed on PI. A value of 0b0 results in the completion being indicated by the assertion of the Transaction Acknowledge signal, while a value of 0b1 results in the completion being indicated by the assertion of the Transaction Error Acknowledge signal.	Protected	0b0

12.11.2.34 Enable Transaction Error Acknowledge On Completion Time-Out Register

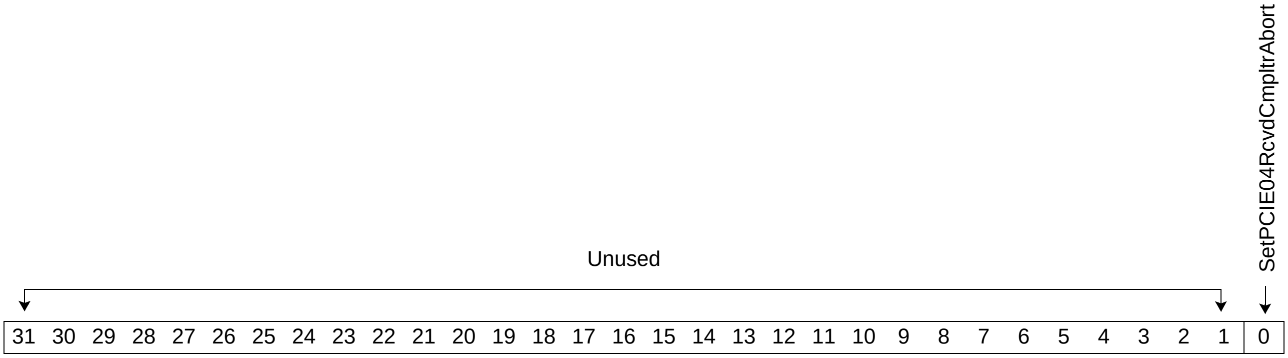
Extended Register Number	0b1111
Register Number	0b011101
Access Type	Protected
Address	0xF0BFFF74, 0xF8093DD0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	EnTEACmpltnTimeOut	Sets the way a completion that has timed out is completed on PI. A value of 0b0 results in the completion being indicated by the assertion of the Transaction Acknowledge signal, while a value of 0b1 results in the completion being indicated by the assertion of the Transaction Error Acknowledge signal.	Protected	0b0

12.11.2.35 Set PCIE04 Received Completer Abort On Completer Abort Register

Extended Register Number	0b1111
Register Number	0b011110
Access Type	Protected
Address	0xF0BFFF78, 0xF8093DE0



Bits	Field Name	Description	Access	Reset
31:1	Unused	Returns 0 on read.	R	0x00000000
0	SetPCIE04RcvdCmpltrAbort	Indicates whether to set the completer abort bit in the Primary Bus Status Registers after a completer abort. A value of 0b0 indicates that the completer abort bit is not set after a completer abort, while a value of 0b1 indicates that it is.	Protected	0b0

12.11.3 PCI Express GCR Registers

Table 12-27. PCI Express GCR Registers

Address	Register Name	Description	Page
0xF8080000	CORE_X	PI Core Interface Parameter Register	586
0xF8080010	BEACON	Beacon Support Register	587
0xF8080020	LOOPBACK	Loopback Control and Status Register	588
0xF8080030	SCRAMBLE	Data Scrambling Configuration Register	589
0xF8080040	SLOT	Slot Management Register	590
0xF8080050	POWER	Power Management Register	591
0xF8080060	VC_STAT	Virtual Channel Status Register	592
0xF8080070	AL_CFG	Application Layer Configuration Register	593
0xF8080080	IO_CFG	I/O Configuration Register	594
0xF8080090	RX_LANE	I/O Receive Configuration Register	594
0xF8080100	IO_LANE0	I/O Control and Status Register for Lane 0	595
0xF8080110	IO_LANE1	I/O Control and Status Register for Lane 1	595
0xF8080120	IO_LANE2	I/O Control and Status Register for Lane 2	595
0xF8080130	IO_LANE3	I/O Control and Status Register for Lane 3	595
0xF8080140	IO_LANE4	I/O Control and Status Register for Lane 4	595
0xF8080150	IO_LANE5	I/O Control and Status Register for Lane 5	595
0xF8080160	IO_LANE6	I/O Control and Status Register for Lane 6	595
0xF8080170	IO_LANE7	I/O Control and Status Register for Lane 7	595
0xF8080180	IO_LANE8	I/O Control and Status Register for Lane 8	595
0xF8080190	IO_LANE9	I/O Control and Status Register for Lane 9	595
0xF80801a0	IO_LANE10	I/O Control and Status Register for Lane 10	595
0xF80801b0	IO_LANE11	I/O Control and Status Register for Lane 11	595
0xF80801c0	IO_LANE12	I/O Control and Status Register for Lane 12	595
0xF80801d0	IO_LANE13	I/O Control and Status Register for Lane 13	595
0xF80801e0	IO_LANE14	I/O Control and Status Register for Lane 14	595
0xF80801f0	IO_LANE15	I/O Control and Status Register for Lane 15	595
0xF8081000	DIAG_IBCPL	Diagnostic Register for Application Layer Inbound Completion (IbCpl).	597
0xF8081010	DIAG_IBCMGR	Diagnostic Register for Application Layer Inbound Completion (IbCMgr).	598
0xF8081020	DIAG_IBRQ	Diagnostic Register for Application Layer Inbound Request (IbRq).	599
0xF8081030	DIAG_IBTLIF	Diagnostic Register for Application Layer Inbound Request (IbTLIf).	600
0xF8081040	DIAG_IBWD	Diagnostic Register for Application Layer Inbound Request (IbWD).	601

Table 12-27. PCI Express GCR Registers (Continued)

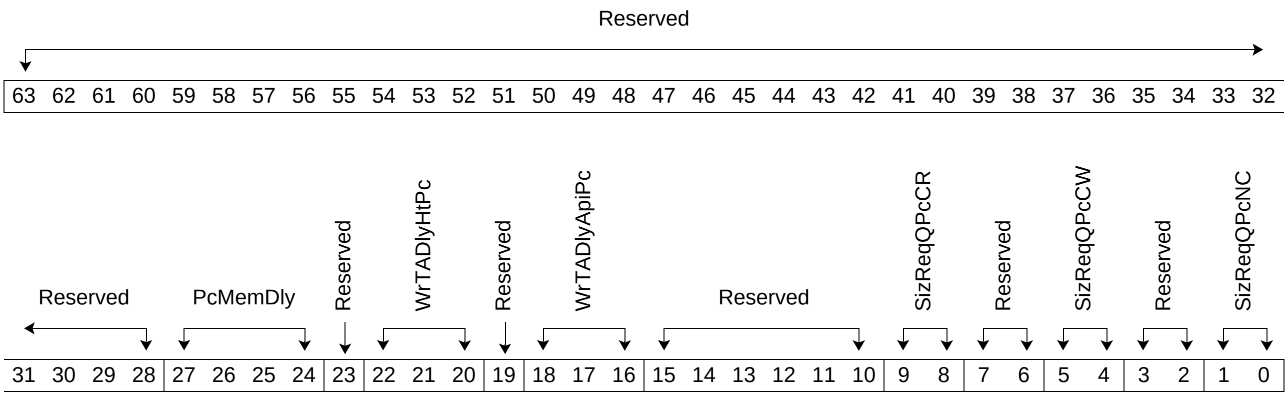
Address	Register Name	Description	Page
0xF8081080	DIAG_OBCPL	Diagnostic Register for Application Layer Outbound Completion (ObCpl).	603
0xF8081090	DIAG_OBMSG	Diagnostic Register for Application Layer Outbound Nonposted (ObMsg).	604
0xF80810a0	DIAG_OBNP	Diagnostic Register for Application Layer Outbound Nonposted (ObNP).	605
0xF80810b0	DIAG_OBP	Diagnostic Register for Application Layer Outbound Posted (ObP).	606
0xF80810c0	DIAG_OBTLIF	Diagnostic Register for Application Layer Outbound Posted (ObTLIf).	607
0xF8081100	MASK_MPIC_IBCPL	MPIC Masking for DIAG_IBCPL	608
0xF8081110	MASK_MPIC_IBCMGR	MPIC Masking for DIAG_IBCMGR	608
0xF8081120	MASK_MPIC_IBRQ	MPIC Masking for DIAG_IBRQ	609
0xF8081130	MASK_MPIC_IBTLIF	MPIC Masking for DIAG_IBTLIF	609
0xF8081140	MASK_MPIC_IBWD	MPIC Masking for DIAG_IBWD	610
0xF8081180	MASK_MPIC_OBCPL	MPIC Masking for DIAG_OBCPL	611
0xF80811a0	MASK_MPIC_OBNP	MPIC Masking for DIAG_OBNP	611
0xF80811b0	MASK_MPIC_OBP	MPIC Masking for DIAG_OBP	612
0xF80811c0	MASK_MPIC_OBTLIF	MPIC Masking for DIAG_OBTLIF	613
0xF8081800	DIAG_AUX0	Supplemental Diagnostic Register 0	597
0xF8081810	DIAG_AUX1	Supplemental Diagnostic Register 1	597

12.11.3.1 CORE_X: PI Core Interface Parameters Register

Reset Value N/A

Address x'F8080000'

Access Type Read, Read/Write



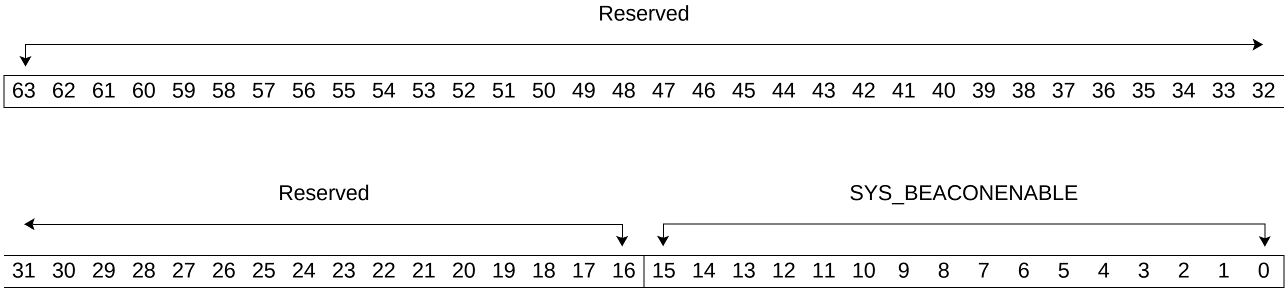
Bits	Field Name	Description	Access	Reset
63:28	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
27:24	PcMemDly	Delay between RdTgV and RdDt on PcDdrMRDI.	R/W	0x6
23	Reserved	Always returns 0 on read; write operations have no effect.	R	0b0
22:20	WrTADlyHtPc	Number of cycles for the WrTA to get Write Data on the HtPcWDI.	R/W	0b1
19	Reserved	Always returns 0 on read; write operations have no effect.	R	0b0
18:16	WrTADlyApiPc	Number of cycles for the WrTA to get Write Data on the ApiPcWDI.	R/W	0b1
15:10	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00
9:8	SizReqQPcCR	Size of request Queue at destination of Request Bus PcieCRRAI. The effective value of this register is calculated using the following table: Programmed Value Effective Value 2'b00 3'b100 2'b01 3'b001 2'b10 3'b010 2'b11 3'b011	R/W	0b0
7:6	Reserved	Always returns 0 on read; write operations have no effect.	R	0b00
5:4	SizReqQPcCW	Size of request Queue at destination of Request Bus PcieCWRAI. The effective value of this register is calculated similar to the above table.	R/W	0b00
3:2	Reserved	Always returns 0 on read; write operations have no effect.	R	0b00
1:0	SizReqQPcNC	Size of request Queue at destination of Request Bus PcieNCRAI. The effective value of this register is calculated similar to the above table.	R/W	0b00

12.11.3.2 BEACON: Beacon Support Register

Reset Value x’0000 0000 0000 0000’

Address x’F8080010’

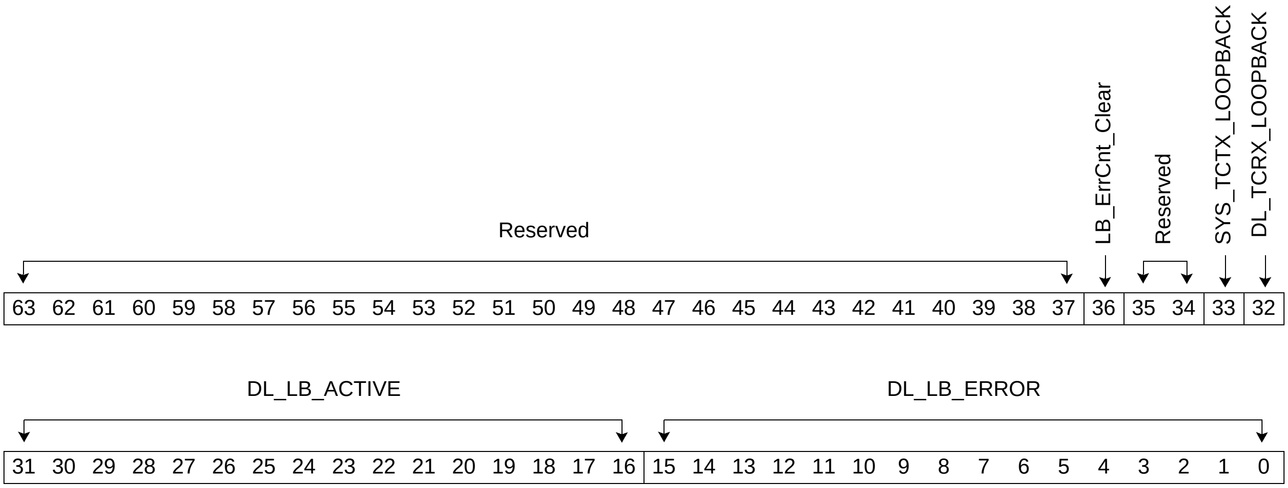
Access Type Read



Bits	Field Name	Description	Access	Reset
63:16	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
15:0	SYS_BEACONENABLE	Reserved for future implementation of beacon support.	R	0x0000

12.11.3.3 LOOPBACK: Loopback Control and Status Register

Reset Value	N/A
Address	x’F8080020’
Access Type	Read, Read/Write



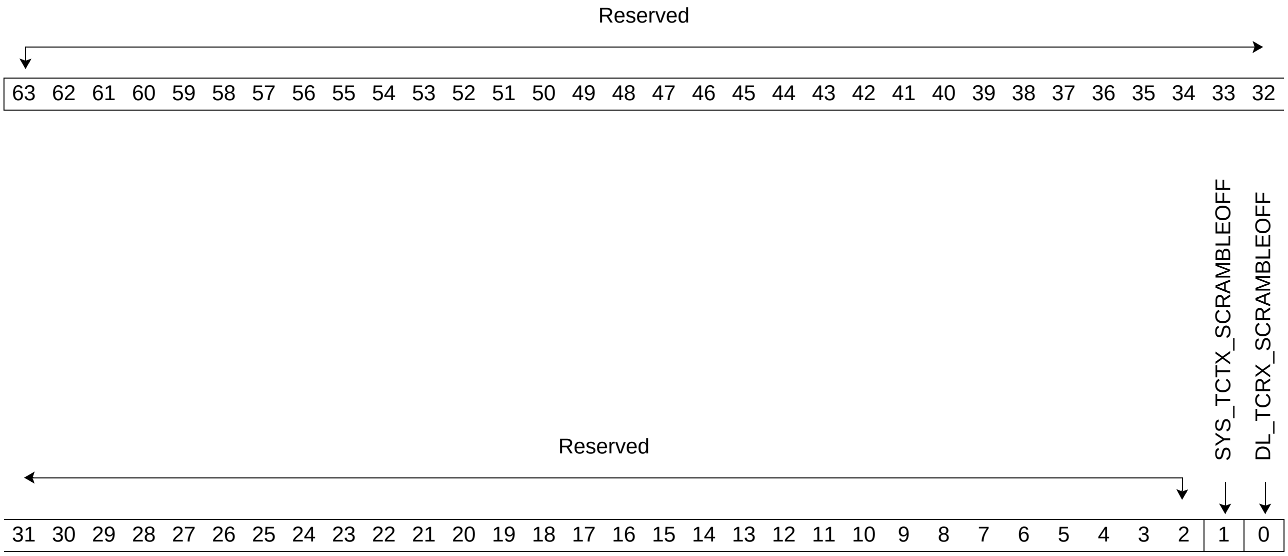
Bits	Field Name	Description	Access	Reset
63:37	Reserved	Always returns 0 on read; write operations have no effect.	R	x0000000
36	LB_ErrCnt_Clear	Always returns 0 on read. Writing a ‘1’ clears all LB error counters.	R/W	0b0
35:34	Reserved	Always returns 0 on read; write operations have no effect.	R	0b00
33	SYS_TCTX_LOOPBACK	Enter loopback mode.	R/W	0b0
32	DL_TCRX_LOOPBACK	Indicates transmit side of DLP is in loopback mode.	R	0b0
31:16	DL_LB_ACTIVE	Indicates lanes on which master mode loopback is active.	R	x0000
15:0	DL_LB_ERROR	Indicates error detected on respective lane.	R	x0000

12.11.3.4 SCRAMBLE: Data Scrambling Configuration Register

Reset Value x’0000 0000 0000 0000’

Address x’F8080030’

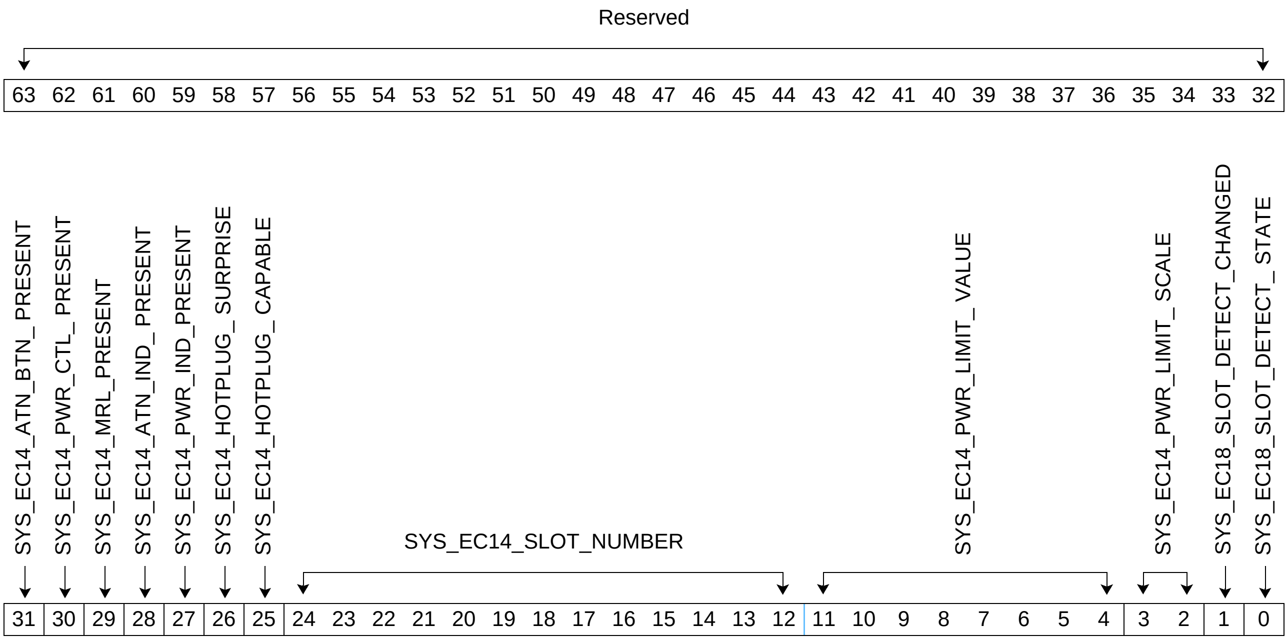
Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:2	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
1	SYS_TCTX_ SCRAMBLEOFF	Turns off data scrambling (for test purposes only).	R/W	0b0
0	DL_TCRX_ SCRAMBLEOFF	Indicates that scrambling is turned off.	R	0b0

12.11.3.5 SLOT: Slot Management Register

Reset Value	N/A
Address	x’F8080040’
Access Type	Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:32	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
31	SYS_EC14_ATN_ BTN_ PRESENT	Attention Button Present to the Slot Capabilities Register.	R/W	0b0
30	SYS_EC14_PWR_CTL_ PRESENT	Power Controller Present to the Slot Capabilities Register.	R/W	0b0
29	SYS_EC14_MRL_ PRESENT	Mechanical Release Latch Present to the Slot Capabilities Register.	R/W	0b0
28	SYS_EC14_ATN_ IND_ PRESENT	Attention Indicator Present to the Slot Capabilities Register.	R/W	0b0
27	SYS_EC14_PWR_ IND_PRESENT	Power Indicator Present to the Slot Capabilities Register.	R/W	0b0
26	SYS_EC14_HOTPLUG_ SURPRISE	Hot-Plug Surprise to the Slot Capabilities Register.	R/W	0b0
25	SYS_EC14_HOTPLUG_ CAPABLE	Hot-Plug Capable to the Slot Capabilities Register.	R/W	0b0
24:12	SYS_EC14_SLOT_ NUMBER	Physical Slot Number to the Slot Capabilities Register.	R/W	0x0000
11:4	SYS_EC14_PWR_LIMIT_ VALUE	Slot Power Limit Value to the Slot Capabilities.	R/W	0x00

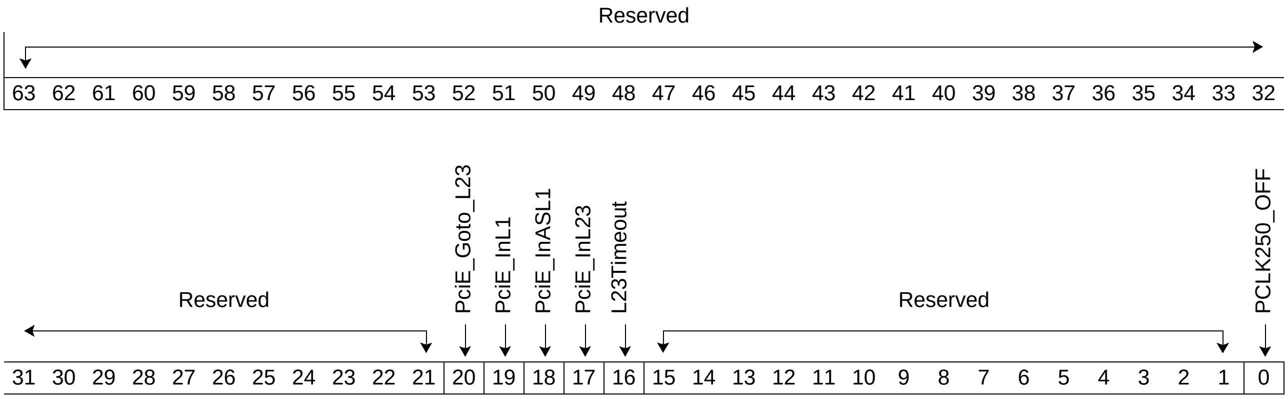
Bits	Field Name	Description	Access	Reset
3:2	SYS_EC14_PWR_LIMIT_ SCALE	Slot power limit scale to the slot capabilities.	R/W	0b00
1	SYS_EC18_SLOT_DETECT _ CHANGED	Indicates state change on $\overline{\text{PCIE\_PRESENTN}}$. The change must be stable for at least 2^{19} PCLK250 cycles. This register reflects a sticky version of the SYS_EC18_SLOT_DETECT_CHANGED single cycle signal and is cleared when writing a ‘1’ to this bit.	R/W	
0	SYS_EC18_SLOT_DETECT _ STATE	Indicates the current state of $\overline{\text{PCIE\_PRESENTN}}$. The state changes only when PCIE_PRESENTN has been stable for 2^{19} PCLK250 cycles.	R	

12.11.3.6 POWER: Power Management Register

Reset Value N/A

Address x’F8080050’

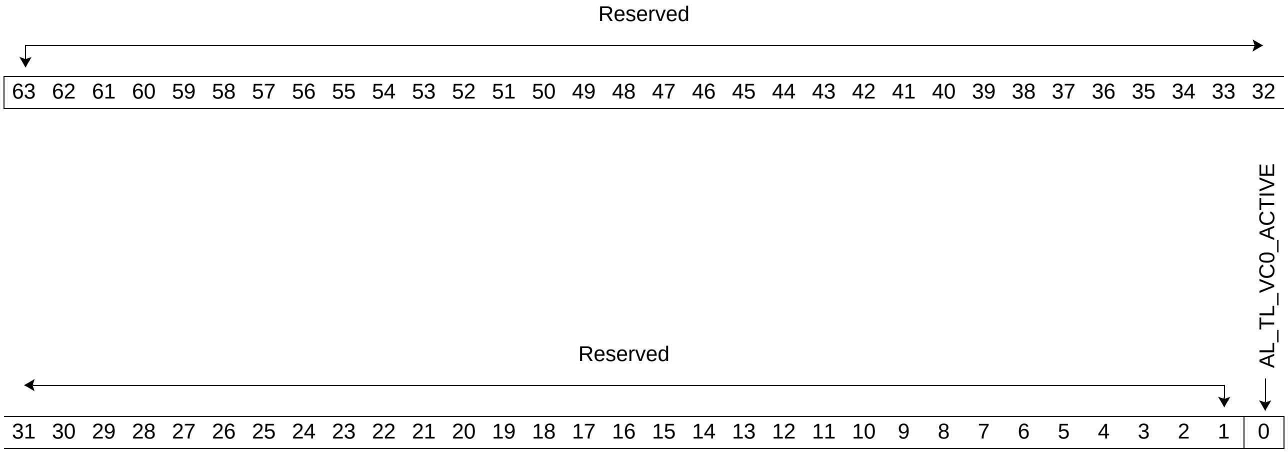
Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:21	Reserved	Always returns 0 on read; write operations have no effect.	R	0x000000
20	PciE_Goto_L23	Indicates that system software has initiated a shutdown of the link and desires it to go into the L2/L3 ready state.	R/W	0b0
19	PciE_InL1	Indicates that the link is in the L1 state	R	
18	PciE_InASL1	Indicates that the link is in the ASPM L1 state	R	
17	PciE_InL23	Indicates that the link is in the L2/L3 ready state	R	
16	L23Timeout	Indicates that the link is forced in the L2/L3 ready state due to PME_TO_Ack timeout counter expired.	R	
15:1	Reserved	Always returns 0 on read; write operations have no effect.	R	0x0000
0	PCLK250_OFF	Indicates that the 250 MHz PCIe stack clock is not currently running (that is, SERDES PLL’s are not locked).	R	

12.11.3.7 VC_STAT: Virtual Channel Status Register

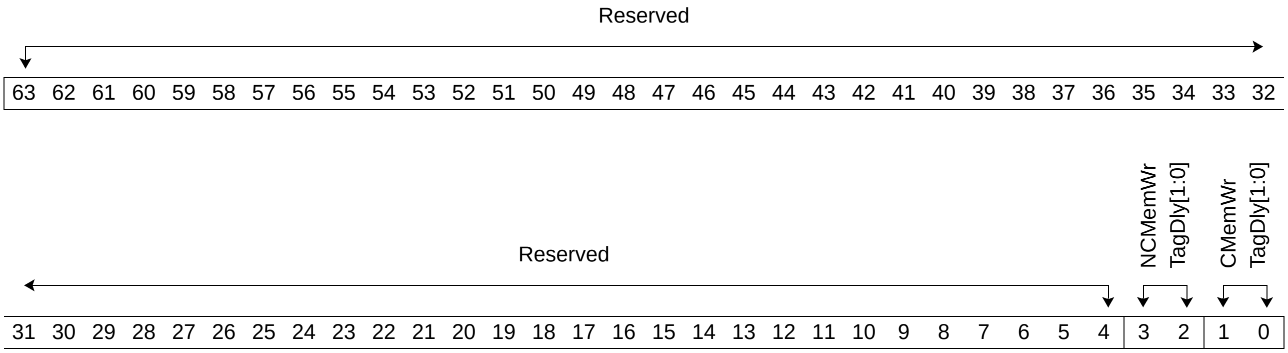
Reset Value	N/A
Address	x'F8080060'
Access Type	Read



Bits	Field Name	Description	Access	Reset
63:1	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
0	AL_TL_VC0_ACTIVE	Indicates that the PCIe stack is up and running and ready to begin transmission of packets.	R	

12.11.3.8 AL_CFG: Application Layer Configuration Register

Reset Value	N/A
Address	x’F8080070’
Access Type	Read, Read/Write



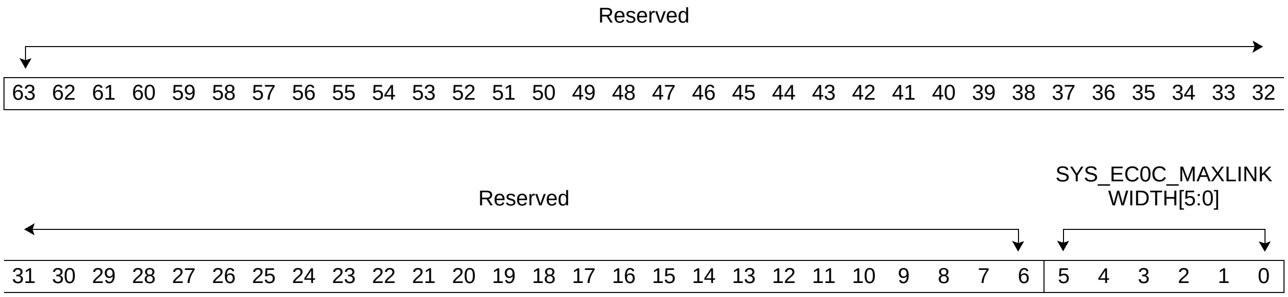
Bits	Field Name	Description	Access	Reset
63:4	Reserved	Always returns 0 on read; write operations have no effect.	R	0x0000000
3:2	NCMemWrTagDly[1:0]	Amount of delay (measured in PCLK250 cycles) to introduce before releasing credits back to the PCI Express master when IbWD responds to IbRq with a noncoherent Memory Write request.	R/W	0b00
1:0	CMemWrTagDly[1:0]	Amount of delay (measured in PCLK250 cycles) to introduce after IbWD has accepted a IbRq_CMemWrRq and begun executing a data push to the memory controller and before the signal IbWD_CMemWrAck is asserted.	R/W	0b00

12.11.3.9 IO_CFG: I/O Configuration Register

Reset Value N/A

Address x’F8080080’

Access Type Read, Read/Write



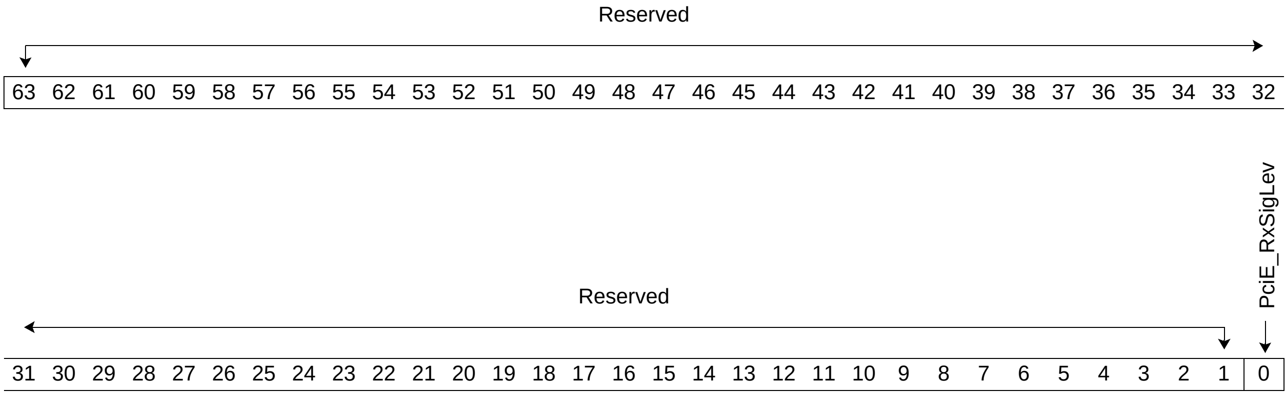
Bits	Field Name	Description	Access	Reset
63:6	Reserved	Always returns 0 on read; write operations have no effect.	R	0x0000000
5:0	SYS_EC0C_MAXLINKWIDTH[5:0]	This register controls the maximum PCIe link width the PCIe stack might negotiate at link initialization time.	R/W	0x10

12.11.3.10 RX_LANE: I/O Receive Configuration Register

Reset Value N/A

Address x’F8080090’

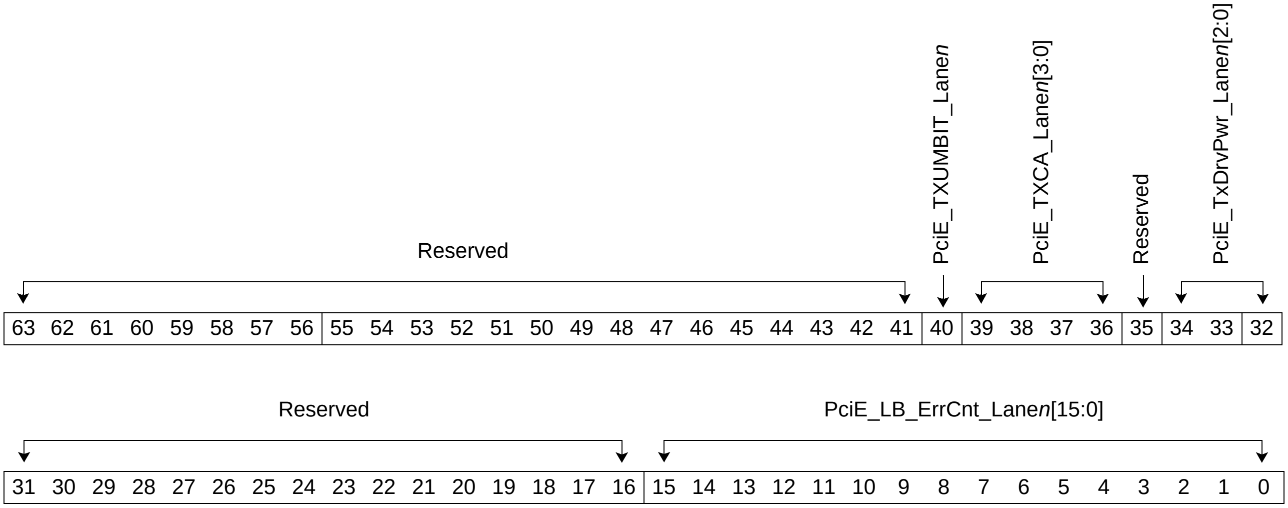
Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:1	Reserved	Always returns 0 on read; write operations have no effect.	R	0x0000000
0	PciE_RxSigLev	Adjusts receiver sensitivity in the SERDES core.	R/W	0b0

12.11.3.11 I/O_LANEn: I/O Status and Control Register for Lane n (n ranges from 0..15)

Reset Value	N/A
Address	x'F80801h0' (Where h is the hexadecimal representation of n)
Access Type	Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:41	Reserved	Always returns 0 on read; write operations have no effect.	R	0x000000
40	PciE_TXUMBIT_Lane*n*	Controls amount of pre-emphasis	R/W	0b0
39:36	PciE_TXCA_Lane*n*[3:0]	Controls SERDES TX FIR coefficients	R/W	0x1
35	Reserved	Always returns 0 on read; write operations have no effect.	R	0b0
34:32	PciE_TxDrvPwr_Lane*n*[2:0]	Controls SERDES TX drive strength.	R/W	0b000
31:16	Reserved	Always returns 0 on read; write operations have no effect.	R	0x0000
15:0	PciE_LB_ErrCnt_Lane*n*[15:0]	Indicates number of loopback errors detected on lane n.	R	

Programmable Driver Power Levels (SERDES TX Drive Strength [34:32])

Control inputs are provided to allow adjustment of the transmit link driver output power. When driving an ideal 100 Ω terminated network, these output power settings effectively establish the differential voltage swings at the driver output. The input signals PciE_TxDrvPwr_Lanen [2:0] (bits 34:32) are encoded inputs that allow the selection of eight discrete power settings. These are adjustable on a per-link basis. *Table 12-29* lists the normalized driver power (NDP) setting of the transmit drivers as a function of the driver power control inputs. The normalized current setting is 2 mA, which corresponds to the normalized power setting of 1.0. Values listed in the normalized driver power setting column are multiples of 2 mA. For example, with inputs at ‘110’, the driver power is 12x2 mA = 24 mA.

Table 12-28. Driver Power Levels Control Encoding for Transmit Function

TXDRVPWR2x	TXDRVPWR1x	TXDRVPWR0x	Normalized Driver Power Setting	Normalized Driver Power Current
0	0	0	5	10 mA
0	0	1	6	12 mA
0	1	0	7	14 mA
0	1	1	8	16 mA
1	0	0	10	20 mA
1	0	1	11	22 mA
1	1	0	12	24 mA
1	1	1	13	26 mA

FIR Pre-emphasis Coefficients

The transmitter employs a sophisticated pre-emphasis technique to reduce inter-symbol interference (ISI) at the receiver. This pre-emphasis technique uses a finite impulse response (FIR) filter mechanism to compensate for the high frequency roll-off of the transmission channel.

$$H(z) = 2\text{ mA} \times NDP(C_0 z^{0} - C_1 z^{-1})$$

This equation computes the time dependent variation in the driver differential current, where NDP represents normalized driver power. Coefficient C0 is automatically controlled, and coefficient C1 is alterable using PciE_TXCA_Lane*n*[3:0]. These control inputs activate the C1 sub-coefficients (See*Table 12-29*). One of two equations apply to the driver, dependent on the NDP setting.

For NDP < 10

$$Hx(z) = 2mA \times NDPx \times (C0xz^{-0} - [1/NDP \times (A + CA0x + CA1x + CA2x + CA3x)]z^{-1})$$

For NDP > or = 10

$$Hx(z) = 2mA \times NDPx \times (C0xz^{-0} - [1/12 \times (A + CA0x + CA1x + CA2x + CA3x+ 1)]z^{-1})$$

Note: The value of C_0 is reduced by C_1 ($C_0 = 1 - C_1$) to maintain a constant output level.

Table 12-29. FIR Coefficient Table

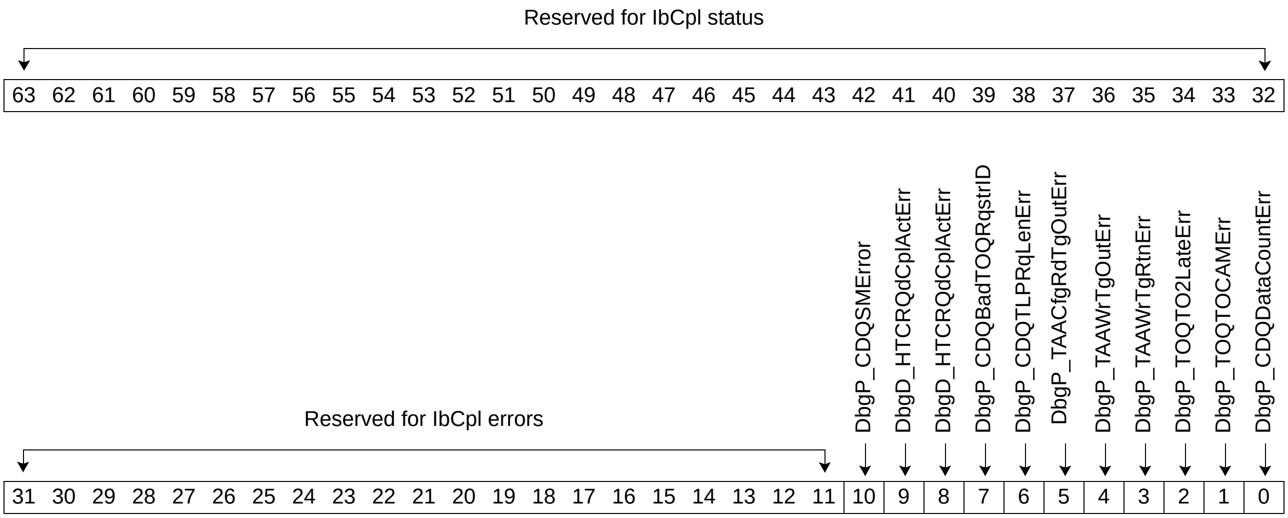
C_O	A	CA3x	CA2x	CA1x	CA0x
$1.000 - C_1$	1/15	1	8/15	4/15	2/15

As an example, a normalized power setting of 10 with TXCA[3:0]=”1111” would yield the transfer function:

$$H(z) = 2\text{ mA} \times 10 \times (0.75z^{-0} - 0.25z^{-1})$$

12.11.3.12 DIAG_IBCPL: Diagnostic Register for Application Layer Inbound Completion (IbCpl)

Reset Value	N/A
Address	x’F8081000’
Access Type	Read



Bits	Field Name	Description	Access	Reset
63:32	Reserved for IbCpl status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:11	Reserved for IbCpl errors	Always returns 0 on read; write operations have no effect.	R	0x000000
10	DbgP_CDQSMError	Indicates that the internal IbCpl_CDQ state machine tracking completions from the ALIbTLIf interface has detected an unexpected state transition.	R	0b0
9	DbgD_APICRQdCplActErr	Indicates that the iMultiCycleCplActive flag internal to the PI instantiation of the IbCpl_CRQddr module is attempting to be set and reset on same clock cycle.	R	0b0
8	DbgD_HTCRQdCplActErr	Indicates that the iMultiCycleCplActive flag internal to the HT instantiation of the IbCpl_CRQddr module is attempting to be set and reset on same clock cycle.	R	0b0
7	DbgP_CDQBadTOQRqstrID	Indicates that an illegal ObNP_RqstrID (2'b11) has been stored in the IbCpl_TOQ module.	R	D0b
6	DbgP_CDQTLPRqLenErr	Indicates an error between the number of dwords indicated in the Completion TLP Header and the number previously stored in the IbCpl_TOQ module.	R	0b0
5	DbgP_TAACfgRdTgOutErr	Indicates an error in the setting or clearing of the configuration read flag.	R	0b0
4	DbgP_TAAWrTgOutErr	Indicates an error in the setting or clearing of the nonposted write flag.	R	0b0
3	DbgP_TAAWrTgRtnErr	Indicates that an outstanding nonposted write Tag has been returned that does not match the value previously stored.	R	0b0

Bits	Field Name	Description	Access	Reset
2	DbgP_TOQTO2LateErr	Indicates that a completion was received that had a valid tag, but that had previously timed out (this would indicate a timeout value that was too small or possibly an endpoint that got hung up and reset itself prior to completing the outbound nonposted transaction.	R	0b0
1	DbgP_TOQTOCAMErr	Indicates that there has been more than one transaction timeout error occurring on the same clock.		0b0
0	DbgP_CDQDataCountErr	Indicates an error between the number of data beats returned in the completion (from the ALIbTLIf interface) and the number previously stored in the IbCpl_TOQ module.	R	0b0

12.11.3.13 DIAG_IBCMGR: Diagnostic Register for Application Layer Inbound Completion Manager (IbCMgr)

Reset Value N/A

Address x’F8081010’

Access Type Read

Reserved

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

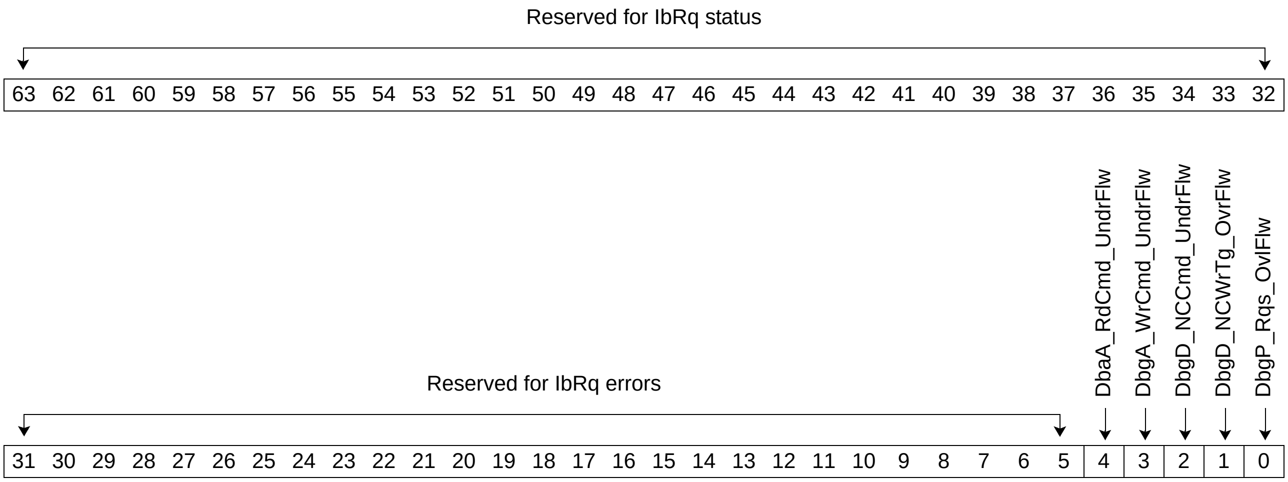
Reserved

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
63:0	Reserved	Reserved for future requirements. Always returns 0 on read; write operations have no effect.	R	0x0000000

12.11.3.14 DIAG_IBRQ: Diagnostic Register for Application Layer Inbound Request (IbRq)

Reset Value	N/A
Address	x’F8081020’
Access Type	Read



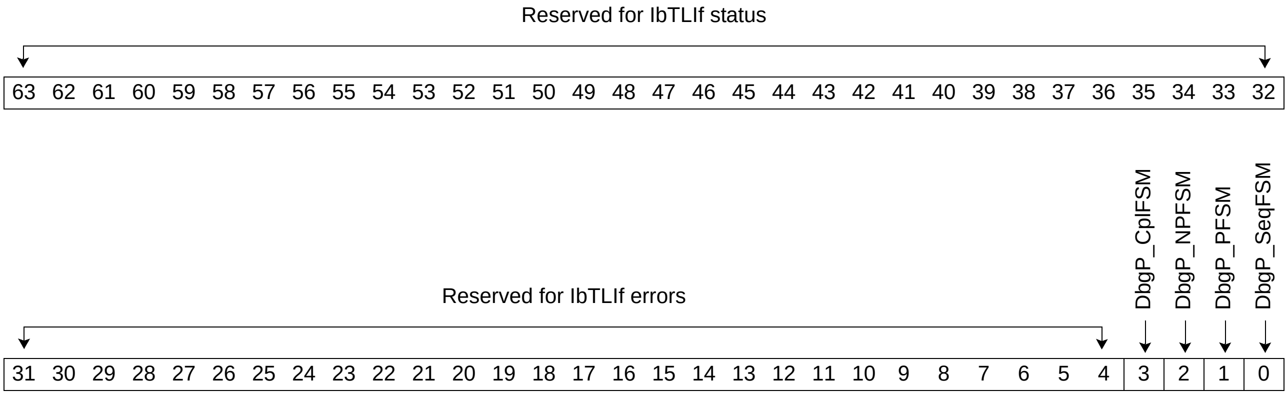
Bits	Field Name	Description	Access	Reset
63:32	Reserved for IbRq status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:5	Reserved for IbRq errors	Always returns 0 on read; write operations have no effect.	R	0x0000000
4	DbgA_RdCmd_UndrFlw	When one, this signal indicates that the target interface has returned more commands credits than the unit holds. This is an underflow condition and is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
3	DbgA_WrCmd_UndrFlw	When one, this signal indicates that the target interface has returned more commands credits than the unit holds. This is an underflow condition and is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
2	DbgD_NCCmd_UndrFlw	When one, this signal indicates the target interface has returned more commands credits than the unit holds. This is an underflow condition and is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
1	DbgD_NCWrTg_OvrFlw	When one, this signal indicates the write tag FIFO has been instructed to accept more tags than it has space. This is an overflow condition and is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
0	DbgP_Rqs_OvlFlw	When one, this signals the detection of an incoming header when no header buffer space is available. This condition is a fatal error. Once set, this bit can only be cleared with a reset.	R	0b0

12.11.3.15 DIAG_IBTLIF: Diagnostic Register for Application Layer Inbound TL Interface (IbTLIf)

Reset Value N/A

Address x’F8081030’

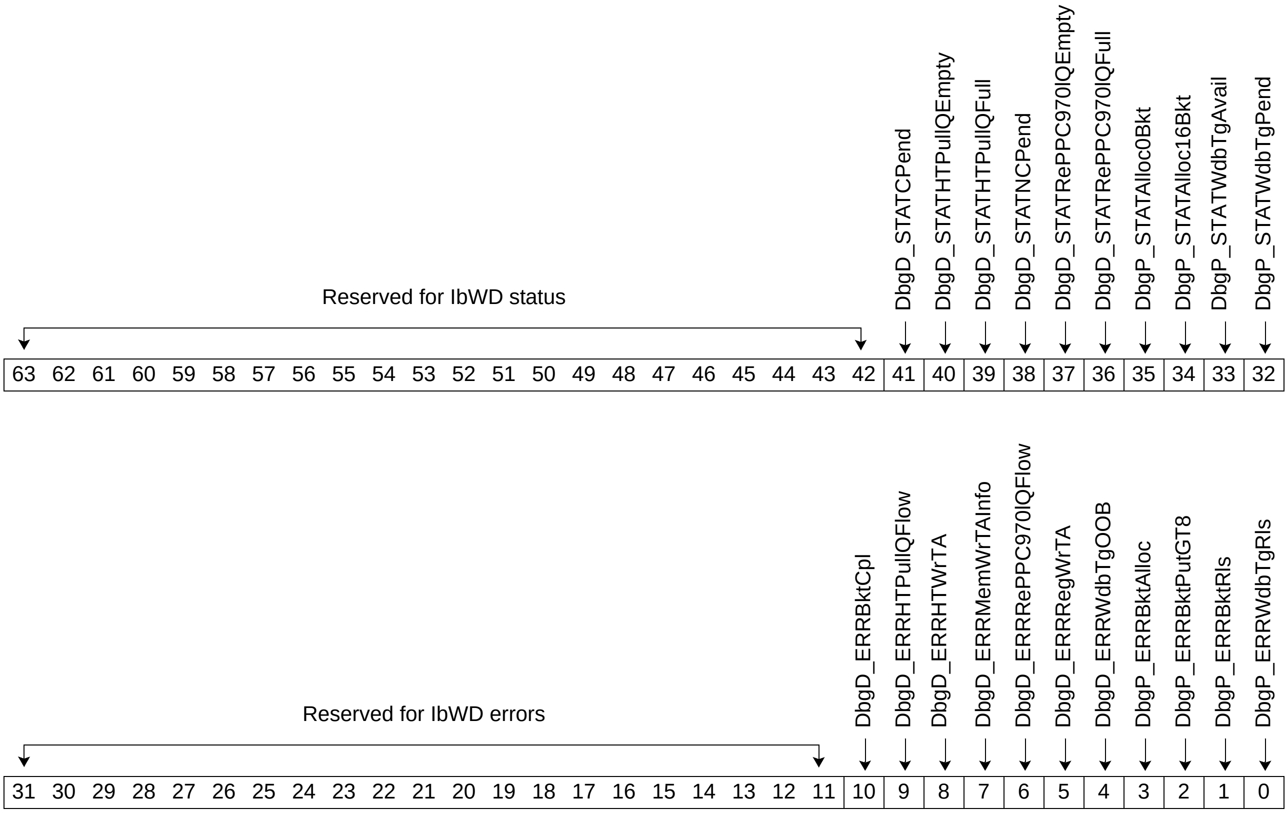
Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:32	Reserved for IbTLIf status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:4	Reserved for IbTLIf errors	Always returns 0 on read; write operations have no effect.	R	0x00000000
3	DbgP_CplFSM	Assertion of this signal indicates that the completion FSM has entered an undefined state.	R	0b0
2	DbgP_NPFSM	Assertion of this signal indicates that the nonposted FSM has entered an undefined state.	R	0b0
1	DbgP_PFSM	Assertion of this signal indicates that the posted FSM has entered an undefined state.	R	0b0
0	DbgP_SeqFSM	Assertion of this signal indicates that the sequence FSM has entered an undefined state.	R	0b0

12.11.3.16 DIAG_IBWD: Diagnostic Register for Application Layer Inbound Request (IbWD)

Reset Value	N/A
Address	x’F8081040’
Access Type	Read

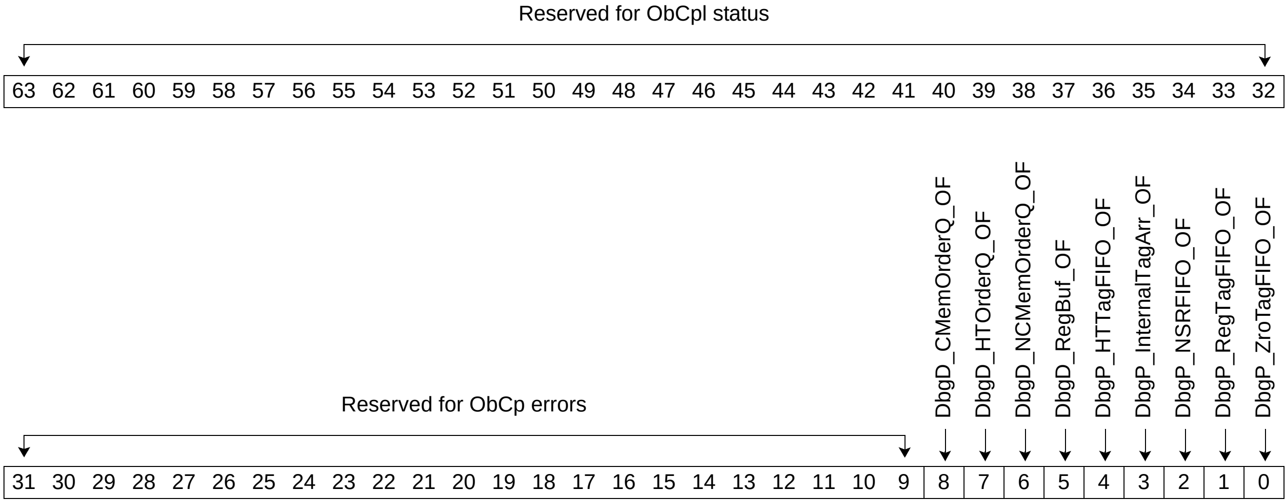


Bits	Field Name	Description	Access	Reset
63:42	Reserved for IbWD status	Always returns 0 on read; write operations have no effect.	R	0x000000
41	DbgD_STATCPend	When ‘1’, this signal indicates that a coherent write request to memory is pending in the push manager.	R	0b0
40	DbgD_STATHTPullQEmpty	When ‘1’, this signal indicates that the pull FIFO used to hold WrTA Information for HyperTransport writes is empty.	R	0b1
39	DbgD_STATHTPullQFull	When ‘1’, this signal indicates that the pull FIFO used to hold WrTA Information for HyperTransport writes is full.	R	0b0
38	DbgD_STATNCPend	When ‘1’, this signal indicates that a noncoherent write request to memory is pending in the push manager.	R	0b0
37	DbgD_STATRePPC970IQ Empty	When ‘1’, this signal indicates that the pull FIFO used to hold WrTA Information for register writes is empty.	R	0b1
36	DbgD_STATRePPC970IQ Full	When ‘1’, this signal indicates that the pull FIFO used to hold WrTA Information for register writes is full.	R	0b0

Bits	Field Name	Description	Access	Reset
35	DbgP_STATAlloc0Bkt	When ‘1’, this signal indicates that 0 outstanding buckets have been allocated in the write data buffer.	R	0b1
34	DbgP_STATAlloc16Bkt	When ‘1’, this signal indicates that all 16 buckets have been allocated in the write data buffer.	R	0b0
33	DbgP_STATWdbTgAvail	When ‘1’, this signal indicates that memory controller WDB tags are available for allocation.	R	0b1
32	DbgP_STATWdbTgPend	When ‘1’, this signal indicates that IbWD is waiting for outstanding WDB tags over the PI bus to be released by the memory controller.	R	0b0
31:11	Reserved for IbWD errors	Always returns 0 on read; write operations have no effect.	R	0x000000
10	DbgD_ERRBktCpl	When ‘1’, this signal indicates that multiple agents are simultaneously attempting to retire same bucket. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
9	DbgD_ERRHTPullQFlow	When ‘1’, this signal indicates that the HT Pull FIFO has overflowed. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
8	DbgD_ERRHTWrTA	When ‘1’, this signal indicates that an unexpected PcieHtWrTA assertion was received. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
7	DbgD_ERRMemWrTAInfo	When ‘1’, this signal indicates that there was a parity mismatch on the Write TA Info being passed from IbRq to the Push Arbiter. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
6	DbgD_ERRRePPC970lQFlow	When ‘1’, this signal indicates that the HT Pull FIFO has overflowed. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
5	DbgD_ERRRegWrTA	When ‘1’, this signal indicates that an unexpected PcieHtWrTA assertion was received. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
4	DbgD_ERRWdbTgOOB	When ‘1’, this signal indicates that an out-of-bounds tag was released by the memory controller to PCI Express. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
3	DbgP_ERRBktAlloc	When ‘1’, this signal indicates that a bucket was taken by IbRq when no buckets were available for allocation. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
2	DbgP_ERRBktPutGT8	When ‘1’, this signal indicates that the Inbound TL Interface has attempted to put more than 8 quad-DWords of data into the write data buffer and a bucket was taken. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
1	DbgP_ERRBktRls	When ‘1’, this signal indicates that the pull or push managers returned a nonallocated bucket. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0
0	DbgP_ERRWdbTgRls	When ‘1’, this signal indicates that the memory controller has attempted to release an unallocated WDB tag. This is a fatal error. Once set, the signal is only released upon a reset.	R	0b0

12.11.3.17 DIAG_OBCPL: Diagnostic Register for Application Layer Outbound Completion (ObCpl)

Reset Value	N/A
Address	x’F8081080’
Access Type	Read



Bits	Field Name	Description	Access	Reset
63:32	Reserved for ObCpl status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:9	Reserved for ObCpl errors	Always returns 0 on read; write operations have no effect.	R	0x000000
8	DbgD_CMemOrderQ_OF	When asserted, this signal indicates that more Cmem completion data have been received than the CMemOrderQ can hold (16 entries). Fatal Error. Level, reset by ObCpl_ddrRst.	R	0b0
7	DbgD_HTOrderQ_OF	When asserted, this signal indicates that more HT completion data have been received than the HTOrderQ can hold (16 entries). Fatal Error. Level, reset by ObCpl_ddrRst.	R	0b0
6	DbgD_NCMemOrderQ_OF	When asserted, this signal indicates that more NCmem completion data have been received than the NCMemOrderQ can hold (16 entries). Fatal Error. Level, reset by ObCpl_ddrRst.	R	0b0
5	DbgD_RegBuf_OF	When asserted, this signal indicates that more Register Read Completion data have been received than the RegBuf can hold (16 entries). Fatal Error. Level, reset by ObCpl_ddrRst.	R	0b0
4	DbgP_HTTagFIFO_OF	When asserted, this signal indicates that more HTTag entries have been received than the HTTagFIFO can hold (16 entries). Fatal Error. Level, reset by ObCpl_PciERst.	R	0b0
3	DbgP_InternalTagArr_OF	When asserted, this signal indicates that more Tags have been returned than the TagArr can hold (16 entries). Fatal Error. Level, reset byObCpl_PciERst.	R	0b0
2	DbgP_NSRFIFO_OF	When asserted, this signal indicates that more NSR headers have been received than the NSRFIFO can hold (16 entries). Fatal Error. Level, reset by ObCpl_PciERst.	R	0b0
1	DbgP_RegTagFIFO_OF	When asserted, this signal indicates that more RegTag entries have been received than the RegTagFIFO can hold (16 entries). Fatal Error. Level, reset by ObCpl_PciERst.	R	0b0

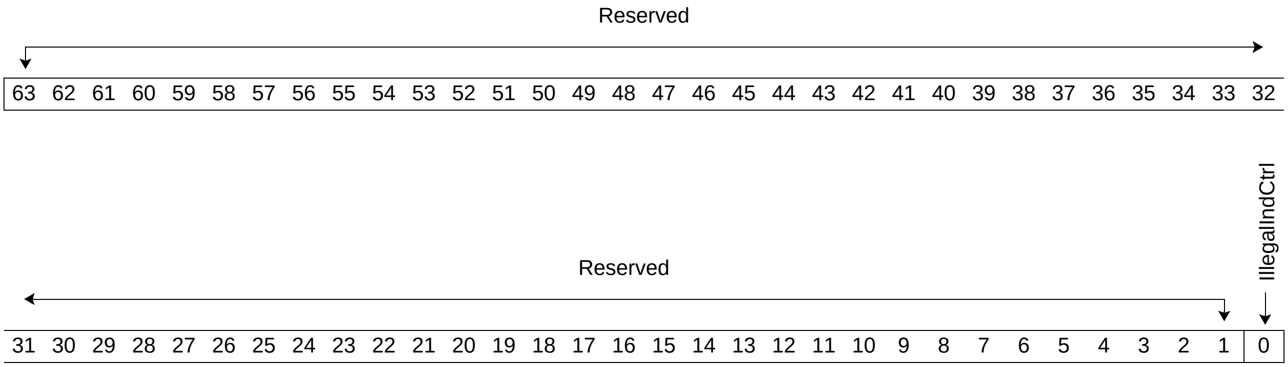
Bits	Field Name	Description	Access	Reset
0	DbgP_ZroTagFIFO_OF	When asserted, this signal indicates that more ZroTag entries have been received than the ZroTagFIFO can hold (16 entries). Fatal Error. Level, reset by ObCpl_PciERst.	R	0b0

12.11.3.18 DIAG_OBMSG: Diagnostic Register for Application Layer Outbound Message (ObMsg)

Reset Value N/A

Address x'F8081090'

Access Type Read, Read/Write



Bits	Field Name	Description	Access	Reset
63:1	Reserved	Always returns 0 on read; write operations have no effect.	R	0x0000000
0	IllegalIndCtrl	This bit, if set, indicates that the ObMsg has received a request to send out either a Power_Indicator message or an Attention_Indicator message with the corresponding control field set to 2'b00. ObMsg has sent out either a Power_Indicator_Off or an Attention_Indicator_Off in response to this request.	R/W	0b0

12.11.3.19 DIAG_OBNP: Diagnostic Register for AL Outbound Nonposted (ObNP)

Reset Value N/A

Address x’F80810A0’

Access Type Read

Reserved for ObNP status
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

Reserved for ObNP errors	DbgP_InvalidRdType	DbgP_InvalidWrType	DbgP_FatalRdType[5:0]
31 30 29 28 27 26 25 24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0

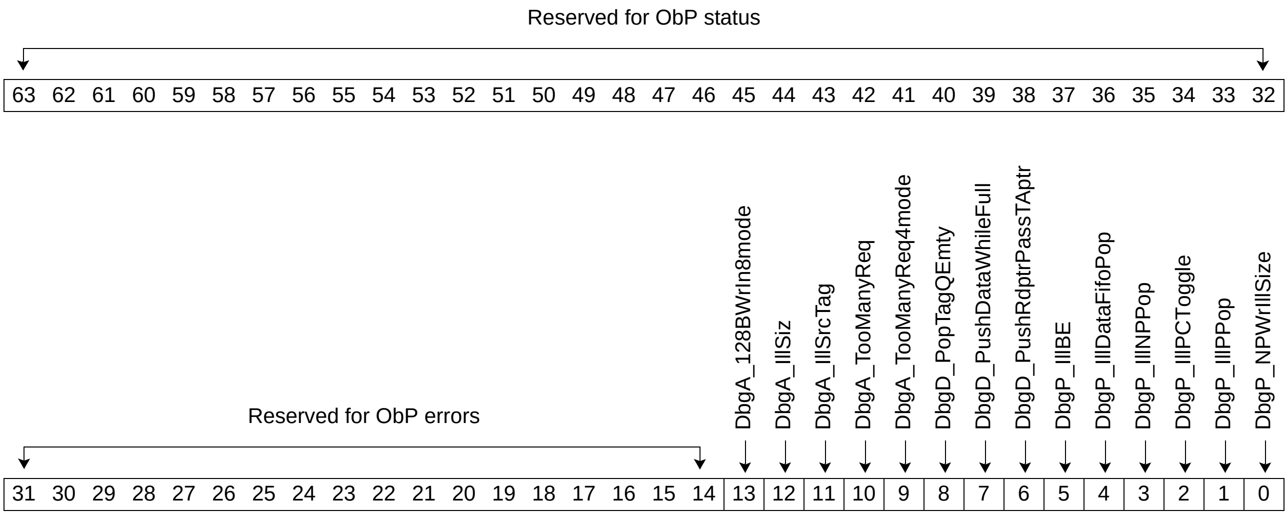
Bits	Field Name	Description	Access	Reset
63:32	Reserved for ObNP status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:18	Reserved for ObNP errors	Always returns 0 on read; write operations have no effect.	R	0x000000
17:12	DbgP_InvalidRdType	An Invalid Read of Type specified in following bits has occurred: 17 A configuration transaction is of an invalid type. 16 A type 0 configuration transaction (not direct) has a 0 cycle bit, in the case of limited direct type 0 configuration. 15 A type 0 configuration transaction (not direct) has a non-zero function number. 14 A type 1 configuration transaction fails the outbound as type 0 and outbound as type 1 checks. 13 An I/O transaction occurs when I/O transactions are not enabled. 12 A memory transaction occurs when memory transactions are not enabled.	R	0b000000
11:6	DbgP_InvalidWrType	An Invalid Write of Type specified in following bits has occurred: 11 A configuration transaction is of an invalid type. 10 A type 0 configuration transaction (not direct) has a 0 cycle bit. In the case of limited direct type 0 Configuration. 9 A type 0 configuration transaction (not direct) has a non-zero function number. 8 A type 1 configuration transaction fails the outbound as type 0 and outbound as type 1 checks. 7 An I/O transaction occurs when I/O transactions are not enabled. 6 A memory transaction occurs when Memory transactions are not enabled.	R	0b000000
5:0	DbgP_FatalRdType[5:0]	A Fatal Read Type of Type specified in following bits has occurred: 5 The read has an unsupported Rid. 4 The read has an unsupported TSiz. 3 The read has no byte enables asserted. 2 The read is for more than 8 bytes or less and crosses an 8-byte boundary. 1 The read targets configuration space and is for more than 4 bytes or crosses a 4-byte boundary. 0 The read targets I/O space and is for more than 4 bytes or crosses a 4-byte boundary.	R	0b000000

12.11.3.20 DIAG_OBP: Diagnostic Register for Application Layer Outbound Posted (ObP)

Reset Value N/A

Address x’F80810B0’

Access Type Read



Bits	Field Name	Description	Access	Reset
63:32	Reserved for ObP status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:14	Reserved for ObP errors	Always returns 0 on read; write operations have no effect.	R	0x00000
13	DbgA_128BWrIn8mode	A 128 byte write has been issued in 8 entry, 64-byte write maximum mode.	R	0b0
12	DbgA_IllSiz	An illegal size has been issued in a PI transaction.	R	0b0
11	DbgA_IllSrcTag	An illegal source tag has been encountered in a PI transaction.	R	0b0
10	DbgA_TooManyReq	Too many REQs have been issued on API without ACKs.	R	0b0
9	DbgA_TooManyReq4mode	There have been more than 4 REQs from Processor Interconnect without any intervening ACKs, and ObP is in 4 mode.	R	0b0
8	DbgD_PopTagQEmty	The tag queue has been popped when it was empty.	R	0b0
7	DbgD_PushDataWhileFull	The data FIFO was pushed when it was full.	R	0b0
6	DbgD_PushRdptrPassTAptr	The push read pointer for the tag queue has passed the TA read pointer.	R	0b0
5	DbgP_IllBE	Illegal byte enables have been issued in a TLP header.	R	0b0
4	DbgP_IllDataFifoPop	There has been an illegal data FIFO pop.	R	0b0
3	DbgP_IllNPPop	There has been an illegal nonposted write pop.	R	0b0
2	DbgP_IllPCToggle	There has been a ObNP_PendingChecked toggle without a previous ObP_CheckPending toggle.	R	0b0
1	DbgP_IllPPop	There has been an illegal posted write pop.	R	0b0
0	DbgP_NPWrIllSize	There has been an illegal nonposted write size.	R	0b0

12.11.3.21 DIAG_OBTLIF: Diagnostic Register for Application Layer Outbound Posted (ObTLIf)

Reset Value	N/A
Address	x’F80810C0’
Access Type	Read

Reserved for ObTLIf status																															
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Reserved for ObTLIf errors																												DbgP_ArbEnFSM	DbgP_DataCnsmFSM	DbgP_DataTxFSM	DbgP_ToAckFSM
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
63:32	Reserved for ObTLIf status	Always returns 0 on read; write operations have no effect.	R	0x00000000
31:4	Reserved for ObTLIf errors	Always returns 0 on read; write operations have no effect.	R	0x0000000
3	DbgP_ArbEnFSM	Arbiter Enable FSM error indicated.	R	0b0
2	DbgP_DataCnsmFSM	Data Consume FSM error indicated.	R	0b0
1	DbgP_DataTxFSM	Data Transmit FSM error indicated.	R	0b0
0	DbgP_ToAckFSM	ToAck Timer Control FSM has error indicated.	R	0b0

12.11.3.22 MASK_MPIC_IBCPL: MPIC Masking for DIAG_IBCPL

Reset Value N/A

Address x’F8081100’

Access Type Read

Reserved																															
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Reserved																					Mask_MPIC_IbCpl										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
63:11	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
10:0	Mask_MPIC_IbCpl	Enable IbCpl error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_IBCPL.	R	0x001

12.11.3.23 MASK_MPIC_IBCMGR: MPIC Masking for DIAG_IBCMGR

Reset Value N/A

Address x’F8081110’

Access Type Read

Reserved for Mask_MPIC_IbCMgr																															
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Reserved for Mask_MPIC_IbCMgr																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

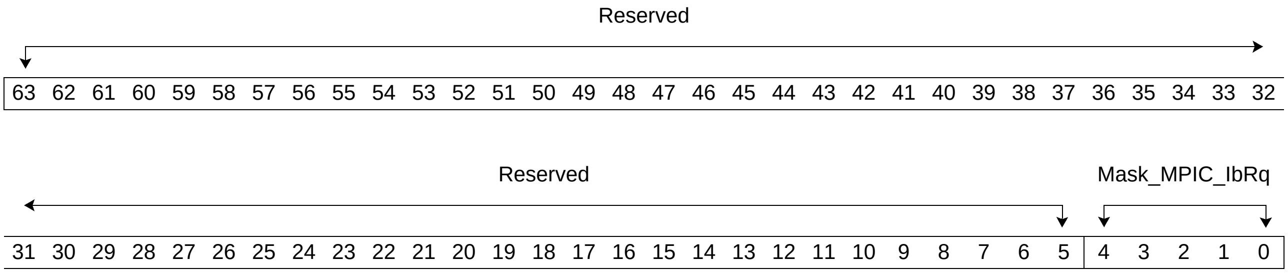
Bits	Field Name	Description	Access	Reset
63:0	Reserved for Mask_MPIC_IbCMgr	Always returns 0 on read; write operations have no effect. Enable IbCMGR error signal to trigger an interrupt in the MPIC controller. These bits correspond to the same bit positions defined in DIAG_IBCMgr.	R	0x00000000

12.11.3.24 MASK_MPIC_IBRQ: MPIC Masking for DIAG_IBRQ

Reset Value N/A

Address x’F8081120’

Access Type Read



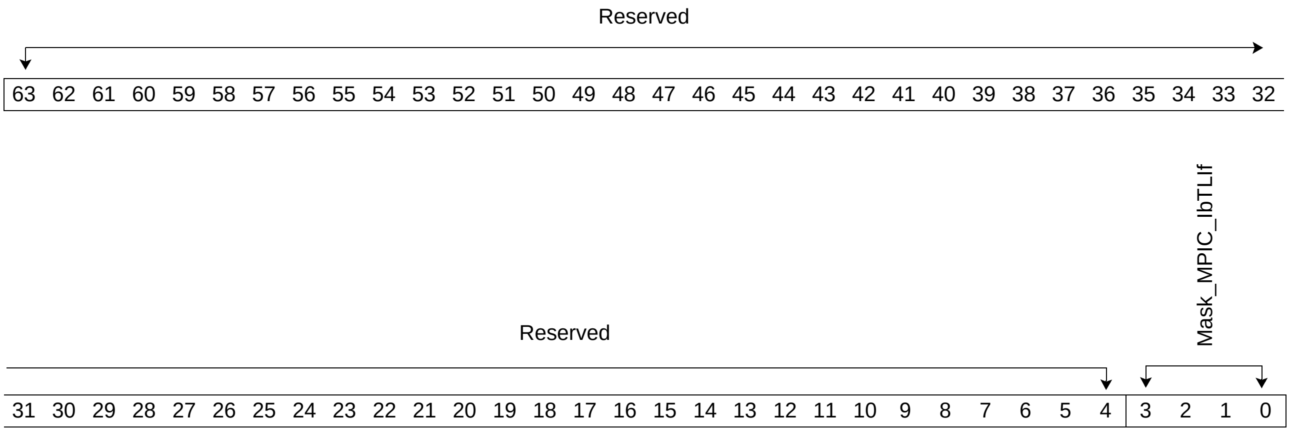
Bits	Field Name	Description	Access	Reset
63:5	Reserved	Always returns 0 on read; write operations have no effect.	R	0x000000
4:0	Mask_MPIC_IbRq	Enable IbRq error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_IBRQ.	R	0b00001

12.11.3.25 MASK_MPIC_IBTLIF: MPIC Masking for DIAG_IBTLIF

Reset Value N/A

Address x’F8081130’

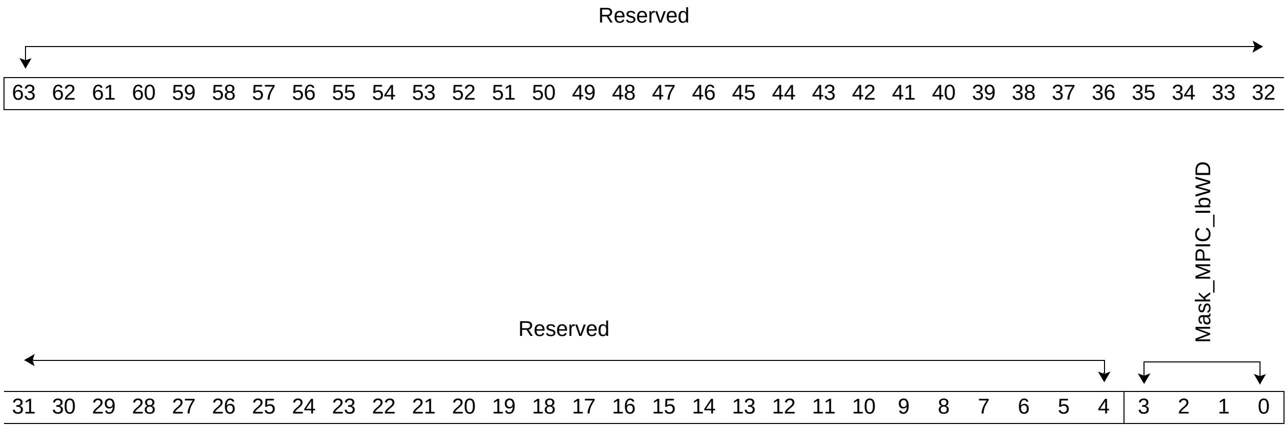
Access Type Read



Bits	Field Name	Description	Access	Reset
63:4	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
3:0	Mask_MPIC_IbTLIf	Enable IbTLIf error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_IBTLIF.	R	0b001

12.11.3.26 MASK_MPIC_IBWD: MPIC Masking for DIAG_IBWD

Reset Value	N/A
Address	x’F8081140’
Access Type	Read



Bits	Field Name	Description	Access	Reset
63:4	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
3:0	Mask_MPIC_IbWD	Enable IbWD error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_IBWD.	R	0b0001

12.11.3.27 MASK_MPIC_OBCPL: MPIC Masking for DIAG_OBCPL

Reset Value	N/A
Address	x’F8081180’
Access Type	Read

Reserved																															
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Reserved																							Mask_MPIC_ObCpl								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
63:9	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
8:0	Mask_MPIC_ObCpl	Enable ObCpl error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_OBCPL.	R	0x01

12.11.3.28 MASK_MPIC_OBNP: MPIC Masking for DIAG_OBNP

Reset Value	N/A
Address	x’F80811A0’
Access Type	Read

Reserved																															
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Reserved																										Mask_MPIC_ObNP					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bits	Field Name	Description	Access	Reset
63:6	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
5:0	Mask_MPIC_ObNP	Enable ObNP error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_OBNP.	R	0b000001

12.11.3.29 MASK_MPIC_OBP: MPIC Masking for DIAG_OBP

Reset Value N/A

Address x’F80811B0’

Access Type Read

Reserved																															
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

Reserved																		Mask_MPIC_ObP													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

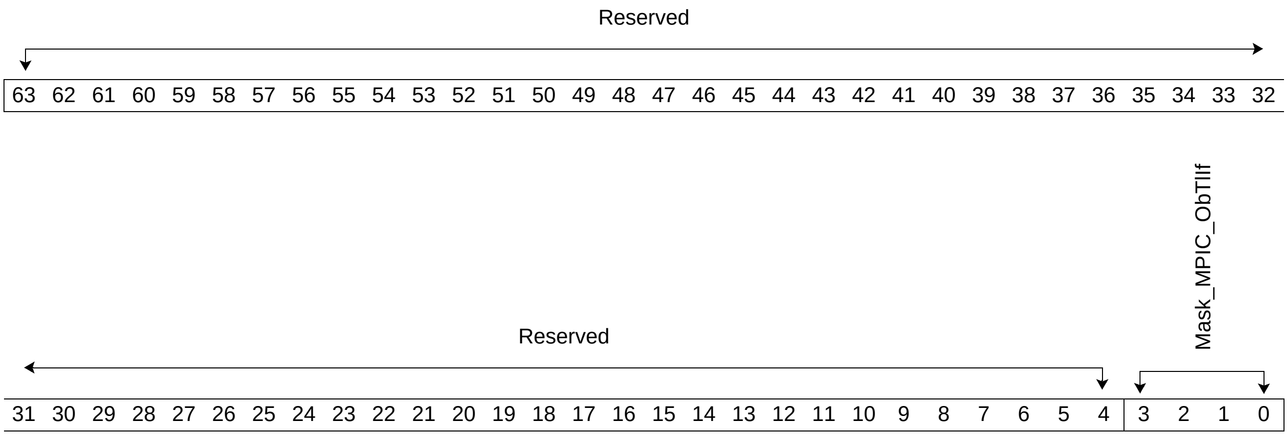
Bits	Field Name	Description	Access	Reset
63:14	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
13:0	Mask_MPIC_ObP	Enable ObP error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_OBP.	R	0x0001

12.11.3.30 MASK_MPIC_OBTLIF: MPIC Masking for DIAG_OBTLIF

Reset Value N/A

Address x’F80811C0’

Access Type Read



Bits	Field Name	Description	Access	Reset
63:4	Reserved	Always returns 0 on read; write operations have no effect.	R	0x00000000
3:0	Mask_MPIC_ObTlIf	Enable ObTlIf error signals to trigger an interrupt in the MPIC controller. Each bit corresponds to the same position defined in DIAG_ObTlIf.	R	0x0001

12.12 HyperTransport Registers (HT1)

The registers listed in *Table 12-34* control the HyperTransport Host bridge logic found in the CPC945 design. The layout of HyperTransport registers is intended to comply as much as possible with the governing specifications (*PCI Local Bus Specification, Revision 2.2 and HyperTransport I/O Link Specification, Revision 1.04*).

HT is a superset of PCI and this block of registers is located in the CPC945 Control Register portion of the address map instead of the PCI Configuration space. Therefore, all of the HyperTransport registers must be spaced 0x10 bytes apart and accessed directly instead of using the indirect method used for PCI Configuration registers. The address of any given register can be determined by left shifting the original configuration index two places and adding 0xF8070000. For example, the Status/Command Register address is ((04h << 2) + 0xF8070000) = F8070010.

The set of registers described here is composed of three subsets of registers, the HyperTransport Header, the HyperTransport Interface Capabilities Block, and implementation specific registers. The header portion of the register block is declared as a Device Header and is similar to the PCI Configuration blocks used in CPC945. One required deviation from the Device header format is the addition of the *“Bridge Control Register (BrCtrl)” on page 636* at address 0xF8070300. The Bridge Control register is used to specify the error and reset behavior of the HyperTransport link connected to the host (primary interface). The HyperTransport Interface Capabilities section of the register block is used for configuration and status of the HyperTransport specific portions of the interface. The final section of registers controls items that fall outside the scope of the PCI and HyperTransport specifications. The top view of the HyperTransport register block is:

- 0xF8070000 - 0xF80700F3h: Device header
- 0xF8070100 - 0xF80701F3h: Capabilities blocks
- 0xF8070200 - 0xF80703F3h: CPC945 specific registers
- 0xF8070500 - 0xF80705F3h: Performance registers

Note: In the tables below, if there is no page number shown, the register is not supported.

Table 12-30. Device Header HyperTransport Register .

0xF8070000 - 0xF80700F3: Device Header					
[0:7]	[8:15]	[16:23]	[24:31]	Address	Page
Device ID		Vendor ID		0xF8070000	619
Status		Command		0xF8070010	620
Class Code			Revision ID	0xF8070020	621
BIST	Header Type	Latency Timer	Cache Line Size	0xF8070030	622
				0xF8070040	
				0xF8070050	
Base Address Registers				0xF8070060	
(Not Supported)				0xF8070070	
				0xF8070080	
				0xF8070090	
Cardbus CIS Pointer				0xF80700A0	
Subsystem ID		Subsystem Vendor ID		0xF80700B0	

Table 12-30. Device Header HyperTransport Register (Continued).

0xF8070000 - 0xF80700F3: Device Header					
[0:7]	[8:15]	[16:23]	[24:31]	Address	Page
Expansion ROM Base Address				0xF80700C0	
Reserved			Capabilities Pointer	0xF80700D0	623
Reserved				0xF80700E0	
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	0xF80700F0	624

Table 12-31. Capabilities Block HyperTransport Registers.

0xF8070100 - 0xF80701F3: Capabilities Block					
[0:7]	[8:15]	[16:23]	[24:31]	Address	Page
Command		Capability Pointer	Capability ID	0xF8070100	625
Link Config		Link Control		0xF8070110	626
LinkFreqCap		Link Error/Frequency	HTRev	0xF8070120	628
Reserved		Feature		0xF8070130	629
Error Handling		Enumeration Scratchpad		0xF8070140	630
Reserved		Mem Limit Upper	MemBase Upper	0xF8070150	

Table 12-32. CPC945-Specific HyperTransport Registers.

0xF8070200 - 0xF80703F3: CPC945-Specific Registers						
[0:3]	[4:7]	[8:15]	[16:23]	[24:31]	Address	Page
HT1 Address Mask					0xF8070200	632
HT1/PI Interface Control					0xF8070210	633
HT1/PI Read Memory Delay					0xF8070220	633
HT1/PI Write Memory Delay					0xF8070230	634
HTG Configuration					0xF8070240	635
Bridge Control			Reserved		0xF8070300	636
Reserved	TxCtl/ BufRelSpace	DataBufAlloc			0xF8070310	638
Reserved			ImpedCtrl (not implemented)		0xF8070320	
Reserved			ImpedCtrlSWCal (not implemented)		0xF8070330	
Reserved		TxBufCountMax			0xF8070340	639
DiagRxCrcExpLane0					0xF8070350	640
DiagRxCrcRcvLane0					0xF8070360	
DiagRxCrcExpLane1					0xF8070370	
DiagRxCrcRcvLane1					0xF8070380	

Table 12-32. CPC945-Specific HyperTransport Registers.

0xF8070200 - 0xF80703F3: CPC945-Specific Registers						
[0:3]	[4:7]	[8:15]	[16:23]	[24:31]	Address	Page
SriRxNumeratorLower					0xF8070390	641
SriRxNumeratorUpper					0xF80703A0	
SriTxNumeratorLower					0xF80703B0	
SriTxNumeratorUpper					0xF80703C0	
SriOveride					0xF80703D0	643
HtPhyCtl					0xF80703E0	644

Table 12-33. HyperTransport Performance Monitor Counter Registers.

0xF8070500 - 0xF80705FFh: HT Performance Monitor					
[0:7]	[8:15]	[16:23]	[24:31]	Address	Page
PM HT Configuration Register 0 (Not Implemented)				0XF8070500	
PM HT Configuration Register 1 (Not Implemented)				0XF8070510	
PM HT Configuration Register 2 (Not Implemented)				0XF8070520	
PM HT Configuration Register 3 (Not Implemented)				0XF8070530	
Undefined (read as 0x00000000, writes have no effect)				0XF8070540- 0XF8070570	
PM HT Data Register 0 (Not Implemented)				0XF8070580	
PM HT Data Register 1 (Not Implemented)				0XF8070590	
PM HT Data Register 2 (Not Implemented)				0XF80705A0	
PM HT Data Register 3 (Not Implemented)				0XF80705B0	
Undefined (read as 0x00000000, writes have no effect)				0XF80705C0- 0XF80705F0	

Table 12-34. HyperTransport Registers.

System Bus Address	Register Name	Bits Used	Description	Page
0xF8070000	Device ID Vendor ID	[0:15] [16:31]	Device Identification Register. Vendor Identification Register.	619
0xF8070010	Status Command	[0:15] [16:31]	Provides status information about the interface. Controls how the bridge responds on the HyperTransport interface.	620
0xF8070020	Class Code Revision ID	[0:23] [24:31]	Class Code specifies device type. Specifies revision level of the chip.	621
0xF8070030	BIST Header Type Latency Timer Cache Line Size	[0:7] [8:15] [16:23] [24:31]	BIST register. Indicates the type of header block used. Not implemented by HyperTransport. Not implemented by HyperTransport.	622
0xF8070040- 0xF8070090	Base Address Registers	[0:31] (each)	BAR’s are not implemented in this design.	

Table 12-34. HyperTransport Registers.

System Bus Address	Register Name	Bits Used	Description	Page
0xF80700A0	Cardbus CIS Pointer	[0:31]	Not implemented by HyperTransport.	
0xF80700B0	Subsystem ID Subsystem Vendor ID	[0:15] [16:31]	Not implemented in this design. Not implemented in this design.	
0xF80700C0	Expansion ROM Base Address	[0:31]	Not implemented in this design.	
0xF80700D0	Reserved Capabilities Pointer	[0:23] [24:31]	Reserved. This is the pointer to the first capability block.	623
0xF80700E0	Reserved	[0:31]	Reserved.	
0xF80700F0	Max_Lat Min_Gnt Interrupt Pin Interrupt Line	[0:7] [7:15] [16:23] [24:31]	Not implemented in this design. Not implemented in this design. Not implemented in this design. Scratchpad to track interrupt routing.	624
0xF8070100	Command Capability Pointer Capability ID	[0:15] [16:23] [24:31]	Host/Secondary Interface Command Register. Pointer to the next capability block. Capability ID.	625
0xF8070110	LinkConfig LinkCtrl	[0:15] [16:31]	Link Configuration Register. Link Control.	626
0xF8070120	LinkFreqCap LinkError LinkFreq HTRev	[0:15] [16:19] [20:23] [24:31]	Link Frequency Capability Register. Link Error Register. Link Frequency Control Register. HyperTransport Revision ID Register. This device conforms to rev 1.03.	628
0xF8070130	Reserved Feature	[0:15] [16:31]	Reserved. Feature Capability Register.	629
0xF8070140	Error Handling Enumeration Scratchpad	[0:15] [16:31]	Error Handling Register. Enumeration Scratchpad Register.	630
0xF8070150	Reserved Mem Limit Upper MemBase Upper	[0:15] [16:13] [24:31]	Reserved. Not implemented in this design. Not implemented in this design.	
0xF8070200	HT1 Address Mask	[0:31]	This register controls which addresses are passed to the HT1 chain.	632
0xF8070210	HT1/PI Interface Control	[0:31]	This register specifies how many requests the PI is capable of receiving.	633
0xF8070220	HT1/PI Memory Read Delay	[28:31]	Memory Read Delay for Memory Read Data Interface.	633
0xF8070230	HT1/PI Memory Write Delay	[26:31]	Write TA delay for Write Data Interface.	634
0xF8070240	HTG Configuration	[30:31]	For fine-tune HTG options.	635
0xF8070300	Bridge Control Reserved	[0:15] [16:31]	Bridge Control Register. Reserved.	636

Table 12-34. HyperTransport Registers.

System Bus Address	Register Name	Bits Used	Description	Page
0xF8070310	Reserved TxCtl DataBufAlloc	[0:11] [12:15] [16:31]	Reserved. Transmit Control. This register contains controls for the HyperTransport transmitters. Rx Data Buffer Allocation. Controls the allocation of the 8 receive data buffers in each link among the 3 virtual channels, allowing performance tuning. The “Need” fields indicate the minimum allocation to each channel, minus one. The “Want” fields indicate how many buffers the allocator should try to have released and outstanding to each channel at all times, minus 1. The total number of buffers “needed” must be less than or equal to the total number (8) of data buffers available. If less, the allocator gets more flexibility handing out buffers dynamically. In the default (reset) case, there are 2 buffers in each category.	638
0xF8070320	Reserved ImpedCtrl (not implemented)	[0:15] [16:31]	Reserved. Link Impedance Control Registers.	
0xF8070330	Reserved ImpedCtrlSWCal (not implemented)	[0:15] [16:31]	Reserved. Software control of calibrator Device 1 Offset 19ch-19fh.	
0xF8070340	Reserved TxBufCountMax	[0:7] [8:31]	Reserved. Maximum threshold for the transmit buffer counters If buffer releases are received in a particular channel which exceed the threshold for that channel, the extras are discarded. This allows a general way to throttle the traffic on the link. The counter value can be lowered in a running system, but the system must go through reset to make an increase take effect.	639
0xF8070350	DiagRxCrcExpLane0	[0:31]	Expected CRC value for Lane 0.	640
0xF8070360	DiagRxCrcRcvLane0	[0:31]	Received CRC value for Lane 0.	
0xF8070370	DiagRxCrcExpLane1	[0:31]	Expected CRC value for Lane 1.	
0xF8070380	DiagRxCrcRcvLane1	[0:31]	Received CRC value for Lane 1.	
0xF8070390	SriRxNumeratorLower	[0:31]	Numerator Values for HT Sync FIFOs.	641
0xF80703A0	SriRxNumeratorUpper	[0:31]	Numerator Values for HT Sync FIFOs.	
0xF80703B0	SriTxNumeratorLower	[0:31]	Numerator Values for HT Sync FIFOs.	
0xF80703C0	SriTxNumeratorUpper	[0:31]	Numerator Values for HT Sync FIFOs.	
0xF80703D0	SriOveride	[0:31]	Override Values for HT Sync FIFOs.	643
0xF80703E0	HT/PI PHY Control	[19:31]	Control bit for fine tuning HT PHY.	644
0XF8070500	PMHC0	[0:27]	PM HT Configuration Register 0.	
0XF8070510	PMHC1	[0:27]	PM HT Configuration Register 1.	
0XF8070520	PMHC2	[0:27]	PM HT Configuration Register 2.	
0XF8070530	PMHC3	[0:27]	PM HT Configuration Register 3.	
0XF8070540- 0XF8070570			Undefined (read as 0x00000000, writes have no effect).	
0XF8070580	PMHD0	[0:31]	PM HT Data Register 0.	
0XF8070590	PMHD1	[0:31]	PM HT Data Register 1.	

Table 12-34. HyperTransport Registers.

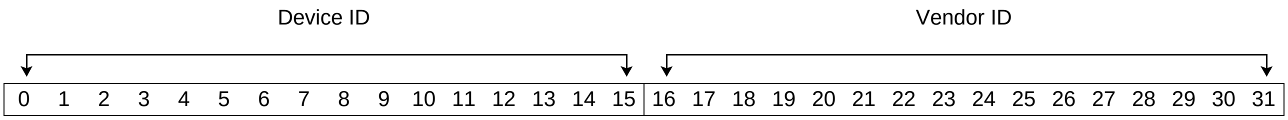
System Bus Address	Register Name	Bits Used	Description	Page
0XF80705A0	PMHD2	[0:31]	PM HT Data Register 2.	
0XF80705B0	PMHD3	[0:31]	PM HT Data Register 3.	
0XF80705C0- 0XF80705F0			Undefined (read as 0x00000000, writes have no effect).	

Note: Some registers have a specialized type of R/W:

```
R/W (R/C) => a write of '1' clears the bit
R/W (R/S) => a write of '1' sets the bit
```

12.12.1 HT Device ID/Vendor ID Register (Device ID/Vendor ID)

Reset Value	0x0074106B
Offset	0xF8070000
Access Type	Read Only

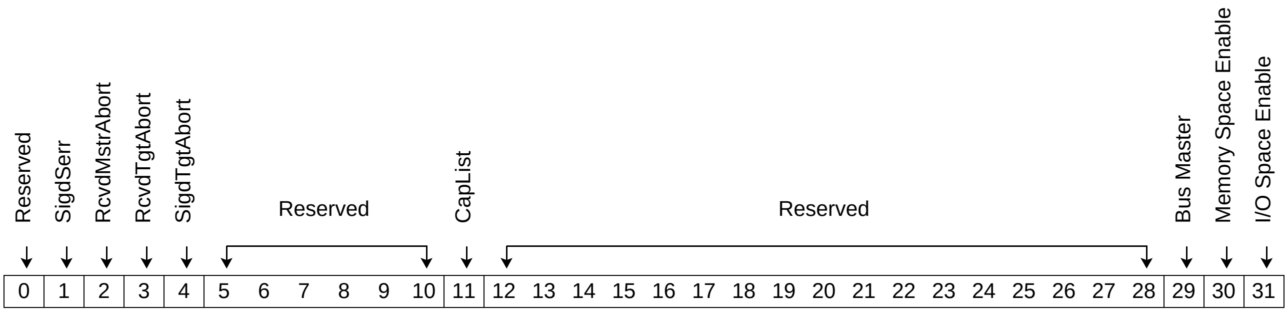


Bits	Field Name	Description	Access	Reset
0:15	Device ID	Device ID assigned to this chip by vendor. CPC945 HT1 (1.0, 1.1, 2.0)= 0x004A	R	0x004A
16:31	Vendor ID	CPC945 = 0x106B	R	0x106B

12.12.2 Status/Command Register (Status/Command)

Controls how the bridge responds on the HyperTransport interface. Only the bits that are used by the host are listed in the Command Register Description.

Reset Value	0x00100006
Offset	0xF8070010 (Bits 0:15 Status Register) 0xF8070010 (Bits 16:23 Command Register)
Access Type	Read/Write, Read Only

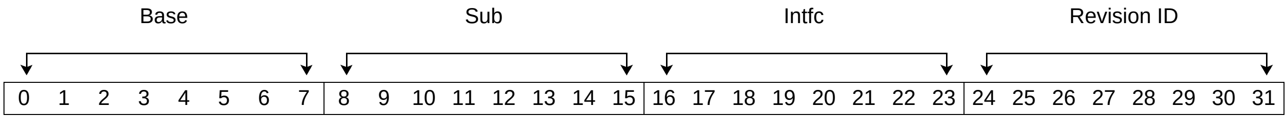


Bits	Field Name	Description	Access	Reset
0	Reserved	This field will read all 0’s.	R	0b0
1	SigdSerr	Not implemented This bit reports the assertion of a system error by a bridge on its primary interface. Since CPC945 HT is a host bridge, there is not primary HT interface.	R	0b0
2	RcvdMstrAbort	This bit reports the detection of a master abort termination by the bridge, when it is the master of a transaction on its primary interface. An equivalent error can exist inside the host. It can be cleared by writing a 1 to it.	R/C	0b0
3	RcvdTgtAbort	This bit reports the detection of a target abort by the bridge when it is the master of a transaction on its primary interface. It can be cleared by writing a 1 to it. An equivalent error can exist inside the host.	R/C	0b0
4	SigdTgtAbort	This bit reports the signalling of a target abort termination by the bridge, when it responds as the target of a transaction on its primary interface. An equivalent error can exist inside the host bit set. It can be cleared by writing a 1 to it.	R/C	0b0
5:10	Reserved	This field will read all 0’s.	R	0x00
11	CapList	Indicates that the bridge supports a capabilities list.	R	0b1
12:28	Reserved	This field will read all 0’s	R	0x0000
29	Bus Master	Controls the devices ability to issue requests onto HyperTransport chain. As a Host Bridge, CPC945 is always able to act as a master.	R	0b1
30	Memory Space Enable	CPC945 always accepts memory cycles.	R	0b1
31	I/O Space Enable	CPC945 never accepts I/O cycles.	R	0b0

12.12.3 Class Code/Revision Register (Class Code/Revision ID)

Specifies class code and revision level of the chip.

Reset Value	0x06000000
Offset	0xF8070020
Access Type	Read Only

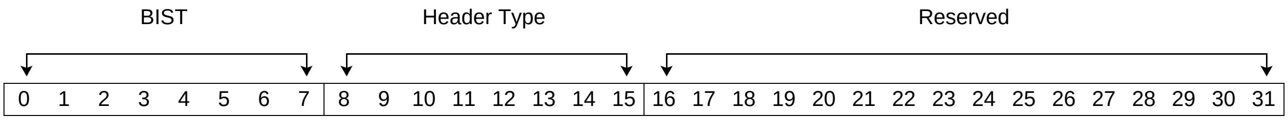


Bits	Field Name	Description	Access	Reset
0:7	Base	Base class of the device – 0x06 indicates a host bridge.	R	0x06
8:15	Sub	Subclass of the device – 0x00 indicates a host bridge.	R	0x00
16:23	Intfc	Interface class of the device – 0x00 indicates a host bridge.	R	0x00
24:31	Revision ID	Revision ID assigned by vendor.	R	0x00

12.12.4 BIST/Header Type Register (BIST/Header Type)

The BIST register is used for control and status of BIST logic. The Header Type register identifies the layout of the second part of the configuration header.

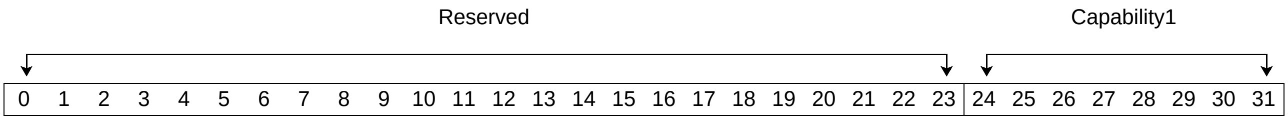
Reset Value 0x00000000
Offset 0xF8070030
Access Type Read Only



Bits	Field Name	Description	Access	Reset
0:7	BIST	Not implemented in this design.	R	0x00
8:15	Header Type	A value of 0x0 indicates that this is a device header.	R	0x00
16:31	Reserved	This field will read all 0’s	R	0x0000

12.12.5 Capabilities Pointer Register (Capability1)

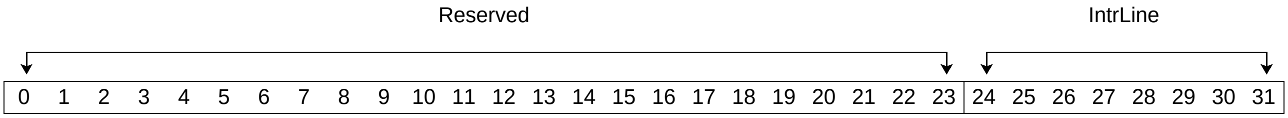
Reset Value	0x00000040
Offset	0xF80700D0
Access Type	Read Only



Bits	Field Name	Description	Access	Reset
0:23	Reserved	This field will read all 0’s	R	0x000000
24:31	Capability1	Register number (less the bottom two bits, which must be 0) of the base of the first capabilities block. All HyperTransport devices have at least an HyperTransport capability block.	R	0x40

12.12.6 Interrupt Line Register (IntrLine)

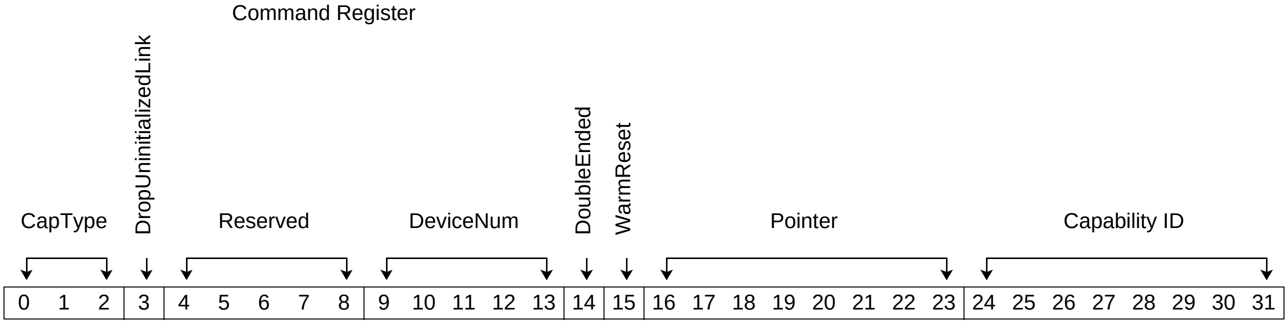
Reset Value	0x000000FF
Offset	0xF80700F0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:23	Reserved	This field will read all 0’s.	R	0x000000
24:31	IntrLine	Used by software as a scratchpad to track interrupt routing.	R/W	0xFF

12.12.7 Command/Pointer/Capability ID Register (HTCapability00)

Reset Value	0X20010008
Offset	0xF8070100
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:2	CapType	Indicates what type of HyperTransport capability block this is. For a secondary block, it is always 001.	R	0b001
3	DropUninitializedLink	Drop on Unitialized Link This bit determines what will happen to packets issued by a device or forwarded from a receiving link interface to a transmitting interface whose Initialization Complete and End of Chain bits are clear. If both are deasserted for a given link, packets to be transmitted on that link will be stalled until either Initialization Complete sets (in which case they will be transmitted) or End Of Chain sets (in which case they will be treated as End Of Chain packets; see *Section 10.1.5 of HT Specification 1.04*). In the case where hardware is broken, it is possible that neither of these events occurs, in which case the packet can hang. If Drop on Unitialized Link is set, a transmitter with its Initialization Complete bit clear will always act as if the End of Chain bit were set. Hosts that use the initialization sequence described in *Section 12.3 of HT spec 1.04* are encouraged to implement a timeout counter to prevent a system-wide initialization error due to link-level initialization problems on a non-default chain. Packet forwarding behavior is described in *Table 37 of the HT specification 1.04*.	R/W	0b0
4:8	Reserved	This field will read all 0’s	R	0x00
9:13	DeviceNum	Device number that this host uses when responding to PCI type 0 configuration cycles from the HyperTransport chain. Double-hosted chain not implemented in this design.	R	0b0000
14	DoubleEnded	Set by a host on the other end of the chain during initialization to indicate that the link has hosts on both ends. Double-hosted chain not implemented in this design.	R	0b0
15	WarmReset	This bit controls whether a reset sequence initiated by writing the Secondary Bus Reset bit of the Bridge Control register is cold or warm. (Cold means that PwrOk is deasserted during the sequence.) 0 Cold 1 Warm	R/W	0b1
16:23	Pointer	Register number (less the bottom two bits, which must be 0) of the base of the next capabilities block. It is 0 if there are no other capabilities blocks.	R	0x00
24:31	Capability ID	Capability ID assigned by the PCI-SIG for HyperTransport.	R	0x08

12.12.8 Link Config/Link Control Register (HTCapability04)

In the HyperTransport Interface capabilities block there are four register fields related to the width of the link. They are LinkWidthOut, LinkWidthIn, MaxLinkWidthOut, and MaxLinkWidthIn. The first two register fields describe the “utilized” width of the link. This is the result of the hardware based link-width negotiation that occurs at cold reset. The latter two register fields (MaxLinkWidthOut and MaxLinkWidthIn) are supposed to reflect the physical width present on the device.

The same encoding is used for all Link Configuration subfields:

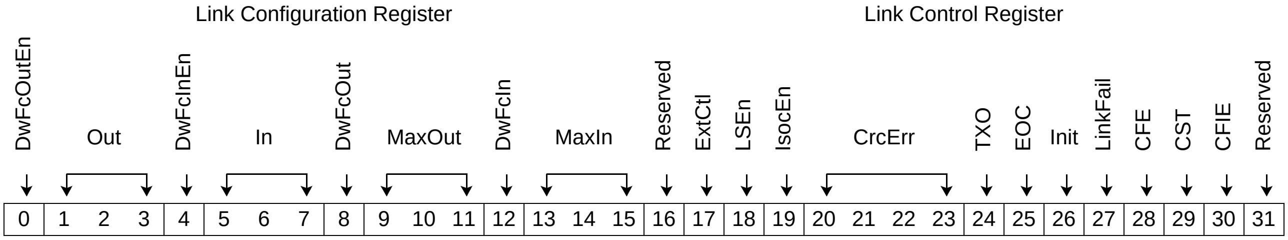
```
000 - 8 bit
001 - 16 bit
011 - 32 bit
100 - 2 bit
101 - 4 bit
111 - disconnected (when applicable)
```

All other encodings are reserved.

Note: Some registers have a specialized type of R/W:

```
R/W (R/C) => a write of ’1’ clears the bit
R/W (R/S) => a write of ’1’ sets the bit
```

Reset Value	0x00110020
Offset	0xF8070110
Access Type	Read/Write, Read Only



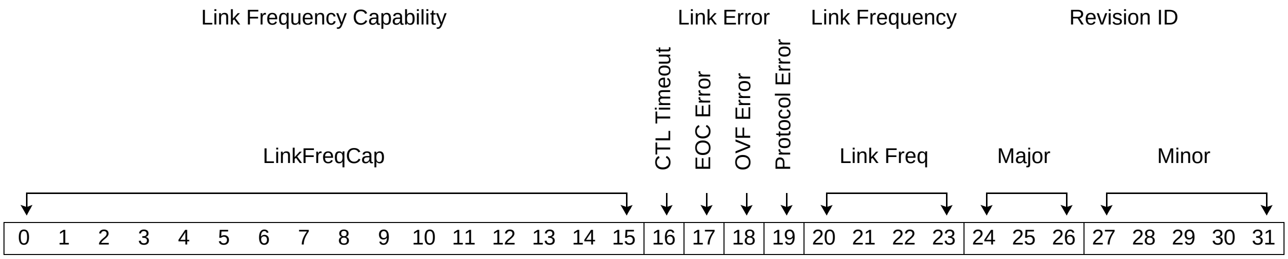
Bits	Field Name	Description	Access	Reset
0	DwFcOutEn	Doubleword Flow Control Out Enable (This optional mode is not supported by CPC945's HyperTransport interface.)	R	0b0
1:3	Out	LinkWidthOut This controls the width used of the outgoing link from this HT device. It must match the used incoming width of the HT device on the other end of the link. A cold software link reset using the SecBusReset bit in the Bridge Control Register resets this bit. Resets to zero, then set by hardware during Link Width Initialization.	R/W	0b000
4	DwFcInEn	Doubleword flow control enable (not supported)	R	0b0
5:7	In	LinkWidthIn This controls the width used of the incoming HT link to this device. It must match the used outgoing width of the HT device on the other end of the link. A cold software link reset using the SecBusReset bit in the Bridge Control Register resets this bit. Resets to zero, then set by hardware during link width initialization.	R/W	0b000

Bits	Field Name	Description	Access	Reset
8	DwFcOut	Doubleword flow control out (This optional mode is not supported by CPC945's HyperTransport interface.)	R	0b0
9:11	MaxOut	Indicates the maximum width of the outgoing link supported by this device. CPC945’s 16-bit bus uses an initial value of 0b001.	R	0b001
12	DwFcIn	Doubleword flow control in (This optional mode is not supported by CPC945's HyperTransport interface.)	R	0b0
13:15	MaxIn	Indicates the maximum width of the incoming link supported by this device. CPC945’s 16-bit bus uses an initial value of 0b001.	R	0b001
16	Reserved	This field will read all 0’s	R	0b0
17	ExtCtl	Extended CTL time If this bit is set, during the link initialization sequence in *Section 12.2. HT Specification 1.04*, CTL will be asserted for 50us after the point where both the transmitting device has asserted CTL and it has sampled CTL asserted from the other side of the link. If this bit is clear, CTL need only be asserted at least 16 bit-times after both sides assert CTL in 8-bit or larger links. (32 bit-times for 4-bit links, 64 bit-times for 2-bit links)	R/W	0b0
18	LSEn	LDTSTOP# tristate enable controls whether the link will be tristated during an LDTSTOP# sequence.	R/W	0b0
19	IsocEn	Enables Isochonous flow control mode for this link. (This optional mode is not supported by CPC945's HyperTransport interface.)	R	0b0
20:23	CrcErr	Each bit is set whenever a CRC error is detected on the corresponding byte lane of the link. Each bit can be cleared by writing a 1 to it. CPC945 only has 2 byte lanes, so only bits 22 and 23 are set.	R/W (R/C)	0b0000
24	TXO	TransmitterOff This bit shuts off the link transmitter to reduce EMI and power. The EOC bit should always be set prior to setting the XmitOff bit. It can only be set, not cleared. In the case of both a cold and warm software link reset using the SecBusReset bit in the Bridge Control Register this bit resets to zero. 0 Transmitter on 1 Transmitter off	R/W (R/S)	0b0
25	EOC	End of chain This bit indicates that this link is not part of the logical HyperTransport chain, and that this device should be considered the end of the chain for packets coming from the other direction. Packets directed towards this link are dropped or result in non-existent address (NXA) responses. It can only be set, not cleared. In the case of both a cold and warm software link reset using the SecBusReset bit in the Bridge Control Register this bit resets to zero.	R/W (R/S)	0b0
26	Init	Initialization complete This read-only bit indicates that low-level link initialization has successfully completed on the link. In the case of both a cold and warm software link reset using the SecBusReset bit in the Bridge Control Register this bit resets to zero when the reset begins and becomes one when the initialization is complete.	R	0b0
27	LinkFail	This bit is set to indicate that a failure has been detected on a link and should not be used. It is set by hardware and cleared by software.	R/W (R/C)	0b0
28	CFE	When this bit is a one, bad CRC is generated on all outgoing traffic on the link.	R/W	0b0
29	CST	Writing a 1 to this bit causes hardware to initiate a CRC test sequence on the link. When the test sequence has completed, hardware will clear the bit.	R/W (R/S)	0b0
30	CFIE	If set, this bit causes CRC errors to be treated as fatal errors. When detected, they cause all HyperTransport links from this device to be flooded with synchronization packets, and the LinkFail bit to be set.	R/W	0b0
31	Reserved	This field will read all 0’s.	R	0b0

12.12.9 LinkFreqCap/Link Error/Link Freq/ Revision ID Register

The name for this register is HTCapability08.

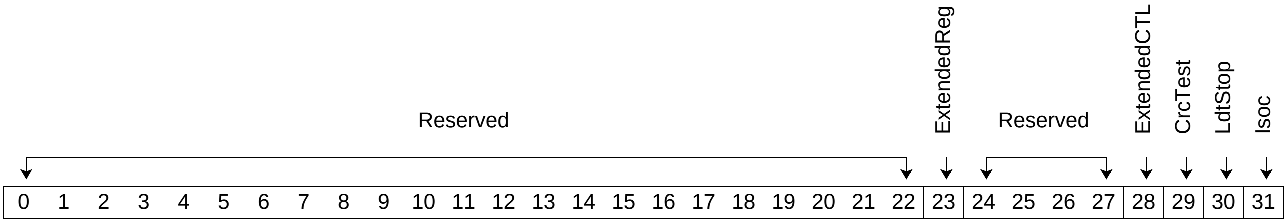
Reset Value	0x003F0024
Offset	0xF8070120
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:15	LinkFreqCap	Each bit in this mask corresponds to one of the 16 encodings in the Link Frequency register. A set bit indicates that the transmitter supports that frequency. For example 0x1F means that frequencies of 200 - 600 MHz are supported.	R	0x001F
16	CTL Timeout	This bit indicates how long CTL can be low before a device indicates a protocol error. 0 1 millisecond; 1 1 second. Not implemented in CPC945, but the HT specification states this bit must stay as a R/W.	R/W	0b0
17	EOC Error	End of chain error	R/C	0b0
18	OVF Error	Overflow error	R/C	0b0
19	Protocol Error	Protocol error	R/C	0b0
20:23	Link Freq	HyperTransport link frequency: 0000 200 MHz 0001 300 MHz 0010 400 MHz 0011 500 MHz 0100 600 MHz 0101 800 MHz 0110 1000 MHz 0111 - 1110 Reserved 1111 Vendor specific	R/W	0b0000
24:26	Major	Major revision of the specification. (Revision 1.04)	R	0b001
27:31	Minor	Minor revision of the specification. (Revision 1.04)	R	0b00100

12.12.10 Feature Capability Register (Feature)

Reset Value	0x00000106
Offset	0xF8070130
Access Type	Read Only



Bits	Field Name	Description	Access	Reset
0:22	Reserved	This field will read all 0’s	R	0x000000
23	ExtendedReg	This device does include the Enumeration Scratchpad, Error Handling, and Memory Base/Limit Upper registers.	R	0b1
24:27	Reserved	This field will read all 0’s	R	0b0000
28	ExtendedCTL	This device does not require CTL to be asserted for 50 µs during the initialization sequence after an LDTSTOP# disconnect.	R	0b0
29	CrcTest	Indicates whether this device supports the CRC test mode.	R	0b1
30	LdtStop	Indicates whether this device supports the link disconnect (LDTSTOP) protocol.	R	0b1
31	Isoc	Indicates whether this device supports isochronous flow control mode. (Not supported)	R	0b0

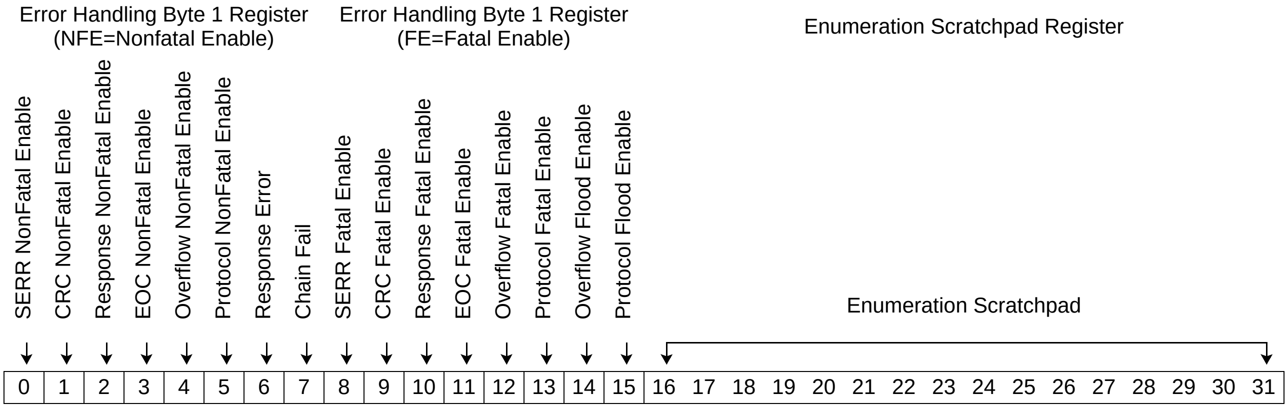
12.12.11 Error Handling/Enumeration Scratchpad Register (ErrCtrl/Enum)

This register contains routing enables from the various error log bits to the various error reporting mechanisms, as well as the Chain Fail and Response Error status bits. For definitions of the reporting mechanisms, see the HyperTransport Specification. Devices that do not check for one or more error conditions should hardwire the log and enable bits for those conditions to 0.

Note: Some registers have a specialized type of R/W

```
R/W (R/C) => a write of ’1’ will clear the bit
R/W (R/S) => a write of ’1’ will set the bit
```

Reset Value	0x00000000
Offset	0xF8070140
Access Type	Read/Write, Read Only

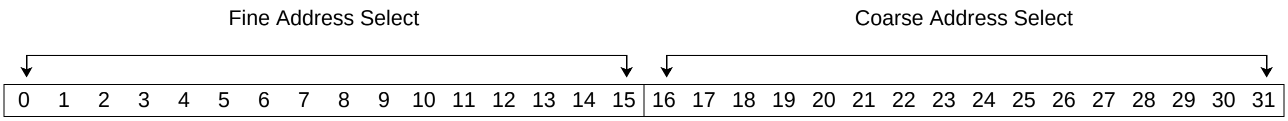


Bits	Field Name	Description	Access	Reset
0	SERR NonFatal Enable	This bit is only implemented for host interfaces. For slave interfaces, it is hardwired to 0.	R/W	0b0
1	CRC NonFatal Enable	If asserted, this bit will cause the nonfatal error interrupt whenever any of the CRC Error bits are asserted in (either of) the Link Control registers. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
2	Response NonFatal Enable	If asserted, this bit will cause the nonfatal error interrupt whenever the Response Error bit (9) is asserted. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
3	EOC NonFatal Enable	If asserted, this bit will cause the nonfatal error interrupt to be asserted whenever the End Of Chain Error bit is asserted in (one of) the Link Error registers, or the Inbound End Of Chain Error bit is set in the Host Command register. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
4	Overflow NonFatal Enable	If asserted, this bit will cause the nonfatal error interrupt to be asserted whenever the Overflow Error bit is asserted in (one of) the Link Error registers. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
5	Protocol NonFatal Enable	If asserted, this bit will cause the nonfatal error interrupt to be asserted whenever the Protocol Error bit is asserted in (one of) the Link Error registers. If the nonfatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
6	Response Error	This bit indicates that the given interface has received a response error.	R/W (R/C)	0b0

Bits	Field Name	Description	Access	Reset
7	Chain Fail	ChainFail - This bit indicates that the chain has gone down. It is set whenever a device detects sync flooding or a sync flood generating error. It is cleared by a cold or warm software link reset using the SecBusReset bit in the Bridge Control Register	R	0b0
8	SERR Fatal Enable	If asserted in a host, this bit will cause the fatal error interrupt whenever the System Error Detected bit is asserted in the Secondary Status register. If the fatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
9	CRC Fatal Enable	If asserted, this bit will cause the fatal error interrupt whenever any of the CRC Error bits are asserted in (either of) the Link Control registers. If the fatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
10	Response Fatal Enable	If asserted, this bit will cause the fatal error interrupt whenever the Response Error bit (9) is asserted. If the fatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
11	EOC Fatal Enable	If asserted, this bit will cause the fatal error interrupt to be asserted whenever the End Of Chain Error bit is asserted in (one of) the Link Error registers, or the Inbound End Of Chain Error bit is set in the Host Command register. If the fatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
12	Overflow Fatal Enable	If asserted, this bit will cause the fatal error interrupt to be asserted whenever the Overflow Error bit is asserted in (one of) the Link Error registers. If the fatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
13	Protocol Fatal Enable	If asserted, this bit will cause the fatal error interrupt to be asserted whenever the Protocol Error bit is asserted in (one of) the Link Error registers. If the fatal error interrupt is not implemented, this bit is hardwired to 0.	R/W	0b0
14	Overflow Flood Enable	If asserted, this bit will cause the link to be sync flooded whenever the Overflow Error bit is asserted in (one of) the Link Error registers.	R/W	0b0
15	Protocol Flood Enable	If asserted, this bit will cause the link to be sync flooded whenever the Protocol Error bit is asserted in (one of) the Link Error registers.	R/W	0b0
16:31	Enumeration Scratchpad	This register provides a scratchpad for enumeration software.	R/W	0x0000

12.12.12 HT Address Mask Register

Reset Value	0x38010000
Offset	0xF8070200
Access Type	Read/Write, Read Only



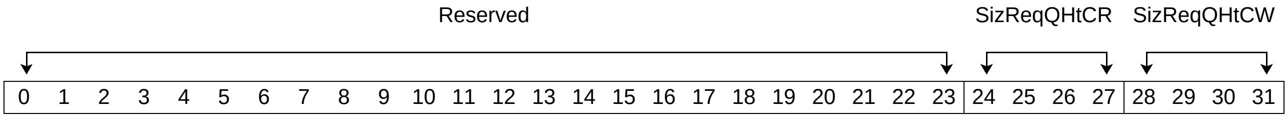
Bits	Field Name	Description	Access	Reset
0:15	Fine Address Select	This field defines which fine (16 MB) regions of memory are mapped from the PI to the HT Bus when PI address bits [0:3] = 0xF. If a given bit in this field is set, addresses on the PI bus with the matching bits 4:7 are passed from the PI bus to the HT Bus, while the same address values on the HT bus are not passed to the PI bus. If an address on the HT bus matches a HT region of memory, then the access is meant for a device on the HT bus and the access is forwarded to the correct device. If a given bit in this field is not set, addresses on the PI with the matching bits 4:7 are not passed from the PI to the HT bus while the same address values on the HT bus are passed to the PI or PCIe. The decode function sets: Bits 2:4, and 15 are read-only and are set to '1'. Bits 0 and 1 should be set to '0' because the PCIe owns that space. The results of setting these bits to '1'is unpredictable.	R [2:4,15] R/W [0:1, 5:14]	0x3801
16:31	Coarse Address Select	This field defines which coarse (256 MB) regions of memory are passed from the PI to the HT Bus. If a given bit in this field is set, Addresses on the PI with the matching address bits [0:3] are passed from the PI bus to the HT Bus, while the equivalent address values on the HT Bus, AD[31:28], are not passed to the PI bus. If a given bit in this field is not set, addresses on PI with the matching address bits [0:3] are not passed from the PI bus to the HT Bus, while the equivalent address values on the HT Bus, AD[31:28], are passed to the PI bus. Bit 31 of this register is disregarded (but please set it to 0) because its position, representing Coarse Space $F, is always used to decode into the Fine Address Select Field. Bits 16:23, and 31 are read only and are always set to initial value as shown. Bits 16:23 represent the first 2 GB of address space and must be set to '0' to avoid conflicts. Bit 31 represents 0xF8000000.	R [16:23,31] R/W [24:30]	0x0000

Note: 0xFxxx_xxxx decoding: See *Section 12.11.2.4 PCI Express 0 Address Mask Register* on page 556, for PCIe access to the 0xFxxx_xxxx space to make sure the two encodings do not conflict.

CPC945 does not allow software to clear the Address Space Select bits corresponding to its assigned HT bus numbers (bits 2,3,4, 15 are always set for HT).

12.12.13 HT1/PI Interface Control Register

Reset Value 0x00000088
Offset 0xF8070210
Access Type Read/Write, Read Only

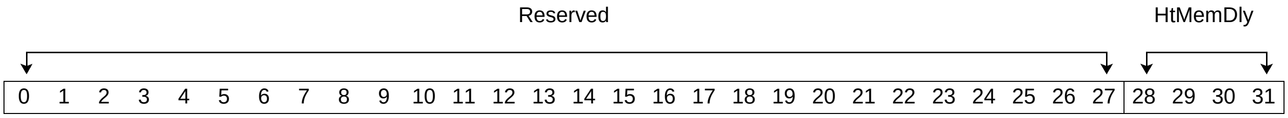


Bits	Field Name	Description	Access	Reset
0:23	Reserved	This field will read all 0's	R	0x000000
24:27	SizReqQHtCR	Size of PI read request queue. Minimum: 1, Maximum: 8, default: 8	R/W	0x8
28:31	SizReqQHtCW	Size of PI write request queue. Minimum: 1, Maximum: 8, default: 8	R/W	0x8

12.12.14 Memory Read Delay for Memory Read Data Interface (HtMemDly)

For upstream read requests from HyperTransport to memory, the read tag valid (RdTgV) signal is sent back from the PI first. After a fixed number of cycles later, the read data is then available on the RdDt bus for the number of consecutive cycles indicated by the RdDtCyc. This fixed number of cycles is specified by the MemDly parameter.

Reset Value 0x00000006
Offset 0xF8070220
Access Type Read/Write, Read Only

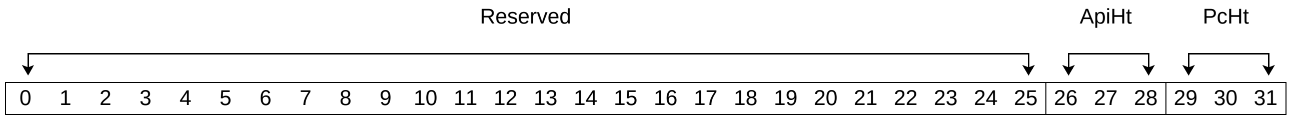


Bits	Field Name	Description	Access	Reset
0:27	Reserved	This field will read all 0's	R	0x000000
28:31	HtMemDly	Delay between RdTgV and RdDt. Min: 4, Max: 12, default: 6	R/W	0x6

12.12.15 Write TA delay for Write Data Interface (WrTADly)

For downstream write requests from PI or PCI Express to HyperTransport, it might take a number of cycles for the WrTA to get to the requestor that is providing the write data. This delay is called the WrTADly and is a configurable parameter. If it is necessary to fix timing problems late in the design due to the requestor and the target being physically far apart, a register can be added to all the signals on this interface. If this happens, all that needs to be changed is the parameter, WrTADly. Two independent parameters can be set for PI and PCI Express.

Reset Value	0x0000000A
Offset	0xF8070230
Access Type	Read/Write, Read Only

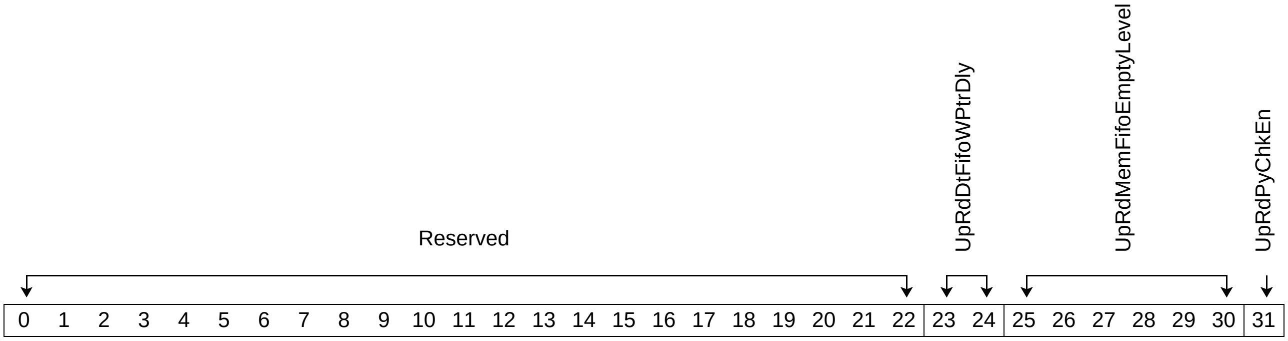


Bits	Field Name	Description	Access	Reset
0:25	Reserved	This field will read all 0’s	R	0x000000
26:28	ApiHt	PI to HT write TA delay. Min: 1, Max: 4, default: 1	R/W	0x1
29:31	PcHt	PCIe to HT write TA delay. Min: 1, Max: 4, default: 2	R/W	0x2

12.12.16 HTG Configuration (HTGCFG)

Configuration register to fine-tune HTG options.

Reset Value	0x00000089
Offset	0xF8070240
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:22	Reserved	This field will read all 0’s	R	0x000000
23:24	UpRdDtFifoWPtrDly	Number of delay cycles for the write pointer before passed to the read side. 3 cycles is the safest setting. Fast DDR2 clocks might use only 1 or 2.	R/W	0x1
25:30	UpRdMemFifoEmptyLevel	Minimum number of available entries in the memory data FIFO necessary before it is considered not full. The memory data FIFO has 32 total entries. The default setting is 4, meaning 4 entries need to be available for the FIFO to be not full. This setting can range in value between 0 and 32.	R/W	0x4
31	UpRdPyChkEn	Enable parity check on upstream memory read interface. default: 1 (enabled)	R/W	0b1

12.12.17 Bridge Control Register (BrCtrl)

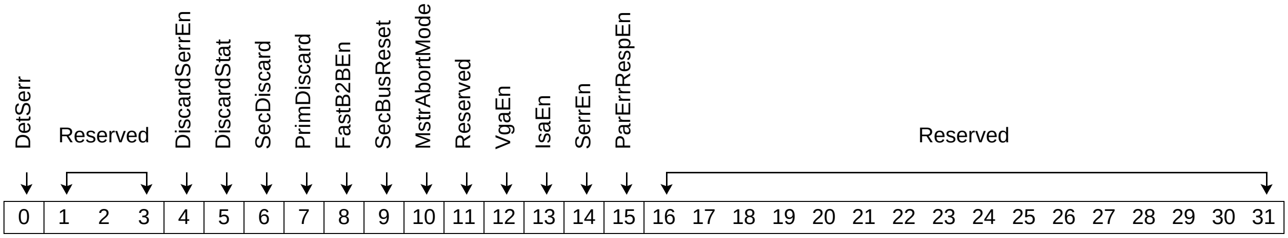
Note: Some registers have a specialized type of R/W.

```
R/W (R/C) => a write of ’1’ clears the bit
R/W (R/S) => a write of ’1’ sets the bit
```

Reset Value 0x00000000

Offset 0xF8070300

Access Type Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0	DetSerr	This bit reports the detection of a system error by the bridge on its secondary interface. This is indicated in HT by a HT device driving sync packets on the link after the completion of the initialization sequence. This bit can be cleared by writing a 1 to it. See discussion in *Section 6.9.4 HyperTransport SERR#* on page 125.	R/W (R/C)	0b0
1:3	Reserved	This field will read all 0’s.	R	0b000
4	DiscardSerrEn	Not meaningful for HT.	R	0b0
5	DiscardStat	Not meaningful for HT.	R	0b0
6	SecDiscard	Not meaningful for HT.	R	0b0
7	PrimDiscard	Not meaningful for HT.	R	0b0
8	FastB2BEn	Not meaningful for HT.	R	0b0
9	SecBusReset	If a 1 is written, hardware performs a reset sequence on the secondary bus. Clearing the bit brings the secondary bus out of reset. The setting of cold vs. warm link reset is controlled by the WarmReset bit in the Command/Pointer/Capability ID Description Register (*Section 12.12.7 Command/Pointer/Capability ID Register (HTCapability00)*). Warning: Though it is ok for reset to assert in the midst of active traffic, the busses must be quiesced before coming back out of reset. If the busses are not quiesced, data might be stuck in the data pipe upon awakening. Leaving write data in the data pipe will cause all subsequent write data to be shifted. After the reset is asserted, no more downstream requests should be issued until the bus is taken out of reset.	R/W	0b0
10	MstrAbortMode	This bit controls the action taken when a transaction takes a Master Abort on the destination bus. If this bit is clear, writes are allowed to complete normally on the source bus, and reads have all 1's returned. If it is set, the Master Abort will be treated as an error, returning a Target Abort Response (indicated on HyperTransport by a set error bit) for nonposted requests, and causing system error assertion (indicated by driving sync packets on the primary interface if enabled, or generating an NMI packet if enabled) for posted requests.	R/W	0b0
11	Reserved	This field will read all 0’s.	R	0b0
12	VgaEn	Not supported by CPC945.	R	0b0

Bits	Field Name	Description	Access	Reset
13	IsaEn	Not supported by CPC945.	R	0b0
14	SerrEn	Not implemented If implemented, this bit controls forwarding of system errors from the secondary interface to the primary interface. If it is set, system errors (indicated on HT by sync packets after the initialization sequence has completed) will propagate, assuming that the SERR enable bit is set for the primary interface in the command register. See discussion in *Section 6.9.4 HyperTransport SERR#* on page 125.	R	0b0
15	ParErrRespEn	Parity error response enable. This bit is reserved because HT does not have parity errors.	R	0b0
16:31	Reserved	This field will read all 0’s.	R	0x0000

12.12.18 TxCtl/Rx Data Buffer Allocation Register (TxCtl/RxDataBufAlloc)

This register contains the default settings for HyperTransport buffer reservations and allows users to tune or adjust the buffer release policy on the link. The TxCtl/BufRelSpace bits [4:7] are used to control how often a "buffer free" message is sent out on the HyperTransport Link.

The TxCtl/Rx Data Buffer Allocation Register controls the allocation of the 8 receive data buffers in each link among the 3 virtual channels, to allow performance tuning. The “Need” fields indicate the minimum allocation at all times to each channel, minus one. That is, a value of 0 indicates a permanent minimum allocation of 1. The “Want” fields indicate how many buffers the allocator should try to have released and outstanding to each channel at all times, minus 1. The total number of buffers “needed” must be less than or equal to the total number (8) of data buffers available. If less, it gives the allocator more flexibility to hand out buffers dynamically. In the default (reset) case, there are 2 buffers in each category.

Reset Value	0x04301106
Offset	0xF8070310
Access Type	Read/Write, Read Only

Reserved	TxCtl/ BufRelSpace	WantPReq	WantNpReq	WantResp	NeedPReq	NeedNpReq	NeedResp
0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31

Bits	Field Name	Description	Access	Reset
0:3	Reserved	This field will read all 0’s.	R	0b0000
4:7	TxCtl/ BufRelSpace	This register controls throttling of buffer release messages on a busy bus. If the bus is idle, buffer releases will always get issued immediately. When the bus is busy, we will force buffer release messages into the packet stream. The field gives the minimum number of cells that must be allowed to pass since a buffer release before another one can be forced in, to prevent them from absorbing too much bandwidth.	R/W	0b0100
8:11	WantPReq	Number of buffers minus 1 to try to keep released in the posted request channel.	R/W	0b0011
12:15	WantNpReq	Number of buffers minus 1 to try to keep released in the nonposted request channel.	R/W	0b0000
16:19	WantResp	Number of buffers minus 1 to try to keep released in the response channel.	R/W	0b0001
20:23	NeedPReq	Minimum data buffer allocation to the posted request channel.	R/W	0b0001
24:27	NeedNpReq	Minimum data buffer allocation to the nonposted request channel.	R/W	0b0000
28:31	NeedResp	Minimum data buffer allocation to the response channel.	R/W	0b0110

12.12.19 Maximum Transmit Buffer Counters Register (TxBufCountMax)

Maximum threshold for the transmit buffer counters If buffer releases are received in a particular channel which exceed the threshold for that channel, the extras are discarded. This allows a general way to throttle the traffic on the link. The counter value can be lowered in a running system, but the system must go through reset to make an increase take effect.

Reset Value	0x00FFFFFF
Offset	0xF8070340
Access Type	Read/Write, Read Only

Reserved								RData				RCmd				NpData				NpCmd				PData				PCmd			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bits	Field Name	Description	Access	Reset
0:7	Reserved	This field will read all 0’s	R	0x00
8:11	RData	Response data buffer threshold	R/W	0xF
12:15	RCmd	Response command buffer threshold	R/W	0xF
16:19	NpData	Nonposted data buffer threshold	R/W	0xF
20:23	NpCmd	Nonposted command buffer threshold	R/W	0xF
24:27	PData	Posted data buffer threshold	R/W	0xF
28:31	PCmd	Posted command buffer threshold	R/W	0xF

12.12.20 Diagnostic CRC Registers (DiagRxCrc)

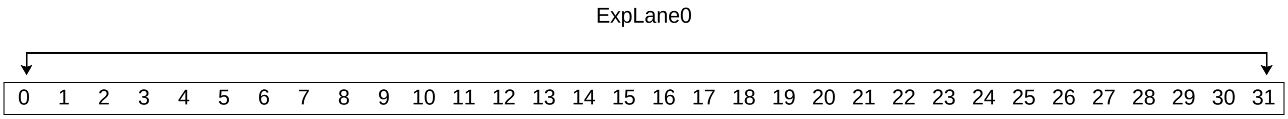
The diagnostic registers record the value of the expected and received CRC value whenever a CRC error is detected. Once the error is latched, further CRC errors do not over-write the initial error value. These registers are not reset by warm or cold reset so that a system can reset the link and still retrieve the error state from the previous run. Note that the CRC is 64 bits long so it is stored within two 32-bit registers.

0xF8070350 DiagRxCrcExpLane0: Diagnostic Expected CRC Register (Bits [0:31] of the CRC)
0xF8070360 DiagRxCrcRcvLane0: Diagnostic Expected CRC Description (Bits [0:31] of the CRC)
0xF8070370 DiagRxCrcExpLane1: Diagnostic Expected CRC Register (Bits [32:63] of the CRC)
0xF8070380 DiagRxCrcRcvLane1: Diagnostic Received CRC Register (Bits [32:63] of the CRC)

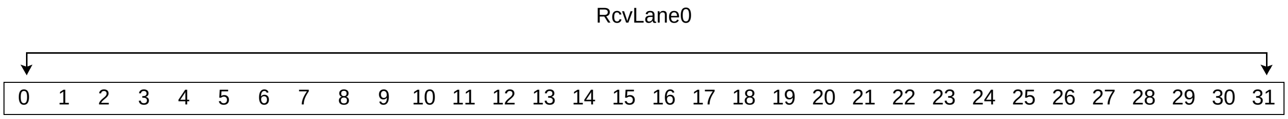
All registers have the same layout as 0xF8070350 (*Table 12-36 DiagRxCrcExpLane0: Diagnostic Expected CRC Description (Bits [0:31] of the CRC)*) so the next three register descriptions have been omitted.

Reset Value	0x00000000
Offset	0xF8070350 (DiagRxCrcExpLane0) 0xF8070360 (DiagRxCrcRcvLane0) 0xF8070370 (DiagRxCrcExpLane1) 0xF8070380 (DiagRxCrcRcvLane1)
Access Type	Read Only

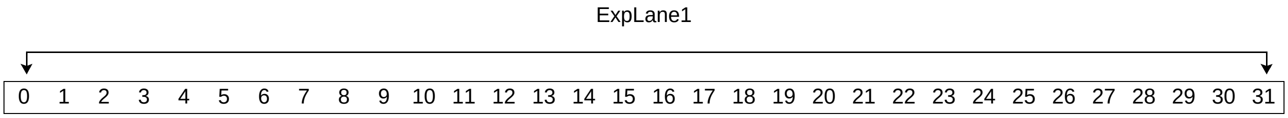
DiagRxCrcExpLane0



DiagRxCrcRcvLane0



DiagRxCrcExpLane1



DiagRxCrcRcvLane1

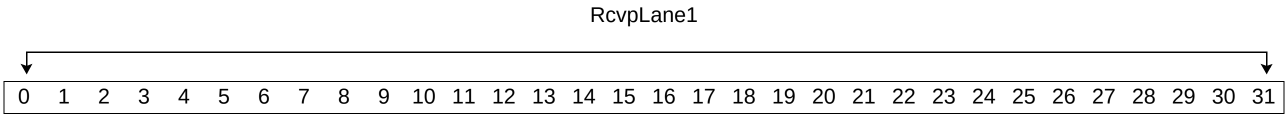


Table 12-36.DiagRxCrcExpLane0: Diagnostic Expected CRC Description (Bits [0:31] of the CRC).

Bits	Field Name	Description	Access	Reset
0:31	ExpLane0	Expected CRC value used for diagnostic purposes (lower).	R	0x00000000

12.12.21 Receive and Transmit Synchronization FIFOs (Sri[Rx/Tx]Numerator)

The receive and transmit synchronization FIFOs require different parameters based on HT data rate and core clock rate. The parameters are normally derived from an internal look up table (LUT), but can be overridden using the values in the Serial Rom Interface (SRI) CSR range for nonstandard frequency combinations, or to tweak margin/offset values to improve latency through the FIFO.

```
0xF8070390 SriRxNumeratorLower
0xF80703A0 SriRxNumeratorUpper
0xF80703B0 SriTxNumeratorLower
0xF80703C0 SriTxNumeratorUpper
0xF80703D0 SriOveride (SriRxNumeratorLower: Receive Synchronization FIFO Register
Description (Bits [0:31]))
```

Here is an example of how the numerator and denominator are calculated for the transmit and receive FIFOs.

```
Clock Ratios
F1 = HT Clock Rate
F2 = Application Clock Rate (ht_clk)

Cad/Ctl  F1 is DDR2 => 2Edges * 8Bits
Cad/Ctl  F2 is 1Edge *64Bits

Ratio = 2*F1/8*F2 => F1/4*F2
Ratio = F1/4*F2, where the ratio must be <= 1
```

Example:

```
HT Data Rate = 1200 MHz
Application Clock Rate = 200 MHz

F1/4*F2 = 600MHz/4*200MHz = ?
```

This indicates that data should be “pushed” (tx) or ”popped” (rx) into/from the FIFO at a rate of ?. To do this the numerator could be set as follows:

```
Numerator = 64’h000000000000000e
```

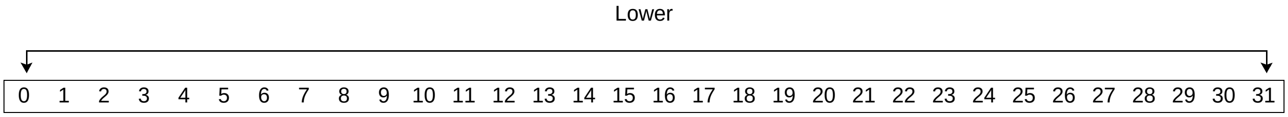
With denominator set to the denominator of the ratio:

```
Denominator = 7’b0000100
```

This causes the push/pop signals to come from the lower nibble of the numerator field, of which, 3 out of the 4 bits are set.

12.12.21.1 SriRxNumeratorLower: Receive Synchronization FIFO Register Description (Bits [0:31])

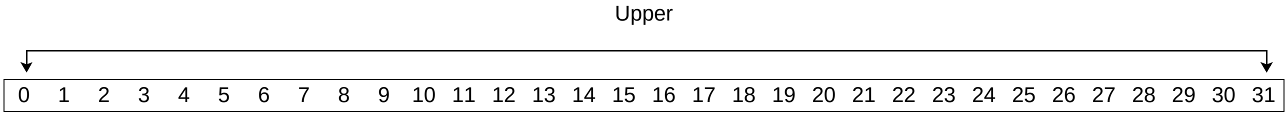
Reset Value	0x00000000
Offset	0xF8070390 (SriRxNumeratorLower) 0xF80703A0 (SriRxNumeratorUpper) 0xF80703B0 (SriTxNumeratorLower) 0xF80703C0 (SriTxNumeratorUpper)
Access Type	Read/Write



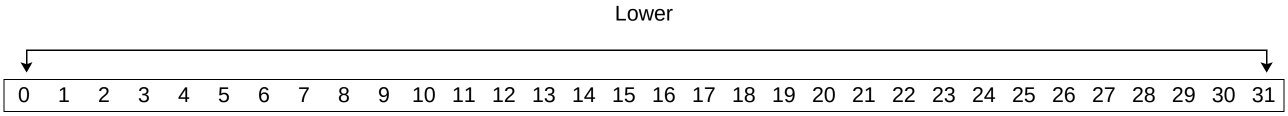
Bits	Field Name	Description	Access	Reset
0:31	Lower	Receive Synchronization FIFO Numerator Value	R/W	0x00000000

All numerator registers have the same layout as 0xF8070390 (*SriRxNumeratorUpper: Receive Synchronization FIFO Register (Bits [32:63])*) so the next three register descriptions have been omitted.

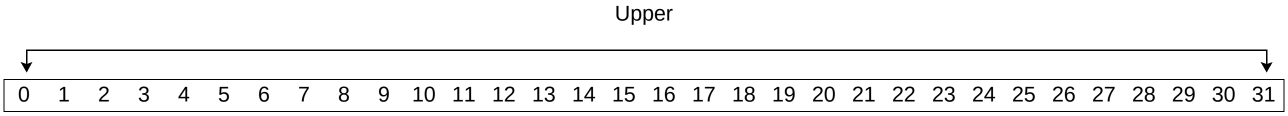
SriRxNumeratorUpper: Receive Synchronization FIFO Register (Bits [32:63])



SriTxNumeratorLower: Transmit Synchronization FIFO Register (Bits [0:31])



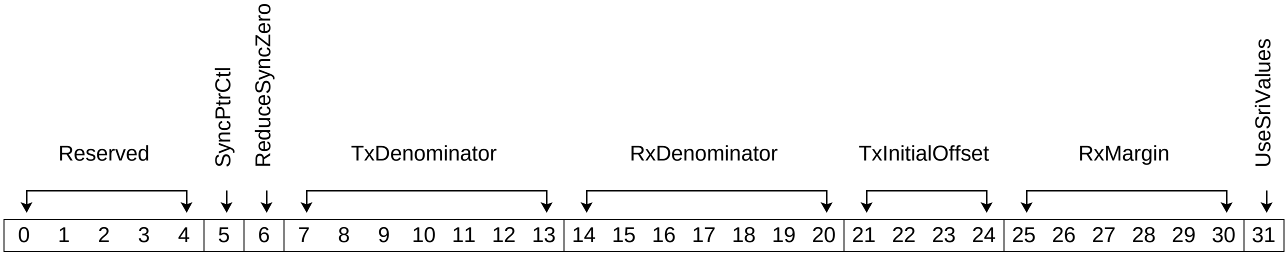
SriTxNumeratorUpper: Transmit Synchronization FIFO Register (Bits [32:63])



12.12.22 Receive and Transmit Synchronization Override Values Register (SriOveride)

See *Section 12.12.21* on page 641.

Reset Value	0x04000000
Offset	0xF80703D0
Access Type	Read/Write, Read Only

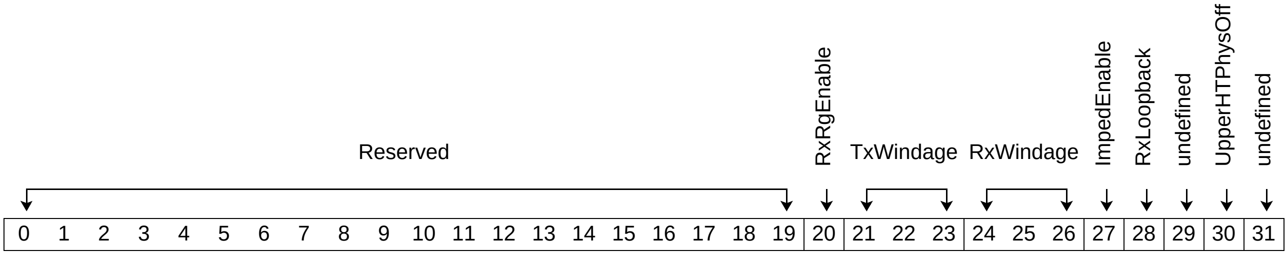


Bits	Field Name	Description	Access	Reset
0:4	Reserved	This field will read all 0’s	R	0b00000
5	SyncPtrCtl	CPC945 only supports synchronized receive logic.	R	0b1
6	ReduceSyncZero	Reserved.	R/W	0b0
7:13	TxDenominator	The csrSriLdtTxDenominator field is used to index into the ldtTxNumerator field, creating the loop point for a circular shift register.	R/W	0b0000000
14:20	RxDenominator	The csrSriLdtRxDenominator field is used to index into the csrSriLdtRxNumerator field, creating the loop point for a circular shift register.	R/W	0b0000000
21:24	TxInitialOffset	The csrSriLdtTxInitialOffset[2:0] signal is used to adjust the pointers at the de-assertion of reset.	R/W	0x0
25:30	RxMargin	Receive Margin. The Rx margin must include the reset synchronization time as well as the desired margin between the pointers during operation. The receive load and unload pointers are reset by the ldtRxSync signal which indicates that a synchronization packet as been seen on the link. LdtRxSync is synchronized to the application clock domain by a four stage synchronizer.	R/W	0b000000
31	UseSriValues	Use SRI Values. When set the LUT is bypassed and the values in the SriLdt CSR range are used.	R/W	0b0

12.12.23 Control bits for the HT PHY (HtPHYCtl)

These pins are used to fine-tune the HT PHY module. Each version of PHY can only use a portion of these pins.

Reset Value	0x00000B70
Offset	0xF80703E0
Access Type	Read/Write, Read Only



Bits	Field Name	Description	Access	Reset
0:19	Reserved	This field will read all 0's	R	0b00000
20	RxRgEnable	Disables the HyperTransport Receive PHYs test receivers for lower power if desired.	R/W	0b1
21:23	TxWindage	Transmit PHY windage pins settings. These settings change the relative timing of the clock and data. The exact range of the windage has not been decided, but the total range would be in the range of +/- 75ps. So a value of 000 would move the clock early by about 75 ps, while a value of 111 would move it later by about 75 ps. A count of 011 will be the neutral position.	R/W	0b011
24:26	RxWindage	Receive PHY windage pins settings. These settings change the relative timing of the clock and data. The exact range of the windage has not been decided, but the total range would be in the range of +/- 75ps. So a value of 000 would move the clock early by about 75 ps, while a value of 111 would move it later by about 75 ps. A count of 011 will be the neutral position.	R/W	0b011
27	ImpedEnable	PHY Impedance Enable	R/W	0b1
28	RxLoopback	Enable PHY Loopback	R/W	0b0
29	undefined	Retains state, but unconnected	R/W	0b0
30	UpperHTPhysOff	Setting this bit active (0b1) forces the most significant byte (MSB) HT PHYs to power down.	R/W	0b0
31	undefined	Retains state, but unconnected	R/W	0b0

13. References

HyperTransport Technology Consortium, “HyperTransport I/O Link Specification”, HTC2002104-0005-0001, Revision 1.05, January 2003.

Philips Semiconductors I^{2}C-Bus Specification Version 2.1, April 1995.

AMD, “HyperTransport Technology Electrical Specification”, 23890, Revision 1.08, August 2001.

AMD, “LDT I/O Link Protocol Specification”, Revision 1.03, 10/10/01.

Francis Chan and Steven Hsu, “(Cu-11) Hot Plug and Supply Sequencing”, IBM, 8/27/01.

IBM, “PCI Express Configuration Space”, SA15-5765-00 Preliminary Copy, IBM. 7/25/03.

IBM, “Phase-Locked Loop, ASIC Products Cu-11”, IBM, 01/20/2003.

IBM, “IBM PowerPC 970MP RISC Microprocessor User’s Manual", IBM. 7/31/06.

IBM, “Standard Cell Nontest I/Os”, IBM, 8/28/2002.

JEDEC STANDARD, “DDR2 SDRAM Specification”, JEDEC Standard JESD79-2B, January 2005.

JEDEC STANDARD, “Stub Series Terminated Logic for 1.8V (SSTL_18)," JEDEC Standard JESD8-15A, September 2003.

PCI-SIG, "PCI Express Base Specification Revision 1.0a", PCI Express, April 15, 2003.

PCI-SIG, "PCI Express Card Electromechanical Specification Revision 1.0a", PCI Express, April 15, 2003.

Sandon, Peter, “Power Management Design Changes for PPC970-2”, November 5, 2002.“PCI Express Configuration Space”, SA15-5765-00 Preliminary Copy, IBM. 7/25/03.

"The I^{2}C-Bus Specification," Version 2.1, January 2000.

“The Open Programmable Interrupt Controller (PIC) Register Interface Specification, Revision 1.2,” Issue Date: October 1995, Issued Jointly by Advanced Micro Devices and Cyrix Corporation.

14. Glossary

AD	address and data
ADI	address/data in
ADO	address/data out
AGP	accelerated graphics port
AL	Application Layer. The functional layer that interfaces between the application logic and the PCIe protocol stack.
API	Advanced Processor Interface
BA	bank address
BCM	balance coding method
BIST	built-in self test
BrCtrl	Bridge Control Register
BUSCONF	Processor-Interface-Bus Configuration Register
CA	column address
CAS	column address strobe
CBGA	ceramic ball grid array
CDR	Configuration Data Register
CFE	CRC force error
CFMR	Chip Fault Mask Register
CK	DDR2 DRAM clock
CKE	clock enable
CptEn	capture enable bit
CPU/HT/PCI	processor, HyperTransport, and PCI buses
DART	DMA address relocation table
DDR2	Double data rate 2. A memory interface technology that provides a data transfer rate that is twice the clock rate. DDR2 is the latest generation of DDR technology that provides a number of evolutionary improvements over DDR. Because of its potential for increased operating frequency in high capacity systems, DDR2 provides higher bandwidth than DDR. Despite its higher frequency, DDR2 operates with lower I/O and core voltages (1.8 instead of 2.5 volts) providing significant power savings.
DERR	data error
DIMM	dual in-line memory module
DLL	delay-locked loop

DMA	Direct Memory Access
DQSDataDelAdj	DQS Data Delay Adjustment Registers
DQSDelAdj	DQS Delay Adjustment Register
DRAM	dynamic random-access memory
DWord (or DW)	Double Word (4 bytes of data)
ECC	error correction code
eciwx	external control in word indexed
ecowx	external control out word indexed
EDC	error detection and correction
EEPROM	electrically erasable programmable ROM
EI	Elastic Interface. Former name for what is now called the PI, or Processor Interconnect. (See PI definition)
EIEIO	enforce in-order execution of I/O
EMI	electromagnetic interference
EMRSRegCntl	extended mode register set
En	trigger enable bit
Endpoint	A downstream component that is not a Switch
Enum	Enumeration Scratch Pad
EOC	end of chain
EOI	end of interrupt
ErrCtrl	Error Handling Register
EST	elastic interface shorts test
FF	full frequency
FIAP	fast initial alignment procedure
FSM	finite state machine
GART	graphics address relocation table
HF	half frequency
HSSide	high-single-side
HT	HyperTransport
I/O	input and output
I^2C	inter-integrated circuit

IAP	initial alignment procedure
IDSEL	ID select
IER	I^{2}C Interrupt Enable Register
IKill	instruction kill
IPI	interprocessor interrupt
IRQ	interrupt request
ISCA	I^{2}C data bus signal
ISCL	I^{2}C clock signal
ISR	I^{2}C Interrupt Status Register
LPN	logical page number
LRU	least recently used
LSSide	low-single-side
LUT	look-up table
Mbps	megabits per second
MCCR	Memory Check Control Register
MemProgCntl	Memory Programming Control Register
MHz	mega hertz
MODE	I^{2}C Mode Register
MPIC	multiprocessor interrupt controller
MRSRegCntl	mode register set
MTps	million transfers per second
NC	noncoherent
NMI	nonmaskable interrupt
NOP	no operation
North Bridge	A generic term for the CPC945 as it connects to higher speed interfaces such as DRAM or PCI Express.
NXA	nonexistent address
OVF	overflow
PAAM	previous adjacent address match
P-Bit	power-up bit
PCHG	precharge

PCI	peripheral component interconnect
PCR	Power Control Register
PI	Processor Interconnect. The interface provided by the CPC945 Bridge and Memory Controller's processor interface bus that connects the system microprocessors to memory and I/O devices. The interface consists of two parts: The slave, which is used by the processor to access memory and I/O The master, which is used to pass coherency information between the I/O devices and the processor.
PLL	phase-locked loop
PMU	power management unit
PORTSEL	port select
PPN	physical page number
NORTH_BRIDGE_RESET_L	power-up reset
QDWord (or QDW)	Quad double word (16 bytes of data)
QF	quarter frequency
QWord (or QW)	Quad word (8 bytes of data)
R/C	read and clear
R/S	read and set
R/W	read or write
RA	row address
RAM	random-access memory
RAS	row address strobe
RDT	random data test
REFWT	waiting to reflect
RIAP	receiver initial alignment procedure
RMW	read-modify-write
RTC	real-time clock
RWITM	read with intent to modify
SBA	side-band addressing
SBus	service bus
SCL	serial clock
SDA	serial data
SDRAM	synchronous DRAM

$\overline{\textbf{SERR}}$	system error
SIOInt	MPIC Interrupt Source 0 Vector/Priority Register
SleepReq	sleep request
SNOOPACC	snoop accumulated response delay
SNOOPLAT	snoop latency
southbridge	A generic term for a device that connects the CPC945 to slower peripheral interfaces. The CPC945's HyperTransport tunnel connects to the southbridge.
SPD	serial presence detect
SPU	service processing unit
SR	snoop response
SRI	snoop response in
TA	transfer acknowledge
$\mathbf{t_{CL}}$	DDR2 CAS latency
TEA	transfer error acknowledge
TH	transfer handshake
THI	transfer handshake in
TLB	translation lookaside buffer
TLBIE	translation lookaside buffer invalidate entry
TLBSYNC	translation lookaside buffer synchronization instruction
TLP	transaction layer packet
$\mathbf{t_{RAS}}$	SDRAM row address strobe
$\mathbf{t_{RCD}}$	RAS-to-CAS delay
$\mathbf{t_{RFC}}$	refresh cycle time
$\mathbf{t_{RP}}$	bank precharge time
TSC	time-stamp counter
TT	transfer type
$\mathbf{t_{WR}}$	write recovery time
$\mathbf{t_{WTR}}$	write-to-read command delay
TXO	transmitter off
VGA	video graphics array
WDB	write data buffer

WIAP transmit initial alignment procedure

WIMGRP The 4 bits that control the processor's accesses to cache and main storage. “W” stands for write through, “I” for cache inhibit, “M” for memory coherence, “G” for guarded storage, “R” for rerunning, and “P” for priority.

Revision Log

Each release of this document supersedes all previously released versions. The revision log lists all significant changes made to the document since its initial release. In the rest of the document, change bars in the margin indicate that the adjacent text was modified from the previous release of this document.

Revision Date	Details of Modification
February 1, 2008	A15-6010-02a • Revised EMRSRegCntl Register (bit 20) noting that RDQS is not supported (see *Section 12.10.4 Mode Register Set (MRS) Register (MRSRegCntl) and Extended Mode Register Set Register (EMRSRegCntl)* on page 457). • Removed DD level information in the description for HtPHYCtl[30] (see *Section 12.12.23 Control bits for the HT PHY (HtPHYCtl)* on page 644).
December 21, 2007	A15-6010-02 • Changed PCIE to PCIe, PCI-Express to PCI Express throughout. • Changed DDR to DDR2. • MiscVernierC0-C3 changed to RstLdEnVerniersC0-C3 throughout. • Added reference to 970MP to *About This Manual* on page 25. • Added reference to 970FX to *Section 1.1 Introduction* on page 27. • Added reference to 970FX to *Section 1.2 Features* on page 27. • Updated *Figure 1-1 CPC945 Bridge and Memory Controller Block Diagram* on page 28 to include 970MP and 970FX processors. • Revised description in *Section 2.2.1.5 Power Manager Clock* on page 32. • Fixed page layout problem in *Section 3 CPC945 Core Interface (API Interface)* on page 39. • Added *Section 4.1 Processor Interface Alignment Procedure* on page 78 and subsections. • Removed *Section 4.3 Interface Alignment Procedure*. • Added *Section 6.2 Initializing HyperTransport Core in the CPC945* on page 115 and subsections. • Added *Section 7.7.2 Memory Controller Bring-up Summary* on page 136. • Clarified *Section 7.19.1 ECC Introduction* on page 191. • Revised *Section 7.25.7.2 ODT Timing* on page 226. • Edited *Section 10.1 Introduction* on page 297 • Replaced *Section 10.3 Reset* with *Section 10.3 Power-On Reset* on page 298. • Edited *Table 12-5 PLL1 Clock Settings* on page 347 highlighting reset default value. • Edited *Table 12-6 PLL2 DDR2 Core Speed* on page 349 highlighting reset default value. • Edited register description in *Section 12.5.7 PLL2 Control Register* on page 350. • Edited *Table 12-8 PLL4 Control Register Default Values* on page 355 highlighting reset default value. • Corrected description for 0xF8070200 Register bits 0:15 in *Section 12.12.12 HT Address Mask Register* on page 632. • Corrected HTCapability04[27] bit name to LinkFail (see *Section 12.12.8 Link Config/Link Control Register (HTCapability04)* on page 626).

Revision Date	Details of Modification
October 31, 2006	A15-6010-01 • Rearranged PCI Express Registers. • Replaced *Table 12-11 PLL1 Clock Settings* and *Table 12-12 PLL1 Clock Settings 2 of 2, with combined table (Table 12-5 PLL1 Clock Settings).* • Fixed a problem (broken line) in *Figure 7-3 Memory Controller Internals*. • Added *Figure 7-4 Chip Select and Interleave Mode Addressing*, *Section 7.14.10 DIMM Configuration Algorithm* and *Section 7.14.11 DIMM Configuration Examples* for clarity on programming the DIMM Configuration registers. • Added *Section 7.15.3 Timing Parameter Examples* for clarity on programming the RAS Timer 0/1 and CAS Timer 0/1 registers. • Added a paragraph to *Section 7.22.3 Relationship to Board Wiring* which notes that systems with x4 DIMMs should have good wiring length matching for nibble pairs because the Calibration logic only measures results for the even nibble of a pair. • Corrected 2 typos in the equation of WrExtMuxDly in *Section 7.30.1 ExtMux Coarse Timing*. • WrtExtMuxDly = WrtExtMuxDly - 2 becomes WrtExtMuxDly = RdExtMuxDly -2. • CL + RegDIMM - 1 becomes CL + RegDIMM - 3 • Added missing section *Section 7.31 DDR2 PHY Calibration Logic*. • *Section 12.10.1 Memory Timing Parameter Registers* added a bullet which notes that RRMux, WRMux, WWWMux and RWMux have a minimum value of ‘1,’ which might need to be increased if external data muxes are use. Also added a bullet referencing the new *Section 7.15 Timing Parameters* and the new *Section 7.15.3 Timing Parameter Examples* for additional information on programming the Memory Timing Parameter Registers. • Fixed an error in section *Section 12.10.1.4 CAS Command Timer1 Register (CASTimer1)*, for bits 25:29, TiRtWSy. The equation is changed from BL/2 + RWMux - 2 to BL/2 + RWMux - 1. • *Section 12.10.7 DIMM Configuration Registers* added cross references to new *Section 7.14.10 DIMM Configuration Algorithm* and *Section 7.14.11 DIMM Configuration Examples* for additional information on programming the DIMM Configuration registers. • Several changes to the Calibration registers, for clarity and to match the new *Section 7.31 DDR2 PHY Calibration Logic*. • Renamed the four "MeasStatusCx" registers to "CalCntlDlyMeasCx". • Renamed the four "CalCx" registers to "CalRsltCx". • Moved the description of the "CalCntlDlyMeasCx" registers from after the description of the IOPadCntl register to just before the description of the CalConfx registers. This groups all of the Calibration registers together. • For Calibration Mode Select = 11 (CalConf0 register, bits 27:28), changed this mode from "Unload Pointer" to "Unload Clock Offset" because the unload monitor does not actually calibrate the Unload Pointer. Instead it has its own Unlock Clock Offset. • Numerous edits to descriptive text for clarity and to match *Section 7.31 DDR2 PHY Calibration Logic*. • Fixed CalConf0 bits 0:3 typo: changed "FieldCalibrationPassResult" to "CalibrationPassResult." • *CPC945 Revision Register* - added DD2.0 to RevNum, changed reset value for bits 24:27 to 0x4 • *APIPhy I/O Control Register (APIPhyIOCTRL)* bit 36 changed from 0b0 to 0b1. • Correction to *Figure 11-5 CPU Power Manager (2 of 2)*. • *Section 12.7.4.13 APIPhy PMR I/O Control Register (APIPhyPMRIOCTRL)*-changed 90 ohms to 55 ohms (bits [4, 61]. • *Section 12.10.22.2 MEAR1 Register (MEAR1)* bits [29:31] changed from “unused” to RA[0:2]. • Clarification of paragraph in *Section 5.2.2 PI Versus TL Data Formatting Differences*. • Clarified *Section 12.12.18 TxCtl/Rx Data Buffer Allocation Register (TxCtl/RxDataBufAlloc)*. • Correction to *Table 12-34 HyperTransport Registers* for definition of TxCtl/RxDataBufAlloc Register.
August 15, 2006	A15-6010-00 - General release.