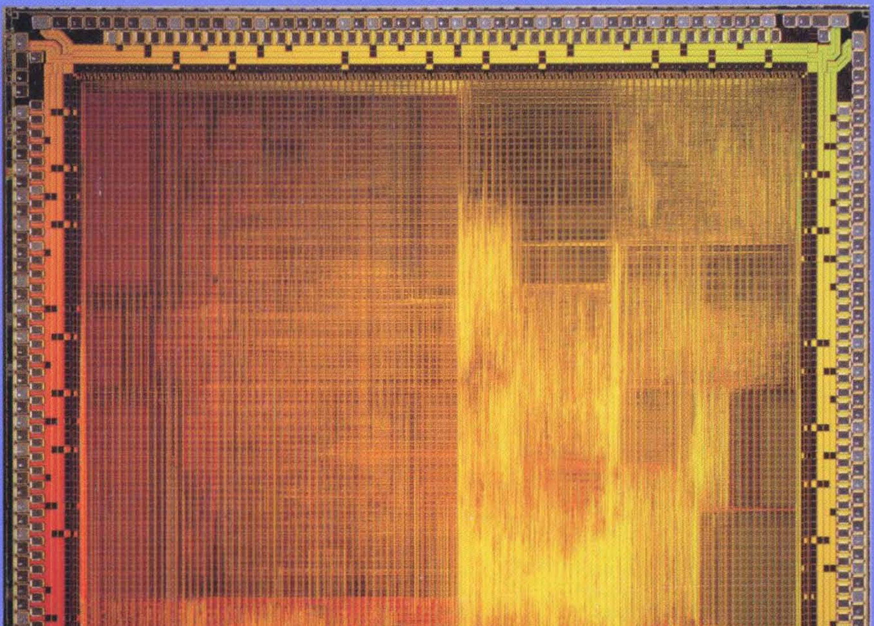
PowerPC 405GP Embedded Processor

User's Manual, Volume 1

Preliminary



IBM

PowerPC

IBM

PowerPC 405GP Embedded Processor User's Manual

Preliminary **Volume 1**

PowerPC™

GK10-3118-03

Sixth Preliminary Edition (June 2000)

This edition of *IBM PowerPC 405GP Embedded Processor User's Manual* applies to the IBM PowerPC^{TM} 405GP 32-bit embedded processor, until otherwise indicated in new versions or application notes.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address technical queries about this product to ppcsupp@us.ibm.com

Address comments about this publication to:

IBM Corporation
Department YM5A
P.O. Box 12195
Research Triangle Park, NC 27709

4 3 2 1

Patents and Trademarks

Contents

Figures

Tables

About This Book

This user's manual provides the architectural overview, programming model, and detailed information about the registers, the instruction set, and operations of the IBM PowerPC 405GP (PPC405GP) 32-bit RISC embedded processor.

The PPC405GP RISC embedded processor features:

- PowerPC Architecture™
- Single-cycle execution for most instructions
- Instruction cache unit and data cache unit
- Support for Little Endian operation
- Interrupt interface for one critical and one non-critical interrupt signal
- JTAG interface
- Extensive development tool support

Who Should Use This Book

This book is for system hardware and software developers, and for application developers who need to understand the PPC405GP. The audience should understand embedded processor design, embedded system design, operating systems, RISC processing, and design for testability.

How to Use This Book

This book describes the PPC405GP device architecture, programming model, external interfaces, internal registers, and instruction set. This book contains the following chapters, arranged in parts:

This book contains the following appendixes:

To help readers find material in these chapters, the book contains:

Conventions

The following is a list of notational conventions frequently used in this manual.

$\overline{\text{ActiveLow}}$	An overbar indicates an active-low signal.
n	A decimal number
0x*n*	A hexadecimal number
0b*n*	A binary number
=	Assignment
$\wedge$	AND logical operator
$\neg$	NOT logical operator
$\vee$	OR logical operator
$\oplus$	Exclusive-OR (XOR) logical operator
+	Twos complement addition
–	Twos complement subtraction, unary minus
$\times$	Multiplication

$\div$	Division yielding a quotient
%	Remainder of an integer division; (33 % 32) = 1.
||	Concatenation
$=, \neq$	Equal, not equal relations
$<, >$	Signed comparison relations
$\overset{u}{<}, \overset{u}{>}$	Unsigned comparison relations
if...then...else...	Conditional execution; if *condition* then *a* else *b*, where *a* and *b* represent one or more pseudocode statements. Indenting indicates the ranges of *a* and *b*. If *b* is null, the else does not appear.
do	Do loop. "to" and "by" clauses specify incrementing an iteration variable; "while" and "until" clauses specify terminating conditions. Indenting indicates the scope of a loop.
leave	Leave innermost do loop or do loop specified in a leave statement.
FLD	An instruction or register field
FLD_b	A bit in a named instruction or register field
$FLD_{b:b}$	A range of bits in a named instruction or register field
$FLD_{b,b,\ldots}$	A list of bits, by number or name, in a named instruction or register field
REG_b	A bit in a named register
$REG_{b:b}$	A range of bits in a named register
$REG_{b,b,\ldots}$	A list of bits, by number or name, in a named register
REG[FLD]	A field in a named register
REG[FLD, FLD$_{\ldots}$]	A list of fields in a named register
REG[FLD:FLD]	A range of fields in a named register
GPR(r)	General Purpose Register (GPR) r, where $0 \leq r \leq 31$.
(GPR(r))	The contents of GPR r, where $0 \leq r \leq 31$.
DCR(DCRN)	A Device Control Register (DCR) specified by the DCRF field in an **mfdcr** or **mtdcr** instruction
SPR(SPRN)	An SPR specified by the SPRF field in an **mfspr** or **mtspr** instruction
TBR(TBRN)	A Time Base Register (TBR) specified by the TBRF field in an **mftb** instruction
GPRs	RA, RB,$_{\ldots}$
(Rx)	The contents of a GPR, where *x* is A, B, S, or T
(RA|0)	The contents of the register RA or 0, if the RA field is 0.
CR_{FLD}	The field in the condition register pointed to by a field of an instruction.
$c_{0:3}$	A 4-bit object used to store condition results in compare instructions.
^{n}b	The bit or bit value *b* is replicated *n* times.

xx	Bit positions which are don't-cares.
CEIL(x)	Least integer $\geq$ x.
EXTS(x)	The result of extending *x* on the left with sign bits.
PC	Program counter.
RESERVE	Reserve bit; indicates whether a process has reserved a block of storage.
CIA	Current instruction address; the 32-bit address of the instruction being described by a sequence of pseudocode. This address is used to set the next instruction address (NIA). Does not correspond to any architected register.
NIA	Next instruction address; the 32-bit address of the next instruction to be executed. In pseudocode, a successful branch is indicated by assigning a value to NIA. For instructions that do not branch, the NIA is CIA +4.
MS(addr, n)	The number of bytes represented by *n* at the location in main storage represented by *addr.*
EA	Effective address; the 32-bit address, derived by applying indexing or indirect addressing rules to the specified operand, that specifies a location in main storage.
EA_b	A bit in an effective address.
$EA_{b:b}$	A range of bits in an effective address.
ROTL((RS),n)	Rotate left; the contents of RS are shifted left the number of bits specified by *n*.
MASK(MB,ME)	Mask having 1s in positions MB through ME (wrapping if MB > ME) and 0s elsewhere.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an EA.

Part I. Introducing the PPC405GP Embedded Processor

Chapter 1. Overview

The IBM PowerPC 405GP 32-bit reduced instruction set computer (RISC) embedded processor, referred to as the PPC405GP, is a system-on-a-chip (SOC) that integrates a PowerPC embedded processor core with a rich set of on-chip peripherals:

- SDRAM controller
- External bus controller (EBC)
- PCI bus interface
- Direct memory access (DMA) with scatter/gather support
- Ethernet and media access layer (MAL) interfaces
- Two serial ports
- Inter-integrated circuit (IIC) interface
- General-purpose input/output (GPIO)

In addition, the PPC405GP supports CodePack™, a code compression scheme that can significantly reduce application code memory requirements, and a variety of debug tools.

This chapter describes:

- PPC405GP features
- The PowerPC Architecture™
- The PPC405GP implementation of the IBM PowerPC Embedded Environment, an extension of the PowerPC Architecture for embedded applications
- PPC405GP organization, including a block diagram and descriptions of the functional units
- PPC405GP registers
- PPC405GP addressing modes

Figure 1-1 illustrates the logical organization of the PPC405GP:

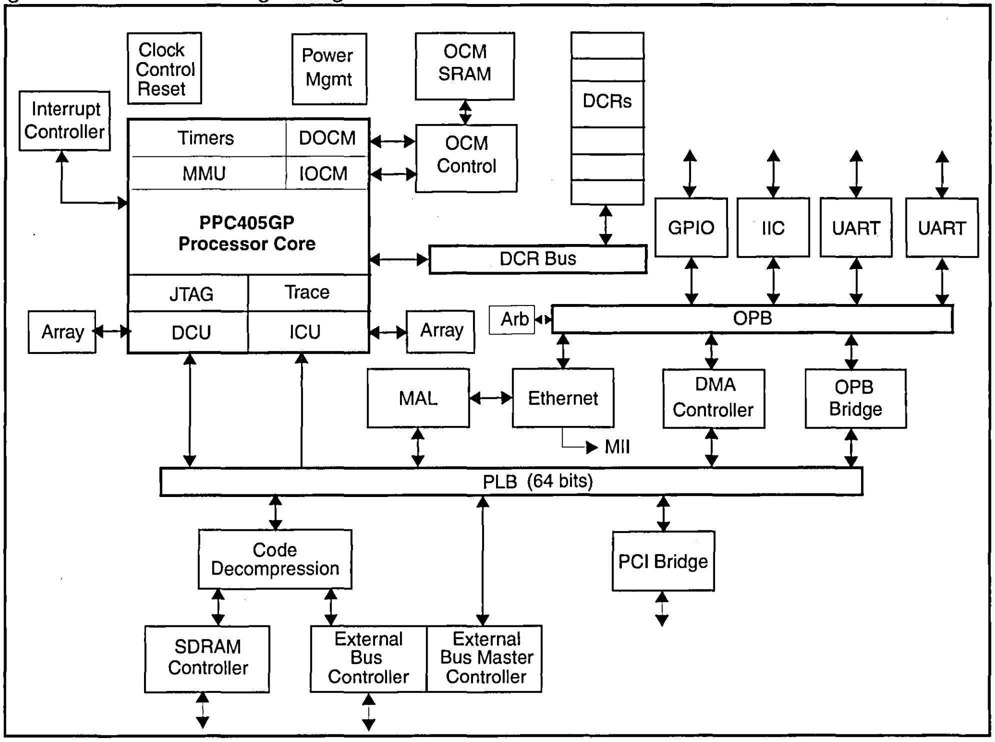


Figure 1-1. PPC405GP Block Diagram

1.1 PPC405GP Features

The PPC405GP provides high performance and low power consumption. The PPC405GP RISC CPU executes at sustained speeds approaching one cycle per instruction. On-chip instruction and data caches reduce chip count and design complexity in systems and improve system throughput.

1.1.1 Bus and Peripheral Features

The PPC405GP multilevel bus architecture and peripherals feature:

- Processor local bus (PLB)
- On-chip peripheral bus (OPB)
- PC-100 synchronous DRAM (SDRAM) controller
  - 32-bit interface for non-ECC applications
  - 40-bit interface (32 data bits and 8 check bits) for ECC applications

- External bus controller (EBC)
  - Flash ROM/Boot ROM interface
  - Direct support for 8-, 16-, or 32-bit SRAM or external peripherals
  - One external master supported
- PCI bus, Revision 2.2 compliant (32 bit, up to 66 MHz)
  - PCI bus interface can be configured to operate synchronously or asynchronously to the PLB
  - Internal PCI bus arbiter that can be disabled for use with an external arbiter
- DMA support for OPB and external peripherals
- Ethernet 10/100 Mbps (full-duplex) controller with media access layer (MAL) support
- Interrupt controller supporting programmable interrupt handling from a variety of sources
- Two 8-bit serial ports (16550 compatible UARTs)
- Inter-integrated circuit (IIC) controller
- General purpose I/O (GPIO) controller

1.1.2 PowerPC Processor Core Features

The PowerPC RISC fixed-point CPU features:

- PowerPC User Instruction Set Architecture (UISA) and extensions for embedded applications
- Thirty-two 32-bit general purpose registers (GPRs)
- Static branch prediction
- Five-stage pipeline with single-cycle execution of most instructions, including loads/stores
- Unaligned load/store support to cache arrays, main memory, and on-chip memory (OCM)
- Hardware multiply/divide for faster integer arithmetic (4-cycle multiply, 35-cycle divide)
- Multiply-accumulate instructions
- Enhanced string and multiple-word handling
- True little endian operation
- Programmable Interval Timer (PIT), Fixed Interval Timer (FIT), and watchdog timer
- Forward and reverse trace from a trigger event
- Storage control
  - Separate, configurable, two-way set-associative instruction and data cache units
  - Eight words (32 bytes) per cache line
  - 16KB instruction and 8KB data cache arrays
  - Instruction cache unit (ICU) non-blocking during line fills, data cache unit (DCU) non-blocking during line fills and flushes
  - Read and write line buffers
  - Instruction fetch hits are supplied from line buffer
  - Data load/store hits are supplied to line buffer
  - Programmable ICU prefetching of next sequential line into line buffer

  - Programmable ICU prefetching of non-cacheable instructions, full line (eight words) or half line (four words)
  - Write-back or write-through DCU write strategies
  - Programmable allocation on loads and stores
  - Operand forwarding during cache line fills
- Memory Management
  - Translation of the 4GB logical address space into physical addresses
  - Independent enabling of instruction and data translation/protection
  - Page level access control using the translation mechanism
  - Software control of page replacement strategy
  - Additional control over protection using zones
  - WIU0GE (write-through, cacheability, user-defined 0, guarded, endian) storage attribute control for each virtual memory region
- WIU0GE storage attribute control for thirty-two real 128MB regions
- PowerPC timer facilities
  - 64-bit time base
  - PIT, FIT, and watchdog timers
  - Synchronous external time base clock input
- Debug Support
  - Enhanced debug support with logical operators
  - Four instruction address compares (IACs)
  - Two data address compares (DACs)
  - Two data value compares (DVCs)
  - JTAG instruction to write to ICU
  - Forward or backward instruction tracing
- Minimized interrupt latency
- Advanced power management support

1.2 PowerPC Architecture

The PowerPC Architecture comprises three levels of standards:

- PowerPC User Instruction Set Architecture (UISA), including the base user-level instruction set, user-level registers, programming model, data types, and addressing modes. This is referred to as Book I of the PowerPC Architecture.
- PowerPC Virtual Environment Architecture, describing the memory model, cache model, cache-control instructions, address aliasing, and related issues. While accessible from the user level, these features are intended to be accessed from within library routines provided by the system software. This is referred to as Book II of the PowerPC Architecture.

- PowerPC Operating Environment Architecture, including the memory management model, supervisor-level registers, and the exception model. These features are not accessible from the user level. This is referred to as Book III of the PowerPC Architecture.

Book I and Book II define the instruction set and facilities available to the application programmer. Book III defines features, such as system-level instructions, that are not directly accessible by user applications. The PowerPC Architecture is described in *The PowerPC Architecture: A Specification for a New Family of RISC Processors*.

The PowerPC Architecture provides compatibility of PowerPC Book I application code across all PowerPC implementations to help maximize the portability of applications developed for PowerPC processors. This is accomplished through compliance with the first level of the architectural definition, the PowerPC UISA, which is common to all PowerPC implementations.

1.3 The PPC405GP as a PowerPC Implementation

The PPC405GP implements the PowerPC UISA, user-level registers, programming model, data types, addressing modes, and 32-bit fixed-point operations. The PPC405GP fully complies with the PowerPC UISA. The UISA 64-bit and floating point operations are not implemented. The floating point operations, which cause exceptions, can then be emulated by software.

Most of the features of the PPC405GP processor core are compatible with the PowerPC Virtual Environment and Operating Environment Architectures, as implemented in PowerPC processors such as the 6xx/7xx family. The PPC405GP processor core also provides a number of optimizations and extensions to these layers of the PowerPC Architecture. The full architecture of the PPC405GP is defined by the PowerPC Embedded Environment and the PowerPC User Instruction Set Architecture.

The primary extensions of the PowerPC Architecture defined in the Embedded Environment are:

- A simplified memory management mechanism with enhancements for embedded applications
- An enhanced, dual-level interrupt structure
- An architected DCR address space for integrated peripheral control
- The addition of several instructions to support these modified and extended resources

Finally, some of the specific implementation features of the PPC405GP are beyond the scope of the PowerPC Architecture. These features are included to enhance performance, integrate functionality, and reduce system complexity in embedded control applications.

1.4 RISC Processor Core Organization

The processor core consists of a 5-stage pipeline, separate instruction and data cache units, virtual memory management unit (MMU), three timers, debug, and interfaces to other functions.

1.4.1 Instruction and Data Cache Controllers

The PPC405GP processor core uses a 16KB instruction cache unit (ICU) and an 8KB data cache unit (DCU) to enable concurrent accesses and minimize pipeline stalls. Both cache units are two-way set-associative and use a 32-byte line size. The instruction set provides a rich assortment of cache control instructions, including instructions to read tag information and data arrays. See Chapter 4, "Cache Operations," for detailed information about the ICU and DCU.

1.4.1.1 Instruction Cache Unit

The ICU provides one or two instructions per cycle to the execution unit (EXU) over a 64-bit bus. A line buffer (built into the output of the array for manufacturing test) enables the ICU to be accessed only once for every four instructions, to reduce power consumption by the array.

The ICU can forward any or all of the words of a line fill to the EXU to minimize pipeline stalls caused by cache misses. The ICU aborts speculative fetches abandoned by the EXU, eliminating unnecessary line fills and enabling the ICU to handle the next EXU fetch. Aborting abandoned requests also eliminates unnecessary PLB activity to increase PLB availability for other on-chip cores, such as the DMA and Ethernet controllers.

1.4.1.2 Data Cache Unit

The DCU transfers 1, 2, 3, 4, or 8 bytes per cycle, depending on the number of byte enables presented by the CPU. The DCU contains a single-element command and store data queue to reduce pipeline stalls; this queue enables the DCU to independently process load/store and cache control instructions. Dynamic PLB request prioritization reduces pipeline stalls even further. When the DCU is busy with a low-priority request while a subsequent storage operation requested by the CPU is stalled, the DCU automatically increases the priority of the current request to the PLB.

The DCU uses a two-line flush queue to minimize pipeline stalls caused by cache misses. Line flushes are postponed until after a line fill is completed. Registers comprise the first position of the flush queue; the line buffer built into the output of the array for manufacturing test serves as the second position of the flush queue. Pipeline stalls are further reduced by forwarding the requested word to the CPU during the line fill. Single-queued flushes are non-blocking. When a flush operation is pending, the DCU can continue to access the array to determine subsequent load or store hits. Under these conditions, load hits can occur concurrently with store hits to write-back memory without stalling the pipeline. Requests abandoned by the CPU can also be aborted by the cache controller.

Additional DCU features enable the programmer to tailor performance for a given application. The DCU can function in write-back or write-through mode, as controlled by the Data Cache Write-through Register (DCWR) or the translation look-aside buffer (TLB). DCU performance can be tuned to balance performance and memory coherency. Store-without-allocate, controlled by the SWOA field of the Core Configuration Register 0 (CCR0), can inhibit line fills caused by store misses to further reduce potential pipeline stalls and unwanted external bus traffic. Similarly, load-without-allocate, controlled by CCR0[LWOA], can inhibit line fills caused by load misses.

1.4.2 Memory Management Unit

The 4GB address space of the PPC405GP is presented as a flat address space.

The MMU provides address translation, protection functions, and storage attribute control for embedded applications. The MMU supports demand paged virtual memory and other management schemes that require precise control of logical to physical address mapping and flexible memory protection. Working with appropriate system level software, the MMU provides the following functions:

- Translation of the 4GB logical address space into physical addresses
- Independent enabling of instruction and data translation/protection
- Page level access control using the translation mechanism
- Software control of page replacement strategy
- Additional control over protection using zones

- Storage attributes for cache policy and speculative memory access control

The MMU can be disabled under software control. If the MMU is not used, the PPC405GP provides other storage control mechanisms.

The translation lookaside buffer (TLB) is the hardware resource that controls translation and protection. It consists of 64 entries, each specifying a page to be translated. The TLB is fully associative; a page entry can be placed anywhere in the TLB. The translation function of the MMU occurs pre-cache for data accesses. Cache tags and indexing use physical addresses for data accesses; instruction fetches are virtually indexed and physically tagged.

Software manages the establishment and replacement of TLB entries. This gives system software significant flexibility in implementing a custom page replacement strategy. For example, to reduce TLB thrashing or translation delays, software can reserve several TLB entries for globally accessible static mappings. The instruction set provides several instructions to manage TLB entries. These instructions are privileged and require the software to be executing in supervisor state. Additional TLB instructions are provided to move TLB entry fields to and from GPRs.

The MMU divides logical storage into pages. Eight page sizes (1KB, 4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB) are simultaneously supported, so that, at any given time, the TLB can contain entries for any combination of page sizes. For a logical to physical translation to occur, a valid entry for the page containing the logical address must be in the TLB. Addresses for which no TLB entry exists cause TLB-Miss exceptions.

To improve performance, 4 instruction-side and 8 data-side TLB entries are kept in shadow arrays. The shadow arrays prevent TLB contention. Hardware manages the replacement and invalidation of shadow-TLB entries; no system software action is required. The shadow arrays can be thought of as level 1 TLBs, with the main TLB serving as a level 2 TLB.

When address translation is enabled, the translation mechanism provides a basic level of protection. Physical addresses not mapped by a page entry are inaccessible when translation is enabled. Read access is implied by the existence of the valid entry in the TLB. The EX and WR bits in the TLB entry further define levels of access for the page, by permitting execute and write access, respectively.

The Zone Protection Register (ZPR) enables the system software to override the TLB access controls. For example, the ZPR provides a way to deny read access to application programs. The ZPR can be used to classify storage by type; access by type can be changed without manipulating individual TLB entries.

The PowerPC Architecture provides WIU0GE (write-back/write through, cacheability, user-defined 0, guarded, endian) storage attributes that control memory accesses, using bits in the TLB or, when address translation is disabled, storage attribute control registers.

When address translation is enabled (MSR[IR, DR] = 1), storage attribute control bits in the TLB control the storage attributes associated with the current page. When address translation is disabled (MSR[IR, DR] = 0), bits in each storage attribute control register control the storage attributes associated with storage regions. Each storage attribute control register contains 32 fields. Each field sets the associated storage attribute for a 128MB memory region. See "Real-mode Storage Attribute Control" on page 6-17 for more information about the storage attribute control registers.

1.4.3 Timer Facilities

The processor core contains a time base and three timers:

- Programmable Interval Timer (PIT)
- Fixed Interval Timer (FIT)
- Watchdog timer

The time base is a 64-bit counter incremented either by an internal signal equal to the CPU clock rate or by a separate external timer clock signal. No interrupts are generated when the time base rolls over.

The PIT is a 32-bit register that is decremented at the same rate as the time base is incremented. The user loads the PIT register with a value to create the desired delay. When a decrement occurs on a PIT count of 1, the timer stops decrementing, a bit is set in the Timer Status Register (TSR), and a PIT interrupt is generated. Optionally, the PIT can be programmed to reload automatically the last value written to the PIT register, after which the PIT begins decrementing again.The Timer Control Register (TCR) contains the interrupt enable for the PIT interrupt.

The FIT generates periodic interrupts based on selected bits in the time base. Users can select one of four intervals for the timer period by setting the appropriate bits in the TCR. When the selected bit in the time base changes from 0 to 1, a bit is set in the TSR and a FIT interrupt is generated. The FIT interrupt enable is contained in the TCR.

The watchdog timer generates a periodic interrupt based on selected bits in the time base. Users can select one of four time periods for the interval and the type of reset generated if the watchdog timer expires twice without an intervening clear from software.

1.4.4 Debug

The processor core debug facilities include debug modes for the various types of debugging used during hardware and software development. Also included are debug events that allow developers to control the debug process. Debug modes and debug events are controlled using debug registers in the chip. The debug registers are accessed either through software running on the processor, or through the JTAG port. The JTAG port can also be used for board test.

The debug modes, events, controls, and interfaces provide a powerful combination of debug facilities for hardware and software development tools

1.4.4.1 Development Tool Support

The PPC405GP supports a wide range of hardware and software development tools.

An operating system debugger is an example of an operating system-aware debugger, implemented using software traps.

RISCWatch is an example of a development tool that uses the external debug mode, debug events, and the JTAG port to support hardware and software development and debugging.

The RISCTrace™ feature of RISCWatch is an example of a development tool that uses the real-time trace capability of the processor core.

1.4.4.2 Debug Modes

The internal, external, real-time-trace, and debug wait modes support a variety of debug tool used in embedded systems development. These debug modes are described in detail in "Debug Modes" on page 12-6.

1.4.5 Processor Core Interfaces

The processor core provides a range of I/O interfaces.

1.4.5.1 Processor Local Bus

The PLB-compliant interface provides separate 32-bit address and 64-bit data buses for the instruction and data sides.

1.4.5.2 Device Control Register Bus

The Device Control Register (DCR) bus interface provides access to on-chip registers for configuration and status of peripherals such as SDRAM, DMA and so on.

These registers are accessed using the **mfdcr** and **mtdcr** instructions.

1.4.5.3 Clock and Power Management

This interface supports several methods of clock distribution and power management.

1.4.5.4 JTAG

The JTAG port is enhanced to support the attachment of a debug tool such as the RISCWatch product from IBM Microelectronics. Through the JTAG test access port, a debug tool can single-step the processor and interrogate internal processor state to facilitate software debugging. The enhancements comply with the IEEE 1149.1 specification for vendor-specific extensions, and are therefore compatible with standard JTAG hardware for boundary-scan system testing.

1.4.5.5 Interrupts

The processor core provides an interface to the UIC, an on-chip interrupt controller that is logically outside the processor core. The UIC combines asynchronous interrupt inputs from on-chip and off-chip sources and presents them to the processor core using a pair of interrupt signals: critical and non-critical.

1.4.5.6 On-Chip Memory

The on-chip memory (OCM) interface supports the implementation of instruction- and data-side memory that can be accessed at performance levels matching the cache arrays.

The PPC405GP provides 4KB of OCM.

1.5 Processor Core Programming Model

The programming model is described in detail in Chapter 3, "Programming Model."

The PowerPC instruction set and Special Purpose Registers (SPRs) provide a high degree of user control over configuration and operation of the processor core functional units.

1.5.1 Data Types

Processor core operands are bytes, halfwords, and words. Multiple words or strings of bytes can be transferred using the load/store multiple and load/store string instructions. Data is represented in twos complement notation or in unsigned fixed-point format.

The address of a multibyte operand is always the lowest memory address occupied by that operand. Byte ordering can be selected as big endian (the lowest memory address of an operand contains its most significant byte) or as little endian (the lowest memory address of an operand contains its least significant byte). See "Byte Ordering" on page 3-28 for more information about big and little endian operation.

1.5.2 Processor Core Register Set Summary

The processor core registers can be grouped into basic categories based on function and access mode: general purpose registers (GPRs), special purpose registers (SPRs), the machine state register (MSR), the condition register (CR), and device control registers (DCRs).

Chapter 25, "Register Summary," provides a register diagram and a register field description table for each register.

1.5.2.1 General Purpose Registers

The processor core contains 32 GPRs; each register contains 32 bits. The contents of the GPRs can be transferred from memory using load instructions and stored to memory using store instructions. GPRs, which are specified as operands in many instructions, can also receive instruction results and the contents of other registers.

1.5.2.2 Special Purpose Registers

Special Purpose Registers (SPRs), which are part of the PowerPC Architecture, are accessed using the **mtspr** and **mfspr** instructions. SPRs control the use of the debug facilities, timers, interrupts, storage control attributes, and other architected processor resources.

All SPRs are privileged (unavailable to user-mode programs), except the Count Register (CTR), the Link Register (LR), SPR General Purpose Registers (SPRG4–SPRG7, read-only), and the Fixed-point Exception Register (XER). Note that access to the Time Base Lower (TBL) and Time Base Upper (TBU) registers, when addressed as SPRs, is write-only and privileged. However, when addressed as Time Base Registers (TBRs), read access to these registers is not privileged. See "Time Base Registers" on page 25-3 for more information.

1.5.2.3 Machine State Register

The processor core contains a 32-bit Machine State Register (MSR). The contents of a GPR can be written to the MSR using the **mtmsr** instruction, and the MSR contents can be read into a GPR using the **mfmsr** instruction. The MSR contains fields that control the operation of the processor core.

1.5.2.4 Condition Register

The processor core contains a 32-bit Condition Register (CR). These bits are grouped into eight 4-bit fields, CR[CR0]–CR[CR7]. Instructions are provided to perform logical operations on CR fields and bits within fields and to test CR bits within fields. The CR fields, which are set by compare instructions, can be used to control branches. CR[CR0] can be set implicitly by arithmetic instructions.

1.5.2.5 Device Control Registers

DCRs, which are architecturally outside of the processor core, are accessed using the **mtdcr** and **mfdcr** instructions. DCRs are used to control, configure, and hold status for various functional units that are not part of the processor core.

The **mtdcr** and **mfdcr** instructions are privileged, for all DCRs. Therefore, all accesses to DCRs are privileged. See “Privileged Mode Operation” on page 3-41.

1.5.3 Memory-Mapped I/O Registers

The memory-mapped I/O (MMIO) registers are accessed using load and store instructions. MMIO registers, which are outside processor core and which are not architected, are used to control, configure, and hold status for various functional units that are not part of the processor core.

1.5.4 Addressing Modes

The processor core supports the following addressing modes, which enable efficient retrieval and storage of data in memory:

- Base plus displacement addressing
- Indexed addressing
- Base plus displacement addressing and indexed addressing, with update

In the base plus displacement addressing mode, an effective address (EA) is formed by adding a displacement to a base address contained in a GPR (or to an implied base of 0). The displacement is an immediate field in an instruction.

In the indexed addressing mode, the EA is formed by adding an index contained in a GPR to a base address contained in a GPR (or to an implied base of 0).

The base plus displacement and the indexed addressing modes also have a “with update” mode. In “with update” mode, the effective address calculated for the current operation is saved in the base GPR, and can be used as the base in the next operation. The “with update” mode relieves the processor from repeatedly loading a GPR with an address for each piece of data, regardless of the proximity of the data in memory.

Chapter 2. On-Chip Buses

The on-chip bus architecture, which consists of the processor local bus (PLB), on-chip peripheral bus (OPB), and device control register (DCR) bus, provides a link between the cache units in the processor core and other PLB and OPB master and slave devices used in the PPC405GP. These devices include the SDRAM controller, PCI bridge, DMA controller, and external bus controller.

The PLB is a high performance bus used to access memory through bus interface units. The PLB master and slave assignments for the PPC405GP are listed in "PLB Masters and Slaves" on page 2-2.

Lower performance peripherals (such as serial ports) are attached to the OPB. A bridge between the PLB and OPB enables data transfers between PLB masters and OPB slaves. DMA peripherals can also be OPB peripherals.

The DCR bus is used primarily to access status and control registers of the various PLB and OPB masters and slaves. The DCR bus offloads status and control read and write transfers from the PLB. The DCR bus is not described further in this chapter.

The following publications, which are available from your IBM representative and in the IBM Microelectronics technical library (www.chips.ibm.com), describe the on-chip bus architecture:

- *The CoreConnect™ Bus Architecture*
- *Processor Local Bus Architecture Specifications*
- *On-Chip Peripheral Bus Architecture Specifications*
- *Device Control Register Bus Architecture Specifications*

The PPC405GP block diagram (Figure 1-1 on page 1-2) illustrates the on-chip bus structure of the PPC405GP.

2.1 Processor Local Bus

The PLB is a high-performance on-chip bus. The PLB supports read and write data transfers between master and slave devices equipped with a PLB interface and connected through PLB signals.

Each PLB master is attached to the PLB through separate address, read data and write data buses, and transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address, read data and write data buses, and transfer control and status signals for each data bus.

Access to the PLB is granted through a central arbitration mechanism that enables masters to compete for bus ownership. This arbitration mechanism provides for fixed and fair priority schemes.

Timing for all PLB signals is provided by a clock source that is shared by all PLB masters and slaves.

2.1.1 PLB Features

- Overlapping of read and write transfers allows two data transfers per clock cycle for maximum bus utilization
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth
- Address pipelining reduces overall bus latency by allowing the latency associated with a new request to be overlapped with an ongoing data transfer in the same direction
- Late master request abort capability reduces latency associated with aborted requests
- Four levels of request priority and selectable arbitration modes provide flexible arbitration policies.
- Support for 16-, 32-, and 64-byte line data transfers
- Sequential burst protocol allows byte, halfword, and word burst data transfers.
- DMA buffered peripheral-to-memory, memory-to-peripheral, and memory-to-memory operations are supported

2.1.2 PLB Masters and Slaves

Table 2-1 lists the PLB masters and slaves provided in the PPC405GP.

Table 2-1. PPC405GP PLB Agents as Masters and Slaves

PLB Agent	PLB Master/Slave
Processor core ICU	Master
Processor core DCU	Master
External bus master interface	Master
Static memory/peripherals	Slaves
PCI bridge (PCI to PLB) PCI bridge (PLB to PCI)	Master Slave
MAL	Master
DMA controller	Master
PLB to OPB bridge	Slave
SDRAM controller	Slave

The SDRAM and EBC slaves are connected in line with the decompression controller. When the decompression controller is disabled, a bypass mode that directly accesses SDRAM and the EBC is used. When the decompression controller is enabled, it intercepts reads from compressed regions before they reach SDRAM and the EBC.

2.1.3 PLB Master Assignments

Each PLB master can be programmed to use one of four priority levels during PLB transfers, enabling the system designer to tune PLB transfer priorities to the requirements of a particular application. For example, if an application always requires PCI to SDRAM transfers to have the lowest latency, the PCI master can be programmed to the highest PLB master priority. This causes the PLB arbiter to grant PCI access requests before granting the access requests of any other master.

Programming Note: PLB master priority assignments, which are application-dependent, must be considered carefully to prevent potential lockouts of lower priority masters. For most applications, assigning a priority of 0b10 to each master is a useful starting point.

A register associated with each master controls the priority of that master. Table 2-2 lists the PLB masters and the register fields controlling the priority of the masters. Priorities range from 0b00 (lowest) to 0b11 (highest).

Table 2-2. Registers Controlling PLB Master Priority Assignments

Master ID	Description	Register Field	Comments
0	Processor core ICU	CCR0[IPP]	
1	Processor core data cache unit DCU	CCR0[DPP1]	The high-order bit of CCR0[DPP1] is controlled by the DCU logic, so only the low-order priority bit can be programmed.
2	External bus master	EBC0_CFG[EMPL] EBC0_CFG[EMPH]	Which field sets external master priority depends upon the setting of the HoldPri signal.
3	PCI bridge	PCIC0_BRDGOPT1[PRP]	Reset value is 0b11
4	MAL	MAL0_CFG[PLBP]	
5	DMA controller	DMA0_CR0[CP]	Unique priorities can be assigned to each DMA channel.
		DMA0_CR1[CP]	
		DMA0_CR2[CP]	
		DMA0_CR3[CP]	

See "PLB Arbiter Control Register (PLB0_ACR)" on page 2-5 for information about programming the PLB0_ACR to control PLB priority mode and priority order, which determine how the PLB arbitrates simultaneous PLB bus access requests having equal priorities.

2.1.4 PLB Transfer Protocol

A PLB transaction is composed of an address cycle and a data cycle.

The address cycle has three phases: request, transfer, and address acknowledge. A PLB transaction begins when a master drives its address and transfer qualifier signals and requests ownership of the bus during the request phase of the address cycle. Once bus ownership has been granted by the PLB arbiter, the master's address and transfer qualifiers are presented to the slave devices during the transfer phase.

During normal operation, the address cycle is terminated by a slave latching the master's address and transfer qualifiers during the address acknowledge phase.

Each data beat in the data cycle has two phases: transfer and data acknowledge. During the transfer phase, the master drives the write data bus for a write transfer or samples the read data bus for a read transfer. Data acknowledge signals are required during the data acknowledge phase for each data beat in a data cycle.

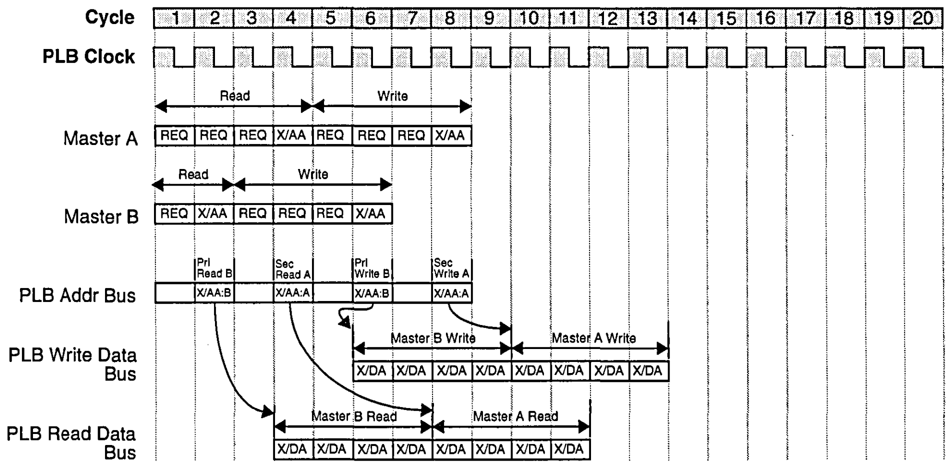
Note: For single-beat transfers, data acknowledge signals also indicate the end of the data transfer. For line or burst transfers, the data acknowledge signals apply to each beat and indicate the end of the data cycle only after the final beat.

2.1.5 Overlapped PLB Transfers

Figure 2-1 shows an example of overlapped PLB transactions on the read/write data buses with pipelining. PLB address, read data, and write data buses are decoupled from one another, allowing for address cycles to be overlapped with read or write data cycles, and for read data cycles to be overlapped with write data cycles. The PLB split-bus transaction capability allows the address and data buses to have different masters at the same time.

PLB address pipelining capability enables a new bus transfer to begin before an ongoing transfer finishes. Address pipelining reduces overall bus latency on the PLB by enabling the latency associated with a new transfer request to be overlapped with an ongoing data transfer in the same direction.

PLB masters A and B each present a read request followed by a write request. Master B gets the bus first, so its read is the primary read transaction. The master A address cycle begins as soon as the master B address cycle ends. The master A read is taken as a secondary transfer. Writes follow reads.



Note: X/AA	= Xfer/AddrAck
X/DA	= Xfer/DataAck
X/AA:A	= X/AA Master A
X/AA:B	= X/AA Master B

Figure 2-1. Overlapped PLB Transfers

Note: A master can begin to request ownership of the PLB in parallel with the address cycle or data cycle of another master bus transfer. Overlapped read and write data transfers and split-bus transactions enable the PLB to operate at a very high bandwidth.

2.1.6 PLB Arbiter Registers

PLB arbiter registers are DCRs accessed using the **mfdcr** and **mtdcr** instructions.

Table 2-3 summarizes the PLB arbiter DCRs.

Table 2-3. PLB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
PLB0_ACR	PLB Arbiter Control Register	0x087	R/W	2-5
PLB0_BEAR	PLB Error Address Register	0x086	R/O	2-5
PLB0_BESR	PLB Error Status Register	0x084	R/Clear	2-6

2.1.6.1 PLB Arbiter Control Register (PLB0_ACR)

The PLB0_ACR controls PLB arbitration priority, which is determined by PLB priority mode and PLB priority order.

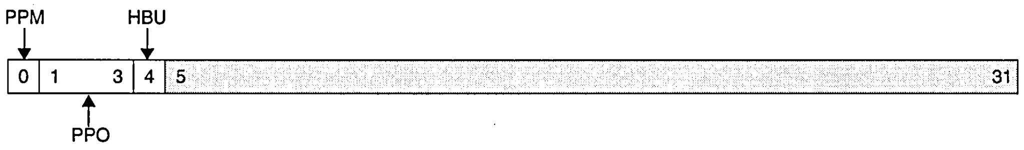


Figure 2-2. PLB Arbiter Control Register (PLB0_ACR)

0	PPM	PLB Priority Mode 0 Fixed 1 Fair
1:3	PPO	PLB Priority Order 000 Masters 0, 1, 2, 3, 4, 5 001 Masters 1, 2, 3, 4, 5, 0 010 Masters 2, 3, 4, 5, 0, 1 011 Masters 3, 4, 5,0, 1, 2 100 Masters 4, 5, 0, 1, 2, 3 101 Masters 5, 0, 1, 2, 3, 4 110 Reserved 111 Reserved
4	HBU	High Bus Utilization 0 Disabled 1 Enabled
5:31	▨	Reserved

2.1.6.2 PLB Error Address Register (PLB0_BEAR)

The read-only PLB0_BEAR contains the address of the access on which a bus timeout error occurred.

The PLB0_BEAR can be locked by the master. Once locked, the PLB0_BEAR cannot be updated, if a subsequent error occurs, until all PLB0_BESR[FLCK*n*] fields are cleared (*n* is the master ID).

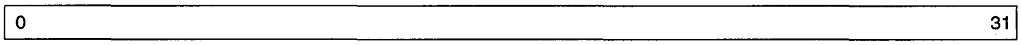


Figure 2-3. PLB Error Address Register (PLB0_BEAR)

0:31		Address of bus timeout error

2.1.6.3 PLB Error Status Register (PLB0_BESR)

The read/clear PLB0_BESR identifies timeout errors on PLB bus transfers, the master initiating the transfer, and the type of transfer.

Each PLB0_BESR[PTE*n*] field (*n* is the master ID) can be locked by the master. Once locked, PLB0_BESR [PTE*n*] fields cannot be updated if a subsequent error occurs until the corresponding PLB0_BESR [FLCK*n*] field is cleared. To clear a PLB0_BESR field, write 1 to the field. Writing 0 to a PLB0_BESR field does not affect the field.

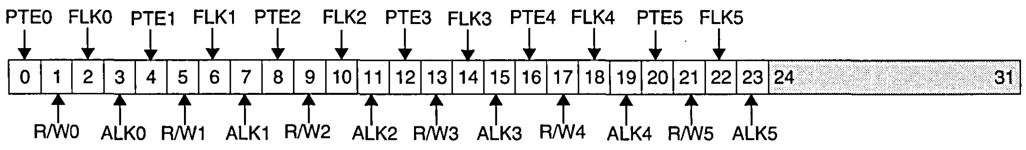


Figure 2-4. PLB Error Status Register (PLB0_BESR)

0	PTE0	Master 0 PLB Timeout Error Status 0 No master 0 timeout error 1 Master 0 timeout error	Master 0 is the processor core ICU.
1	R/W0	Master 0 Read/Write Status 0 Master 0 error operation was a write 1 Master 0 ICU error operation was a read	
2	FLK0	Master 0 PLB0_BESR Field Lock 0 Master 0 PLB0_BESR field is unlocked 1 Master 0 field is locked	
3	ALK0	Master 0 PLB0_BEAR Address Lock 0 Master 0 PLB0_BEAR is unlocked 1 Master 0 PLB0_BEAR is locked	
4	PTE1	Master 1 PLB Timeout Error Status 0 No master 1 timeout error 1 Master 1 timeout error	Master 1 is the processor core DCU.
5	R/W1	Master 1 Read/Write Status 0 Master 1 error operation was a write 1 Master 1 error operation was a read	

6	FLK1	Master 1PLB0_BESR Field Lock 0 Master 1 PLB0_BESR field is unlocked 1 Master 1 PLB0_BESR field is locked	
7	ALK1	Master 1 PLB0_BEAR Address Lock 0 Master 1 PLB0_BEAR is unlocked 1 Master 1 PLB0_BEAR is locked	
8	PTE2	Master 2 PLB Timeout Error Status 0 No master 2 timeout error 1 Master 2 timeout error	Master 2 is the external master.
9	R/W2	Master 2 Read/Write Status 0 Master 2 error operation was a write 1 Master 2 error operation was a read	
10	FLK2	Master 2 PLB0_BESR Field Lock 0 Master 2 PLB0_BESR field is unlocked 1 Master 2 PLB0_BESR field is locked	
11	AL2	Master 2 PLB0_BEAR Address Lock 0 Master 2 PLB0_BEAR is unlocked 1 Master 2 PLB0_BEAR is locked	
12	PTE3	Master 3 PLB Timeout Error Status 0 No Master 3 timeout error 1 Master 3 timeout error	Master 3 is PCI.
13	R/W3	Master 3 Read/Write Status 0 Master 3 error operation was a write 1 Master 3 error operation was a read	
14	FLK3	Master 3 PLB0_BESR Field Lock 0 Master 3 PLB0_BESR field is unlocked 1 Master 3 PLB0_BESR field is locked	
15	ALK3	Master 3 PLB0_BEAR Address Lock 0 Master 3 PLB0_BEAR is unlocked 1 Master 3 PLB0_BEAR is locked	
16	PTE4	Master 4 PLB Timeout Error Status 0 No master 4 timeout error 1 Master 4 timeout error	Master 4 is MAL.
17	R/W4	Master 4 Read/Write Status 0 Master 4 error operation was a write 1 Master 4 error operation was a read	
18	FLK4	Master 4 PLB0_BESR Field Lock 0 Master 4 PLB0_BESR field is unlocked 1 Master 4 field is locked	
19	ALK4	Master 4 PLB0_BEAR Address Lock 0 Master 4 PLB0_BEAR is unlocked 1 Master 4 PLB0_BEAR is locked	
20	PTE5	Master 5 PLB Timeout Error Status 0 No master 5 timeout error 1 Master 5 timeout error	Master 5 is DMA.
21	R/W5	Master 5 Read/Write Status 0 Master 5 error operation was a write 1 Master 5 error operation was a read	

22	FLK5	Master 5 PLB0_BESR Field Lock 0 Master 5 PLB0_BESR field is unlocked 1 Master 5 PLB0_BESR field is locked
23	ALK5	Master 5 PLB0_BEAR Address Lock 0 Master 5 PLB0_BEAR is unlocked 1 Master 5 PLB0_BEAR is locked
24:31	▨	Reserved

2.1.7 PLB to OPB Bridge Registers

The PLB to OPB bridge registers are DCRs accessed using the **mfdcr** and **mtdcr** instructions.

Table 2-4 lists the PLB to OPB bridge registers.

Table 2-4. PLB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
POB0_BEAR	Bridge Error Address Register	0x0A2	R/O	2-5
POB0_BESR0	Bridge Error Status Register 0 (Master IDs 0,1, 2, 3)	0x0A0	R/Clear	2-5
POB0_BESR1	Bridge Error Status Register (Master IDs 4, 5)	0x0A4	R/Clear	2-6

2.1.7.1 Bridge Error Address Register (POB0_BEAR)

The read-only POB0_BEAR reports the address of a PLB to OPB transfer that results in an error. The PLB to OPB bridge writes the error address in the POB0_BEAR, unless the associated POB0_BESR*m*[ALCK*n*] field is set (*m* is either 0 or 1, depending on the master ID specified by *n*). Once locked, the PLB to OPB bridge cannot write POB0_BEAR until all POB0_BESR*m*[ALCK*n*] fields that are set are cleared.

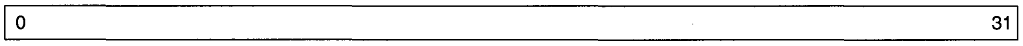


Figure 2-5. Bridge Error Address Register (POB0_BEAR)

0:31	0x0B2	Address of bus error

2.1.7.2 Bridge Error Status Registers (POB0_BESR0–POB0_BESR1)

The PLB to OPB bridge writes error information into the appropriate POB0_BESR*m* register. (For master IDs 0, 1, 2, and 3, *m* = 0; for master IDs 4 and 5, *m* = 1.)

POB0_BESR*m* fields can be locked using the POB0_BESR*m*[FLK*n*] and POB0_BESR*m*[ALK*n*] fields (*n* is the master ID). Once locked, the POB0_BESR*m* fields associated with a master cannot be overwritten if a subsequent error occurs until the locking fields are cleared. To clear a lock, write 1 to the POB0_BESR*m*[FLK*n*] and POB0_BESR*m*[ALK*n*] fields that are set. Writing 0 to a lock field does not affect the field.

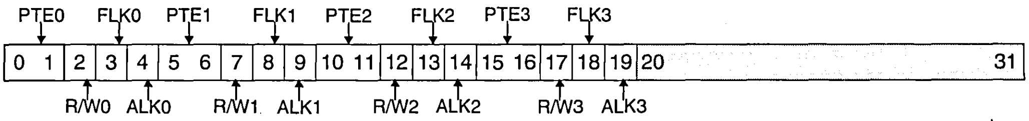


Figure 2-6. Bridge Error Status Register 0 (POB0_BESR0)

0:1	PTE0	PLB Timeout Error Status Master 0 00 No master 0 error occurred 01 Master 0 timeout error occurred 10 Master 0 slave error occurred 11 Reserved	Master 0 is the processor core ICU.
2	R/W0	Read Write Status Master 0 0 Master 0 error operation is a read 1 Master 0 error operation is a write	
3	FLK0	POB0_BESR0 Field Lock Master 0 0 Master 0 POB0_BESR0 field is unlocked 1 Master 0 POB0_BESR0 field is locked	
4	ALK0	POB0_BEAR Address Lock Master 0 0 Master 0 POB0_BEAR address is unlocked 1 Master 0 POB0_BEAR address is locked	
5:6	PTE1	PLB Timeout Error Status Master 1 00 No master 1 error occurred 01 Master 1 timeout error occurred 10 Master 1 slave error occurred 11 Reserved	Master 1 is the processor core DCU.
7	R/W1	Read/Write Status Master 1 0 Master 1 error operation is a read 1 Master 1 error operation is a write	
8	FLK1	POB0_BESR0 Field Lock Master 1 0 Master 1 POB0_BESR0 field is unlocked 1 Master 1 POB0_BESR0 field is locked	
9	ALK1	POB0_BEAR Address Lock Master 1 0 Master 1 POB0_BEAR address is unlocked 1 Master 1 POB0_BEAR address is locked	
10:11	PTE2	PLB Timeout Error Status Master 2 00 No master 2 error occurred 01 Master 2 timeout error occurred 10 Master 2 slave error occurred 11 Reserved	Master 2 is the external master.
12	R/W2	Read/Write Status Master 2 0 Master 2 error operation is a read 1 Master 2 error operation is a write	

13	FLK2	POB0_BESR0 Field Lock Master 2 0 Master 2 POB0_BESR0 field is unlocked 1 Master 2 POB0_BESR0 field is locked	
14	ALK2	POB0_BEAR Address Lock Master 2 0 Master 2 POB0_BEAR address is unlocked 1 Master 2 POB0_BEAR address is locked	
15:16	PTE3	PLB Timeout Error Status Master 3 00 No master 3 error occurred 01 Master 3 timeout error occurred 10 Master 3 slave error occurred 11 Reserved	Master 3 is PCI.
17	R/W3	Read/Write Status Master 3 0 Master 3 error operation is a read 1 Master 3 error operation is a write	
18	FLK3	POB0_BESR0 Field Lock Master 3 0 Master 3 POB0_BESR0 field is unlocked 1 Master 3 POB0_BESR0 field is locked	
19	ALK3	POB0_BEAR Address Lock Master 3 0 Master 3 POB0_BEAR address is unlocked 1 Master 3 POB0_BEAR address is locked	
20:31	▨	Reserved	

Figure 2-7 illustrates POB0_BESR1.

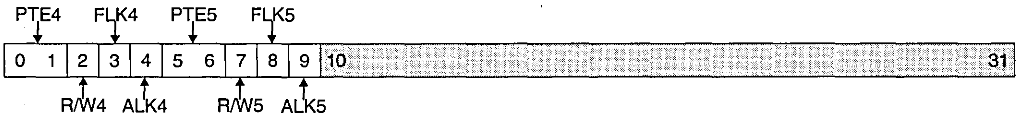


Figure 2-7. Bridge Error Status Register 1 (POB0_BESR1)

0:1	PTE4	PLB Timeout Error Status Master 4 00 No Master 4 error occurred 01 Master 4 timeout error occurred 10 Master 4 slave error occurred 11 Reserved	Master 4 is MAL.
2	R/W4	Read/Write Status Master 4 0 Master 4 error operation is a read 1 Master 4 error operation is a write	
3	FLK4	POB0_BESR1 Field Lock Master 4 0 Master 4 POB0_BESR1 field is unlocked 1 Master 4 POB0_BESR1 field is locked	

4	ALK4	POB0_BEAR Address Lock Master 4 0 Master 4 POB0_BEAR address is unlocked 1 Master 4 POB0_BEAR address is locked	
5:6	PTE5	PLB Timeout Error Status Master 5 00 No Master 5 error occurred 01 Master 5 timeout error occurred 10 Master 5 slave error occurred 11 Reserved	Master 5 is DMA.
7	R/W5	Read/Write Status Master 5 0 Master 5 error operation is a read 1 Master 5 error operation is a write	
8	FLK5	POB0_BESR1 Field Lock Master 5 0 Master 5 POB0_BESR1 field is unlocked 1 Master 5 POB0_BESR1 field is locked	
9	ALK5	POB0_BEAR Address Lock Master 5 0 Master 5 POB0_BEAR address is unlocked 1 Master 5 POB0_BEAR address is locked	
10:31		Reserved	

2.1.7.3 On-Chip Peripheral Bus

The OPB is used to attach peripherals that do not require the bandwidth of the PLB. The OPB does not connect directly to the PPC405GP processor core, which accesses peripherals attached to the OPB through a PLB-to-OPB bridge.

2.1.8 OPB Features

The on-chip peripheral bus features:

- A 32-bit address bus and a 32-bit data bus
- Dynamic bus sizing; byte, halfword, and word transfers
- Byte and halfword duplication for byte and halfword transfers
- Single-cycle transfer of data between OPB bus master and OPB slaves
- Sequential address (burst) protocol support
- Devices on the OPB may be memory mapped, act as DMA peripherals, or support both transfer methods
- A 16-cycle fixed bus timeout provided by the OPB arbiter
- Bus parking for reduced latency
- Bus arbitration overlapped with last cycle of bus transfers

2.1.9 OPB Master Assignments

Table 2-1 lists the OPB masters. (The numbering reflects that the PPC405GP implements two masters; the OPB can support four masters.)

Table 2-5. PPC405GP OPB Master Assignments

OPB Agents	Description
DMA controller	DMA (master 0)
OPB to PLB bridge	OPB to PLB bridge (master 2)

2.1.10 OPB Arbiter Registers

The OPB arbiter contains the MMIO registers summarized in Table 2-6.

Table 2-6. PLB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
OPBA0_CR	OPB Arbiter Control Register	0xEF600601	R/W	2-5
OPBA0_PR	OPB Arbiter Priority Register	0xEF600600	R/W	2-5

2.1.10.1 OPB Arbiter Control Register (OPBA0_CR)

The OPBA0_CR fields controls updating of the OPBA0_PR (described in "OPB Arbiter Priority Register (OPBA0_PR)" on page 2-13). Because the PPC405GP provides two masters, master IDs 1 and 3 are ignored.

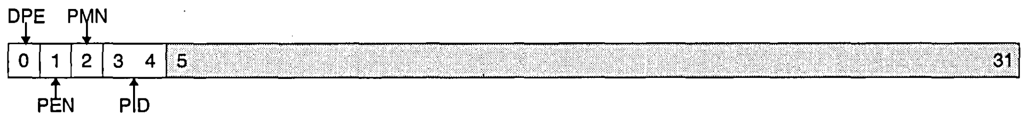


Figure 2-8. OPB Arbiter Control Register (OPBA0_CR)

0	DPE	Dynamic Priority Enable 0 Dynamic priority disabled 1 Dynamic priority enabled
1	PEN	Park Enable 0 Park disabled 1 Park enabled

2	PMN	Park on Master Not Last 0 Park on master last 1 Park on master not last	
3:4	PID	Parked Master ID 00 Master ID 0 01 Reserved 10 Master ID 2 11 Reserved	Master 0 is DMA; master 2 is the OPB to PLB bridge.
5:31		Reserved	

2.1.10.2 OPB Arbiter Priority Register (OPBA0_PR)

The OPBA0_PR assigns priorities to the OPB master IDs. Because the PPC405 provides two masters, master IDs 1 and 3 are ignored. At reset, master ID 0 (DMA) has a higher priority than master 2 (OPB to PLB bridge).

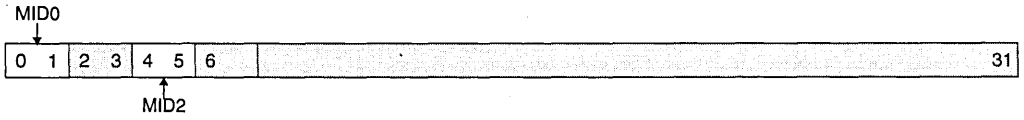


Figure 2-9. OPB Arbiter Priority Register (OPBA0_PR)

0:1	MID0	High Priority Master ID 00 Master ID 0 01 Reserved 10 Reserved 11 Reserved	At reset, this priority is assigned to DMA.
2:3		Reserved	
4:5	MID2	Low Priority master ID 00 Reserved 01 Reserved 10 Master ID 2 11 Reserved	At reset, this priority is assigned to the OPB to PLB bridge.
6:31		Reserved	

Part II. The PPC405GP RISC Processor

Chapter 3. Programming Model

The programming model of the PPC405GP embedded processor describes the following features and operations:

- Memory organization and addressing, starting on page 3-1
- Registers, starting on page 3-3
- Data types and alignment, starting on page 3-26
- Byte ordering, starting on page 3-28
- Instruction processing, starting on page 3-33
- Branching control, starting on page 3-34
- Speculative accesses, starting on page 3-37
- Privileged mode operation, starting on page 3-41
- Synchronization, starting on page 3-43
- Instruction set, starting on page 3-47

3.1 User and Privileged Programming Models

The PPC405GP executes programs in two modes, also referred to as states. Programs running in *privileged mode* (also referred to as the supervisor state) can access any register and execute any instruction. These instructions and registers comprise the privileged programming model. In *user mode*, certain registers and instructions are unavailable to programs. This is also called the problem state. Those registers and instructions that are available comprise the user programming model.

Privileged mode provides operating system software access to all processor resources. Because access to certain processor resources is denied in user mode, application software runs in user mode. Operating system software and other application software is protected from the effects of an errant application program.

Throughout this book, the terms user program and privileged programs are used to associate programs with one of the programming models. Registers and instructions are described as user or privileged. Privileged mode operation is described in detail in "Privileged Mode Operation" on page 3-41.

3.2 Memory Organization and Addressing

The PowerPC Architecture defines a 32-bit, 4-gigabyte (GB) address space for instructions and data.

3.2.1 Physical Address Map

Table 3-1 illustrates the physical address map.

Table 3-1. PPC405GP Address Space

Function	Start Address	End Address	Size
Local Memory/Peripherals[1]	0x00000000	0x7FFFFFFF	2GB
PCI Bridge (Total)	0x80000000	0xEF5FFFFF	1.744GB
PCI Memory	0x80000000	0xE7FFFFFF	1.625GB
PCI I/O	0xE8000000	0xE800FFFF	64KB
Reserved	0xE8010000	0xE87FFFFF	8MB–64KB
PCI I/O	0xE8800000	0xEBFFFFFF	56MB
Reserved (PCI does not respond)	0xEC000000	0xEEBFFFFF	44MB
PCI Configuration Registers	0xEEC00000	0xEEC00007	8B
Reserved	0xEEC00008		1MB–8B
PCI Interrupt Acknowledge (read)	0xEED00000	0xEED00003	4B
PCI Special Cycle (write)	0xEED00000	0xEED00003	4B
Reserved	0xEED00004	0xEEDFFFFF	1MB–4B
Reserved (PCI does not respond)	0xEEE00000	0xEF3FFFFF	6MB
PCI Local Configuration Registers	0xEF400000	0xEF40003F	64B
Reserved	0xEF400040	0xEF5FFFFF	2MB–64B
Internal Peripherals (Total)	0xEF600000	0xEFFFFFFF	10MB
UART0 Registers	0xEF600300	0xEF600307	8B
UART1 Registers	0xEF600400	0xEF600407	8B
IIC Registers	0xEF600500	0xEF600510	17B
OPB Arbiter Registers	0xEF600600	0xEF600601	2B
GPIO Controller Registers	0xEF600700	0xEF60077F	128B
Ethernet MAC Registers	0xEF600800	0xEF600867	104B
Expansion ROM[2]	0xF0000000	0xFFDFFFFF	254MB
Boot ROM[2, 3]	0xFFE00000	0xFFFFFFFF	2MB

1. The local memory/peripheral area of the memory map can be configured for SDRAM, ROM, or peripherals.
2. The boot ROM and expansion ROM areas of the memory map are intended for ROM or flash devices. The controller supports volatile memory devices such as SDRAM and SRAM in these areas.
3. When booting from PCI memory, the boot ROM address space begins at 0xFFFE 0000 (size is 128KB).

3.2.2 Storage Attributes

The PowerPC Architecture defines storage attributes that control data and instruction accesses. Storage attributes are provided to control cache write-through policy (the W storage attribute), cachability (the I storage attribute), memory coherency in multiprocessor environments (the M storage attribute), and guarding against speculative memory accesses (the G storage attribute). The IBM PowerPC Embedded Environment defines additional storage attributes for storage compression (the U0 storage attribute) and byte ordering (the E storage attribute).

The PPC405GP provides two control mechanisms for the W, I, U0, G, and E attributes. Because the PPC405GP does not provide hardware support for multiprocessor environments, the M storage attribute, when present, has no effect.

When the PPC405GP operates in virtual mode (address translation is enabled), each storage attribute is controlled by the W, I, U0, G, and E fields in the translation lookaside buffer (TLB) entry for each memory page. The size of memory pages, and hence the size of storage attribute control regions, is variable. Multiple sizes can be in effect simultaneously on different pages.

When the PPC405GP operates in real mode (address translation is disabled), storage attribute control registers control the corresponding storage attributes. These registers are:

- Data Cache Write-through Register (DCWR)
- Data Cache Cachability Register (DCCR)
- Instruction Cache Cachability Register (ICCR)
- Storage Guarded Register (SGR)
- Storage Little-Endian Register (SLER)
- Storage User-defined 0 Register (SU0R)

Each storage attribute control register contains 32 bits; each bit controls one of thirty-two 128MB storage attribute control regions. Bit 0 of each register controls the lowest-order region, with ascending bits controlling ascending regions in memory. The storage attributes in each storage attribute region are set independently of each other and of the storage attributes for other regions.

3.3 Registers

All PPC405GP registers are listed in this section. Some of the frequently-used registers are described in detail. Other registers are covered in their respective topic chapters (for example, the cache registers are described in Chapter 4, "Cache Operations"). All registers are summarized in Chapter 25, "Register Summary."

The registers are grouped into categories: General Purpose Registers (GPRs), Special Purpose Registers (SPRs), Time Base Registers (TBRs), the Machine State Register (MSR), the Condition Register (CR), Device Control Registers (DCRs), and memory-mapped I/O registers (MMIO). Different instructions are used to access each category of registers.

For all registers with fields marked as *reserved*, the reserved fields should be written as 0 and read as *undefined*. That is, when writing to a register with a reserved field, write a 0 to the reserved field. When reading from a register with a reserved field, ignore that field.

Programming Note: A good coding practice is to perform the initial write to a register with reserved fields as described, and to perform all subsequent writes to the register using a read-modify-write strategy: read the register, use logical instructions to alter defined fields, leaving reserved fields unmodified, and write the register.

Figure 3-1 on page 3-5 illustrates the registers in the user and supervisor programming models.

User Model

General-Purpose Registers

GPR0
GPR1
• • •
GPR31

SPR General Registers (read-only)

SPRG4	SPR 0x104
SPRG5	SPR 0x105
SPRG5	SPR 0x106
SPRG7	SPR 0x107

User SPR General Register 0 (read/write)

USPRG0	SPR 0x100

Condition Register

CR

Fixed-Point Exception Register

XER	SPR 0x001

Link Register

LR	SPR 0x008

Count Register

CTR	SPR 0x009

Time Base Registers (read-only)

TBL	TBR 0x10C
TBU	TBR 0x10D

Storage Attribute Control Registers

DCCR	SPR 0x3FA
DCWR	SPR 0x3BA
ICCR	SPR 0x3FB
SGR	SPR 0x3B9
SLER	SPR 0x3BB
SU0R	SPR 0x3BC

Supervisor Model

Machine State Register

MSR

Core Configuration Register

CCR0	SPR 0x3B3

SPR General Registers

SPRG0	SPR 0x110
SPRG1	SPR 0x111
SPRG2	SPR 0x112
SPRG3	SPR 0x113
SPRG4	SPR 0x114
SPRG5	SPR 0x115
SPRG6	SPR 0x116
SPRG7	SPR 0x117

Exception Handling Registers

Exception Vector Prefix Register

EVPR	SPR 0x3D5

Exception Syndrome Register

ESR	SPR 0x3D4

Data Exception Address Register

DEAR	SPR 0x3D5

Save/Restore Registers

SRR0	SPR 0x01A
SRR1	SPR 0x01B
SRR2	SPR 0x3DE
SRR3	SPR 0x3DF

Memory Management Registers

Process ID

PID	SPR 0x3B1

Zone Protection Register

ZPR	SPR 0x3B0

Processor Version Register

PVR	SPR 0x11F

Timer Facilities

Time Base Registers

TBL	SPR 0x11C
TBU	SPR 0x11D

Timer Control Register

TCR	SPR 0x3DA

Timer Status Register

TSR	SPR 0x3D8

Programmable Interval Timer

PIT	SPR 0x3DB

Debug Registers

Debug Status Register

DBSR	SPR 0x3F0

Debug Control Registers

DBCR0	SPR 0x3F2
DBCR1	SPR 0x3BD

Data Address Compares

DAC1	SPR 0x3F6
DAC2	SPR 0x3F7

Data Value Compares

DVC1	SPR 0x3B6
DVC2	SPR 0x3B7

Instruction Address Compares

IAC1	SPR 0x3F4
IAC2	SPR 0x3F5
IAC3	SPR 0x3B4
IAC4	SPR 0x3B5

Instruction Cache Debug Data Register

ICDBR	SPR 0x3D3

Figure 3-1. PPC405GP Programming Model—Registers

3.3.1 General Purpose Registers (R0-R31)

The PPC405GP contains thirty-two 32-bit general purpose registers (GPRs). Data from memory can be read into GPRs using load instructions and the contents of GPRs can be written to memory using store instructions. Most integer instructions use GPRs for source and destination operands. See Table 25-1, "PPC405GP General Purpose Registers," on page 25-1 for the numbering of the GPRs.

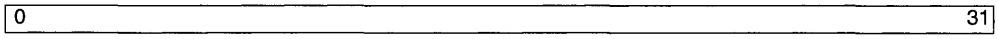


Figure 3-2. General Purpose Registers (R0-R31)

0:31		General Purpose Register data

3.3.2 Special Purpose Registers

Special purpose registers (SPRs), which are part of the PowerPC Architecture and the IBM PowerPC Embedded Environment, are accessed using the **mtspr** and **mfspr** instructions.

SPRs control the operation of debug facilities, timers, interrupts, storage control attributes, and other architected processor resources. Table 25-2, "Special Purpose Registers," on page 25-2 shows the mnemonic, name, and number for each SPR. Table 3-2, "PPC405GP SPRs," on page 3-7 lists the PPC405GP SPRs by function and indicates the pages where the SPRs are described more fully.

Except for the Link Register (LR), the Count Register (CTR), the Fixed-point Exception Register (XER), User SPR General 0 (USPRG0, and read access to SPR General 4–7 (SPRG4–SPRG7), all SPRs are privileged. As SPRs, the registers TBL and TBU are privileged write-only; as TBRs, these registers can be read in user mode. Unless used to access non-privileged SPRs, attempts to execute **mfspr** and **mtspr** instructions while in user mode cause privileged violation program interrupts. See "Privileged SPRs" on page 3-42.

Table 3-2. PPC405GP SPRs

Function	Register				Access	Page
Configuration	CCR0				Privileged	4-11
Branch Control	CTR				User	3-7
	LR				User	3-8
Debug	DAC1	DAC2			Privileged	12-14
	DBCR0	DBCR1			Privileged	12-9
	DBSR				Privileged	12-12
	DVC1	DVC2			Privileged	12-15
	IAC1	IAC2	IAC3	IAC4	Privileged	12-14
	ICDBDR				Privileged	4-14
Fixed-point Exception	XER				User	3-8
General-Purpose SPR	SPRG0	SPRG1	SPRG2	SPRG3	Privileged	3-11
	SPRG4	SPRG5	SPRG6	SPRG7	User read, privileged write	3-11
	USPRG0				User	3-11
Interrupts and Exceptions	DEAR				Privileged	10-34
	ESR				Privileged	10-31
	EVPR				Privileged	10-31
	SRR0	SRR1			Privileged	10-29
	SRR2	SRR3			Privileged	10-30
Processor Version	PVR				Privileged, read-only	3-12
Storage Attribute Control	DCCR				Privileged	6-17
	DCWR				Privileged	6-17
	ICCR				Privileged	6-17
	SGR				Privileged	6-17
	SLER				Privileged	6-17
	SU0R				Privileged	6-17
Timer Facilities	TBL	TBU			Privileged, write-only	11-2
	PIT				Privileged	11-2
	TCR				Privileged	11-9
	TSR				Privileged	11-8
Zone Protection	ZPR				Privileged	6-13

3.3.2.1 Count Register (CTR)

The CTR is written from a GPR using **mtspr**. The CTR contents can be used as a loop count that is decremented and tested by some branch instructions. Alternatively, the CTR contents can specify a target address for the **bcctr** instruction, enabling branching to any address.

The CTR is in the user programming model.

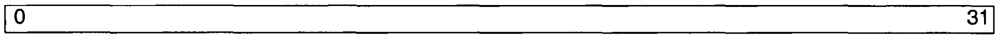


Figure 3-3. Count Register (CTR)

0:31		Count	Used as count for branch conditional with decrement instructions, or as address for branch-to-counter instructions.

3.3.2.2 Link Register (LR)

The LR is written from a GPR using **mtspr**, and by branch instructions that have the LK bit set to 1. Such branch instructions load the LR with the address of the instruction following the branch instruction. Thus, the LR contents can be used as the return address for a subroutine that was called using the branch.

The LR contents can be used as a target address for the **bclr** instruction. This allows branching to any address.

When the LR contents represent an instruction address, $LR_{30:31}$ are assumed to be 0, because all instructions must be word-aligned. However, when LR is read using **mfspr**, all 32 bits are returned as written.

The LR is in the user programming model.

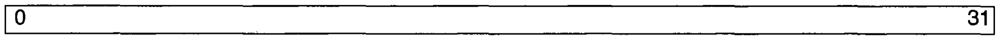


Figure 3-4. Link Register (LR)

0:31		Link Register contents	If (LR) represents an instruction address, $LR_{30:31}$ should be 0.

3.3.2.3 Fixed Point Exception Register (XER)

The XER records overflow and carry conditions generated by integer arithmetic instructions.

The Summary Overflow (SO) field is set to 1 when instructions cause the Overflow (OV) field to be set to 1. The SO field does not necessarily indicate that an overflow occurred on the most recent arithmetic operation, but that an overflow occurred since the last clearing of XER[SO]. **mtspr**(XER) sets XER[SO, OV] to the value of bit positions 0 and 1 in the source register, respectively.

Once set, XER[SO] is not reset until an **mtspr**(XER) is executed with data that explicitly puts a 0 in the SO bit, or until an **mcrxr** instruction is executed.

XER[OV] is set to indicate whether an instruction that updates XER[OV] produces a result that "overflows" the 32-bit target register. XER[OV] = 1 indicates overflow. For arithmetic operations, this occurs when an operation has a carry-in to the most-significant bit of the result that does not equal the carry-out of the most-significant bit (that is, the exclusive-or of the carry-in and the carry-out is 1).

The following instructions set XER[OV] differently. The specific behavior is indicated in the instruction descriptions in Chapter 24, "Instruction Set."

- Move instructions:

  mcrxr, mtspr(XER)

- Multiply and divide instructions:

  mullwo, mullwo., divwo, divwo., divwuo, divwuo

The Carry (CA) field is set to indicate whether an instruction that updates XER[CA] produces a result that has a carry-out of the most-significant bit. XER[CA] = 1 indicates a carry.

The following instructions set XER[CA] differently.The specific behavior is indicated in the instruction descriptions in Chapter 24, "Instruction Set."

- Move instructions

  mcrxr, mtspr(XER)

- Shift-algebraic operations

  sraw, srawi

The Transfer Byte Count (TBC) field is the byte count for load/store string instructions.

The XER is part of the user programming model.

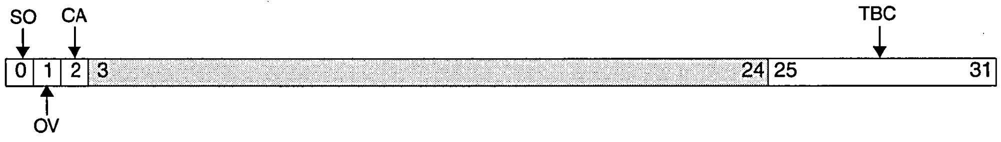


Figure 3-5. Fixed Point Exception Register (XER)

0	SO	Summary Overflow 0 No overflow has occurred. 1 Overflow has occurred.	Can be *set* by **mtspr** or by using "o" form instructions; can be *reset* by **mtspr** or by **mcrxr**.
1	OV	Overflow 0 No overflow has occurred. 0 Overflow has occurred.	Can be *set* by **mtspr** or by using "o" form instructions; can be *reset* by **mtspr**, by **mcrxr**, or "o" form instructions.
2	CA	Carry 0 Carry has not occurred. 1 Carry has occurred.	Can be *set* by **mtspr** or arithmetic instructions that update the CA field; can be *reset* by **mtspr**, by **mcrxr**, or by arithmetic instructions that update the CA field.
3:24	▨	Reserved	
25:31	TBC	Transfer Byte Count	Used by **lswx** and **stswx**; written by **mtspr.**

Table 3-3 and Table 3-4 list the PPC405GP instructions that update the XER. In the tables, the syntax "[**o**]" indicates that the instruction has an "o" form that updates XER[SO,OV], and a "non-o" form. The syntax "[.]" indicates that the instruction has a "record" form that updates CR[CR0] (see "Condition Register (CR)" on page 3-12), and a "non-record" form.

Table 3-3. XER[CA] Updating Instructions

Integer Arithmetic		**Integer Shift**	**Processor Control**
Add	**Subtract**	**Shift Right Algebraic**	**Register Management**
addc[o][.] **adde[o][.]** **addic[.]** **addme[o][.]** **addze[o][.]**	**subfc[o][.]** **subfe[o][.]** **subfic** **subfme[o][.]** **subfze[o][.]**	**sraw[.]** **srawi[.]**	**mtspr** **mcrxr**

Table 3-4. XER[SO,OV] Updating Instructions

Integer Arithmetic					Auxiliary Processor		Processor Control
Add	Subtract	Multiply	Divide	Negate	Multiply-Accumulate	Negative Multiply-Accumulate	Register Management
addo[.]	subfo[.]	mullwo[.]	divwo[.]	nego[.]	macchwo[.]	nmacchwo[.]	mtspr
addco[.]	subfco[.]		divwuo[.]		macchwso[.]	nmacchwso[.]	mcrxr
addeo[.]	subfeo[.]				macchwsuo[.]	nmachhwo[.]	
addmeo[.]	subfmeo[.]				macchwuo[.]	nmachhwso[.]	
addzeo[.]	subfzeo[.]				machhwo[.]	nmaclhwo[.]	
					machhwso[.]	nmaclhwso[.]	
					machhwsuo[.]		
					machhwuo[.]		
					maclhwo[.]		
					maclhwso[.]		
					maclhwsuo[.]		
					maclhwuo[.]		

3.3.2.4 Special Purpose Register General (SPRG0–SPRG7)

USPRG0 and SPRG0–SPRG7 are provided for general purpose software use. For example, these registers are used as temporary storage locations. For example, an interrupt handler might save the contents of a GPR to an SPRG, and later restore the GPR from it. This is faster than a save/restore to a memory location. These registers are written using **mtspr** and read using **mfspr**.

Access to USPRG0 is non-privileged for both read and write.

SPRG0–SPRG7 provide temporary storage locations. For example, an interrupt handler might save the contents of a GPR to an SPRG, and later restore the GPR from it. This is faster than performing a save/restore to memory. These registers are written by **mtspr** and read by **mfspr**.

Access to SPRG0–SPRG7 is privileged, except for read access to SPRG4–SPRG7. See “Privileged SPRs” on page 3-42 for more information.

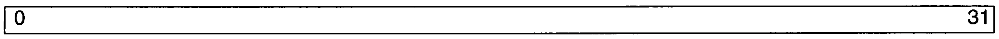


Figure 3-6. Special Purpose Register General (SPRG0–SPRG7)

0:31		General data	Software value; no hardware usage.

3.3.2.5 Processor Version Register (PVR)

The PVR is a read-only register that uniquely identifies a standard product or Core+ASIC implementation. Software can examine the PVR to recognize implementation-dependent features and determine available hardware resources.

Access to the PVR is privileged. See "Privileged SPRs" on page 3-42 for more information.

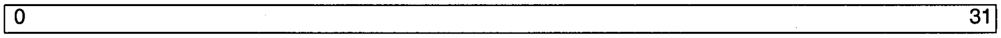


Figure 3-7. Processor Version Register (PVR)

0:31		Assigned PVR value

3.3.3 Condition Register (CR)

The CR contains eight 4-bit fields (CR0–CR7), as shown in Figure 3-8. The fields contain conditions detected during the execution of integer or logical compare instructions, as indicated in the instruction descriptions in Chapter 24, "Instruction Set." The CR contents can be used in conditional branch instructions.

The CR can be modified in any of the following ways:

- **mtcrf** sets specified CR fields by writing to the CR from a GPR, under control of a mask specified as an instruction field.
- **mcrf** sets a specified CR field by copying another CR field to it.
- **mcrxr** copies certain bits of the XER into a designated CR field, and then clears the corresponding XER bits.
- The "with update" forms of integer instructions implicitly update CR[CR0].
- Integer compare instructions update a specified CR field.
- The CR-logical instructions update a specified CR bit with the result of a logical operation on a specified pair of CR bit fields.
- Conditional branch instructions can test a CR bit as one of the branch conditions.

If a CR field is set by a compare instruction, the bits are set as described in "CR Fields after Compare Instructions."

The CR is part of the user programming model.

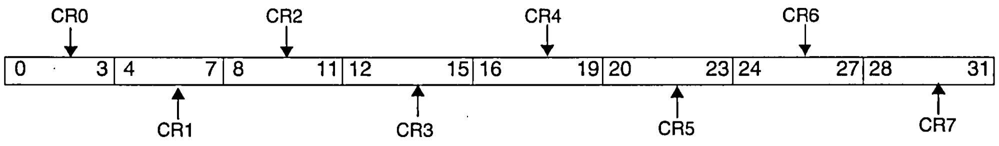


Figure 3-8. Condition Register (CR)

0:3	CR0	Condition Register Field 0
4:7	CR1	Condition Register Field 1
8:11	CR2	Condition Register Field 2
12:15	CR3	Condition Register Field 3
16:19	CR4	Condition Register Field 4
20:23	CR5	Condition Register Field 5
24:27	CR6	Condition Register Field 6
28:31	CR7	Condition Register Field 7

3.3.3.1 CR Fields after Compare Instructions

Compare instructions compare the values of two 32-bit registers. The two types of compare instructions, *arithmetic* and *logical*, are distinguished by the interpretation given to the 32-bit values. For *arithmetic* compares, the values are considered to be signed, where 31 bits represent the magnitude and the most-significant bit is a sign bit. For *logical* compares, the values are considered to be unsigned, so all 32 bits represent magnitude. There is no sign bit. As an example, consider the comparison of 0 with 0xFFFF FFFF. In an *arithmetic* compare, 0 is larger, because 0xFFFF FFFF represents –1; in a *logical* compare, 0xFFFF FFFF is larger.

A compare instruction can direct its CR update to any CR field. The first data operand of a compare instruction specifies a GPR. The second data operand specifies another GPR, or immediate data derived from the IM field of the immediate instruction form. The contents of the GPR specified by the first data operand are compared with the contents of the GPR specified by the second data operand (or with the immediate data). See descriptions of the compare instructions (page 24-34 through page 24-37) for precise details.

After a compare, the specified CR field is interpreted as follows:

LT (bit 0)	The first operand is less than the second operand.
GT (bit 1)	The first operand is greater than the second operand.
EQ (bit 2)	The first operand is equal to the second operand.
SO (bit 3)	Summary overflow; a copy of XER[SO].

3.3.3.2 The CR0 Field

After the execution of compare instructions that update CR[CR0], CR[CR0] is interpreted as described in "CR Fields after Compare Instructions" on page 3-13. The "dot" forms of arithmetic and logical instructions also alter CR[CR0]. After most instructions that update CR[CR0], the bits of CR0 are interpreted as follows:

LT (bit 0)	Less than 0; set if the most-significant bit of the 32-bit result is 1.
GT (bit 1)	Greater than 0; set if the 32-bit result is non-zero and the most-significant bit of the result is 0.
EQ (bit 2)	Equal to 0; set if the 32-bit result is 0.
SO (bit 3)	Summary overflow; a copy of XER[SO] at instruction completion.

The CR[CR0]$_{LT, GT, EQ}$ subfields are set as the result of an algebraic comparison of the instruction result to 0, regardless of the type of instruction that sets CR[CR0]. If the instruction result is 0, the EQ subfield is set to 1. If the result is not 0, either LT or GT is set, depending on the value of the most-significant bit of the result.

When updating CR[CR0], the most significant bit of an instruction result is considered a sign bit, even for instructions that produce results that are not usually thought of as signed. For example, logical instructions such as **and.**, **or.**, and **nor.** update CR[CR0]$_{LT, GT, EQ}$ using such an arithmetic comparison to 0, although the result of such a logical operation is not actually an arithmetic result.

If an arithmetic overflow occurs, the "sign" of an instruction result indicated in CR[CR0]$_{LT, GT, EQ}$ might not represent the "true" (infinitely precise) algebraic result of the instruction that set CR0. For example, if an **add.** instruction adds two large positive numbers and the magnitude of the result cannot be represented as a twos-complement number in a 32-bit register, an overflow occurs and CR[CR0]$_{LT, SO}$ are set, although the infinitely precise result of the add is positive.

Adding the largest 32-bit twos-complement negative number, 0x8000 0000, to itself results in an arithmetic overflow and 0x0000 0000 is recorded in the target register. CR[CR0]$_{EQ, SO}$ is set, indicating a result of 0, but the infinitely precise result is negative.

The CR[CR0]$_{SO}$ subfield is a copy of XER[SO]. Instructions that do not alter the XER[SO] bit cannot cause an overflow, but even for these instructions CR[CR0]$_{SO}$ is a copy of XER[SO].

Some instructions set CR[CR0] differently or do not specifically set any of the subfields. These instructions include:

- Compare instructions

  cmp, **cmpi**, **cmpl**, **cmpli**
- CR logical instructions

  crand, **crandc**, **creqv**, **crnand**, **crnor**, **cror**, **crorc**, **crxor**, **mcrf**
- Move CR instructions

  mtcrf, **mcrxr**
- **stwcx.**

The instruction descriptions provide detailed information about how the listed instructions alter CR[CR0].

3.3.4 The Time Base

The PowerPC Architecture provides a 64-bit time base. "Time Base" on page 11-2 describes the architected time base. Access to the time base is through two 32-bit time base registers (TBRs). The least-significant 32 bits of the time base are read from the Time Base Lower (TBL) register and the most-significant 32 bits are read from the Time Base Upper (TBU) register.

User-mode access to the time base is read-only, and there is no explicitly privileged read access to the time base.

The **mftb** instruction reads from TBL and TBU. Writing the time base is accomplished by moving the contents of a GPR to a pair of SPRs, which are also called TBL and TBU, using **mtspr**.

Table 3-5 shows the mnemonics and names of the TBRs.

Table 3-5. Time Base Registers

Mnemonic	Register Name	Access
TBL	Time Base Lower (Read-only)	Read-only
TBU	Time Base Upper (Read-only)	Read-only

3.3.5 Machine State Register (MSR)

The Machine State Register (MSR) controls processor core functions, such as the enabling or disabling of interrupts and address translation.

The MSR is written from a GPR using the **mtmsr** instruction. The contents of the MSR can be read into a GPR using the **mfmsr** instruction. MSR[EE] is set or cleared using the **wrtee** or **wrteei** instructions.

The MSR contents are automatically saved, altered, and restored by the interrupt-handling mechanism. See "Machine State Register (MSR)" on page 10-28.

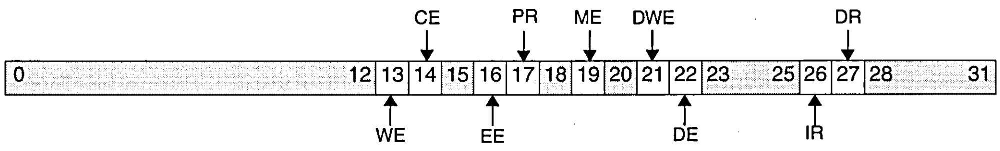


Figure 3-9. Machine State Register (MSR)

0:12		Reserved	
13	WE	Wait State Enable 0 The processor is not in the wait state. 1 The processor is in the wait state.	If MSR[WE] = 1, the processor remains in the wait state until an interrupt is taken, a reset occurs, or an external debug tool clears WE.
14	CE	Critical Interrupt Enable 0 Critical interrupts are disabled. 1 Critical interrupts are enabled.	Controls the critical interrupt input and watchdog timer first time-out interrupts.
15		Reserved	

16	EE	External Interrupt Enable 0 Asynchronous interrupts (external to the processor core) are disabled. 1 Asynchronous interrupts are enabled.	Controls the non-critical external interrupt input, PIT, and FIT interrupts.
17	PR	Problem State 0 Supervisor state (all instructions allowed). 1 Problem state (some instructions not allowed).	
18		Reserved	
19	ME	Machine Check Enable 0 Machine check interrupts are disabled. 1 Machine check interrupts are enabled.	
20		Reserved	
21	DWE	Debug Wait Enable 0 Debug wait mode is disabled. 1 Debug wait mode is enabled.	
22	DE	Debug Interrupts Enable 0 Debug interrupts are disabled. 1 Debug interrupts are enabled.	
23:25		Reserved	
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.	
27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.	
28:31		Reserved	

3.3.6 Device Control Registers

Device Control Registers (DCRs), on-chip registers that exist architecturally outside the processor core, are not part of the IBM PowerPC Embedded Environment. The Embedded Environment simply defines the existence of a DCR address space and the instructions that access the DCRs, but does not define any DCRs. The instructions that access the DCRs are **mtdcr** (move to device control register) and **mfdcr** (move from device control register).

DCRs are used to control the operations of on-chip buses, peripherals, and some processor behavior.

3.3.6.1 Directly Accessed DCRs

The following DCRs are directly accessed; that is, they are accessed using their DCR numbers.

Table 3-6. Directly Accessed DCRs

Register	DCR Number	Access	Description
DCRs Used for Indirect Access			
SDRAM0_CFGADDR	0x010	R/W	Memory Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	Memory Controller Data Register
EBC0_CFGADDR	0x012	R/W	Peripheral Controller Address Register
EBC0_CFGDATA	0x013	R/W	Peripheral Controller Data Register
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register
On-Chip Memory			
OCM0_ISARC	0x018	R/W	OCM Instruction-Side Address Range Compare Register
OCM0_ISCNTL	0x019	R/W	OCM Instruction-Side Control Register
OCM0_DSARC	0x01A	R/W	OCM Data-Side Address Range Compare Register
OCM0_DSCNTL	0x01B	R/W	OCM Data-Side Control Register
On-Chip Buses			
PLB0_BESR	0x084	R/Clear	PLB Bus Error Status Register
PLB0_BEAR	0x086	R/W	PLB Bus Error Address Register
PLB0_ACR	0x087	R/W	PLB Arbiter Control Register
POB0_BESR0	0x0A0	R/Clear	PLB to OPB Bus Error Status Register 0
POB0_BEAR	0x0A2	R	PLB to OPB Bus Error Address Register
POB0_BESR1	0x0A4	R/Clear	PLB to OPB Bus Error Status Register 1
Clocking, Power Management, and Chip Control			
CPC0_PLLMR	0x0B0	R/W	PLL Mode Register
CPC0_CR0	0x0B1	R/W	Chip Control Register 0
CPC0_CR1	0x0B2	R/W	Chip Control Register 1
CPC0_PSR	0x0B4	R	Chip Pin Strapping Register
CPC0_JTAGID	0x0B5	R	JTAG ID Register
CPC0_SR	0x0B8	R	CPM Status Register
CPC0_ER	0x0B9	R/W	CPM Enable Register
CPC0_FR	0x0BA	R/W	CPM Force Register
Universal Interrupt Controller			
UIC0_SR	0x0C0	R/Clear	UIC Status Register
UIC0_ER	0x0C2	R/W	UIC Enable Register
UIC0_CR	0x0C3	R/W	UIC Critical Register
UIC0_PR	0x0C4	R/W	UIC Polarity Register
UIC0_TR	0x0C5	R/W	UIC Triggering Register
UIC0_MSR	0x0C6	R	UIC Masked Status Register
UIC0_VR	0x0C7	R	UIC Vector Register
UIC0_VCR	0x0C8	W	UIC Vector Configuration Register

Table 3-6. Directly Accessed DCRs (continued)

Register	DCR Number	Access	Description
Direct Memory Access			
DMA0_CR0	0x100	R/W	DMA Channel Control Register 0
DMA0_CT0	0x101	R/W	DMA Count Register 0
DMA0_DA0	0x102	R/W	DMA Destination Address Register 0
DMA0_SA0	0x103	R/W	DMA Source Address Register 0
DMA0_SG0	0x104	R/W	DMA Scatter/Gather Descriptor Address Register 0
DMA0_CR1	0x108	R/W	DMA Channel Control Register 1
DMA0_CT1	0x109	R/W	DMA Count Register 1
DMA0_DA1	0x10A	R/W	DMA Destination Address Register 1
DMA0_SA1	0x10B	R/W	DMA Source Address Register 1
DMA0_SG1	0x10C	R/W	DMA Scatter/Gather Descriptor Address Register 1
DMA0_CR2	0x110	R/W	DMA Channel Control Register 2
DMA0_CT2	0x111	R/W	DMA Count Register 2
DMA0_DA2	0x112	R/W	DMA Destination Address Register 2
DMA0_SA2	0x113	R/W	DMA Source Address Register 2
DMA0_SG2	0x114	R/W	DMA Scatter/Gather Descriptor Address Register 2
DMA0_CR3	0x118	R/W	DMA Channel Control Register 3
DMA0_CT3	0x119	R/W	DMA Count Register 3
DMA0_DA3	0x11A	R/W	DMA Destination Address Register 3
DMA0_SA3	0x11B	R/W	DMA Source Address Register 3
DMA0_SG3	0x11C	R/W	DMA Scatter/Gather Descriptor Address
DMA0_SR	0x120	R/Clear	DMA Status Register
DMA0_SGC	0x123	R/W	DMA Scatter/Gather Command Register
DMA0_SLP	0x125	R/W	DMA Sleep Mode Register
DMA0_POL	0x126	R/W	DMA Polarity Configuration Register
Media Access Layer			
MAL0_CFG	0x180	R/W	MAL Configuration Register
MAL0_ESR	0x181	R/Clear	Error Status Register
MAL0_IER	0x182	R/W	Interrupt Enable Register
MAL0_TXCASR	0x184	R/W	Tx Channel Active Register (Set)
MAL0_TXCARR	0x185	R/W	Tx Channel Active Register (Reset)
MAL0_TXEOBISR	0x186	R/Clear	Tx End of Buffer Interrupt Status Register
MAL0_TXDEIR	0x187	R/Clear	Tx Descriptor Error Interrupt Register
MAL0_RXCASR	0x190	R/W	Rx Channel Active Register (Set)
MAL0_RXCARR	0x191	R/W	Rx Channel Active Register (Reset)
MAL0_RXEOBISR	0x192	R/Clear	Rx End of Buffer Interrupt Status Register
MAL0_RXDEIR	0x193	R/Clear	Rx Descriptor Error Interrupt Register
MAL0_TXCTP0R	0x1A0	R/W	Channel Tx 0 Channel Table Pointer Register
MAL0_TXCTP1R	0x1A1	R/W	Channel Tx 1 Channel Table Pointer Register
MAL0_RXCTP0R	0x1C0	R/W	Channel Rx 0 Channel Table Pointer Register
MAL0_RCBS0	0x1E0	R/W	Channel RX 0 Channel Buffer Size Register

3.3.6.2 Indirectly Accessed DCRs

The DCRs for the SDRAM controller, external bus controller (EBC), and decompression controller are indirectly accessed.

3.3.7 Indirect Access of SDRAM Controller DCRs

The following procedure accesses the SDRAM controller DCRs listed in Table 3-7.

1. Write the offset from Table 3-8 to the Memory Controller Address Register (SDRAM0_CFGADDR).
2. Read data from or write data to the Memory Controller Data Register (SDRAM0_CFGDATA).

Table 3-7. SDRAM Controller DCR Usage

Register	DCR Number	Access	Description
SDRAM0_CFGADDR	0x010	R/W	Memory Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	Memory Controller Data Register

Table 3-8. Offsets for SDRAM Controller Registers

Register	Offset	R/W	Description
SDRAM0_BESR0	0x00	R/Clear	Bus Error Syndrome Register 0
SDRAM0_BESR1	0x08	R/Clear	Bus Error Syndrome Register 1
SDRAM0_BEAR	0x10	R/W	Bus Error Address Register
SDRAM0_CFG	0x20	R/W	Memory Controller Options 1
SDRAM0_RTR	0x30	R/W	Refresh Timer Register
SDRAM0_PMIT	0x34	R/W	Power Management Idle Timer
SDRAM0_B0CR	0x40	R/W	Memory Bank 0 Configuration Register
SDRAM0_B1CR	0x44	R/W	Memory Bank 1 Configuration Register
SDRAM0_B2CR	0x48	R/W	Memory Bank 2 Configuration Register
SDRAM0_B3CR	0x4C	R/W	Memory Bank 3 Configuration Register
SDRAM0_TR	0x80	R/W	SDRAM Timing Register 1
SDRAM0_ECCCFG	0x94	R/W	ECC Configuration
SDRAM0_ECCESR	0x98	R	ECC Error Status Register

3.3.8 Indirect Access of External Bus Controller DCRs

The following procedure accesses the EBC DCRs listed in Table 3-9.

1. Write the offset from Table 3-10 to the Peripheral Controller Address Register (EBC0_CFGADDR).
2. Read data from or write data to the Peripheral Controller Data Register (EBC0_CFGDATA).

Table 3-9. External Bus Controller DCR Usage

Register	DCR Number	Access	Description
EBC0_CFGADDR	0x012	R/W	Peripheral Controller Address Register
EBC0_CFGDATA	0x013	R/W	Peripheral Controller Data Register

Table 3-10. Offsets for External Bus Controller Registers

Register	Offset	Access	Description
EBC0_B0CR	0x00	R/W	Peripheral Bank 0 Configuration Register
EBC0_B1CR	0x01	R/W	Peripheral Bank 1 Configuration Register
EBC0_B2CR	0x02	R/W	Peripheral Bank 2 Configuration Register
EBC0_B3CR	0x03	R/W	Peripheral Bank 3 Configuration Register
EBC0_B4CR	0x04	R/W	Peripheral Bank 4 Configuration Register
EBC0_B5CR	0x05	R/W	Peripheral Bank 5 Configuration Register
EBC0_B6CR	0x06	R/W	Peripheral Bank 6 Configuration Register
EBC0_B7CR	0x07	R/W	Peripheral Bank 7 Configuration Register
EBC0_B0AP	0x10	R/W	Peripheral Bank 0 Access Parameters
EBC0_B1AP	0x11	R/W	Peripheral Bank 1 Access Parameters
EBC0_B2AP	0x12	R/W	Peripheral Bank 2 Access Parameters
EBC0_B3AP	0x13	R/W	Peripheral Bank 3 Access Parameters
EBC0_B4AP	0x14	R/W	Peripheral Bank 4 Access Parameters
EBC0_B5AP	0x15	R/W	Peripheral Bank 5 Access Parameters
EBC0_B6AP	0x16	R/W	Peripheral Bank 6 Access Parameters
EBC0_B7AP	0x17	R/W	Peripheral Bank 7 Access Parameters
EBC0_BEAR	0x20	R/W	Peripheral Bus Error Address Register
EBC0_BESR0	0x21	R/W	Peripheral Bus Error Status Register 0
EBC0_BESR1	0x22	R/W	Peripheral Bus Error Status Register 1
EBC0_CFG	0x23	R/W	External Peripheral Control Register

3.3.9 Indirect Access of Decompression Controller DCRs

The following procedure accesses the decompression controller DCRs listed in Table 3-11.

1. Write the offset from Table 3-12 to the Decompression Controller Address Register (DCP0_CFGADDR).
2. Read data from or write data to the Decompression Controller Data Register (DCP0_CFGDATA).

Table 3-11. Decompression Controller DCR Usage

Register	DCR Number	Access	Description
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register

Table 3-12. Offsets for Decompression Controller Registers

Register	Offset	R/W	Description
DCP0_ITOR0	0x00	R/W	Index Table Origin Register 0
DCP0_ITOR1	0x01	R/W	Index Table Origin Register 1
DCP0_ITOR2	0x02	R/W	Index Table Origin Register 2
DCP0_ITOR3	0x03	R/W	Index Table Origin Register 3
DCP0_ADDR0	0x04	R/W	Address Decode Definition Register 0
DCP0_ADDR1	0x05	R/W	Address Decode Definition Register 1
DCP0_CFG	0x40	R/W	Decompression Controller Configuration Register
DCP0_ID	0x41	R	Decompression Controller ID Register
DCP0_VER	0x42	R	Decompression Controller Version Number Register
DCP0_PLBBEAR	0x50	R	Bus Error Address Register (PLB address)
DCP0_MEMBEAR	0x51	R	Bus Error Address Register (DCP to EBC address)
DCP0_ESR	0x52	R/Clear	Bus Error Status Register 0 (masters 0-3)
DCP0_RAM0–DCP0_RAM3FF	0x400–0x7FF	R/W	Decode Tables

3.3.10 Memory-Mapped Input/Output Registers

Some registers associated with on-chip peripherals are memory-mapped input/output (MMIO) registers. Such registers are mapped into the system memory space and are accessed using load/store instructions.

3.3.10.1 Directly Accessed MMIO Registers

Directly-accessed MMIO registers are accessed using load/store instructions that contain the register addresses. Table 3-13 lists the directly-accessed MMIO registers.

Table 3-13. Directly Accessed MMIO Registers

Register	Address	Access	Description
MMIO Registers Used for Indirect Access			
PCIC0_CFGADDR	0xEEC00000	R/W	PCI Configuration Address Register
PCIC0_CFGDATA	0xEEC00004	R/W	PCI Configuration Data Register
PCI-to-PLB Bridge			
PCIL0_PMM0LA	0xEF400000	R/W	PMM 0 Local Address
PCIL0_PMM0MA	0xEF400004	R/W	PMM 0 Mask/Attribute
PCIL0_PMM0PCILA	0xEF400008	R/W	PMM 0 PCI Low Address
PCIL0_PMM0PCIHA	0xEF40000C	R/W	PMM 0 PCI High Address
PCIL0_PMM1LA	0xEF400010	R/W	PMM 1 Local Address
PCIL0_PMM1MA	0xEF400014	R/W	PMM 1 Mask/Attribute
PCIL0_PMM1PCILA	0xEF400018	R/W	PMM 1 PCI Low Address
PCIL0_PMM1PCIHA	0xEF40001C	R/W	PMM 1 PCI High Address
PCIL0_PMM2LA	0xEF400020	R/W	PMM 2 Local Address
PCIL0_PMM2MA	0xEF400024	R/W	PMM 2 Mask/Attribute
PCIL0_PMM2PCILA	0xEF400028	R/W	PMM 2 PCI Low Address
PCIL0_PMM2PCIHA	0xEF40002C	R/W	PMM 2 PCI High Address
PCIL0_PTM1MS	0xEF400030	R/W	PTM 1 Memory Size
PCIL0_PTM1LA	0xEF400034	R/W	PTM 1 Local Address
PCIL0_PTM2MS	0xEF400038	R/W	PTM 2 Memory Size
PCIL0_PTM2LA	0xEF40003C	R/W	PTM 2 Local Address
Serial Ports			
UART0_RBR	0xEF600300	R	UART 0 Receiver Buffer Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_THR		W	UART 0 Transmitter Holding Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_DLL		R/W	UART 0 Baud-rate Divisor Latch LSB **Note:** Set UART0_LCR[DLAB] = 1 to access.
UART0_IER	0xEF600301	R/W	UART 0 Interrupt Enable Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_DLM		R/W	UART 0 Baud-rate Divisor Latch MSB **Note:** Set UART0_LCR[DLAB] = 1 to access.
UART0_IIR	0xEF600302	R	UART 0 Interrupt Identification Register
UART0_FCR	0xEF600302	W	UART 0 FIFO Control Register
UART0_LCR	0xEF600303	R/W	UART 0 Line Control Register

Table 3-13. Directly Accessed MMIO Registers (continued)

Register	Address	Access	Description
UART0_MCR	0xEF600304	R/W	UART 0 Modem Control Register
UART0_LSR	0xEF600305	R/W	UART 0 Line Status Register
UART0_MSR	0xEF600306	R/W	UART 0 Modem Status Register
UART0_SCR	0xEF600307	R/W	UART 0 Scratch Register
UART1_RBR	0xEF600400	R	UART 1 Receiver Buffer Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_THR		W	UART 1 Transmitter Holding Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_DLL		R/W	UART 1 Baud-rate Divisor Latch LSB **Note:** Set UART1_LCR[DLAB] = 1 to access.
UART1_IER	0xEF600401	R/W	UART 1 Interrupt Enable Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_DLM		R/W	UART 1 Baud-rate Divisor Latch MSB **Note:** Set UART1_LCR[DLAB] = 1 to access.
UART1_IIR	0xEF600402	R	UART 1 Interrupt Identification Register
UART1_FCR	0xEF600402	W	UART 1 FIFO Control Register
UART1_LCR	0xEF600403	R/W	UART 1 Line Control Register
UART1_MCR	0xEF600404	R/W	UART 1 Modem Control Register
UART1_LSR	0xEF600405	R/W	UART 1 Line Status Register
UART1_MSR	0xEF600406	R/W	UART 1 Modem Status Register
UART1_SCR	0xEF600407	R/W	UART 1 Scratch Register
Inter-Integrated Circuit			
IIC0_MDBUF	0xEF600500	R/W	IIC0 Master Data Buffer
IIC0_SDBUF	0xEF600502	R/W	IIC0 Slave Data Buffer
IIC0_LMADR	0xEF600504	R/W	IIC0 Low Master Address
IIC0_HMADR	0xEF600505	R/W	IIC0 High Master Address
IIC0_CNTL	0xEF600506	R/W	IIC0 Control
IIC0_MDCNTL	0xEF600507	R/W	IIC0 Mode Control
IIC0_STS	0xEF600508	R/W	IIC0 Status
IIC0_EXTSTS	0xEF600509	R/W	IIC0 Extended Status
IIC0_LSADR	0xEF60050A	R/W	IIC0 Low Slave Address
IIC0_HSADR	0xEF60050B	R/W	IIC0 High Slave Address
IIC0_CLKDIV	0xEF60050C	R/W	IIC0 Clock Divide
IIC0_INTRMSK	0xEF60050D	R/W	IIC0 Interrupt Mask
IIC0_XFRCNT	0xEF60050E	R/W	IIC0 Transfer Count
IIC0_XTCNTLSS	0xEF60050F	R/W	IIC0 Extended Control and Slave Status
IIC0_DIRECTCNTL	0xEF600510	R/W	IIC0 Direct Control
OPB Arbiter			
OPBA0_PR	0xEF600600	R/W	OPB Arbiter Priority Register
OPBA0_CR	0xEF600601	R/W	OPB Arbiter Control Register

Table 3-13. Directly Accessed MMIO Registers (continued)

Register	Address	Access	Description
General-Purpose I/O			
GPIO0_OR	0xEF600700	R/W	GPIO0_IRO Output Register
GPIO0_TCR	0xEF600704	R/W	GPIO0_IRO Three-State Control Register
GPIO0_ODR	0xEF600718	R/W	GPIO0_IRO Open Drain Register
GPIO0_IR	0xEF60071C	R	GPIO0_IRO Input Register
Ethernet			
EMAC0_MR0	0xEF600800	R/W	Mode Register 0
EMAC0_MR1	0xEF600804	R/W	Mode Register 1
EMAC0_TMR0	0xEF600808	R/W	Transmit Mode Register 0
EMAC0_TMR1	0xEF60080C	R/W	Transmit Mode Register 1
EMAC0_RMR	0xEF600810	R/W	Receive Mode Register
EMAC0_ISR	0xEF600814	R/W	Interrupt Status Register
EMAC0_ISER	0xEF600818	R/W	Interrupt Status Enable Register
EMAC0_IAHR	0xEF60081C	R/W	Individual Address High
EMAC0_IALR	0xEF600820	R/W	Individual Address Low
EMAC0_VTPID	0xEF600824	R/W	VLAN TPID Register
EMAC0_VTCI	0xEF600828	R/W	VLAN TCI Register
EMAC0_PTR	0xEF60082C	R/W	Pause Timer Register
EMAC0_IAHT1	0xEF600830	R/W	Individual Address Hash Table 1
EMAC0_IAHT2	0xEF600834	R/W	Individual Address Hash Table 2
EMAC0_IAHT3	0xEF600838	R/W	Individual Address Hash Table 3
EMAC0_IAHT4	0xEF60083C	R/W	Individual Address Hash Table 4
EMAC0_GAHT1	0xEF600840	R/W	Group Address Hash Table 1
EMAC0_GAHT2	0xEF600844	R/W	Group Address Hash Table 2
EMAC0_GAHT3	0xEF600848	R/W	Group Address Hash Table 3
EMAC0_GAHT4	0xEF60084C	R/W	Group Address Hash Table 4
EMAC0_LSAH	0xEF600850	R	Last Source Address Low
EMAC0_LSAL	0xEF600854	R	Last Source Address High
EMAC0_IPGVR	0xEF600858	R/W	Inter-Packet Gap Value Register
EMAC0_STACR	0xEF60085C	R/W	STA Control Register
EMAC0_TRTR	0xEF600860	R/W	Transmit Request Threshold Register
EMAC0_RWMR	0xEF600864	R/W	Receive Low/High Water Mark Register

3.3.10.2 Indirectly Accessed MMIO Registers

The PCI configuration registers, listed in Table 3-15, are indirectly accessed.

The following procedure accesses the PCI configuration registers, using the address and data registers listed in Table 3-14:

1. OR the Enable, Bus, Device, and Function fields of the PCI Configuration Address Register (PCIC0_CFGADDR) with the high-order 6 bits of the offset from Table 3-15 and write the result to the PCIC0_CFGADDR.
2. OR the low-order 2 bits of the offset from Table 3-15 with the address of the PCI Configuration Data Register (PCIC0_CFGDATA) to form an address.
3. Read data from or write data to the address.

Table 3-14. PCI Configuration Address and Data Registers

Register	Address	Access	Description
PCIC0_CFGADDR	0xEEC00000	R/W	PCI Configuration Address Register
PCIC0_CFGDATA	0xEEC00004	R/W	PCI Configuration Data Register

Table 3-15. PCI Configuration Registers

Register	Offset	Access		Description
		PLB	PCI	
PCIC0_VENDID	0x01–0x00	R/W	R	PCI Vendor ID
PCIC0_DEVID	0x03–0x02	R/W	R	PCI Device ID
PCIC0_CMD	0x05–0x04	R/W	R/W	PCI Command Register
PCIC0_STATUS	0x07–0x06	R/W	R/W	PCI Status Register
PCIC0_REVID	0x08	R/W	R/W	PCI Revision ID
PCIC0_CLS	0x0B–0x09	R/W	R	PCI Class Register
PCIC0_CACHELS	0x0C	R	R	PCI Cache Line Size
PCIC0_LATTIM	0x0D	R/W	R/W	PCI Latency Timer
PCIC0_HDTYPE	0x0E	R	R	PCI Header Type
PCIC0_BIST	0x0F	R	R	PCI Built In Self Test Control
PCIC0_ BAR0	0x13–0x10	R	R	PCI Reserved BAR 0
PCIC0_PTM1BAR	0x17–0x14	R/W	R/W	PCI PTM 1 BAR
PCIC0_PTM2BAR	0x1B–0x18	R/W	R/W	PCI PTM 2 BAR
PCIC0_ BAR3	0x1F–0x1C	—	—	PCI Reserved BAR 3
PCIC0_ BAR4	0x23–0x20	—	—	PCI Reserved BAR 4
PCIC0_ BAR5	0x24–0x27	—	—	PCI Reserved BAR 5
PCIC0_CISPTR	0x2B–0x28	—	—	Unused Cardbus CIS Pointer
PCIC0_SBSYSVID	0x2D–0x2C	R/W	R	PCI Subsystem Vendor ID
PCIC0_SBSYSID	0x2F–0x2E	R/W	R	PCI Subsystem ID
PCIC0_EROMBA	0x33–0x30	—	—	Unused Expansion ROM Base Address
PCIC0_CAP	0x34	R	R	PCI Capabilities Pointer
PCIC0_INTLN	0x3C	R/W	R/W	PCI Interrupt Line
PCIC0_INTPN	0x3D	R	R	PCI Interrupt Pin
PCIC0_MINGNT	0x3E	R	R	PCI Minimum Grant

Table 3-15. PCI Configuration Registers (continued)

Register	Offset	Access		Description
		PLB	PCI	
PCIC0_MAXLTNCY	0x3F	R	R	PCI Maximum Latency
PCIC0_ICS	0x44	R/W	R/W	PCI Interrupt Control/Status
PCIC0_ERREN	0x48	R/W	R/W	Error Enable
PCIC0_ERRSTS	0x49	R/W	R/W	Error Status
PCIC0_BRDGOPT1	0x4B–0x4A	R/W	R/W	PCI Bridge Options 1
PCIC0_PLBBESR0	0x4F–0x4C	R/W	R/W	PLB Slave Error Syndrome 0
PCIC0_PLBBESR1	0x53–0x50	R/W	R/W	PLB Slave Error Syndrome 1
PCIC0_PLBBEAR	0x57–0x54	R/W	R/W	PLB Slave Error Address Register
PCIC0_CAPID	0x58	R	R	Capability Identifier
PCIC0_NEXTIPTR	0x59	R	R	Next Item Pointer
PCIC0_PMC	0x5B–0x5A	R	R	Power Management Capabilities
PCIC0_PMCSR	0x5D–0x5C	R/W	R/W	Power Management Control Status
PCIC0_PMCSRBSE	0x5E	R	R	PMCSR PCI to PCI bridge Support Extensions
PCIC0_DATA	0x5F	—	—	Unused Data
PCIC0_BRDGOPT2	0x63–0x60	R/W	R/W	PCI Bridge Options 2
PCIC0_PMSCRR	0x64	R/W	R/W	Power Management State Change Request Register

3.4 Data Types and Alignment

The data types consist of bytes (eight bits), halfwords (two bytes), words (four bytes), and strings (1 to 128 bytes). Figure 3-10 shows the byte, halfword, and word data types and their bit and byte definitions for big endian representations of values. Note that PowerPC bit numbering is reversed from industry conventions; bit 0 represents the most significant bit of a value.

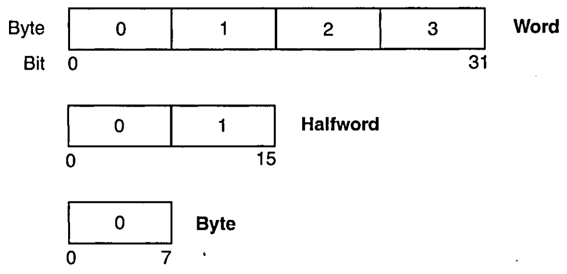


Figure 3-10. PPC405GP Data Types

Data is represented in either twos-complement notation or in an unsigned integer format; data representation is independent of alignment issues.

The address of an a data object is always the lowest address of any byte comprising the object.

All instructions are words, and are word-aligned (the lowest byte address is divisible by 4).

3.4.1 Alignment for Storage Reference and Cache Control Instructions

The storage reference instructions (loads and stores; see Table 3-23, "Storage Reference Instructions," on page 3-48) move data to and from storage. The data cache control instructions listed in Table 3-32, "Cache Management Instructions," on page 3-52, control the contents and operation of the data cache unit (DCU). Both types of instructions form an effective address (EA). The method of calculating the EA for the storage reference and cache control instructions is detailed in the description of those instructions. See Chapter 24, "Instruction Set," for more information.

Cache control instructions ignore the four least significant bits of the EA; no alignment restrictions exist in the DCU because of EAs. However, storage control attributes can cause alignment exceptions. When data address translation is disabled and a **dcbz** instruction references a storage region that is non-cachable, or for which write-through caching is the write strategy, an alignment exception is taken. Such exceptions result from the storage control attributes, not from EA alignment. The alignment exception enables system software to emulate the write-through function.

Alignment requirements for the storage reference instructions and the **dcread** instruction depend on the particular instruction. Table 3-16, "Alignment Exception Summary," on page 3-28, summarizes the instructions that cause alignment exceptions.

The data targets of instructions are of types that depend upon the instruction. The load/store instructions have the following "natural" alignments:

- Load/store word instructions have word targets, word-aligned.
- Load/ store halfword instructions have halfword targets, halfword-aligned.
- Load/store byte instructions have byte targets, byte-aligned (that is, any alignment).

Misalignments are addresses that are not naturally aligned on data type boundaries. An address not divisible by four is misaligned with respect to word instructions. An address not divisible by two is misaligned with respect to halfword instructions. The PPC405GP implementation handles misalignments within and across word boundaries, but there is a performance penalty because additional bus cycles are required.

3.4.2 Alignment and Endian Operation

The endian storage control attribute does not affect alignment behavior. In little endian storage regions, the alignment of data is treated as it is in big endian storage regions; no special alignment exceptions occur when accessing data in little endian storage regions. Note that the alignment exceptions that apply to big endian region accesses also apply to little endian storage region accesses.

3.4.3 Summary of Instructions Causing Alignment Exceptions

Table 3-16 summarizes the instructions that cause alignment exceptions and the conditions under which the alignment exceptions occur.

Table 3-16. Alignment Exception Summary

Instructions Causing Alignment Exceptions	Conditions
dcbz	EA in non-cachable or write-through storage
dcread, lwarx, stwcx.	EA not word-aligned

3.5 Byte Ordering

The following discussion describes the "endianness" of the PPC405GP core, which, by default and in normal use is "big endian." The PPC405GP also contains "little endian" peripherals and supports the attachment of external little endian peripherals.

If scalars (individual data items and instructions) were indivisible, "byte ordering" would not be a concern. It is meaningless to consider the order of bits or groups of bits within a byte, the smallest addressable unit of storage; nothing can be observed about such order. Only when scalars, which the programmer and processor regard as indivisible quantities, can comprise more than one addressable unit of storage does the question of byte order arise.

For a machine in which the smallest addressable unit of storage is the 32-bit word, there is no question of the ordering of bytes within words. All transfers of individual scalars between registers and storage are of words, and the address of the byte containing the high-order eight bits of a scalar is the same as the address of any other byte of the scalar.

For the PowerPC Architecture, as for most computer architectures currently implemented, the smallest addressable unit of storage is the 8-bit byte. Other scalars are halfwords, words, or doublewords, which consist of groups of bytes. When a word-length scalar is moved from a register to storage, the scalar is stored in four consecutive byte addresses. It thus becomes meaningful to discuss the order of the byte addresses with respect to the value of the scalar: that is, which byte contains the highest-order eight bits of the scalar, which byte contains the next-highest-order eight bits, and so on.

Given a scalar that contains multiple bytes, the choice of byte ordering is essentially arbitrary. There are 4! = 24 ways to specify the ordering of four bytes within a word, but only two of these orderings are commonly used:

- The ordering that assigns the lowest address to the highest-order ("leftmost") eight bits of the scalar, the next sequential address to the next-highest-order eight bits, and so on.

  This ordering is called *big endian* because the "big end" of the scalar, considered as a binary number, comes first in storage.

- The ordering that assigns the lowest address to the lowest-order ("rightmost") eight bits of the scalar, the next sequential address to the next-lowest-order eight bits, and so on.

  This ordering is called *little endian* because the "little end" of the scalar, considered as a binary number, comes first in storage.

3.5.1 Structure Mapping Examples

The following C language structure, *s*, contains an assortment of scalars and a character string. The comments show the value assumed to be in each structure element; these values show how the bytes comprising each structure element are mapped into storage.

```
struct {
            int a;          /* 0x1112_1314 word */
            long long b;    /* 0x2122_2324_2526_2728 doubleword */
            char *c;        /* 0x3132_3334 word */
            char d[7];      /* 'A','B','C','D','E','F','G' array of bytes */
            short e;        /* 0x5152 halfword */
            int f;          /* 0x6162_6364 word */
} s;
```

C structure mapping rules permit the use of padding (skipped bytes) to align scalars on desirable boundaries. The structure mapping examples show each scalar aligned at its natural boundary. This alignment introduces padding of four bytes between *a* and *b*, one byte between *d* and *e*, and two bytes between *e* and *f*. The same amount of padding is present in both big endian and little endian mappings.

3.5.1.1 Big Endian Mapping

The big endian mapping of structure *s* follows. (The data is highlighted in the structure mappings. Addresses, in hexadecimal, are below the data stored at the address. The contents of each byte, as defined in structure *s*, is shown as a (hexadecimal) number or character (for the string elements).

11	**12**	**13**	**14**				
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
31	**32**	**33**	**34**	**'A'**	**'B'**	**'C'**	**'D'**
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
'E'	**'F'**	**'G'**		**51**	**52**		
0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
61	**62**	**63**	**64**				
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27

3.5.1.2 Little Endian Mapping

Structure *s* is shown mapped little endian.

14 0x00	**13** 0x01	**12** 0x02	**11** 0x03	 0x04	 0x05	 0x06	 0x07
28 0x08	**27** 0x09	**26** 0x0A	**25** 0x0B	**24** 0x0C	**23** 0x0D	**22** 0x0E	**21** 0x0F
34 0x10	**33** 0x11	**32** 0x12	**31** 0x13	**'A'** 0x14	**'B'** 0x15	**'C'** 0x16	**'D'** 0x17
'E' 0x18	**'F'** 0x19	**'G'** 0x1A	 0x1B	**52** 0x1C	**51** 0x1D	 0x1E	 0x1F
64 0x20	**63** 0x21	**62** 0x22	**61** 0x23	 0x24	 0x25	 0x26	 0x27

3.5.2 Support for Little Endian Byte Ordering

Except as noted, this book describes the processor as if it operated only in a big endian fashion. In fact, the IBM PowerPC Embedded Environment also supports little endian operation.

The PowerPC little endian mode, defined in the PowerPC Architecture, is not implemented.

3.5.3 Endian (E) Storage Attribute

The endian (E) storage attribute supports direct connection of the PPC405GP to little endian peripherals and to memory containing little endian instructions and data. For every storage reference (instruction fetch or load/store access), an E storage attribute is associated with the storage region of the reference. The E attribute specifies whether that region is organized as big endian (E = 0) or little endian (E = 1).

When address translation is enabled (MSR[IR] = 1 or MSR[DR] = 1), the E field in the corresponding TLB entry controls the endianness of a memory region. When address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0), the SLER controls the endianness of a memory region.

Bytes in storage that are accessed as little endian are arranged in true little endian format. The PPC405GP does not support the little endian mode defined in the PowerPC architecture and used in PPC401xx and PPC403xx processors. Furthermore, no address modification is performed when accessing storage regions programmed as little endian. Instead, the PPC405GP reorders the bytes as they are transferred between the processor and memory.

The on-the-fly reversal of bytes in little endian storage regions is handled in one of two ways, depending on whether the storage access is an instruction fetch or a data access (load/store). The following sections describe byte reordering for the two kinds of storage accesses.

3.5.3.1 Fetching Instructions from Little Endian Storage Regions

Instructions are words (four bytes) that are aligned on word boundaries in memory. As such, instructions in a big endian memory region are arranged with the most significant byte (MSB) of the instruction word at the lowest address.

Consider the big endian mapping of instruction *p* at address 00, where, for example, *p* = add r7, r7, r4:

MSB			LSB
0x00	0x01	0x02	0x03

On the other hand, in the little endian mapping instruction *p* is arranged with the least significant byte (LSB) of the instruction word at the lowest numbered address:

LSB			MSB
0x00	0x01	0x02	0x03

When an instruction is fetched from memory, the instruction must be placed in the instruction queue in the proper order. The execution unit assumes that the MSB of an instruction word is at the lowest address. Therefore, when instructions are fetched from little endian storage regions, the four bytes of an instruction word are reversed before the instruction is decoded. In the PPC405GP, the byte reversal occurs between memory and the instruction cache unit (ICU). The ICU always stores instructions in big endian format, regardless of whether the memory region containing the instruction is programmed as big endian or little endian. Thus, the bytes are already in the proper order when an instruction is transferred from the ICU to the decode stage of the pipeline.

If a storage region is reprogrammed from one endian format to the other, the storage region must be reloaded with program and data structures in the appropriate endian format. If the endian format of instruction memory changes, the ICU must be made coherent with the updates. The ICU must be invalidated and the updated instruction memory using the new endian format must be fetched so that the proper byte ordering occurs before the new instructions are placed in the ICU.

3.5.3.2 Accessing Data in Little Endian Storage Regions

Unlike instruction fetches from little endian storage regions, data accesses from little endian storage regions are *not* byte-reversed between memory and the DCU. Data byte ordering, in memory, depends on the data type (byte, halfword, or word) of a specific data item. It is only when moving a data item *of a specific type* from or to a GPR that it becomes known what type of byte reversal is required. Therefore, byte reversal during load/store accesses is performed between the DCU and the GPR.

When accessing data in a little endian storage region:

- For byte loads/stores, no reordering occurs.
- For halfword loads/stores, bytes are reversed within the halfword.
- For word loads/stores, bytes are reversed within the word.

Note that this applies, regardless of data alignment.

The big endian and little endian mappings of the structure *s*, shown in “Structure Mapping Examples” on page 3-29, demonstrate how the size of an item determines its byte ordering. For example:

- The word *a* has its four bytes reversed within the word spanning addresses 0x00–0x03.
- The halfword *e* has its two bytes reversed within the halfword spanning addresses 0x1C–0x1D.

Note that the array of bytes *d*, where each data item is a byte, is not reversed when the big endian and little endian mappings are compared. For example, the character 'A' is located at address 14 in both the big endian and little endian mappings.

In little endian storage regions, the alignment of data is treated as it is in big endian storage regions. Unlike PowerPC little endian mode, no special alignment exceptions occur when accessing data in little endian storage regions.

3.5.3.3 PowerPC Byte-Reverse Instructions

For big endian storage regions, normal load/store instructions move the more significant bytes of a register to and from the lower-numbered memory addresses. The load/store with byte-reverse instructions move the more significant bytes of the register to and from the higher numbered memory addresses.

As Figure 3-11 through Figure 3-14 illustrate, a normal store to a big endian storage region is the same as a byte-reverse store to a little endian storage region. Conversely, a normal store to a little endian storage region is the same as a byte-reverse store to a big endian storage region.

Figure 3-11 illustrates the contents of a GPR and memory (starting at address 00) after a normal load/store in a big endian storage region.

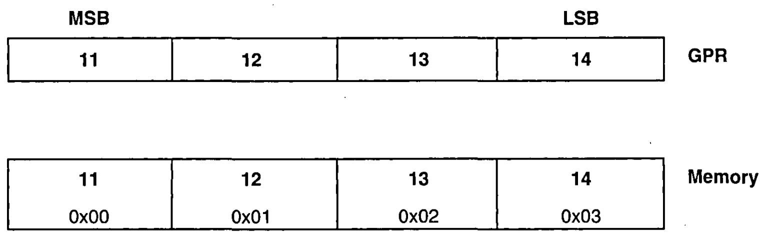


Figure 3-11. Normal Word Load or Store (Big Endian Storage Region)

Note that the results are identical to the results of a load/store with byte-reverse in a little endian storage region, as illustrated in Figure 3-12.

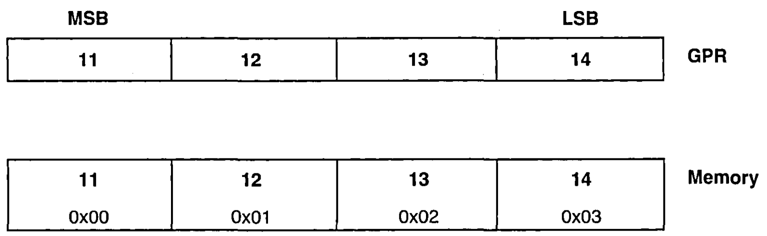


Figure 3-12. Byte-Reverse Word Load or Store (Little Endian Storage Region)

Figure 3-13 illustrates the contents of a GPR and memory (starting at address 00) after a load/store with byte-reverse in a big endian storage region.

MSB			LSB	
11	12	13	14	GPR

14	13	12	11	Memory
0x00	0x01	0x02	0x03	

Figure 3-13. Byte-Reverse Word Load or Store (Big Endian Storage Region)

Note that the results are identical to the results of a normal load/store in a little endian storage region, as illustrated in Figure 3-14.

MSB			LSB	
11	12	13	14	GPR

14	13	12	11	Memory
0x00	0x01	0x02	0x03	

Figure 3-14. Normal Word Load or Store (Little Endian Storage Region)

The E storage attribute augments the byte-reverse load/store instructions in two important ways:

- The load/store with byte-reverse instructions do not solve the problem of fetching instructions from a storage region in little endian format.

  Only the endian storage attribute mechanism supports the fetching of little endian program images.

- Typical compilers cannot make general use of the byte-reverse load/store instructions, so these instructions are ordinarily used only in device drivers written in hand-coded assembler.

  Compilers can, however, take full advantage of the endian storage attribute mechanism, enabling application programmers working in a high-level language, such as C, to compile programs and data structures into little endian format.

3.6 Instruction Processing

The instruction pipeline, illustrated in Figure 3-15, contains three queue locations: prefetch buffer 1 (PFB1), prefetch buffer 0 (PFB0), and decode (DCD). This queue implements a pipeline with the following functional stages: fetch, decode, execute, write-back and load write-back. Instructions are fetched from the instruction cache unit (ICU), placed in the instruction queue, and eventually dispatched to the execution unit (EXU).

Instructions are fetched from the ICU at the request of the EXU. Cachable instructions are forwarded directly to the instruction queue and stored in the ICU cache array. Non-cachable instructions are also forwarded directly to the instruction queue, but are not stored in the ICU cache array. Fetched instructions drop to the empty queue location closest to the EXU. When there is room in the queue, Instructions can be returned from the ICU two at a time. If the queue is empty and the ICU is returning two instructions, one instruction drops into DCD while the other drops into PFB0. PFB1 buffers instructions when the pipeline stalls.

Branch instructions are examined in DCD and PFB0 while all other instructions are decoded in DCD. All instructions must pass through DCD before entering the EXU. The EXU contains the execute, write-back and load write-back stages of the pipe. The results of most instructions are calculated during the execute stage and written to the GPR file during the write back stage. Load instructions write the GPR file during the load write-back stage.

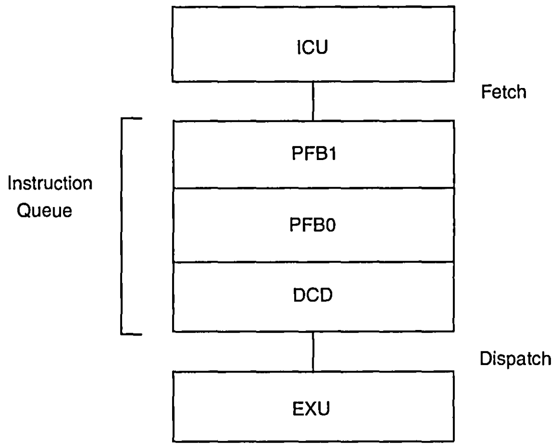


Figure 3-15. PPC405GP Instruction Pipeline

3.7 Branch Processing

The PPC405GP, which provides a variety of conditional and unconditional branching instructions, uses the branch prediction techniques described in "Branch Prediction" on page 3-35.

3.7.1 Unconditional Branch Target Addressing Options

The unconditional branches (**b**, **ba**, **bl**, **bla**) carry the displacement to the branch target address as a signed 26-bit value (the 24-bit LI field right-extended with 0b00). The displacement enables unconditional branches to cover an address range of ±32MB.

For the relative (AA = 0) forms (**b**, **bl**), the target address is the current instruction address (CIA, the address of the branch instruction) plus the signed displacement.

For the absolute (AA = 1) forms (**ba**, **bla**), the target address is 0 plus the signed displacement. If the sign bit (LI[0]) is 0, the displacement is the target address. If the sign bit is 1, the displacement is a negative value and wraps to the highest memory addresses. For example, if the displacement is 0x3FF FFFC (the 26-bit representation of –4), the target address is 0xFFFF FFFC (0 – 4B, or 4 bytes below the top of memory).

3.7.2 Conditional Branch Target Addressing Options

The conditional branches (**bc**, **bca**, **bcl**, **bcla**) carry the displacement to the branch target address as a signed 16-bit value (the 14-bit BD field right-extended with 0b00). The displacement enables conditional branches to cover an address range of ±32KB.

For the relative (AA = 0) forms (**bc**, **bcl**), the target address is the CIA plus the signed displacement.

For the absolute (AA = 1) forms (**bca**, **bcla**), the target address is 0 plus the signed displacement. If the sign bit (BD[0]) is 0, the displacement is the target address. If the sign bit is 1, the displacement is negative and wraps to the highest memory addresses. For example, if the displacement is 0xFFFC (the 16-bit representation of –4), the target address is 0xFFFF FFFC (0 – 4B, or 4 bytes from the top of memory).

3.7.3 Conditional Branch Condition Register Testing

Conditional branch instructions can test a CR bit. The value of the BI field specifies the bit to be tested (bit 0–31). The BO field controls whether the CR bit is tested, as described in the following section.

3.7.4 BO Field on Conditional Branches

The BO field of the conditional branch instruction specifies the conditions used to control branching, and specifies how the branch affects the CTR.

Conditional branch instructions can test one bit in the CR. This option is selected when BO[0] = 0; if BO[0] = 1, the CR does not participate in the branch condition test. If this option is selected, the condition is satisfied (branch can occur) if CR[BI] = BO[1].

Conditional branch instructions can decrement the CTR by one, and after the decrement, test the CTR value. This option is selected when BO[2] = 0. If this option is selected, BO[3] specifies the condition that must be satisfied to allow a branch to be taken. If BO[3] = 0, CTR ≠ 0 is required for a branch to occur. If BO[3] = 1, CTR = 0 is required for a branch to occur.

If BO[2] = 1, the contents of the CTR are left unchanged, and the CTR does not participate in the branch condition test.

Table 3-17 summarizes the usage of the bits of the BO field. BO[4] is further discussed in "Branch Prediction."

Table 3-17. Bits of the BO Field

BO Bit	Description
BO[0]	CR Test Control 0 Test CR bit specified by BI field for value specified by BO[1] 1 Do not test CR
BO[1]	CR Test Value 0 Test for CR[BI] = 0. 1 Test for CR[BI] = 1.
BO[2]	CTR Test Control 0 Decrement CTR by one and test whether CTR satisfies the condition specified by BO[3]. 1 Do not change CTR, do not test CTR.

Table 3-17. Bits of the BO Field (continued)

BO Bit	Description
BO[3]	CTR Test Value 0 Test for CTR ≠ 0. 1 Test for CTR = 0.
BO[4]	Branch Prediction Reversal 0 Apply standard branch prediction. 1 Reverse the standard branch prediction.

Table 3-18 lists specific BO field contents, and the resulting actions; *z* represents a mandatory value of 0, and *y* is a branch prediction option discussed in “Branch Prediction.”

Table 3-18. Conditional Branch BO Field

BO Value	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0 and CR[BI]=0.
0001*y*	Decrement the CTR, then branch if the decremented CTR = 0 and CR[BI] = 0.
001*zy*	Branch if CR[BI] = 0.
0100*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0 and CR[BI] = 1.
0101*y*	Decrement the CTR, then branch if the decremented CTR=0 and CR[BI] = 1.
011*zy*	Branch if CR[BI] = 1.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

3.7.5 Branch Prediction

Conditional branches present a problem to the instruction fetcher. A branch might be taken. The branch EXU attempts to predict whether or not a branch is taken before all information necessary to determine the branch direction is available. This decision is called a *branch prediction*. The fetcher can then prefetch instructions starting at the predicted branch target address. If the prediction is correct, time is saved because the branched-to instruction is available in the instruction queue. Otherwise, the instruction pipeline stalls while the correct instruction is fetched into the instruction queue. To be effective, branch prediction must be correct most of the time.

The PowerPC Architecture enables software to reverse the default branch prediction, which is defined as follows:

Predict that the branch is to be taken if ((BO[0] $\wedge$ BO[2]) $\vee$ *s*) = 1

where *s* is the sign bit of the displacement for conditional branch (**bc**) instructions, and 0 for **bclr** and **bcctr** instructions.

(BO[0] $\wedge$ BO[2]) = 1 only when the conditional branch tests nothing (the “branch always” condition). Obviously, the branch should be predicted taken for this case.

If the branch tests anything, (BO[0] $\wedge$ BO[2]) = 0, and *s* entirely controls the prediction. The default prediction for this case was decided by considering the relative form of **bc**, which is commonly used at the end of loops to control the number of times that a loop is executed. The branch is taken every time the loop is executed except the last, so it is best if the branch is predicted taken. The branch target is the beginning of the loop, so the branch displacement is negative and *s* = 1.

If branch displacements are positive (*s* = 0), the branch is predicted not taken. If the branch instruction is any form of **bclr** or **bcctr** except the "branch always" forms, then *s* = 0, and the branch is predicted not taken.

There is a peculiar consequence of this prediction algorithm for the absolute forms of **bc** (**bca** and **bcla**). As described in "Unconditional Branch Target Addressing Options" on page 3-34, if the algebraic sign of the displacement is negative (*s* = 1), the branch target address is in high memory. If the algebraic sign of the displacement is positive (s = 0), the branch target address is in low memory. Because these are absolute-addressing forms, there is no reason to treat high and low memory differently. Nevertheless, for the high memory case the default prediction is taken, and for the low memory case the default prediction is not taken.

BO[4] is the *prediction reversal bit.* If BO[4] = 0, the default prediction is applied. If BO[4] = 1, the reverse of the standard prediction is applied. For the cases in Table 3-17 where BO[4] = *y*, software can reverse the default prediction. This should only be done when the default prediction is likely to be wrong. Note that for the "branch always" condition, reversal of the default prediction is not allowed.

The PowerPC Architecture requires assemblers to provide a way to conveniently control branch prediction. For any conditional branch mnemonic, a suffix may be added to the mnemonic to control prediction, as follows:

+	Predict branch to be taken
–	Predict branch to be not taken

For example, **bcctr+** causes BO[4] to be set appropriately to force the branch to be predicted taken.

3.8 Speculative Accesses

The PowerPC Architecture permits implementations to perform speculative accesses to memory, either for instruction fetching, or for data loads. A speculative access is defined as any access which is not required by a sequential execution model.

For example, prefetching instructions beyond an undetermined conditional branch is a speculative fetch; if the branch is not in the predicted direction, the program, as executed, never needs the instructions from the predicted path.

Sometimes speculative accesses are inappropriate. For example, attempting to fetch instructions from addresses that cannot contain instructions can cause problems.To protect against errant accesses to "sensitive" memory or I/O devices, the PowerPC Architecture provides the G (guarded) storage attribute, which can be used to specify memory pages from which speculative accesses are prohibited. (Actually, speculative accesses to guarded storage are allowed in certain limited circumstances; if an instruction in a cache block will be executed, the rest of the cache block can be speculatively accessed.)

3.8.1 Speculative Accesses in the PPC405GP

The PPC405GP does not perform speculative loads.

Two methods control speculative instruction fetching. If instruction address translation is enabled (MSR[IR] = 1), the G (guarded) field in the translation lookaside buffer (TLB) entries controls speculative accesses.

If instruction address translation is disabled (MSR[IR] = 0), the Storage Guarded Register (SGR) controls speculative accesses for regions of memory. When a region is guarded (speculative fetching is disallowed), instruction prefetching is disabled for that region. A fetch request must be completely resolved (no longer speculative) before it is issued. There is a considerable performance penalty for fetching from guarded storage, so guarding should be used only when required.

Note that, following any reset, the PPC405GP operates with all of storage guarded.

Note that when address translation is enabled, attempts to access guarded storage result in instruction storage exceptions. Guarded memory is in most often needed with peripheral status registers that are cleared automatically after being read, because an unintended access resulting from a speculative fetch would cause the loss of status information. Because the MMU provides 64 pages with a wide range of page sizes as small as 1KB, fetching instructions from guarded storage should be unnecessary.

3.8.1.1 Prefetch Distance Down an Unresolved Branch Path

The fetcher will speculatively access up to five instructions down a predicted branch path, whether taken or sequential. The unresolved branch is in the DCD stage of the instruction queue (see "Instruction Processing" on page 3-33 for a description of the instruction pipeline). If PFB0 and PFB1 are full, no further speculative accesses occur. If PFB0 or PFB1 is empty, the fetcher requests the next speculative instruction from the ICU; that instruction is placed in PFB0 or PFB1. If the fetched instruction is at the end of a cache line, and if PFB1 is empty, the fetcher requests the next cache line. The instruction at the beginning of the cache line is placed in PFB1. In this case, five instructions are speculatively accessed. The fetcher can speculatively access no more than four instructions (a cache line) from the cache with a single request, assuming the speculative address is cachable.

If the address is non-cachable (as controlled by the I storage attribute), no more than two instructions are speculatively accessed.

3.8.1.2 Prefetch of Branches to the CTR and Branches to the LR

When the instruction fetcher predicts that a **bctr** or **blr** instruction will be taken, the fetcher does not attempt to fetch an instruction from the target address in the CTR or LR if an executing instruction updates the register ahead of the branch. (See "Instruction Processing" on page 3-33 for a description of the instruction pipeline). The fetcher recognizes that the CTR or LR contains data left from an earlier use and that such data is probably not valid.

In such cases, the fetcher does not fetch the instruction at the target address until the instruction that is updating the CTR or LR completes. Only then are the "correct" CTR or LR contents known. This prevents the fetcher from speculatively accessing a completely "random" address. After the CTR or LR contents are known to be correct, the fetcher accesses no more than five instructions down the sequential or taken path of an unresolved branch, or at the address contained in the CTR or LR.

3.8.2 Preventing Inappropriate Speculative Accesses

A memory-mapped I/O peripheral, such as a serial port having a status register that is automatically reset when read provides a simple example of storage that should not be speculatively accessed. If code is in memory at an address adjacent to the peripheral (for example, code goes from

0x0000 0000 to 0x0000 0FFF, and the peripheral is at 0x0000 1000), prefetching past the end of the code will read the peripheral.

Guarding storage also prevents prefetching past the end of memory. If the highest memory address is left unguarded, the fetcher could attempt to fetch past the last valid address, potentially causing machine checks on the fetches from invalid addresses. While the machine checks do not actually cause an exception until the processor attempts to execute an instruction at an invalid address, some systems could suffer from the attempt to access such an invalid address. For example, an external memory controller might log an error.

System designers can avoid problems from speculative fetching without using the guarded storage attributes. The rest of this section describes ways to prevent speculative instruction fetches to sensitive addresses in unguarded memory regions.

3.8.2.1 Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction

Suppose a **bctr** or **blr** instruction closely follows an interrupt-causing or interrupt-returning instruction (**sc**, **rfi**, or **rfci**). The fetcher does not prevent speculatively fetching past one of these instructions. In other words, the fetcher does not treat the interrupt-causing and interrupt-returning instructions specially when deciding whether to predict down a branch path. Instructions after an **rfi**, for example, are considered to be on the determined branch path.

To understand the implications of this situation, consider the code sequence:

```
  handler:    aaa
              bbb
              rfi
 subroutine: bctr
```

When executing the interrupt handler, the fetcher does not recognize the **rfi** as a break in the program flow, and speculatively fetches the target of the **bctr**, which is really the first instruction of a subroutine that has not been called. Therefore, the CTR might contain an invalid pointer.

To protect against such a prefetch, the software must insert an unconditional branch hang (**b $**) just after the **rfi**. This prevents the hardware from prefetching the invalid target address used by **bctr**.

Consider also the above code sequence, with the **rfi** instruction replaced by an **sc** instruction used to initialize the CTR with the appropriate value for the **bctr** to branch to, upon return from the system call. The **sc** handler returns to the instruction following the **sc**, which can't be a branch hang. Instead, software could put a **mtctr** just before the **sc** to load a non-sensitive address into the CTR. This address will be used as the prediction address before the **sc** executes. An alternative would be to put a **mfctr** or **mtctr** between the **sc** and the **bctr**; the **mtctr** prevents the fetcher from speculatively accessing the address contained in the CTR before initialization.

3.8.2.2 Fetching Past tw or twi Instructions

The interrupt-causing instructions, **tw** and **twi**, do not require the special handling described in "Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction" on page 3-39. These instructions are typically used by debuggers, which implement software breakpoints by substituting a trap instruction for the instruction originally at the breakpoint address. In a code sequence **mtlr** followed by **blr** (or **mtctr** followed by **bctr**), replacement of **mtlr/mtctr** by **tw** or **twi** leaves the LR/CTR uninitialized. It would be inappropriate to fetch from the **blr/bctr** target address. This situation is common, and the fetcher is designed to prevent the problem.

3.8.2.3 Fetching Past an Unconditional Branch

When an unconditional branch is in DCD in the instruction queue, the fetcher recognizes that the sequential instructions following the branch are unnecessary. These sequential addresses are not accessed. Addresses at the branch target are accessed instead.

Therefore, placing an unconditional branch just before the start of a sensitive address space (for example, at the "end" of a memory area that borders an I/O device) guarantees that addresses in the sensitive area will not be speculatively fetched.

3.8.2.4 Suggested Locations of Memory-Mapped Hardware

The preferred method of protecting memory-mapped hardware from inadvertent access is to use address translation, with hardware isolated to guarded pages (the G storage attribute in the associated TLB entry is set to 1.) The pages can be as small as 1KB. Code should never be stored in such pages.

If address translation is disabled, the preferred protection method is to isolate memory-mapped hardware into regions guarded using the SGR. Code should never be stored in such regions. The disadvantage of this method, compared to the preferred method, is that each region guarded by the SGR consumes 128MB of the address space.

Table 3-19 shows two address regions of the PPC405GP. Suppose a system designer can map all I/O devices and all ROM and SRAM devices into any location in either region. The choices made by the designer can prevent speculative accesses to the memory-mapped I/O devices.

Table 3-19. Example Memory Mapping

0x7800 0000 – 0x7FFF FFFF (SGR bit 15)	128MB Region 2
0x7000 0000 – 0x77FF FFFF (SGR bit 14)	128MB Region 1

A simple way to avoid the problem of speculative reads to peripherals is to map all storage containing code into Region 2, and all I/O devices into Region 1. Thus, accesses to Region 2 would only be for code and program data. Speculative fetches occuring in Region 2 would never access addresses in Region 1. Note that this hardware organization eliminates the need to use of the G storage attribute to protect Region 1. However, Region 1 could be set as guarded with no performance penalty, because there is no code to execute or variable data to access in Region 1.

The use of these regions could be reversed (code in Region 1 and I/O devices in Region 2), if Region 2 is set as guarded. Prefetching from the highest addresses of Region 1 could cause an attempt to speculatively access the bottom of Region 2, but guarding prevents this from occurring. The performance penalty is slight, under the assumption that code infrequently executes the instructions in the highest addresses of Region 1.

3.8.3 Summary

Software should take the following actions to prevent speculative accesses to sensitive data areas, if the sensitive data areas are not in guarded storage:

- Protect against accesses to "random" values in the LR or CTR on **blr** or **bctr** branches following **rfi**, **rfci**, or **sc** instructions by putting appropriate instructions before or after the **rfi**, **rfci**, or **sc** instruction. See "Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction" on page 3-39.
- Protect against "running past" the end of memory into a bordering I/O device by putting an unconditional branch at the end of the memory area. See "Fetching Past an Unconditional Branch" on page 3-40.
- Recognize that a maximum of five words (20 bytes) can be prefetched past an unresolved conditional branch, either down the target path or the sequential path. See "Prefetch Distance Down an Unresolved Branch Path" on page 3-38.

Of course, software should not code branches with known unsafe targets (either relative to the instruction counter, or to addresses contained in the LR or CTR), on the assumption that the targets are "protected" by code guaranteeing that the unsafe direction is not taken. The fetcher assumes that if a branch is predicted to be taken, it is safe to fetch down the target path.

3.9 Privileged Mode Operation

In the PowerPC Architecture, several terms describe two operating modes that have different instruction execution privileges. When a processor is in "privileged mode," it can execute all instructions in the instruction set. This mode is also called the "supervisor state." The other mode, in which certain instructions cannot be executed, is called the "user mode," or "problem state." These terms are used in pairs:

Privileged	Non-privileged
Privileged Mode	User Mode
Supervisor State	Problem State

The architecture uses MSR[PR] to control the execution mode. When MSR[PR] = 1, the processor is in user mode (problem state); when MSR[PR] = 0, the processor is in privileged mode (supervisor state).

After a reset, MSR[PR] = 0.

3.9.1 MSR Bits and Exception Handling

The current value of MSR[PR] is saved, along with all other MSR bits, in the SRR1 (for non-critical interrupts) or SRR3 (for critical interrupts) upon any interrupt, and MSR[PR] is set to 0. Therefore, all exception handlers operate in privileged mode.

Attempting to execute a privileged instruction while in user mode causes a privileged violation program exception (see "Program Interrupt" on page 10-40). The PPC405GP does not execute the instruction, and the least-significant 16 bits of the program counter are loaded with 0x0700, the address of an exception processing routine.

The PRR field of the Exception Syndrome Register (ESR) is set when an interrupt was caused by a privileged instruction program exception. Software is not required to clear ESR[PRR].

3.9.2 Privileged Instructions

The instructions listed in Table 3-20 are privileged and cannot be executed while in user mode (MSR[PR] = 1).

Table 3-20. Privileged Instructions

dcbi	
dccci	
dcread	
iccci	
icread	
mfdcr	
mfmsr	
mfspr	For all SPRs except CTR, LR, SPRG4–SPRG7, and XER. See "Privileged SPRs" on page 3-42
mtdcr	
mtmsr	
mtspr	For all SPRs except CTR, LR, XER. See "Privileged SPRs" on page 3-42
rfci	
rfi	
tlbia	
tlbre	
tlbsx	
tlbsync	
tlbwe	
wrtee	
wrteei	

3.9.3 Privileged SPRs

All SPRs are privileged, except for the LR, the CTR, the XER, USPRG0, and read access to SPRG4–SPRG7. Reading from the time base registers Time Base Lower (TBL) and Time Base Upper (TBU) is not privileged. These registers are read using the **mftb** instruction, rather than the **mfspr** instruction. TBL and TBU are written (with different addresses) using **mtspr**, which is privileged for these registers. Except for moves to and from non-privileged SPRs, attempts to execute **mfspr** and **mtspr** instructions while in user mode result in privileged violation program exceptions.

In a **mfspr** or **mtspr** instruction, the 10-bit SPRN field specifies the SPR number of the source or destination SPR. The SPRN field contains two five-bit subfields, $SPRN_{0:4}$ and $SPRN_{5:9}$. The assembler handles the unusual register number encoding to generate the SPRF field. In the *machine code* for the **mfspr** and **mtspr** instructions, the SPRN subfields are *reversed* (ending up as $SPRF_{5:9}$ and $SPRF_{0:4}$) for compatibility with the POWER Architecture.

In the PowerPC Architecture, SPR numbers having a 1 in the most-significant bit of the SPRF field are privileged.

The following example illustrates how SPR numbers appear in assembler language coding and in machine coding of the **mfspr** and **mtspr** instructions.

In assembler language coding, SRR0 is SPR 26. Note that the assembler handles the unusual register number encoding to generate the SPRF field.

```
mfspr r5,26
```

When the SPR number is considered as a binary number (0b0000011010), the most-significant bit is 0. However, the machine code for the instruction reverses the subfields, resulting in the following SPRF field: 0b1101000000. The most-significant bit is 1; SRR0 is privileged.

When an SPR number is considered as a hexadecimal number, the second digit of the three-digit hexadecimal number indicates whether an SPR is privileged. If the second digit is odd (1, 3, 5, 7, 9, B, D, F), the SPR is privileged.

For example, the SPR number of SRR0 is 26 (0x01A). The second hexadecimal digit is odd; SRR0 is privileged. In contrast, the LR is SPR 8 (0x008); the second hexadecimal digit is not odd; the LR is non-privileged.

3.9.4 Privileged DCRs

The **mtdcr** and **mfdcr** instructions themselves are privileged, in all cases. All DCRs are privileged.

3.10 Synchronization

The PPC405GP supports the synchronization operations of the PowerPC Architecture. The following book, chapter, and section numbers refer to related information in *The PowerPC Architecture: A Specification for a New Family of RISC Processors*:

- Book II, Section 1.8.1, "Storage Access Ordering" and "Enforce In-order Execution of I/O"
- Book III, Section 1.7, "Synchronization"
- Book III, Chapter 7, "Synchronization Requirements for Special Registers and Lookaside Buffers"

3.10.1 Context Synchronization

The context of a program is the environment (for example, privilege and address translation) in which the program executes. Context is controlled by the content of certain registers, such as the Machine State Register (MSR), and includes the content of all GPRs and SPRs.

An instruction or event is context synchronizing if it satisfies the following requirements:

1. All instructions that *precede* a context synchronizing operation must complete in the context that existed *before* the context synchronizing operation.
2. All instructions that *follow* a context synchronizing operation must complete in the context that exists *after* the context synchronizing operation.

Such instructions and events are called "context synchronizing operations." In the PPC405GP, these include any interrupt, except a non-recoverable instruction machine check, and the **isync**, **rfci**, **rfi**, and **sc** instructions.

However, context specifically excludes the contents of memory. A context synchronizing operation does not guarantee that subsequent instructions observe the memory context established by previous instructions. To guarantee memory access ordering in the PPC405GP, one must use either an **eieio** instruction or a **sync** instruction. Note that for the PPC405GP, the **eieio** and **sync** instructions are implemented identically. See "Storage Synchronization" on page 3-46.

The contents of DCRs are not considered as part of the processor "context" managed by a context synchronizing operation. DCRs are not part of the processor core, and are analogous to memory-mapped registers. Their context is managed in a manner similar to that of memory contents.

Finally, implementations of the PowerPC Architecture can exempt the machine check exception from context synchronization control. If the machine check exception is exempted, an instruction that *precedes* a context synchronizing operation can cause a machine check exception *after* the context synchronizing operation occurs and additional instructions have completed.

The following scenarios use pseudocode examples to illustrate these limitations of context synchronization. Subsequent text explains how software can further guarantee "storage ordering."

1. Consider the following instruction sequence:

   ```
   STORE non-cachable to address XYZ
   isync
   XYZ instruction
   ```

   In this sequence, the **isync** instruction does not guarantee that the XYZ instruction is fetched after the STORE has occurred to memory. There is no guarantee which XYZ instruction will execute; either the old version or the new (stored) version might.

2. Consider the following instruction sequence, which assumes that the PPC405GP uses DCRs to provide bus region control:

   ```
   STORE non-cachable to address XYZ
   isync
   MTDCR to change a bus region containing XYZ
   ```

   In this sequence, there is no guarantee that the STORE will occur before the **mtdcr** changing the bus region control DCR. The STORE could fail because of a configuration error.

Consider an interrupt that changes privileged mode. An interrupt is a context synchronizing operation, because interrupts cause the MSR to be updated. The MSR is part of the processor context; the context synchronizing operation guarantees that all instructions that precede the interrupt complete using the preinterrupt value of MSR[PR], and that all instructions that follow the interrupt complete using the postinterrupt value.

Consider, on the other hand, some code that uses **mtmsr** to change the value of MSR[PR], which changes the privileged mode. In this case, the MSR is changed, changing the context. It is possible, for example, that prefetched privileged instructions expect to execute after the **mtmsr** has changed the operating mode from privileged mode to user mode. To prevent privileged instruction program exceptions, the code must execute a context synchronization operation, such as **isync**, immediately after the **mtmsr** instruction to prevent further instruction execution until the **mtmsr** completes.

eieio or **sync** can ensure that the contents of memory and DCRs are synchronized in the instruction stream. These instructions guarantee storage ordering because all memory accesses that precede **eieio** or **sync** are completed before subsequent memory accesses. Neither **eieio** nor **sync** guarantee that instruction prefetching is delayed until the **eieio** or **sync** completes. The instructions do not cause the prefetch queues to be purged and instructions to be refetched. See "Storage Synchronization" on page 3-46 for more information.

Instruction cache state is part of context. A context synchronization operation is required to guarantee instruction cache access ordering.

3. Consider the following instruction sequence, which is required for creating self-modifying code:

STORE	Change data cache contents
dcbst	Flush the new data cache contents to memory
sync	Guarantee that **dcbst** completes before subsequent instructions begin
icbi	Context changing operation; invalidates instruction cache contents.
isync	Context synchronizing operation; causes refetch using new instruction cache context text and new memory context, due to the previous **sync**.

If software wishes to ensure that all storage accesses are complete before executing a **mtdcr** to change a bus region (Example 2), the software must issue a **sync** after all storage accesses and before the **mtdcr**. Likewise, if the software is to ensure that all instruction fetches after the **mtdcr** use the new bank register contents, the software must issue an **isync**, after the **mtdcr** and before the first instruction that should be fetched in the new context.

isync guarantees that all subsequent instructions are fetched and executed using the context established by all previous instructions. **isync** is a context synchronizing operation; **isync** causes all subsequently prefetched instructions to be discarded and refetched.

The following example illustrates the use of **isync** with debug exceptions:

mtdbcr0	Enable an instruction address compare (IAC) event
isync	Wait for the new Debug Control Register 0 (DBCR0) context to be established
XYZ	This instruction is at the IAC address; an **isync** was necessary to guarantee that the IAC event occurs at the execution of this instruction

3.10.2 Execution Synchronization

For completeness, consider the definition of execution synchronizing as it relates to context synchronization. Execution synchronization is architecturally a subset of context synchronization.

Execution synchronization guarantees that the following requirement is met:

> All instructions that *precede* an execution synchronizing operation must complete in the context that existed *before* the execution synchronizing operation.

The following requirement need not be met:

> All instructions that *follow* an execution synchronizing operation must complete in the context that exists *after* the execution synchronizing operation.

Execution synchronization ensures that preceding instructions execute in the old context; subsequent instructions might execute in either the new or old context (indeterminate). The PPC405GP provides three execution synchronizing operations: the **eieio**, **mtmsr**, and **sync** instructions.

Because **mtmsr** is execution synchronizing, it guarantees that previous instructions complete using the old MSR value. (For example, using **mtmsr** to change the endian mode.) However, to guarantee that subsequent instructions use the new MSR value, we have to insert a context synchronization operation, such as **isync**.

Note that the PowerPC Architecture requires MSR[EE] (the external interrupt bit) to be, in effect, execution synchronizing: if a **mtmsr** sets MSR[EE] = 1, and an external interrupt is pending, the exception must be taken before the instruction that follows **mtmsr** is executed. However, the **mtmsr** instruction is not a context synchronizing operation, so the PPC405GP does not, for example, discard prefetched instructions and refetch. Note that the **wrtee** and **wrteei** instructions can change the value of MSR[EE], but are not execution synchronizing.

Finally, while **sync** and **eieio** are execution synchronizing, they are also more restrictive in their requirement of memory ordering. Stating that an operation is execution synchronizing does not imply storage ordering. This is an additional specific requirement of **sync** and **eieio**.

3.10.3 Storage Synchronization

The **sync** instruction guarantees that all previous storage references complete with respect to the PPC405GP before the **sync** instruction completes (therefore, before any subsequent instructions begin to execute). The **sync** instruction is execution synchronizing.

Consider the following use of **sync**:

```
stw      Store to peripheral
sync     Wait for store to actually complete
mtdcr    Reconfigure device
```

The **eieio** instruction guarantees the order of storage accesses. All storage accesses that precede **eieio** complete before any storage accesses that follow the instruction, as in the following example:

stb X	Store to peripheral, address X; this resets a status bit in the device
eieio	Guarantee **stb** X completes before next instruction
lbz Y	Load from peripheral, address Y; this is the status register updated by **stb** X. **eieio** was necessary, because the read and write addresses are different, but affect each other

The PPC405GP implements both **sync** and **eieio** identically, in the manner described above for **sync**. In the PowerPC Architecture, **sync** can function across all processors in a multiprocessor environment; **eieio** functions only within its executing processor. The PPC405GP does not provide hardware support for multiprocessor memory coherency, so **sync** does not guarantee memory ordering across multiple processors.

3.11 Instruction Set

The PPC405GP instruction set contains instructions defined in the PowerPC Architecture and instructions specific to the IBM PowerPC 400 family of embedded processors.

Chapter 24, "Instruction Set," contains detailed descriptions of each instruction.

Appendix A, "Instruction Summary," alphabetically lists each instruction and extended mnemonic and provides a short-form description. Appendix B, "Instructions by Category," provides short-form descriptions of instructions, grouped by the instruction categories listed in Table 3-21, "PPC405GP Instruction Set Summary," on page 3-47.

Table 3-21 summarizes the PPC405GP instruction set functions by categories. Instructions within each category are described in subsequent sections.

Table 3-21. PPC405GP Instruction Set Summary

Storage Reference	load, store
Arithmetic	add, subtract, negate, multiply, multiply-accumulate, multiply halfword, divide
Logical	and, andc, or, orc, xor, nand, nor, xnor, sign extension, count leading zeros
Comparison	compare, compare logical, compare immediate
Branch	branch, branch conditional, branch to LR, branch to CTR
CR Logical	crand, crandc, cror, crorc, crnand, crnor, crxor, crxnor, move CR field
Rotate	rotate and insert, rotate and mask, shift left, shift right
Shift	shift left, shift right, shift right algebraic
Cache Management	invalidate, touch, zero, flush, store, read
Interrupt Control	write to external interrupt enable bit, move to/from MSR, return from interrupt, return from critical interrupt
Processor Management	system call, synchronize, trap, move to/from DCRs, move to/from SPRs, move to/from CR

3.11.1 Instructions Specific to IBM PowerPC Embedded Processors

To support functions required in embedded real-time applications, the IBM PowerPC 400 family of embedded processors defines instructions that are not defined in the PowerPC Architecture.

Table 3-22 lists the instructions specific to IBM PowerPC embedded processors. Programs using these instructions are not portable to PowerPC implementations that are not part of the IBM PowerPC 400 family of embedded processors.

In the table, the syntax [**s**] indicates that the instruction has a signed form. The syntax [**u**] indicates that the instruction has an unsigned form. The syntax "[**.**]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-22. Implementation-specific Instructions

dccci **dcread** **iccci** **icread**	**macchw[s][u]** **machhw[s][u]** **maclhw[s][u]** **nmacchw[s]** **nmachhw[s]** **nmaclhw[s]**	**mulchw[u]** **mulhhw[u]** **mullhw[u]**	**mfdcr** **mtdcr** **rfci** **tlbre** **tlbsx[.]** **tlbwe** **wrtee** **wrteei**

3.11.2 Storage Reference Instructions

Table 3-23 lists the PPC405GP storage reference instructions. Load/store instructions transfer data between memory and the GPRs. These instructions operate on bytes, halfwords, and words. Storage reference instructions also support loading or storing multiple registers, character strings, and byte-reversed data.

In the table, the syntax "[**u**]" indicates that an instruction has an "update" form that updates the RA addressing register with the calculated address, and a "non-update" form. The syntax "[**x**]" indicates that an instruction has an "indexed" form, which forms the address by adding the contents of the RA and RB GPRs) and a "base + displacement" form (in which the address is formed by adding a 16-bit signed immediate value (included as part of the instruction word) to the contents of RA GPR.

Table 3-23. Storage Reference Instructions

Loads				Stores			
Byte	Halfword	Word	Multiple/String	Byte	Halfword	Word	Multiple/String
lbz[u][x]	**lha[u][x]** **lhbrx** **lhz[u][x]**	**lwarx** **lwbrx** **lwz[u][x]**	**lmw** **lswi** **lswx**	**stb[u][x]**	**sth[u][x]** **sthbrx**	**stw[u][x]** **stwbrx** **stwcx.**	**stmw** **stswi** **stswx**

3.11.3 Arithmetic Instructions

Arithmetic operations are performed on integer operands stored in GPRs. Instructions that perform operations on two operands are defined in a three-operand format; an operation is performed on the operands, which are stored in two GPRs. The result is placed in a third, operand, which is stored in a GPR. Instructions that perform operations on one operand are defined using a two-operand format; the operation is performed on the operand in a GPR and the result is placed in another GPR. Several instructions also have immediate formats in which an operand is contained in a field in the instruction word.

Most arithmetic instructions have versions that can update CR[CR0] and XER[SO, OV], based on the result of the instruction. Some arithmetic instructions also update XER[CA] implicitly. See “Condition Register (CR)” on page 3-12 and “Fixed Point Exception Register (XER)” on page 3-8 for more information.

Table 3-24 lists the PPC405GP arithmetic instructions. In the table, the syntax “[**o**]” indicates that an instruction has an “o” form that updates XER[SO,OV], and a “non-o” form. The syntax “[.]” indicates that the instruction has a “record” form that updates CR[CR0], and a “non-record” form.

Table 3-24. Arithmetic Instructions

Add	Subtract	Multiply	Divide	Negate
add[o][.]	**subf[o][.]**	**mulhw[.]**	**divw[o][.]**	**neg[o][.]**
addc[o][.]	**subfc[o][.]**	**mulhwu[.]**	**divwu[o][.]**	
adde[o][.]	**subfe[o][.]**	**mulli**		
addi	**subfic**	**mullw[o][.]**		
addic[.]	**subfme[o][.]**			
addis	**subfze[o][.]**			
addme[o][.]				
addze[o][.]				

Table 3-25 lists additional arithmetic instructions for multiply-accumulate and multiply halfword operations. In the table, the syntax "[o]" indicates that an instruction has an "o" form that updates XER[SO,OV], and a "non-o" form. The syntax "[.]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-25. Multiply-Accumulate and Multiply Halfword Instructions

Multiply-Accumulate	Negative-Multiply-Accumulate	Multiply Halfword
macchw[o][.]	nmacchw[o][.]	mulchw[.]
macchws[o][.]	nmacchws[o][.]	mulchwu[.]
macchwsu[o][.]	nmachhw[o][.]	mulhhw[.]
macchwu[o][.]	nmachhws[o][.]	mulhhwu[.]
machhw[o][.]	nmaclhw[o][.]	mullhw[.]
machhws[o][.]	nmaclhws[o][.]	mullhwu[.]
machhwsu[o][.]		
machhwu[o][.]		
maclhw[o][.]		
maclhws[o][.]		
maclhwsu[o][.]		
maclhwu[o][.]		

3.11.4 Logical Instructions

Table 3-26 lists the PPC405GP logical instructions. In the table, the syntax "[.]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-26. Logical Instructions

And	And with complement	Nand	Or	Or with complement	Nor	Xor	Equivalence	Extend sign	Count leading zeros
and[.]	andc[.]	nand[.]	or[.]	orc[.]	nor[.]	xor[.]	eqv[.]	extsb[.]	cntlzw[.]
andi.			ori			xori		extsh[.]	
andis.			oris			xoris			

3.11.5 Compare Instructions

These instructions perform arithmetic or logical comparisons between two operands and update the CR with the result of the comparison.

Table 3-27 lists the PPC405GP compare instructions.

Table 3-27. Compare Instructions

Arithmetic	Logical
cmp	cmpl
cmpi	cmpli

3.11.6 Branch Instructions

These instructions unconditionally or conditionally branch to an address. Conditional branch instructions can test condition codes set by a previous instruction and branch accordingly. Conditional branch instructions can also decrement and test the CTR as part of branch determination, and can save the return address in the LR.The target address for a branch can be a displacement from the current instruction address (a relative address), an absolute address, or contained in the CTR or LR.

See "Branch Processing" on page 3-34 for more information on branch operations.

Table 3-28 lists the PPC405GP branch instructions. In the table, the syntax "[**l**]" indicates that the instruction has a "link update" form that updates LR with the address of the instruction after the branch, and a "non-link update" form. The syntax "[**a**]" indicates that the instruction has an "absolute address" form, in which the target address is formed directly using the immediate field specified as part of the instruction, and a "relative" form, in which the target address is formed by adding the immediate field to the address of the branch instruction).

Table 3-28. Branch Instructions

Branch
b[l][a] **bc[l][a]** **bcctr[l]** **bclr[l]**

3.11.6.1 CR Logical Instructions

These instructions perform logical operations on a specified pair of bits in the CR, placing the result in another specified bit. These instructions can logically combine the results of several comparisons without incurring the overhead of conditional branch instructions. Software performance can significantly improve if multiple conditions are tested at once as part of a branch decision.

Table 3-29 lists the PPC405GP condition register logical instructions.

Table 3-29. CR Logical Instructions

crand **crandc** **creqv** **crnand**	**crnor** **cror** **crorc** **crxor** **mcrf**

3.11.6.2 Rotate Instructions

These instructions rotate operands stored in the GPRs. Rotate instructions can also mask rotated operands.

Table 3-30 lists the PPC405GP rotate instructions. In the table, the syntax "[**.**]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-30. Rotate Instructions

Rotate and Insert	Rotate and Mask
rlwimi[.]	**rlwinm[.]** **rlwnm[.]**

3.11.6.3 Shift Instructions

These instructions rotate operands stored in the GPRs.

Table 3-31 lists the PPC405GP shift instructions. Shift right algebraic instructions implicitly update XER[CA]. In the table, the syntax "[.]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-31. Shift Instructions

Shift Left	Shift Right	Shift Right Algebraic
slw[.]	srw[.]	sraw[.] srawi[.]

3.11.6.4 Cache Management Instructions

These instructions control the operation of the ICU and DCU. Instructions are provided to fill or invalidate instruction cache blocks. Instructions are also provided to fill, flush, invalidate, or zero data cache blocks, where a block is defined as a 32-byte cache line.

Table 3-32 lists the PPC405GP cache management instructions.

Table 3-32. Cache Management Instructions

DCU	ICU
dcba dcbf dcbi dcbst dcbt dcbtst dcbz dccci dcread	icbi icbt iccci icread

3.11.7 Interrupt Control Instructions

mfmsr and **mtmsr** read and write data between the MSR and a GPR to enable and disable interrupts. **wrtee** and **wrteei** enable and disable external interrupts. **rfi** and **rfci** return from interrupt handlers. Table 3-33 lists the PPC405GP interrupt control instructions.

Table 3-33. Interrupt Control Instructions

mfmsr mtmsr rfi rfci wrtee wrteei

3.11.8 TLB Management Instructions

The TLB management instructions read and write entries of the TLB array in the MMU, search the TLB array for an entry which will translate a given address, and invalidate all TLB entries. There is also an instruction for synchronizing TLB updates with other processors, but because the PPC405GP is for use in uniprocessor environments, this instruction performs no operation.

Table 3-34 lists the TLB management instructions. In the table, the syntax "[.]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-34. TLB Management Instructions

tlbia
tlbre
tlbsx[.]
tlbsync
tlbwe

3.11.9 Processor Management Instructions

These instructions move data between the GPRs and SPRs, the CR, and DCRs in the PPC405GP, and provide traps, system calls, and synchronization controls.

Table 3-35 lists the processor management instructions in the PPC405GP.

Table 3-35. Processor Management Instructions

eieio	**mcrxr**	**mtcrf**
isync	**mfcr**	**mtdcr**
sync	**mfdcr**	**mtspr**
	mfspr	**sc**
		tw
		twi

3.11.10 Extended Mnemonics

In addition to mnemonics for instructions supported directly by hardware, the PowerPC Architecture defines numerous *extended mnemonics*.

An extended mnemonic translates directly into the mnemonic of a hardware instruction, typically with carefully specified operands. For example, the PowerPC Architecture does not define a "shift right word immediate" instruction, because the "rotate left word immediate then AND with mask," **(rlwinm)** instruction can accomplish the same result:

rlwinm RA,RS,32–n,n,31

However, because the required operands are not obvious, the PowerPC Architecture defines an extended mnemonic:

srwi RA,RS,n

Extended mnemonics transfer the problem of remembering complex or frequently used operand combinations to the assembler, and can more clearly reflect a programmer's intentions. Thus, programs can be more readable.

Refer to the following chapter and appendixes for lists of the extended mnemonics:

- Chapter 24, “Instruction Set,” lists extended mnemonics under the associated hardware instruction mnemonics.
- Appendix A, “Instruction Summary,” lists extended mnemonics alphabetically, along with the hardware instruction mnemonics.
- Table B-5 in Appendix B, “Instructions by Category,” lists all extended mnemonics.

Chapter 4. Cache Operations

The PPC405GP incorporates two internal caches, a16KB instruction cache and an 8KB data cache. Instructions and data can be accessed in the caches much faster than in main memory.

The instruction cache unit (ICU) controls instruction accesses to main memory and stores frequently used instructions to reduce the overhead of instruction transfers between the instruction pipeline and external memory. Using the instruction cache minimizes access latency for frequently executed instructions.

The data cache unit (DCU) controls data accesses to main memory and stores frequently used data to reduce the overhead of data transfers between the GPRs and external memory. Using the data cache minimizes access latency for frequently used data.

The ICU features:

- Programmable address pipelining and prefetching for cache misses and non-cachable lines
- Support for non-cachable hits from lines contained in fill buffer
- Programmable non-cachable requests to memory as 4 or 8 words (line or half line)
- Bypass path for critical words
- Non-blocking cache for hits during fills
- Flash invalidate (one instruction invalidates entire cache)
- Programmable allocation for fetch fills, enabling program control of cache contents using the **icbt** instruction
- Virtually indexed, physically tagged cache arrays
- Support for 64-bit and 32-bit PLB slaves
- A rich set of cache control instructions

The DCU features:

- Address pipelining for line fills
- Support for load hits from non-cachable and non-allocated lines contained in fill buffer
- Bypass path for critical words
- Non-blocking cache for hits during fills
- Write-back and write-through write strategies controlled by storage attributes
- Programmable non-cachable load requests to memory as lines or words.
- Handling of up to two pending line flushes.
- Holding of up to three stores before stalling the core pipeline
- Physically indexed, physically tagged cache arrays
- Support for 64-bit and 32-bit PLB slaves
- A rich set of cache control instructions

"ICU Organization" on page 4-2 and "DCU Organization" on page 4-5 describe the organization and provide overviews of the ICU and the DCU.

4.1 ICU Organization

The ICU manages instruction transfers between external cachable memory and the instruction queue in the execution unit.

The ICU contains a two-way set-associative 16KB cache memory. Each way is organized in 256 lines of eight words (eight instructions) each.

As shown in Table 4-1, tag ways A and B store instruction address bits $A_{0:21}$ for each line in cache ways A and B. Instruction address bits $A_{19:26}$ serve as the index to the cache array. The two cache lines that correspond to the same line index (one in each way) are called a congruence class.

Table 4-1. Instruction Cache Organization

Tags (Two-way Set)		Instructions (Two-way Set)	
Way A	**Way B**	**Way A**	**Way B**
$A_{0:21}$ Line 0 A	$A_{0:21}$ Line 0 B	Line 0 A	Line 0 B
$A_{0:21}$ Line 1 A	$A_{0:21}$ Line 1 B	Line 1 A	Line 1 B
⋮	⋮	⋮	⋮
$A_{0:21}$ Line 254 A	$A_{0:21}$ Line 254 B	Line 254 A	Line 254 B
$A_{0:21}$ Line 255 A	$A_{0:21}$ Line 255 B	Line 255 A	Line 255 B

When a cache line is to be loaded, the cache way to receive the line is determined by using an least-recently-used (LRU) policy. The index, determined by the instruction address, selects a congruence class. Within a congruence class, the line which was accessed most recently is retained, and the other line is marked as LRU, using an LRU bit in the tag array. The line to receive the incoming data is the LRU line. After the cache line fill, the LRU bit is then set to identify as least-recently-used the line opposite the line just filled.

Figure 4-1 shows the relationships between the ICU and the instruction pipeline.

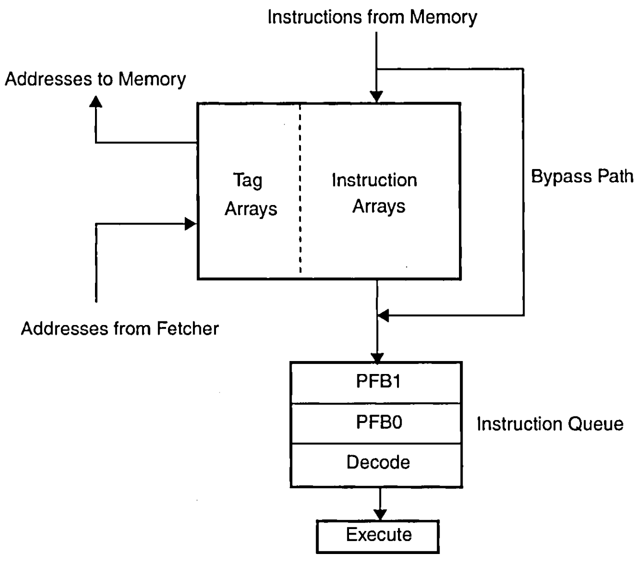


Figure 4-1. Instruction Flow

4.1.1 ICU Operations

Instructions from cachable memory regions are copied into the instruction cache array. The fetcher can access instructions much more quickly from a cache array than from memory. Cache lines are loaded either target-word-first or sequentially. Target-word-first fills start at the requested word, continue to the end of the line, and then wrap to fill the remaining words at the beginning of the line. Sequential fills start at the first word of the cache line and proceed sequentially to the last word of the line.

The bypass path handles instructions in cache-inhibited memory and improves performance during line fill operations. If a request from the fetcher obtains an entire line from memory, the queue does not have to wait for the entire line to reach the cache. The target word (the word requested by the fetcher) is sent on the bypass path to the queue while the line fill proceeds, even if the selected line fill order is not target-word-first.

Cache line fills always run to completion, even if the instruction stream branches away from the rest of the line. As requested instructions are received, they go to the fetcher from the fill register before the line fills in the cache. The filled line is always placed in the ICU; if an external memory subsystem error occurs during the fill, the line is not written to the cache. During a clock cycle, the ICU can send two instruction to the fetcher.

4.1.2 Instruction Cachability Control

When instruction address translation is enabled (MSR[IR] = 1), instruction cachability is controlled by the I storage attribute in the translation lookaside buffer (TLB) entry for the memory page. If TLB_entry[I] = 1, caching is inhibited; otherwise caching is enabled. Cachability is controlled separately for each page, which can range in size from 1KB to 16MB. "Translation Lookaside Buffer (TLB)" on page 6-2 describes the TLB.

When instruction address translation is disabled (MSR[IR] = 0), instruction cachability is controlled by the Instruction Cache Cachability Register (ICCR). Each field in the ICCR (ICCR[S0:S31]) controls the cachability of a 128MB region (see "Real-mode Storage Attribute Control" on page 6-17). If ICCR[S*n*] = 1, caching is enabled for the specified region; otherwise, caching is inhibited.

The performance of the PPC405GP is significantly lower while fetching instructions from cache-inhibited regions.

Following system reset, address translation is disabled and all ICCR bits are reset to 0 so that no memory regions are cachable. Before regions can be designated as cachable, the ICU cache array must be invalidated. The **iccci** instruction must execute before the cache is enabled. Address translation can then be enabled, if required, and the TLB or the ICCR can then be configured for the required cachability.

4.1.3 Instruction Cache Synonyms

The following information applies only if instruction address translation is enabled (MSR[IR] = 1) and 1KB or 4KB page sizes are used. See Chapter 6, "Memory Management," for information about address translation and page sizes.

An instruction cache synonym occurs when the instruction cache array contains multiple cache lines from the same real address. Such synonyms result from combinations of:

- Cache array size
- Cache associativity
- Page size
- The use of effective addresses (EAs) to index the cache array

For example, the instruction cache array has a "way size" of 8KB (16KB array/2 ways). Thus, 11 bits ($EA_{19:29}$) are needed to select a word (instruction) in each way. For the minimum page size of 1KB, the low order 8 bits ($EA_{22:29}$) address a word in a page. The high order address bits ($EA_{0:21}$) are translated to form a real address (RA), which the ICU uses to perform the cache tag match. Cache synonyms could occur because the index bits ($EA_{19:29}$) overlap the translated RA bits. For 1KB pages, overlap in $EA_{19:21}$ and $RA_{19:21}$ could result in as many as 8 synomyms. In other words, data from the same RA could occur as as manyas 8 llocations in the cache array. Similarly, for 4KB pages, $EA_{0:19}$ are translated. Differences in EA_{19} and RA_{19} could result in as many as 2 synonyms. For the next largest page size (16KB), only EA $_{0:17}$ are translated. Because there is no overlap with index bits $EA_{19:21}$, synonyms do not occur.

In practice, cache synonyms occur when a real instruction page having multiple virtual mappings exists in multiple cache lines. For 1KB pages, all EAs differing in $EA_{19:21}$ must be cast out of cache, using an **icbi** instruction for each such EA (up to 8 per cache line in the page). For 4KB pages, all EAs differing in EA_{19} must be cast out in the same manner (up to 2 per cache line in the page). For larger pages, cache synonyms do not occur, and casting out any of the multiple EAs removes the physical information from the cache.

Programming Note: To prevent the occurrence of cache synonyms, use only page sizes greater than the cache way size (8KB), if possible. For the PPC405GP, the minimum such page size is 16KB.

4.1.4 ICU Coherency

The ICU does not "snoop" external memory or the DCU. Programmers must follow special procedures for ICU synchronization when self-modifying code is used or if a peripheral device updates memory containing instructions.

The following code example illustrates the necessary steps for self-modifying code. This example assumes that *addr1* is both data and instruction cachable.

```
stw     regN, addr1   # the data in regN is to become an instruction at addr1
dcbst   addr1         # forces data from the data cache to memory
sync                  # wait until the data actually reaches the memory
icbi    addr1         # the previous value at addr1 might already be in
                        the instruction cache; invalidate it in the cache
isync                 # the previous value at addr1 may already have been
                        pre-fetched into the queue; invalidate the queue
                        so that the instruction must be re-fetched
```

4.2 DCU Organization

The DCU manages data transfers between external cachable memory and the general-purpose registers in the execution unit.

The DCU contains a two-way set-associative 8KB1KB cache memory. Each way is organized in 128 lines of eight words (32 bytes) each.

As shown in Table 4-2, tag ways A and B store data address bits $A_{0:19}$ for each line in cache ways A and B. Data address bits $A_{20:26}$ serve as the index to the cache array. The two cache lines that correspond to the same line index (one in each way) are called a congruence class.

Table 4-2. Data Cache Organization

Tags (Two-way Set)		Data (Two-way Set)	
Way A	**Way B**	**Way A**	**Way B**
$A_{0:19}$ Line 0 A	$A_{0:19}$ Line 0 B	Line 0 A	Line 0 B
$A_{0:19}$ Line 1 A	$A_{0:19}$ Line 1 B	Line 1 A	Line 1 B
• • •	• • •	• • •	• • •
$A_{0:19}$ Line 126 A	$A_{0:19}$ Line 126 B	Line 126 A	Line 126 B
$A_{0:19}$ Line 127 A	$A_{0:19}$ Line 127 B	Line 127 A	Line 127 B

When a cache line is to be loaded, the cache way to receive the line is determined by using an LRU policy. The index, determined by the data address, selects a congruence class. Within a congruence class, the line which was accessed most recently is retained, and the other line is marked as LRU, using an LRU bit in the tag array. The line to receive the incoming data is the LRU line. After the cache line fill, the LRU bit is then set to identify as least-recently-used the line opposite the line just filled.

A bypass path handles data operations in cache-inhibited memory and improves performance during line fill operations.

4.2.1 DCU Operations

Data from cachable memory regions are copied from external memory into lines in the data cache array so that subsequent cache operations result in cache hits. Loads and stores that hit in the DCU are completed in one cycle. For loads, GPRs receive the requested byte, halfword, or word of data from the data cache array. The DCU supports byte-writeability to improve the performance of byte and halfword store operations.

Cache operations require a line fill when they require data from cachable memory regions that are not currently in the DCU. A line fill is the movement of a cache line (eight words) from external memory to the data cache array. Eight words are copied from external memory into the fill buffer, either target-word-first or sequentially. Loading order is controlled by the PLB slave. Target-word-first fills start at the requested word, continue to the end of the line, and then wrap to fill the remaining words at the beginning of the line. Sequential fills start at the first word of the cache line and proceed sequentially to the last word of the line. In both types of fills, the fill buffer, when full, is transferred to the data cache array. The cache line is marked valid when it is filled.

Loads that result in a line fill, and loads from non-cachable memory, are sent to a GPR. The requested byte, halfword, or word is sent from the DCU to the GPR from the fill buffer, using a cache bypass mechanism. Additional loads for data in the fill buffer can be bypassed to the GPR until the data is moved into the data array.

Stores that result in a line fill have their data held in the fill buffer until the line fill completes. Additional stores to the line being filled will also have their data placed in the fill buffer before being transferred into the data cache array.

To complete a line fill, the DCU must access the tag and data arrays. The tag array is read to determine the tag addresses, the LRU line, and whether the LRU line is dirty. A dirty cache line is one that was accessed by a store instruction after the line was established, and can be inconsistent with external memory. If the line being replaced is dirty, the address and the cache line must be saved so that external memory can be updated. During the cache line fill, the LRU bit is set to identify the line opposite the line just filled as LRU.

When a line fill completes and replaces a dirty line, a line flush begins. A flush copies updated data in the data cache array to main storage. Cache flushes are always sequential, starting at the first word of the cache line and proceeding sequentially to the end of the line.

Cache lines are always completely flushed or filled, even if the program does not request the rest of the bytes in the line, or if a bus error occurs after a bus interface unit accepts the request for the line fill. If a bus error occurs during a line fill, the line is filled and the data is marked valid. However, the line can contain invalid data, and a machine check exception occurs.

4.2.2 DCU Write Strategies

DCU operations can use write-back or write-through strategies to maintain coherency with external cachable memory.

The write-back strategy updates only the data cache, not external memory, during store operations. Only modified data lines are flushed to external memory, and then only when necessary to free up locations for incoming lines, or when lines are explicitly flushed using **dcbf** or **dcbst** instructions. The write-back strategy minimizes the amount of external bus activity and avoids unnecessary contention for the external bus between the ICU and the DCU.

The write-back strategy is contrasted with the write-through strategy, in which stores are written simultaneously to the cache and to external memory. A write-through strategy can simplify maintaining coherency between cache and memory.

When data address translation is enabled (MSR[DR] = 1), the W storage attribute in the TLB entry for the memory page controls the write strategy for the page. If TLB_entry[W] = 0, write-back is selected; otherwise, write-through is selected. The write strategy is controlled separately for each page. "Translation Lookaside Buffer (TLB)" on page 6-2 describes the TLB.

When data address translation is disabled (MSR[DR] = 0), the Data Cache Write-through Register (DCWR) sets the storage attribute. Each bit in the DCWR (DCWR[W0:W31]) controls the write strategy of a 128MB storage region (see "Real-mode Storage Attribute Control" on page 6-17). If DCWR[W*n*] = 0, write-back is enabled for the specified region; otherwise, write-through is enabled.

Programming Note: The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

4.2.3 DCU Load and Store Strategies

The DCU can control whether a load receives one word or one line of data from main memory.

For cachable memory, the load without allocate (LWOA) field of the CCR0 controls the type of load resulting from a load miss. If CCR0[LWOA] = 0, a load miss causes a line fill. If CCR0[LWOA] = 1, load misses do not result in a line fill, but in a word load from external memory. For infrequent reads of non-contiguous memory, setting CCR0[LWOA] = 1 may provide a small performance improvement.

For non-cachable memory and for loads misses when CCR0[LWOA] = 1, the load word as line (LWL) field in the CCR0 affects whether load misses are satisfied with a word, or with eight words (the equivalent of a cache line) of data. If CCR0[LWL] = 0, only the target word is bypassed to the core. If CCR0[LWL] = 1, the DCU saves eight words (one of which is the target word) in the fill buffer and bypasses the target data to the core to satisfy the load word request. The fill buffer is not written to the data cache array.

Setting CCR0[LWL] = 1 provides the fastest accesses to sequential non-cachable memory. Subsequent loads from the same line are bypassed to the core from the fill buffer and do not result in additional external memory accesses. The load data remains valid in the fill buffer until one of the following occurs: the beginning of a subsequent load that requires the fill buffer, a store to the target address, a **dcbi** or **dccci** instruction issued to the target address, or the execution of a **sync** instruction. Non-cachable loads to guarded storage never cause a line transfer on the PLB even if CCR0[LWL] = 1. Subsequent loads to the same non-cachable storage are always requested again from the PLB.

For cachable memory, the store without allocate (SWOA) field of the CCR0 controls the type of store resulting from a store miss. If CCR0[SWOA] = 0, a store miss causes a line fill. If CCR0[SWOA] = 1, store misses do not result in a line fill, but in a single word store to external memory.

4.2.4 Data Cachability Control

When data address translation is disabled (MSR[DR] = 0), data cachability is controlled by the Data Cache Cachability Register (DCCR). Each bit in the DCCR (DCCR[S0:S31]) controls the cachability of a 128MB region (see "Real-mode Storage Attribute Control" on page 6-17). If DCCR[S*n*] = 1, caching is enabled for the specified region; otherwise, caching is inhibited.

When data address translation is enabled (MSR[DR] = 1), data cachability is controlled by the I bit in the TLB entry for the memory page. If TLB_entry[I] = 1, caching is inhibited; otherwise caching is enabled. Cachability is controlled separately for each page, which can range in size from 1KB to 16MB. "Translation Lookaside Buffer (TLB)" on page 6-2 describes the TLB.

> **Programming Note:** The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

The performance of the PPC405GP is significantly lower while accessing memory in cache-inhibited regions.

Following system reset, address translation is disabled and all DCCR bits are reset to 0 so that no memory regions are cachable. The **dccci** instruction must execute 128before regions can be designated as cachable. This invalidates all congruence classes before enabling the cache. Address translation can then be enabled, if required, and the TLB or the DCCR can then be configured for the desired cachability.

> **Programming Note:** If a data block corresponding to the effective address (EA) exists in the cache, but the EA is non-cachable, loads and stores (including **dcbz**) to that address are considered programming errors (the cache block should previously have been flushed). The only instructions that can legitimately access such an EA in the data cache are the cache management instructions **dcbf**, **dcbi**, **dcbst**, **dcbt**, **dcbtst**, **dccci**, and **dcread**.

4.2.5 DCU Coherency

The DCU does not provide snooping. Application programs must carefully use cache-inhibited regions and cache control instructions to ensure proper operation of the cache in systems where external devices can update memory.

4.3 Cache Instructions

For detailed descriptions of the instructions described in the following sections, see Chapter 24, "Instruction Set."

In the instruction descriptions, the term "block" is synonymous with cache line. A block is the unit of storage operated on by all cache block instructions.

4.3.1 ICU Instructions

The following instructions control instruction cache operations:

icbi	Instruction Cache Block Invalidate Invalidates a cache block.
icbt	Instruction Cache Block Touch Initiates a block fill, enabling a program to begin a cache block fetch before the program needs an instruction in the block. The program can subsequently branch to the instruction address and fetch the instruction without incurring a cache miss. This is a privileged instruction.
iccci	Instruction Cache Congruence Class Invalidate Invalidates the instruction cache array. This is a privileged instruction.
icread	Instruction Cache Read Reads either an instruction cache tag entry or an instruction word from an instruction cache line, typically for debugging. Fields in CCR0 control instruction behavior (see "Cache Control and Debugging Features" on page 4-11). This is a privileged instruction.

4.3.2 DCU Instructions

Data cache flushes and fills are triggered by load, store and cache control instructions. Cache control instructions are provided to fill, flush, or invalidate cache blocks.

The following instructions control data cache operations.

dcba	Data Cache Block Allocate Speculatively establishes a line in the cache and marks the line as modified. If the line is not currently in the cache, the line is established and marked as modified without actually filling the line from external memory. If dcba references a non-cachable address, **dcba** is treated as a no-op. If dcba references a cachable address, write-through required (which would otherwise cause an alignment exception), **dcba** is treated as a no-op.
dcbf	Data Cache Block Flush Flushes a line, if found in the cache and marked as modified, to external memory; the line is then marked invalid. If the line is found in the cache and is not marked modified, the line is marked invalid but is not flushed. This operation is performed regardless of whether the address is marked cachable.
dcbi	Data Cache Block Invalidate Invalidates a block, if found in the cache, regardless of whether the address is marked cachable. Any modified data is not flushed to memory. This is a privileged instruction.
dcbst	Data Cache Block Store Stores a block, if found in the cache and marked as modified, into external memory; the block is not invalidated but is no longer marked as modified. If the block is marked as not modified in the cache, no operation is performed. This operation is performed regardless of whether the address is marked cachable.
dcbt	Data Cache Block Touch Fills a block with data, if the address is cachable and the data is not already in the cache. If the address is non-cachable, this instruction is a no-op.
dcbtst	Data Cache Block Touch for Store Implemented identically to the **dcbt** instruction for compatibility with compilers and other tools.

dcbz	Data Cache Block Set to Zero Fills a line in the cache with zeros and marks the line as modified. If the line is not currently in the cache (and the address is marked as cachable and non-write-through), the line is established, filled with zeros, and marked as modified without actually filling the line from external memory. If the line is marked as either non-cachable or write-through, an alignment exception results.
dccci	Data Cache Congruence Class Invalidate Invalidates a congruence class (both cache ways). This is a privileged instruction.
dcread	Data Cache Read Reads either a data cache tag entry or a data word from a data cache line, typically for debugging. Bits in CCR0 control instruction behavior (see “Cache Control and Debugging Features” on page 4-11). This is a privileged instruction.

4.4 Cache Control and Debugging Features

Registers and instructions are provided to control cache operation and help debug cache problems. For ICU debug, the **icread** instruction and the Instruction Cache Debug Data Register (ICDBDR) are provided. See “ICU Debugging” on page 4-14 for more information. For DCU debug, the **dcread** instruction is provided. See “DCU Debugging” on page 4-15 for more information.

CCR0 controls the behavior of the **icread** and the **dcread** instructions.

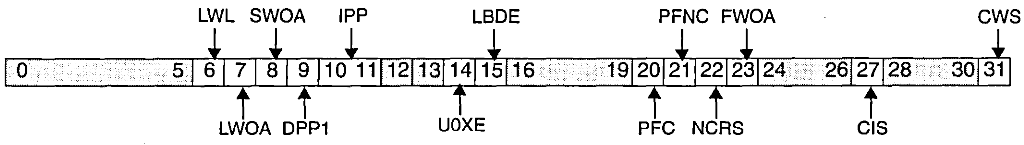


Figure 4-2. Core Configuration Register 0 (CCR0)

0:5		Reserved
6	LWL	Load Word as Line 0 The DCU performs load misses or non-cachable loads as words, halfwords, or bytes, as requested 1 For load misses or non-cachable loads, the DCU moves eight words (including the target word) into the line buffer

7	LWOA	Load Without Allocate 0 Load misses result in line fills 1 Load misses do not result in a line fill, but in non-cachable loads	
8	SWOA	Store Without Allocate 0 Store misses result in line fills 1 Store misses do not result in line fills, but in non-cachable stores	
9	DPP1	DCU PLB Priority Bit 1 0 DCU PLB priority 0 on bit 1 1 DCU PLB priority 1 on bit 1	**Note:** DCU logic dynamically controls DCU priority bit 0.
10:11	IPP	ICU PLB Priority Bits 0:1 00 Lowest ICU PLB priority 01 Next to lowest ICU PLB priority 10 Next to highest ICU PLB priority 11 Highest ICU PLB priority	
12:13	▨	Reserved	
14	U0XE	Enable U0 Exception 0 Enables the U0 exception 1 Disables the U0 exception	
15	LDBE	Load Debug Enable 0 Load data is invisible on data-side (on-chip memory (OCM) 1 Load data is visible on data-side OCM	
16:19	▨	Reserved	
20	PFC	ICU Prefetching for Cachable Regions 0 Disables prefetching for cachable regions 1 Enables prefetching for cachable regions	
21	PFNC	ICU Prefetching for Non-Cachable Regions 0 Disables prefetching for non-cachable regions 1 Enables prefetching for non-cachable regions	
22	NCRS	Non-cachable ICU request size 0 Requests are for four-word lines 1 Requests are for eight-word lines	
23	FWOA	Fetch Without Allocate 0 An ICU miss results in a line fill. 1 An ICU miss does not cause a line fill, but results in a non-cachable fetch.	
24:26	▨	Reserved	
27	CIS	Cache Information Select 0 Information is cache data. 1 Information is cache tag.	
28:30	▨	Reserved	

31	CWS	Cache Way Select 0 Cache way is A. 1 Cache way is B.

4.4.1 CCR0 Programming Guidelines

Several fields in CCR0 affect ICU and DCU operation. Altering these fields while the cache units are involved in PLB transfers can cause errant operation, including a processor hang.

To guarantee correct ICU and DCU operation, specific code sequences must be followed when altering CCR0 fields.

CCR0[IPP, FWOA] affect ICU operation. When these fields are altered, execution of the following code sequence (Sequence 1) is required.

```
    ! SEQUENCE 1 Altering CCR0[IPP, FWOA]
    ! Turn off interrupts
    mfmsr     RM
    addis     RZ,r0,0x0002  ! CE bit
    ori       RZ,RZ,0x8000  ! EE bit
    andc      RZ,RM,RZ      ! Turn off MSR[CE,EE]
    mtmsr     RZ
    ! sync
    sync
    ! Touch code sequence into i-cache
    addis     RX,r0,seq1@h
    ori       RX,RX,seq1@l
    icbt       r0,RX
! Call function to alter CCR0 bits
    b seq1
back:
! Restore MSR to original value
    mtmsr     RM
              •
              •
              •
! The following function must be in cacheable memory
    .align 5     ! Align CCR0 altering code on a cache line boundary.
    seq1:
    icbt      r0,RX          ! Repeat ICBT and execute an ISYNC to guarantee CCR0
    isync                    ! altering code has been completely fetched across the PLB.
    mfspr     RN,CCR0       ! Read CCR0.
    andi/ori RN,RN,0xXXXX    ! Execute and/or function to change any CCR0 bits.
                             ! Can use two instructions before having to touch
                             ! in two cache lines.
    mtspr     CCR0, RN ! Update CCR0.
    isync          ! Refetch instructions under new processor context.
    b    back      ! Branch back to initialization code.
```

CCR0[DPP1, U0XE] affect DCU operation. When these fields are altered, execution of the following code sequence (Sequence 2) is required. Note that Sequence 1 includes Sequence 2, so Sequence 1 can be used to alter any CCR0 fields.

In the following sample code, registers RN, RM, RX, and RZ are any available GPRs.

```
! SEQUENCE 2 Alter CCR0[DPP1, U0XE)
! Turn off interrupts
    mfmsr     RM
    addis     RZ,r0,0x0002  ! CE bit
    ori       RZ,RZ,0x8000  ! EE bit
    andc      RZ,RM,RZ     ! Turn off MSR[CE,EE]
    mtmsr     RZ
! sync
    sync
! Alter CCR0 bits
    mfspr     RN,CCR0      ! Read CCR0.
    andi/ori  RN,RN,0xXXXX   ! Execute and/or function to change any CCR0 bits.
    mtspr     CCR0, RN     ! Update CCR0.
    isync          ! Refetch instructions under new processor context.
! Restore MSR to original value
    mtmsr    RM
```

CCR0[CIS, CWS] do not require special programming.

4.4.2 ICU Debugging

The **icread** instruction enables the reading of the instruction cache entries for the congruence class specified by $EA_{19:26}$. The cache information is read into the ICDBDR; from there it can subsequently be moved, using a **mfspr** instruction, into a GPR.

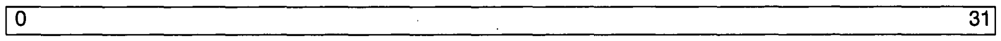


Figure 4-3. Instruction Cache Debug Data Register (ICDBDR)

0:31		Instruction cache information See **icread**, page 24-68.

ICU tag information is placed into the ICDBDR as shown:

0:21	TAG	Cache Tag
22:26		Reserved
27	V	Cache Line Valid 0 Not valid 1 Valid
28:30		Reserved
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU

If CCR0[CIS] = 0, the data is a word of ICU data from the addressed line, specified by $EA_{27:29}$. If CCR0[CWS] = 0, the data is from the A-way; otherwise; the data from the B-way.

If CCR0[CIS] = 1, the cache information is the cache tag. If CCR0[CWS] = 0, the tag is from the A-way; otherwise, the tag is from the B-way.

Programming Note: The instruction pipeline does not wait for data from an **icread** instruction to arrive before attempting to use the contents the ICDBDR. The following code sequence ensures proper results:

```
icread r5,r6# read cache information
isync       # ensure completion of icread
mficdbdr r7# move information to GPR
```

4.4.3 DCU Debugging

The **dcread** instruction provides a debugging tool for reading the data cache entries for the congruence class specified by $EA_{20:26}$. The cache information is read into a GPR.

If CCR0[CIS] = 0, the data is a word of DCU data from the addressed line, specified by $EA_{27:29}$. If CCR0[CWS] = 0, the data is from the A-way; otherwise; the data is from the B-way.

If CCR0[CIS] = 1, the cache information is the cache tag. If CCR0[CWS] = 0, the tag is from the A-way; otherwise the tag is from the B-way.

DCU tag information is placed into the GPR as shown:

0:19	TAG	Cache Tag
20:25		Reserved
26	D	Cache Line Dirty 0 Not dirty 1 Dirty
27	V	Cache Line Valid 0 Not valid 1 Valid
28:30		Reserved
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU

Note: A “dirty” cache line is one which has been accessed by a store instruction after it was established, and can be inconsistent with external memory.

4.5 DCU Performance

DCU performance depends upon the application, but, in general, cache hits complete in one cycle without stalling the CPU pipeline. Under certain conditions and limitations of the DCU, the pipeline stalls (stops executing instructions) until the DCU completes current operations.

Several factors affect DCU performance, including:

- Pipeline stalls
- DCU priority
- Simultaneous cache operations
- Sequential cache operations

4.5.1 Pipeline Stalls

The CPU issues commands for cache operations to the DCU. If the DCU can immediately perform the requested cache operation, no pipeline stall occurs. In some cases, however, the DCU cannot immediately perform the requested cache operation, and the pipeline stalls until the DCU can perform the pending cache operation.

In general, the DCU, when hitting in the cache array, can execute a load/store every cycle. If a cache miss occurs, the DCU must retrieve the line from main memory. For cache misses, the DCU stores the cache line in a line buffer until the entire cache line is received. The DCU can accept new DCU commands while the fill progresses. If the instruction causing the line fill is a load, the target word is bypassed to the GPR during the cycle after it becomes available in the fill buffer. When the fill buffer is full, it must be moved into the tag and data arrays. During this time, the DCU cannot begin a new cache operation and stalls the pipeline if new DCU commands are presented. Storing a line in the line buffer takes 3 cycles, unless the line being replaced has been modified. In that case, the operation takes 4 cycles.

The DCU can accept up to two load commands. If the data for the first load command is not immediately available, the DCU can still accept the second load command. If the load data is not required by subsequent instructions, those instructions will continue to execute. If data is required from either load command, the CPU pipeline will stall until the load data has been delivered. The pipeline will also stall until the second load has read the data array if a subsequent data cache command is issued.

In general, if the fill buffer is being used and the next load or store command requires the fill buffer, only one additional command can be accepted before causing additional DCU commands to stall the pipeline.

The DCU can accept up to three outstanding store commands before stalling the CPU pipeline for additional data cache commands.

The DCU can have two flushes pending before stalling the CPU pipeline.

DCU cache operations other than loads and stores stall the CPU pipeline until all prior data cache operations complete. Any subsequent data cache command will stall the pipeline until the prior operation is complete.

The pipeline stalls when on-chip memory (OCM) asserts a hold signal. For loads or stores that are held, the DCU can still accept one additional load or store command before stalling the pipeline for subsequent commands.

4.5.2 Cache Operation Priorities

The DCU uses a priority signal to improve performance when pipeline stalls occur. When the pipeline is stalled because of a data cache operation, the DCU asserts the priority signal to the PLB. The priority signal tells the external bus that the DCU requires immediate service, and is valid only when the data cache is requesting access to the PLB. The priority signal is asserted for all loads that require external data, or when the data cache is requesting the PLB and stalling an operation that is being presented to the data cache.

Table 4-3 provides examples of when the priority is asserted and deasserted.

Table 4-3. Priority Changes With Different Data Cache Operations

Instruction Requesting PLB	Priority	Next Instruction	Priority
Any load from external memory	1	N/A	N/A
Any store	0	Any other cache operation not being accepted by the DCU.	1
dcbf	0	Any cache hit.	0
dcbf/dcbst	0	Load non-cache.	1
dcbf/dcbst	0	Another command that requires a line flush.	1
dcbt	0	Any cache hit.	0
dcbi/dccci/dcbz	0	N/A	N/A

4.5.3 Simultaneous Cache Operations

Some cache operations can occur simultaneously to improve DCU performance. For example, combinations of line fills, line flushes, word load/stores, and operations that hit in the cache can occur simultaneously. Cache operations other than loads/stores cannot begin until the PLB completes all previous operations.

4.5.4 Sequential Cache Operations

Some common cache operations, when performed sequentially, can limit DCU performance: sequential loads/stores to non-cachable storage regions, sequential line fills, and sequential line flushes.

In the case of sequential cache hits, the most commonly occurring operations, the DCU loads or stores data every cycle. In such cases, the DCU does not limit performance.

However, when a load from a non-cachable storage region is followed by multiple loads from non-cachable regions, the loads can complete no faster than every four cycles, assuming that the addresses are accepted during the same cycle in which it is requested, and that the data is returned during the cycle after the load is accepted.

Similarly, when a store to a non-cachable storage region is followed by multiple stores to non-cachable regions the fastest that the stores can complete is every other cycle. The DCU can have accepted up to three stores before additional DCU commands will stall waiting for the prior stores to complete.

Sequential line fills can limit DCU performance. Line fills occur when a load/store or **dcbt** instruction misses in the cache, and can be pipelined on the PLB interface such that up to two requests can be accepted before stalling subsequent requests. The subsequent operations will wait in the DCU until the first line fill completes. The line fills must complete in the order that they are accepted.

Sequential line flushes from the DCU to main memory also limit DCU performance. Flushes occur when a line fill replaces a valid line that is marked dirty (modified), or when a **dcbf** instruction flushes a specific line. If two flushes are pending, the DCU stalls any new data cache operations until the first flush finishes and the second flush begins.

Chapter 5. On-Chip Memory

The on-chip memory (OCM) subsystem consists of a memory controller that connects the PPC405GP processor core to a one-port, 4KB on-chip SRAM array. OCM is ideal for applications requiring low-latency access to critical instructions and data. OCM can provide performance that is identical to cache hits, yet, unlike a cache, the OCM never misses. Instructions and data stored in the OCM are always available because OCM contents only change under program control. Therefore, if the programmer avoids instruction-side and data-side OCM access contention, OCM can provide information availability that is superior to a cache line locking scheme. OCM is superior because it can provide single cycle performance identical to cache hits without locking down portions of the cache. This results in more effective cache utilization for the processor.

Instructions and data returned from OCM interface do not flow through the PPC405GP core caches. The caches remain available for caching from other memory sources accessed across the PLB interface. The system designer must ensure that each address has a single access path into the PPC405GP core for a given software process. Each address that is requested should be found in either the OCM address space or the PLB address space, *but not in both.*

Code to initialize OCM should execute in non-OCM address space in a region marked as non-cachable. The initialization code should invalidate the cache arrays (in the instruction cache unit (ICU) and data cache unit (DCU), as appropriate) to ensure that no addresses to be programmed as OCM space are in the cache. After programming the OCM address and control registers, the OCM address space should remain marked as non-cachable. Chip initialization for OCM usage is described in "OCM Initialization" on page 8-13.

Read and write accesses to the OCM array share a single access port. OCM accesses have the following priorities:

1. Data-side OCM reads (loads)
2. Data-side OCM writes (stores)
3. Instruction-side OCM read (fetches)

Data-side OCM reads occur in one cycle. Data-side writes also complete in one cycle, though they can be pre-empted by higher priority data-side reads. Instruction-side OCM reads occur by default (that is, after a reset) in two cycles. However, when the Instruction-Side Two-Cycle Mode field of the OCM Instruction-Side Control Register is set to 0 (OCM0_ISCNTL[ISTCM] = 0), instruction-side OCM reads occur in one cycle, unless pre-empted by higher priority data-side transfers. Two-cycle mode is provided for chips that cannot make instruction-side timing to the processor core. The PPC405GP, however, meets the timing requirement. Therefore, programmers should set OCM0_ISCNTL[ISTCM] = 0 during chip initialization, as described in "OCM Initialization" on page 8-13.

5.1 OCM Addressing

The address space for the instruction-side OCM and the data side OCM are defined by the OCM Instruction-Side Address Range Compare Register (OCM0_ISARC) and OCM Data-Side Address Range Compare Register (OCM0_DSARC), respectively. These registers are implemented as 6-bit registers that define the most significant address bits of the respective OCM address space. Using 6 bits defines a 64MB address space. The instruction side and data side can share a 64MB address space, or each can have its own 64MB address space. The address spaces are fully relocatable on 64MB boundaries within the 4GB address space of the PPC405GP, but the programmer must assign OCM address space to avoid conflicts with other assigned addresses. See "Programming Model" on page 3-1 for information about the PPC405GP memory map.

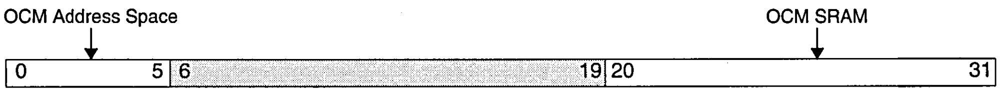


Figure 5-1. OCM Address Usage

Figure 5-1 illustrates OCM address usage. The OCM SRAM array size is 4KB. Address bits 20:31 select byte addresses for data-side accesses. Address bits 30:31 are ignored for instruction-side accesses, because instruction-side accesses return either one or two words per transfer.

Note that the instruction-side and data-side OCM address spaces overlap physically, even if defined as distinct logical address spaces, because the 4KB SRAM is shared. There is no distinction between data space or instruction space, except as defined by the programmer.

Addresses in the OCM array are aliased throughout the larger OCM address spaces. The larger OCM address spaces are filled with multiple images of the 4KB SRAM. Aliased addresses refer to the same physical memory locations.

> **Programming Note:** To avoid possible memory coherency problems when using aliased addresses, align aliased addresses on 16KB boundaries rather than on 4KB boundaries. See "Store Data Bypass Behavior and Memory Coherency" on page 5-3 for details.

If address translation is enabled (MSR[IR, DR] = 1), one or more TLB entries for the OCM address space must exist to validate accesses. However, the virtual addresses are *not* translated, and 32-bit effective addresses (virtual addresses) are presented to OCM.

Data-side OCM contents can use big endian or little endian byte ordering. Instruction-side OCM contents *must* use big endian byte ordering. See "Byte Ordering" on page 3-28 for detailed information about byte ordering.

5.2 OCM Programming Guidelines

The following guidelines prevent potential problems associated with using OCM:

- Code that uses **mtdcr** to disable instruction-side OCM access should not run out of the instruction-side OCM.

  Instructions following an **mtdcr** are not guaranteed to be fetched before the instruction-side OCM is disabled.

- Do not change the value in OCM0_ISARC while fetching from the instruction-side OCM.
- To change the value in OCM0_ISARC or OCM0_DSARC:

  1. Set OCM0_ISCNTL[ISEN] = 0 to disable instruction-side OCM accesses, or set OCM0_DSCNTL[DSEN] = 0 to disable data-side OCM accesses.

2. Clear MSR[EE] and MSR[CE] to mask interrupts, to ensure that interrupts do not interfere with the cache invalidation described in Step 4. This avoids a potential problem with "dirty" cache addresses that would not be fetched from the cache because they have been marked as non-cachable.
3. Mark the address region to be programmed as OCM address space as noncachable.
4. Invalidate the cache array that corresponds to the OCM (instruction-side or data-side) whose address range compare register is to be modified to ensure that no addresses to be programmed as OCM addresses exist in the cache. A single **iccci** instruction invalidates the ICU cache array. To invalidate the DCU cache array, use a sequence of **dcbf** instructions (one per cache line).
5. Modify the value in OCM0_ISARC or OCM0_DSARC.
6. Set OCM0_ISCNTL[ISEN] = 1 to enable instruction-side OCM accesses, or set OCM0_DSCNTL[DSEN] = 1 to enable data-side OCM accesses.

- Self-modifying code that accesses OCM to update instructions should not fetch instructions from the area being modified until a **sync** instruction executes, followed by an **isync** instruction.

  The **sync** instruction ensures that instructions are updated. The **isync** instruction ensures that only updated instructions are fetched into the pipeline.

  Instructions in OCM can be updated while instructions from non-OCM addresses execute. A **sync-isync** pair should still be used whenever such self-modifying code is updated.

- The CPU can become less efficient when instructions and data in OCM are accessed at the same time, because the SRAM has only one access port and instruction fetches have the lowest priority.

  For example, instructions fetched from OCM that contain several sequential data-side loads accessing OCM can result in bubbles in the instruction pipeline. The sequential data-side loads dominate OCM accesses, resulting in the inability to fetch instructions from OCM.

- If aliased addresses are used, the aliased 4KB address spaces should be aligned on 16KB boundaries to eliminate potential store data bypass problems, as described in "Store Data Bypass Behavior and Memory Coherency."

5.3 Store Data Bypass Behavior and Memory Coherency

The OCM subsystem provides only one mechanism, data-side store operations, for writing both instructions and data into the OCM array. However, two independent mechanisms request read access of OCM contents; one for instruction-side fetches and the other for data-side loads.

The following description applies only to applications that alias the OCM address space and perform a mix of data-side loads and stores. It does not apply to applications that use data-side stores only to initialize OCM with instructions.

If a data-side OCM store is followed in the next cycle by a data-side load, the load actually accesses the OCM array before the store. This is due to the nature of the processor pipeline, the cycle availability of the store data, and the fact that data-side loads have a higher priority than data-side stores. In this scenario, store data is queued in a register while the load accesses the array. Further, if the store is immediately followed by a sequence of consecutive loads, it remains in the queue until the last of the consecutive loads has accessed the OCM array. The queued store data is written into the OCM array in the first cycle that does not have a data-side load operation accessing the array.

Consider a scenario where such a situation causes store data to be held in the store data queue. If any of the loads access the same address as the address of the store operation whose data is being held in the store data queue, there is a need to bypass the store data from the store data queue to provide the correct data to the load operation.

A bypass is determined to be required by comparing the pending store address with the load address. However, the comparison is done with a 16KB address representation for the load and store operations, not the 4KB address (the physical size of the PPC405GP OCM array). If the 16KB address compares, the store data is bypassed to the load operation. This implies that a bypass results for address aliasing only when the OCM addresses match at a 16KB multiple, which corresponds to a match of address bits 18:29 (a word address that is further specified by byte enables). Although the physical address space is aliased at 4KB multiples, the bypass determination is made at 16KB multiples. Therefore, if bits 18:19 of an aliased load address do not match bits 18:19 of the 16KB store address of the data being held in the store data queue, the load data will not be coherent. Instead of returning the most recently stored data, which is being held in the store data queue, the load returns "old" data previously stored in and accessed from the OCM array.

Table 5-1 provides examples that describe bypass behavior when address aliasing is used.

Table 5-1. Examples of Store Data Bypass

Example	Store Address	Load Address	4KB Aliased Address	16KB Aliased Address	Bypass
1	0x00000100	0x00000100	Same	Same	Yes
2	0x00000100	0x00000400	No	No	No
3	0x00000100	0x00001100	Yes	No, loads old data	No
4	0x00000100	0x00005100	Yes	No, loads old data	No
5	0x00000100	0x00004100	Yes	Yes	Yes
6	0x00000100	0x00008100	Yes	Yes	Yes

Example 1 provides the most basic example, in which the load and store addresses are the same. This results in the load accessing the queued store data, bypassing the OCM array to satisfy the load.

Example 2 shows two different addresses that are *not* aliased (both addresses are in the 4KB SRAM address space). No bypass occurs, and the load returns the correct data from the OCM array.

Examples 3 and 4 show aliased addresses that do not bypass data because the addresses do not compare within a 16KB address space. In both examples, address bits 18:19 do not match. In both examples, the load does not return the most recently stored data from the store data queue; the load returns the "old' data from the array. To avoid such problems, alias on 16KB boundaries. If addresses are aliased on 4KB boundaries, place at least one instruction that does not access the data-side OCM between a load and a store to the same aliased address so the store data has a cycle to be written into the array.

Examples 5 and 6 bypass data out of the store data queue because the aliased addresses compare within a 16KB address space. In both examples, address bits 18:29 match, and load data is returned from the store data queue.

5.4 Registers

The OCM controller uses the Device Control Registers (DCRs) listed in Table 5-2.

Table 5-2. OCM DCRs

Register	Mnemonic	DCR Number	Access	Page
OCM Instruction-Side Address Range Compare Register	OCM0_ISARC	0x018	R/W	5-5
OCM Instruction-Side Control Register	OCM0_ISCNTL	0x019	R/W	5-6
OCM Data-Side Address Range Compare Register	OCM0_DSARC	0x01A	R/W	5-6
OCM Data-Side Control Register	OCM0_DSCNTL	0x01B	R/W	5-7

5.4.1 OCM Instruction-Side Address Range Compare Register (OCM0_ISARC)

OCM0_ISARC defines the address range of the OCM controller when it is presented with instruction fetch requests. OCM0_ISARC[ISAR] is compared to the high-order 6 bits of the requested instruction address, providing a 64MB address space. The address space can be shared with, or distinct from, the data-side OCM address space.

The OCM controller returns requested instructions if instruction-side OCM access is enabled (OCM0_ISCNTL[ISEN] = 1) and OCM0_ISARC[ISAR] matches the high-order 6 bits of the requested instruction address.

OCM0_ISARC must be initialized before OCM0_ISCNTL[ISEN] is set to 1 to enable instruction-side OCM accesses. See "OCM Initialization" on page 8-13 for details

.

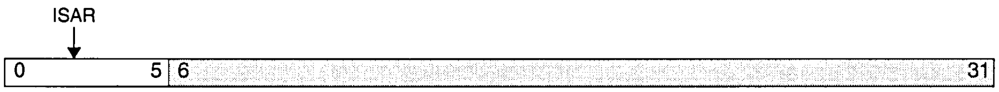


Figure 5-2. OCM Instruction-Side Address Range Compare Register (OCM0_ISARC)

0:5	ISAR	Instruction-side OCM address range
6:31	▨	Reserved

5.4.2 OCM Instruction-Side Control Register (OCM0_ISCNTL)

OCM0_ISCNTL enables and disables instruction-side OCM access and controls whether instruction requests are satisfied in one or two cycles.

OCM0_ISCNTL[ISEN] enables the OCM controller to respond to requests for instruction fetches to addresses in the instruction-side OCM address range defined by OCM0_ISARC[ISAR]. At reset, OCM0_ISCNTL[ISEN] = 0; instruction-side OCM is not enabled. If instruction-side OCM is to be accessed, this field must be set to 1 during chip initialization, as described in “OCM Initialization” on page 8-13.

Setting OCM0_ISCNTL[ISTCM] = 1 places the instruction-side OCM in a mode in which accesses complete in no fewer than two cycles. Two-cycle mode is provided for chips that cannot make instruction-side timing to the processor core. The PPC405GP, however, meets the timing requirement. At reset, OCM0_ISCNTL[ISTCM] = 1. This field should be set to 0 during chip initialization so that instruction-side accesses can complete in one cycle. OCM0_ISCNTL[ISTCM] does not affect data-side OCM operation.

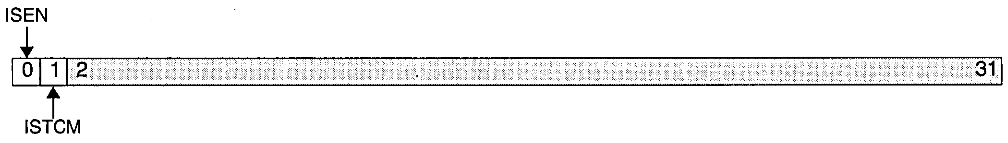


Figure 5-3. OCM Instruction-Side Control Register (OCM0_ISCNTL)

0	ISEN	Instruction-Side OCM Enable 0 Instruction-side OCM accesses are disabled. 1 Instruction-side OCM accesses are enabled.	
1	ISTCM	Instruction-Side Two Cycle Mode 0 Instruction-side OCM accesses are returned in one cycle. 1 Instruction-side OCM accesses are returned in two cycles.	OCM0_ISCNTL[ISTCM], which has a reset value of 1, should be set to OCM0_ISCNTL[ISTCM] = 0 during chip initialization.
2:31	▨	Reserved	

5.4.3 OCM Data-Side Address Range Compare Register (OCM0_DSARC)

OCM0_DSARC defines the address range of the OCM controller when it is presented with load and store requests. OCM0_DSARC[DSAR] is compared to the high-order 6 bits of the requested data address, providing a 64MB address space. The address space can be shared with, or distinct from, the instruction-side OCM address space.

The OCM controller transfers the requested load/store data if data-side OCM access is enabled (OCM0_DSCNTL[DSEN] = 1) and OCM0_DSARC[DSAR] matches the high-order 6 bits of the requested data address.

OCM0_DSARC must be initialized before OCM0_DSCNTL[DSEN] is set to 1 to enable data-side OCM accesses. See "OCM Initialization" on page 8-13 for details.

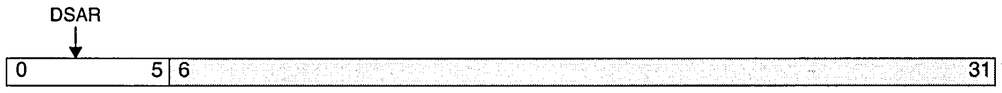


Figure 5-4. OCM Data-Side Address Range Compare Register (OCM0_DSARC)

0:5	DSAR	Data-side OCM address range
6:31	▨	Reserved

5.4.4 OCM Data-Side Control Register (OCM0_DSCNTL)

OCM0_DSCNTL enables and disables data-side OCM access.

OCM0_DSCNTL[DSEN] enables the OCM controller to respond to requests for data loads and stores within the data-side OCM address range defined by OCM0_DSARC[DSAR]. At reset, OCM0_DSCNTL[DSEN] = 0; data-side OCM is not enabled. If data-side OCM is to be accessed, this field must be set to 1 during chip initialization, as described in "OCM Initialization" on page 8-13.

The reset value of the DOF field is 1. This field should always remain set to 1 when writing OCM0_DSCNTL.

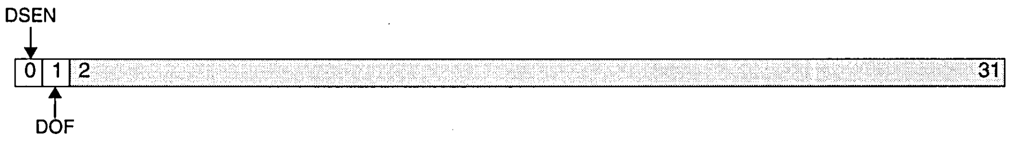


Figure 5-5. OCM Data-Side Control Register (OCM0_DSCNTL)

0	DSEN	Data-Side OCM Enable 0 Data-side OCM accesses are disabled. 1 Data-side OCM accesses are enabled.
1	DOF	This field should remain set to 1.
2:31	▨	Reserved

Chapter 6. Memory Management

The PPC405GP memory management unit (MMU) performs address translation and protection functions. With appropriate system software, the MMU supports:

- Translation of effective addresses to real addresses
- Independent enabling of instruction and data address translation and protection
- Page-level access control using the translation mechanism
- Software control of page replacement strategy
- Additional virtual-mode control of protection using zones
- Real-mode write protection

6.1 MMU Overview

The instruction and integer units generate 32-bit effective addresses (EAs) for instruction fetches and data accesses, respectively. Instruction EAs are generated for sequential instruction fetches, and for instruction fetches causing changes in program flow (branches and interrupts). Data EAs are generated for load/store and cache control instructions. The MMU translates EAs into real addresses; the instruction cache unit (ICU) and data cache unit (DCU) use real addresses to access memory.

The PPC405GP MMU supports demand-paged virtual memory and other memory management schemes that depend on precise control of effective to real address mapping and flexible memory protection. Translation misses and protection faults cause precise interrupts. Sufficient information is available to correct the fault and restart the faulting instruction.

The MMU divides storage into pages. A page represents the granularity of EA translation and protection controls. Eight page sizes (1KB, 4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB) are simultaneously supported. A valid entry for a page containing the EA to be translated must be in the translation lookaside buffer (TLB) for address translation to be performed. EAs for which no valid TLB entry exists cause TLB-miss interrupts.

6.2 Address Translation

Fields in the Machine State Register (MSR) control the use of the MMU for address translation. The instruction relocate (IR) field of the MSR controls translation for instruction accesses. The data relocate (DR) field of the MSR controls the translation mechanism for data accesses. These fields, specified independently, can be changed at any time by a program in supervisor state. Note that all interrupts clear MSR[IR, DR] and place the processor in the supervisor state. Subsequent discussion about translation and protection assumes that MSR[IR, DR] are set, enabling address translation.

The processor references memory when it fetches an instruction, and when it executes load/store, branch, and cache control instructions. Processor accesses to memory use EAs to references a memory location. When translation is enabled, the EA is translated into a real address, as illustrated in Figure 6-1 on page 6-2. The ICU or DCU uses the real address for the access. (When translation is not enabled, the EA is already a real address.)

In address translation, the EA is combined with an 8-bit process ID (PID) to create a 40-bit virtual address. The virtual address is compared to all of the TLB entries. A matching entry supplies the real address for the storage reference. Figure 6-1 illustrates the process.

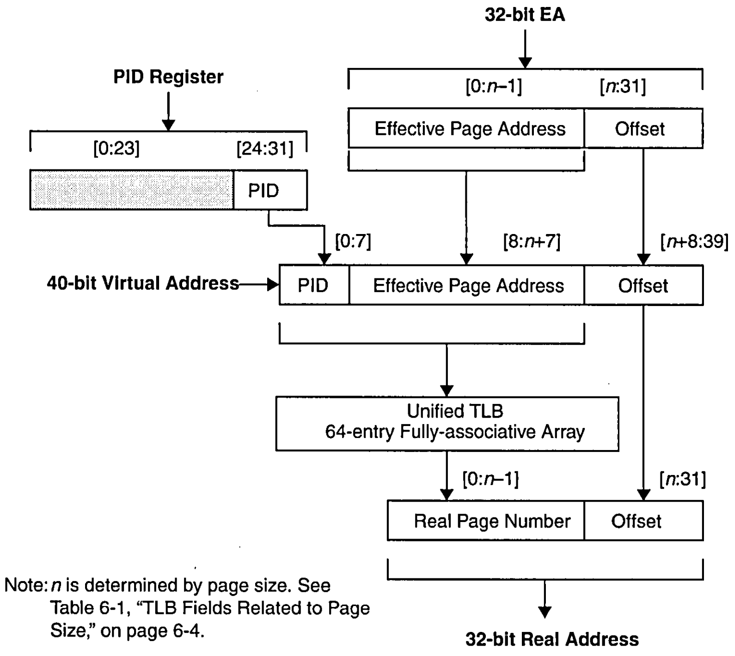


Figure 6-1. Effective to Real Address Translation Flow

6.3 Translation Lookaside Buffer (TLB)

The TLB is hardware that controls translation, protection, and storage attributes. The instruction and data units share a unified fully-associative TLB, in which any page entry (TLB entry) can be placed anywhere in the TLB. TLB entries are maintained under program control. System software determines the TLB entry replacement strategy and the format and use of page state information. A TLB entry contains the information required to identify the page, to specify translation and protection controls, and to specify the storage attributes.

6.3.1 Unified TLB

The unified TLB (UTLB) contains 64 entries; each has a TLBHI (tag) portion and a TLBLO (data) portion, as described in Figure 6-2 on page 6-3. TLBHI contains 36 bits; TLBLO contains 32 bits. When translation is enabled, the UTLB tag portion compares some or all of $EA_{0:21}$ with some or all of the effective page number $EPN_{0:21}$, based on the size bits $SIZE_{0:2}$. All 64 entries are simultaneously checked for a match. If an entry matches, the corresponding data portion of the UTLB provides the real page number (RPN), access control bits (ZSEL, EX, WR), and storage attributes (W, I, M, G, E, U0)..

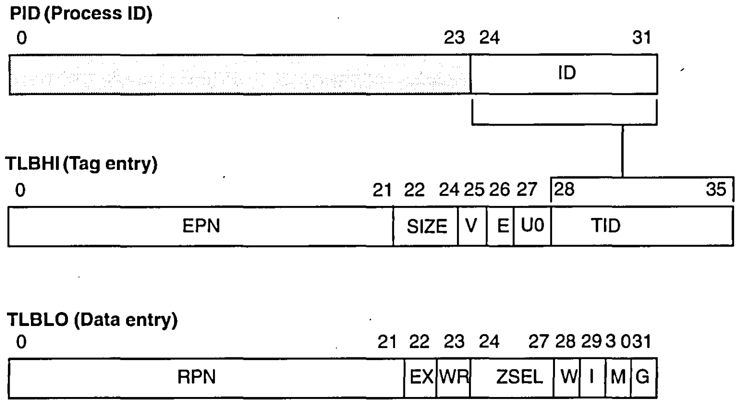


Figure 6-2. TLB Entries

The virtual address space is extended by adding an 8-bit translation ID (TID) loaded from the Process ID (PID) register during a TLB access. The PID identifies one of 255 unique software entities, usually used as a process or thread ID. TLBHI[TID] is compared to the PID during a TLB look-up.

Tag and data entries are written by copying data from GPRs and the PID, using the **tlbwe** instruction. Tag and data entries are read by copying data to GPRs and the PID, using the **tlbre** instruction. Software can search for specific entries using the **tlbsx** instruction.

6.3.2 TLB Fields

Each TLB entry describes a page that is enabled for translation and access controls. Fields in the TLB entry fall into four categories:

- Information required to identify the page to the hardware translation mechanism
- Control information specifying the translation
- Access control information
- Storage attribute control information

6.3.2.1 Page Identification Fields

When an EA is presented to the MMU for processing, the MMU applies several selection criteria to each TLB entry to select the appropriate entry. Although it is possible to place multiple entries into the TLB to match a specific EA and PID, this is considered a programming error, and the result of a TLB lookup for such an EA is undefined. The following fields in the TLB entry identify the page. Except as noted, all comparisons must succeed to validate an entry for subsequent use.

EPN (effective page number, 22 bits)

Compared to some number of the $EA_{0:21}$ bits presented to the MMU. The number of bits corresponds to the page size.

The exact comparison depends on the page size, as shown in Table 6-1.

Table 6-1. TLB Fields Related to Page Size

Page Size	SIZE Field	*n* Bits Compared	EPN to EA Comparison	RPN Bits Set to 0
1KB	000	22	$EPN_{0:21} \leftrightarrow EA_{0:21}$	—
4KB	001	20	$EPN_{0:19} \leftrightarrow EA_{0:19}$	$RPN_{20:21}$
16KB	010	18	$EPN_{0:17} \leftrightarrow EA_{0:17}$	$RPN_{18:21}$
64KB	011	16	$EPN_{0:15} \leftrightarrow EA_{0:15}$	$RPN_{16:21}$
256KB	100	14	$EPN_{0:13} \leftrightarrow EA_{0:13}$	$RPN_{14:21}$
1MB	101	12	$EPN_{0:11} \leftrightarrow EA_{0:11}$	$RPN_{12:21}$
4MB	110	10	$EPN_{0:9} \leftrightarrow EA_{0:9}$	$RPN_{10:21}$
16MB	111	8	$EPN_{0:7} \leftrightarrow EA_{0:7}$	$RPN_{8:21}$

SIZE (page size, 3 bits)

Selects one of the eight page sizes, 1KB–16MB, listed in Table 6-1.

V (valid,1 bit)

Indicates whether a TLB entry is valid and can be used for translation.

A valid TLB entry implies read access, unless overridden by zone protection. TLB_entry[V] can be written using a **tlbwe** instruction. The **tlbia** instruction invalidates all TLB entries.

TID (translation ID, 8 bits)

Loaded from the PID register during a **tlbwe** operation. The TID value is compared with the PID value during a TLB access. The TID provides a convenient way to associate a translation with one of 255 unique software entities, typically a process or thread ID maintained by operating system software. Setting TLBHI[TID] = 0x00 disables TID-PID comparison and identifies a TLB entry as valid for all processes; the value of the PID register is then irrelevant.

6.3.2.2 Translation Field

When a TLB entry is identified as matching an EA (and possibly the PID), TLBLO[RPN] defines how the EA is translated.

RPN (real page number, 22 bits)

Replaces some, or all, of $EA_{0:21}$, depending on page size. For example, a 16KB page uses $EA_{0:17}$ for comparison. The translation mechanism replaces $EA_{0:17}$ with $TLBLO[RPN]_{0:17}$ to form the physical address, and $EA_{18:31}$ becomes the real page offset, as illustrated in Figure 6-1.

Programming Note: Software must set all unused bits of RPN (as determined by page size) to 0. See Table 6-1.

6.3.2.3 Access Control Fields

Several access controls are available in the UTLB entries.

ZSEL (zone select, 4 bits)

Selects one of 16 zone fields (Z0—Z15) from the Zone Protection Register (ZPR). The ZPR field bits can modify the access protection specified by the TLB_entry[V, EX, WR] bits of a TLB entry. Zone protection is described in detail in "Zone Protection" on page 6-13.

EX (execute enable, 1 bit)

When set (TLBLO[EX] = 1), enables instruction execution at addresses within a page. ZPR settings can override TLBLO[EX]; see "Zone Protection" on page 6-13, for more information.

WR (write-enable 1 bit)

When set (TLBLO[WR] = 1), enables store operations to addresses in a page. ZPR settings can override TLBLO[WR]; see "Zone Protection" on page 6-13.

6.3.2.4 Storage Attribute Fields

TLB entries contain bits that control and provide information about the storage control attributes. Four of the attributes (W, I, M, and G) are defined in the PowerPC Architecture. The E storage attribute is defined in the IBM PowerPC Embedded Environment. The U0 attribute is implementation-specific. For the PPC405GP, U0 controls the CodePack instruction decompression function.

W (write-through,1 bit)

When set (TLBLO[W] = 1), stores are specified as write-through. If data in the referenced page is in the data cache, a store updates the cached copy of the data and the external memory location. Contrast this with a write-back strategy, which updates memory only when a cache line is flushed.

In real mode, the Data Cache Write-through Register (DCWR) controls the write strategy.

Note that the PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited. It is considered a programming error to use these memory models; the results are undefined.

I (caching inhibited,1 bit)

When set (TLBLO[I] = 1), a memory access is completed by using the location in main memory, bypassing the cache arrays. During the access, the accessed location is not put into the cache arrays.

In real mode, the Instruction Cache Cachability Register (ICCR) and Data Cache Cachability Register (DCCR) control cachability. In these registers, the setting of the bit is reversed; 1 indicates that a storage control region is cachable, rather than caching inhibited.

Note that the PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited. It is considered a programming error to use these memory models; the results are undefined.

It is considered a programming error if the target location of a load/store, **dcbz**, or fetch access to caching inhibited storage is in the cache; the results are undefined. It is *not* considered a programming error for the target locations of other cache control instructions to be in the cache when caching is inhibited.

M (memory coherent,1 bit)

For implementations that support multiprocessing, the M storage attribute improves the performance of memory coherency management. Because the PPC405GP does not provide multi-processor support or hardware support for data coherency, the M bit is implemented, but has no effect.

G (guarded,1 bit)

When set (TLBLO[G] = 1), indicates that the hardware cannot speculatively access the location for pre-fetching or out-of-order load access. The G storage attribute is typically used to protect memory-mapped I/O from inadvertent access. Attempted execution of an instruction from a guarded data storage address while instruction address translation is enabled results in an instruction storage interrupt because data storage and memory mapped I/O (MMIO) addresses are not used to contain instructions.

An instruction fetch from a guarded region does not occur until the execution pipeline is empty, thus guaranteeing that the access is necessary and therefore not speculative. For this reason, performance is degraded when executing out of guarded regions, and software should avoid unnecessarily marking regions of instruction storage as guarded.

In real mode, the Storage Guarded Register (SGR) controls guarding.

U0 (user-defined attribute, 1 bit)

When set (TLBLO[U0] = 1), indicates the user-defined attribute applies to the data in the associated page. This storage attribute controls CodePack decompression for a page.

In real mode, the Storage User-defined 0 Register (SU0R) controls the setting of the U0 storage attribute.

E (endian, 1 bit)

When set (TLBLO[E] = 1), indicates that data in the associated page is stored in true little endian format.

In real mode, the Storage Little-Endian Register (SLER) controls the setting of the E storage attribute.

6.3.3 Shadow Instruction TLB

To enhance performance, four instruction-side TLB entries are kept in a four-entry fully-associative shadow array. This array, called the instruction TLB (ITLB), helps to avoid TLB contention between instruction accesses to the TLB and load/store operations. Replacement and invalidation of the ITLB entries is managed by hardware (see "Shadow TLB Consistency" on page 6-8 for details).

The ITLB can be considered a level-1 instruction-side TLB; the UTLB serves as the level-2 instruction-side TLB. The ITLB is used only by instruction fetches for storing instruction address translations. Each ITLB entry contains the translation information for a page. The processor uses the ITLB for address translation of instruction accesses when MSR[IR] = 1.

6.3.3.1 ITLB Accesses

The instruction unit accesses the ITLB independently of the rest of the MMU. ITLB accesses are transparent to the executing program, except that ITLB hits contribute to higher overall instruction throughput by allowing data address translations to occur in parallel. Therefore, when instruction accesses hit in the ITLB, the address translation mechanisms in the UTLB are available for use by data accesses simultaneously.

The ITLB requests a new entry from the UTLB when an ITLB miss occurs. A four-cycle latency occurs at each ITLB miss that is also a UTLB hit; the latency is longer if it is also a UTLB miss, or if there is contention for the UTLB from the data side. A round-robin replacement algorithm replaces existing entries with new entries.

6.3.4 Shadow Data TLB

To enhance performance, eight data-side TLB entries are kept in a eight-entry fully-associative shadow array. This array, called the data TLB (DTLB), helps to avoid TLB contention between instruction accesses to the TLB and load/store operations. Replacement and invalidation of the DTLB entries is managed by hardware. See "Shadow TLB Consistency" on page 6-8 for details.

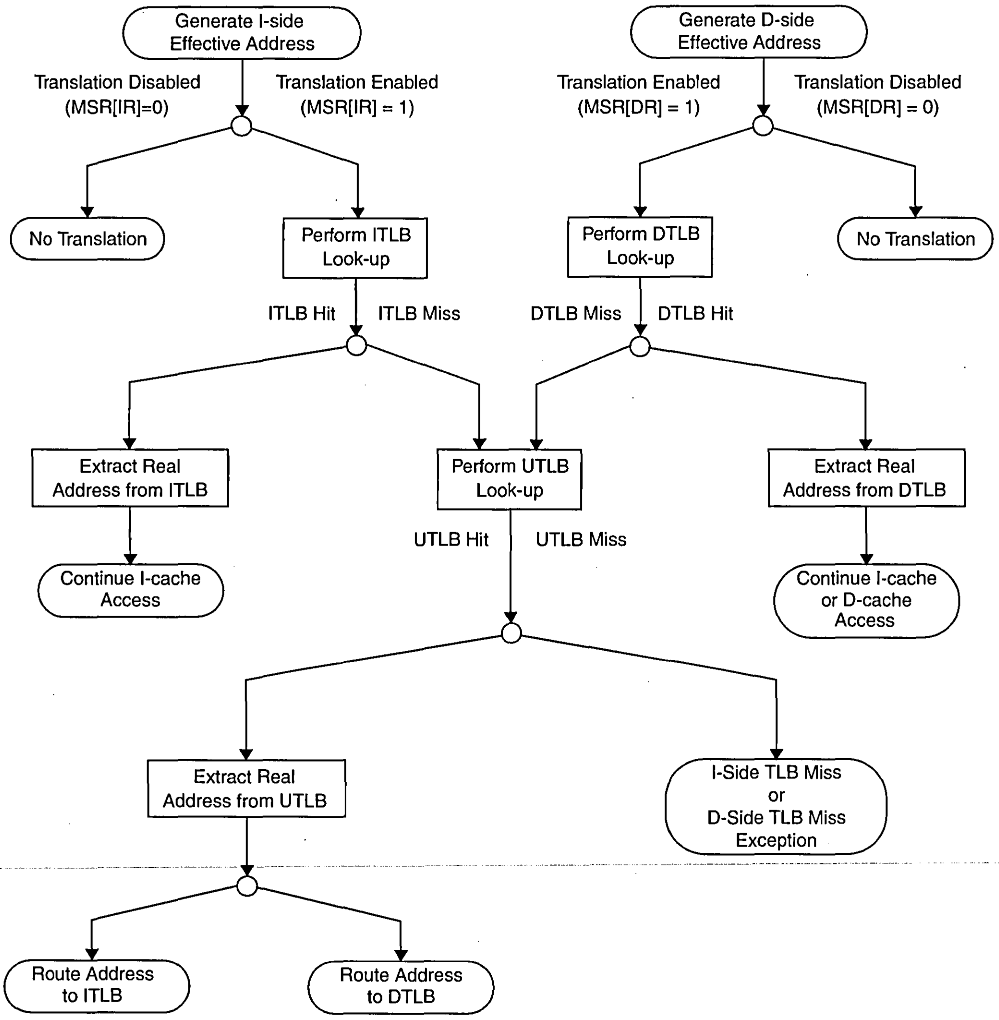
The DTLB can be considered a level-1 data-side TLB; the UTLB serves as the level-2 data-side TLB. The DTLB is used only by instructions in execute for storing data address translations. Each DTLB entry contains the translation information for a page. The processor uses the DTLB for address translation of data accesses when MSR[DR] = 1.

6.3.4.1 DTLB Accesses

The execute unit accesses the DTLB independently of the rest of the MMU. DTLB accesses are transparent to the executing program, except that DTLB hits contribute to higher overall instruction throughput by allowing instruction address translations to occur in parallel. Therefore, when data accesses hit in the DTLB, the address translation mechanisms in the UTLB are available for use by instruction accesses simultaneously.

The DTLB requests a new entry from the UTLB when a DTLB miss occurs. A three-cycle latency occurs at each DTLB miss that is also a UTLB hit; the latency is longer if it is also a UTLB miss. If there is contention for the UTLB from the instruction side, the data side has priority. A round-robin replacement algorithm replaces existing entries with new entries.

Figure 6-3 illustrates the relationship of the shadow TLBs and UTLB in address translation:



Generate I-side Effective Address
Translation Disabled (MSR[IR]=0)
Translation Enabled (MSR[IR] = 1)
Generate D-side Effective Address
Translation Enabled (MSR[DR] = 1)
Translation Disabled (MSR[DR] = 0)
No Translation
Perform ITLB Look-up
Perform DTLB Look-up
No Translation
ITLB Hit
ITLB Miss
DTLB Miss
DTLB Hit
Extract Real Address from ITLB
Perform UTLB Look-up
Extract Real Address from DTLB
UTLB Hit
UTLB Miss
Continue I-cache Access
Continue I-cache or D-cache Access
Extract Real Address from UTLB
I-Side TLB Miss or D-Side TLB Miss Exception
Route Address to ITLB
Route Address to DTLB

Figure 6-3. ITLB/DTLB/UTLB Address Resolution

6.3.4.2 Shadow TLB Consistency

The processor invalidates the entire ITLB contents when the following context-synchronizing events occur, to help to maintain ITLB integrity.

- **isync** instruction
- Processor context switch (all interrupts, **rfi**, **rfci**)

If software updates a translation/protection mechanism (UTLB, PID, ZPR, or MSR) and must synchronize these updates with the ITLB, the *software* must perform the necessary context synchronization.

A typical example is the manipulation of the TLB by an operating system within an interrupt handler for a TLB miss. Upon entry to the interrupt handler, the ITLB is invalidated and translation is disabled. If the operating system simply made the TLB updates and returned from the handler (using **rfi**), no additional explicit software action would be required to synchronize the ITLB.

If, instead, the operating system re-enables translation within the handler, and then performs TLB updates within the handler, those updates would not be effective in the ITLB until **rfi** is executed to return from the handler. For those TLB updates to be reflected in the ITLB *within* the handler, an **isync** must be issued after TLB updates finish. Failure to properly synchronize the ITLB can cause unexpected behavior.

Programming Note: As a rule of thumb, follow software manipulation of an translation mechanism (if performed while translation is active) with a context-synchronizing operation (usually **isync**).

6.4 TLB-Related Interrupts

The processor relies on interrupt handling software to implement paged virtual memory, and to enforce protection of specified memory pages.

When an interrupt occurs, the processor clears MSR[IR, DR]. Therefore, at the start of all interrupt handlers, the processor operates in real mode for instruction accesses and data accesses. Note that when address translation is disabled for an instruction fetch or load/store, the EA is equal to the real address and is passed directly to the memory subsystem (including cache units). Such untranslated addresses bypass all memory protection checks that would otherwise be performed by the MMU.

When translation is enabled, MMU accesses can result in the following interrupts:

- Data storage interrupt
- Instruction storage interrupt
- Data TLB miss interrupt
- Instruction TLB miss interrupt

6.4.1 Data Storage Interrupt

A data storage interrupt is generated when data address translation is active, and the desired access to the EA is not permitted for one of the following reasons:

- In the problem state
  - **icbi**, load/store, **dcbz**, or **dcbf** with an EA whose zone field is set to no access (ZPR[Z*n*] = 00). In this case, **dcbt** and **dcbtst** no-op, rather than cause an interrupt. Privileged instructions cannot cause data storage interrupts.
  - Stores, or **dcbz**, to an EA having TLB_entry[WR] = 0 (write access disabled) and ZPR[Z*n*] ≠ 11. (The privileged instructions **dcbi** and **dccci** are treated as “stores”, but cause program interrupts, rather than data storage interrupts.)

- In supervisor state
  - Data store, **dcbi**, **dcbz**, or **dccci** to an EA having TLB_entry[WR] = 0 and ZPR[Z*n*] other than 11 or 10.

dcba does not cause data storage exceptions. If conditions occur that would otherwise cause such an exception, **dcba** is treated as a no-op.

"Zone Protection" on page 6-13 describes zone protection in detail. See "Data Storage Interrupt" on page 10-36 for a detailed discussion of the data storage interrupt.

6.4.2 Instruction Storage Interrupt

An instruction storage interrupt is generated when instruction address translation is active and the processor attempts to execute an instruction at an EA for which fetch access is not permitted, for any of the following reasons:

- In the problem state
  - Instruction fetch from an EA with ZPR[Z*n*] = 00.
  - Instruction fetch from an EA having TLB_entry[EX] = 0 and ZPR[Z*n*] ≠ 11.
  - Instruction fetch from an EA having TLB_entry[G] = 1.
- In the supervisor state
  - Instruction fetch from an EA having TLB_entry[EX] = 0 and ZPR[Z*n*] other than 11 or 10.
  - Instruction fetch from an EA having TLB_entry[G] = 1.

See "Zone Protection" on page 6-13 for a detailed discussion of zone protection. See "Instruction Storage Interrupt" on page 10-38 for a detailed discussion of the instruction storage interrupt.

6.4.3 Data TLB Miss Interrupt

A data TLB miss interrupt is generated if data address translation is enabled and a valid TLB entry matching the EA and PID is not present. The interrupt applies to data access instructions and cache operations (excluding cache touch instructions).

See "Data TLB Miss Interrupt" on page 10-43 for a detailed discussion.

6.4.4 Instruction TLB Miss Interrupt

The instruction TLB miss interrupt is generated if instruction address translation is enabled and execution is attempted for an instruction for which a valid TLB entry matching the EA and PID for the instruction fetch is not present.

See "Instruction TLB Miss Interrupt" on page 10-44 for a detailed discussion.

6.5 TLB Management

The processor does not imply any format for the page tables or the page table entries because there is no hardware support for page table management. Software has complete flexibility in implementing a replacement strategy, because software does the replacing. For example, software can "lock" TLB entries that correspond to frequently used storage by electing to never replace them, so that those entries are never cast out of the TLB.

TLB management is performed by software with some hardware assist, consisting of:

- Storage of the missed EA in the Save/Restore Register 0 (SRR0) for an instruction-side miss, or in the Data Exception Address Register (DEAR) for a data-side miss.
- Instructions for reading, writing, searching, and invalidating the TLB, as described briefly in the following subsections. See Chapter 24, "Instruction Set," for detailed instruction descriptions.

6.5.1 TLB Search Instructions (tlbsx/tlbsx.)

tlbsx locates entries in the TLB, to find the TLB entry associated with an interrupt, or to locate candidate entries to cast out. **tlbsx** searches the UTLB array for a matching entry. The EA is the value to be matched; EA = (RA|0)+(RB).

If the TLB entry is found, its index is placed in $RT_{26:31}$. RT can then serve as the source register for a **tlbre** or **tlbwe** instruction to read or write the entry, respectively. If no match is found, the contents of RT are undefined.

tlbsx. sets the Condition Register (CR) bit $CR0_{EQ}$. The value of $CR0_{EQ}$ depends on whether an entry is found: $CR0_{EQ} = 1$ if an entry is found; $CR0_{EQ} = 0$ if no entry is found.

6.5.2 TLB Read/Write Instructions (tlbre/tlbwe)

TLB entries can be accessed for reading and writing by **tlbre** and **tlbwe**, respectively. Separate extended mnemonics are available for the TLBHI (tag) and TLBLO (data) portions of a TLB entry.

6.5.3 TLB Invalidate Instruction (tlbia)

tlbia sets TLB_entry[V] = 0 to invalidate all TLB entries. All other TLB entry fields remain unchanged.

Using **tlbwe** to set TLB_entry[V] = 0 invalidates a specific TLB entry.

6.5.4 TLB Sync Instruction (tlbsync)

tlbsync guarantees that all TLB operations have completed for all processors in a multi-processor system. PPC405GP provides no multiprocessor support, so this instruction performs no function. The instruction is included to facilitate code portability.

6.6 Recording Page References and Changes

When system software manages virtual memory, the software views physical memory as a collection of pages. Each page is associated with at least one TLB entry. To manage memory effectively, system software often must know whether a particular page has been referenced or modified. Note that this involves more than knowing whether a particular TLB entry was used to reference or alter memory, because multiple TLB entries can translate to the same page.

When system software manages a demand-paged environment, and the software needs to replace the contents of a page with other data, previously referenced pages (accessed for any purpose) are more likely to be maintained than pages that were never referenced. If the contents of a page must be replaced, and data contained in that page was modified, system software generally must write the contents of the modified page to the backing store before replacing its contents. System software must maintain records to control the environment.

Similarly, when system software manages TLB entries, the software often must know whether a particular TLB entry was referenced. When the system software must select a TLB entry to cast out, previously referenced entries are more likely to be maintained than entries which were never referenced. System software must also maintain records for this purpose.

The PPC405GP does not provide hardware reference or change bits, but TLB miss interrupts and data storage interrupts enable system software to maintain reference information for TLB entries and their associated pages, respectively.

A possible algorithm follows. First, the TLB entries are built, with each TLB_entry[V, WR] = 0. System software retains the index and EPN of each entry.

The first attempt by application code to access a page causes a TLB miss interrupt, because its TLB entry is marked invalid. The TLB miss handler records the reference to the TLB entry (and to the associated page) in a data structure, then sets TLB_entry[V] = 1. (Note that TLB_entry[V] can be considered a reference bit for the TLB entry.) Subsequent read accesses to the page associated with the TLB entry proceed normally.

In the example just given for recording TLB entry references, the first write access to the page using the TLB entry, after the entry is made valid, causes a data storage interrupt because write access was turned off. The TLB miss handler records the write to the page in a data structure, for use as a "changed" flag, then sets TLB_entry[WR] = 1 to enable write access. (Note that TLB_entry[WR] can be considered a change bit for the page.) Subsequent write accesses to the page proceed normally.

6.7 Access Protection

The PPC405GP provides virtual-mode access protection. The TLB entry enables system software to control general access for programs in the problem state, and control write and execute permissions for all pages. The TLB entry can specify zone protection that can override the other access control mechanisms supported in the TLB entries.

TLB entry and zone protection methods also support access controls for cache operation and string loads/stores.

6.7.1 Access Protection Mechanisms in the TLB

For MMU access protection to be in effect, one or both of MSR[IR] or MSR[DR] must be set to one to enable address translation. MSR[IR] enables protection on instruction fetches, which are inherently read-only. MSR[DR] enables protection on data accesses (loads/stores).

6.7.1.1 General Access Protection

The translation ID (TLB_entry[TID]) provides the first level of MMU access protection. This 8-bit field, if non-zero, is compared to the contents of TLB_entry[PID]. These fields must match in a valid TLB entry if any access is to be allowed. In typical use, it is assumed that a program in the supervisor state, such as a real-time operating system, sets the PID before starting a problem state program that is subject to access control.

If TLB_entry[TID] = 0x00, the associated memory page is accessible to all programs, regardless of their PID. This enables multiple processes to share common code and data. The common area is still subject to all other access protection mechanisms. Figure 6-4 illustrates the PID.

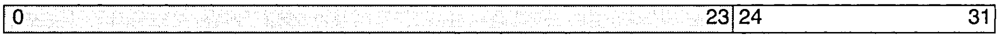


Figure 6-4. Process ID (PID)

0:23	▨	Reserved
24:31		Process ID

6.7.1.2 Execute Permissions

If instruction address translation is enabled, instruction fetches are subject to MMU translation and have MMU access protection. Fetches are inherently read-only, so write protection is not needed. Instead, using TLB_entry[EX], a memory page is marked as executable (contains instructions) or not executable (contains only data or memory-mapped control hardware).

If an instruction is pre-fetched from a memory page for which TLB_entry[EX] = 0, the instruction is tagged as an error. If the processor subsequently attempts to execute this instruction, an instruction storage interrupt results. This interrupt is precise with respect to the attempted execution. If the fetcher discards the instruction without attempting to execute it, no interrupt will result.

Zone protection can alter execution protection.

6.7.1.3 Write Permissions

If MSR[DR] = 1, data loads and stores are subject to MMU translation and are afforded MMU access protection. The existence of a TLB entry describing a memory page implies read access; write access is controlled by TLB_entry[WR].

If a store (including those caused by **dcbz**, **dcbi**, or **dccci**) is made to an EA having TLB_entry[WR] = 0, a data storage interrupt results. This interrupt is precise.

Zone protection can alter write protection (see "Zone Protection" on page 6-13). In addition, only zone protection can prevent read access of a page defined by a TLB entry.

6.7.1.4 Zone Protection

Each TLB entry contains a 4-bit zone select (ZSEL) field. A zone is an arbitrary identifier for grouping TLB entries (memory pages) for purposes of protection. As many as 16 different zones may be defined. Any zone can have any number of member pages.

Each zone is associated with a 2-bit field (Z0–Z15) in the ZPR. The values of the field define how protection is applied to all pages that are member of that zone. Changing the value of the ZPR field can alter the protection attributes of all pages in the zone. Without ZPR, the change would require finding, reading, altering, and rewriting the TLB entry for each page in a zone, individually. The ZPR provides a much faster means of altering the protection for groups of memory pages.

The ZSEL values 0–15 select ZPR fields Z0–Z15, respectively.

The fields are defined within the ZPR as follows:

While it is common for TLB_entry[EX, WR] to be identical for all member pages in a group, this is not required. The ZPR field alters the protection defined by TLB_entry[EX] and TLB_entry[WR], on a page-by-page basis, as shown in the ZPR illustration. An application program (presumed to be running in the problem state) can have execute and write permissions as defined by TLB_entry[EX] and TLB_entry[WR] for the individual pages, or no access (denies loads, as well as stores and execution), or complete access.

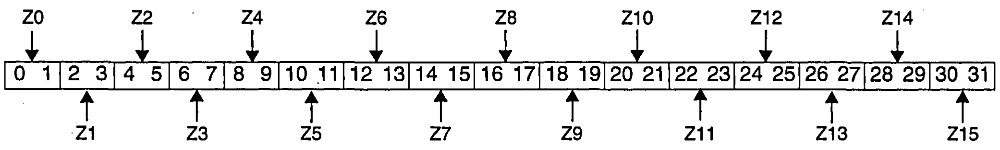


Figure 6-5. Zone Protection Register (ZPR)

0:1	Z0	TLB page access control for all pages in this zone.	
		In the problem state (MSR[PR] = 1): 00 No access 01 Access controlled by applicable TLB_entry[EX, WR] 10 Access controlled by applicable TLB_entry[EX, WR] 11 Accessed as if execute and write permissions (TLB_entry[EX, WR]) are granted.	In the supervisor state (MSR[PR] = 0): 00 Access controlled by applicable TLB_entry[EX, WR] 01 Access controlled by applicable TLB_entry[EX, WR] 10 Access controlled by applicable TLB_entry[EX, WR] 11 Accessed as if execute and write permissions (TLB_entry[EX, WR]) are granted.
2:3	Z1	See the description of Z0.	
4:5	Z2	See the description of Z0.	
6:7	Z3	See the description of Z0.	
8:9	Z4	See the description of Z0.	
10:11	Z5	See the description of Z0.	
12:13	Z6	See the description of Z0.	
14:15	Z7	See the description of Z0.	
16:17	Z8	See the description of Z0.	
18:19	Z9	See the description of Z0.	
20:21	Z10	See the description of Z0.	
22:23	Z11	See the description of Z0.	
24:25	Z12	See the description of Z0.	
26:27	Z13	See the description of Z0.	
28:29	Z14	See the description of Z0.	

30:31	Z15	See the description of Z0.

Setting ZPR[Z*n*] = 00 for a ZPR field is the only way to deny read access to a page defined by an otherwise valid TLB entry. TLB_entry[EX] and TLB_entry[WR] do not support read protection. Note that the **icbi** instruction is considered a load with respect to access protection; executed in user mode, it causes a data storage interrupt if MSR[DR] = 1 and ZPR[Z*n*] = 00 is associated with the EA.

For a given ZPR field value, a program in supervisor state always has equal or greater access than a program in the problem state. System software can never be denied read (load) access for a valid TLB entry.

6.7.2 Access Protection for Cache Control Instructions

Architecturally the instructions **dcba**, **dcbi**, and **dcbz** are treated as "stores" because they can change data, or cause loss of data by invalidating a dirty line (a modified cache block).

Table 6-2 summarizes the conditions under which the cache control instructions can cause data storage interrupts.

Table 6-2. Protection Applied to Cache Control Instructions

Instruction	Possible Data Storage interrupt	
	When ZPR[Zn] = 00	When TLB_entry[WR] = 0
dcba	No (instruction no-ops)	No (instruction no-ops)
dcbf	Yes	No
dcbi	No	Yes
dcbst	Yes	No
dcbt	No (instruction no-ops)	No
dcbtst	No (instruction no-ops)	No
dcbz	Yes	Yes
dccci	No	Yes
dcread	No	No
icbi	Yes	No
icbt	No	No
iccci	No	No
icread	No	No

If data address translation is enabled, and write permission is denied (TLB_entry[WR] = 0), **dcbi** and **dcbz** can cause data storage interrupts. **dcbz** can cause a data storage interrupt when executed in the problem state and all access is denied (ZPR[Z*n*] = 00); **dcbi** cannot cause a data storage interrupt because it is a privileged instruction.

The **dcba** instruction enables "speculative" line establishment in the cache arrays; the established lines do not cause a line fill. Because the effects of **dcba** are speculative, interrupts that would otherwise result when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0 do not occur. In such cases, **dcba** is treated as a no-op.

The **dccci** instruction can also be considered a "store" because it can change data by invalidating a dirty line; however, **dccci** is not address-specific (it affects an entire congruence class regardless of the operand address of the instruction). To restrict possible damage from an instruction which can change data and yet avoids the protection mechanism, the **dccci** instruction is privileged.

If data address translation is enabled, **dccci** can cause data storage interrupts when TLB_entry[WR] = 0; the operand is treated as if it were address-specific. **dccci** cannot cause a data storage interrupt when ZPR[Z*n*] = 00, because it is a privileged instruction.

Because **dccci** can cause data storage and TLB -miss interrupts, use of **dccci** is not recommended when MSR[DR] = 1; if **dccci** is used. Note that the specific operand address can cause an interrupt.

Architecturally, **dcbt** and **dcbtst** are treated as "loads" because they do not change data; they cannot cause data storage interrupts when TLB_entry[WR] = 0.

The cache block touch instructions **dcbt** and **dcbtst** are considered "speculative" loads; therefore, if a data storage interrupt would otherwise result from the execution of **dcbt** or **dcbtst** when ZPR[Z*n*] = 00, the instruction is treated as a no-op and the interrupt does not occur. Similarly, TLB miss interrupts do not occur for these instructions.

Architecturally, **dcbf** and **dcbst** are treated as "loads". Flushing or storing a line from the cache is not architecturally considered a "store" because a store was performed to update the cache, and **dcbf** or **dcbst** only update main memory. Therefore, neither **dcbf** nor **dcbst** can cause data storage interrupts when TLB_entry[WR] = 0. Because neither instruction is privileged, they can cause data storage interrupts when ZPR[Z*n*] = 00 and data address translation is enabled.

dcread is a "load" from a non-specific address, and is privileged. Therefore, it cannot cause data storage interrupts when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0.

icbi and **icbt** are considered "loads" and cannot cause data storage interrupts when TLB_entry[WR] = 0. **icbi** can cause data storage interrupts when ZPR[Z*n*] = 00. Because **icbt** is privileged, it cannot cause data storage interrupts when ZPR[Z*n*] = 00.

The **iccci** instruction cannot change data; an instruction cache line cannot be dirty. The **iccci** instruction is privileged and is considered a load. It does not cause data storage interrupts when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0.

Because **iccci** can cause a TLB miss interrupt, using **iccci** is not recommended when data address translation is enabled; if it is used, note that the specific operand address can cause an interrupt.

icread is considered a "load" from a non-specific address, and is privileged. Therefore, it cannot cause data storage interrupts when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0.

6.7.3 Access Protection for String Instructions

The **stswx** instruction with string length equal to 0(XER[TBC] = 0) is a no-op.

When data address translation is enabled and the Transfer Byte Count (TBC) field of the Fixed Point Exception Register (XER) is 0, neither **lswx** nor **stswx** can cause TLB miss interrupts, or data storage interrupts when ZPR[Z*n*] = 0 or TLB_entry[WR] = 0.

6.8 Real-mode Storage Attribute Control

The PowerPC Architecture and the PowerPC Embedded Environment define several SPRs to control the following storage attributes in real mode: W, I, G,U0, and E. Note that the U0 and E attributes are not defined in the PowerPC Architecture. The E attribute is defined in the IBM PowerPC Embedded Environment, and the U0 attribute is implementation-specific. No storage attribute control register is implemented for the M storage attribute because the PPC405GP does not provide multi-processor support or hardware support for data coherency.

These SPRs, called storage attribute control registers, control the various storage attributes when address translation is disabled. When address translation is enabled, these registers are ignored, and the storage attributes supplied by the TLB entry are used (see “TLB Fields” on page 6-3).

The storage attribute control registers divide the 4GB real address space into thirty-two 128MB regions. In a storage attribute control register, bit 0 controls the lowest addressed 128MB region, bit 1 the next higher-addressed 128MB region, and so on. $EA_{0:4}$ specify a storage control region.

For detailed information on the function of the storage attributes, see “Storage Attribute Fields” on page 6-5.

6.8.1 Storage Attribute Control Registers

Figure 6-6 shows a generic storage attribute control register. The storage attribute control registers have the same bit numbering and address ranges.

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

Figure 6-6. Generic Storage Attribute Control Register

Bit	Address Range	Bit	Address Range
0	0x0000 0000–0x07FF FFFF	16	0x8000 0000–0x87FF FFFF
1	0x0800 0000–0x0FFF FFFF	17	0x8800 0000–0x8FFF FFFF
2	0x1000 0000–0x17FF FFFF	18	0x9000 0000–0x97FF FFFF
3	0x1800 0000–0x1FFF FFFF	19	0x9800 0000–0x9FFF FFFF
4	0x2000 0000–0x27FF FFFF	20	0xA000 0000–0xA7FF FFFF
5	0x2800 0000–0x2FFF FFFF	21	0xA800 0000–0xAFFF FFFF
6	0x3000 0000–0x37FF FFFF	22	0xB000 0000–0xB7FF FFFF
7	0x3800 0000–0x3FFF FFFF	23	0xB800 0000–0xBFFF FFFF
8	0x4000 0000–0x47FF FFFF	24	0xC000 0000–0xC7FF FFFF
9	0x4800 0000–0x4FFF FFFF	25	0xC800 0000–0xCFFF FFFF
10	0x5000 0000–0x57FF FFFF	26	0xD000 0000–0xD7FF FFFF
11	0x5800 0000–0x5FFF FFFF	27	0xD800 0000–0xDFFF FFFF
12	0x6000 0000–0x67FF FFFF	28	0xE000 0000–0xE7FF FFFF
13	0x6800 0000–0x6FFF FFFF	29	0xE800 0000–0xEFFF FFFF
14	0x7000 0000–0x77FF FFFF	30	0xF000 0000–0xF7FF FFFF
15	0x7800 0000–0x7FFF FFFF	31	0xF800 0000–0xFFFF FFFF

6.8.1.1 Data Cache Write-through Register (DCWR)

The DCWR controls write-through policy (the W storage attribute) for the data cache unit (DCU). Write-through is not applicable to the instruction cache unit (ICU).

After any reset, all DCWR bits are set to 0, which establishes a write-back write strategy for all regions.

The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

6.8.1.2 Data Cache Cachability Register (DCCR)

The DCCR controls the I storage attribute for data accesses and cache management instructions. Note that the polarity of the bits in this register is opposite to that of the I attribute in the TLB; DCCR[S*n*] = 1 enables caching, while TLB_entry[I] = 1 inhibits caching.

After any reset, all DCCR bits are set to 0. No memory regions are cachable. Before memory regions can be designated as cachable in the DCCR, it is necessary to execute the **dccci** instruction once for each congruence class in the DCU cache array. This procedure invalidates all congruence classes. The DCCR can then be reconfigured, and the DCU can begin normal operation.

The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

6.8.1.3 Instruction Cache Cachability Register (ICCR)

The ICCR controls the I storage attribute for instruction fetches. Note that the polarity of the bits in this register is opposite of that of the I attribute (ICCR[S*n*] = 1 enables caching, while TLB_entry[I] = 1 inhibits caching).

After any reset, all ICCR bits are set to 0. No memory regions are cachable. Before memory regions can be designated as cachable in the ICCR, it is necessary to execute the **iccci** instruction. This invalidates all congruence classes. The ICCR can then be reconfigured, and the ICU can begin normal operation.

6.8.1.4 Storage Guarded Register (SGR)

The SGR controls the G storage attribute for instruction and data accesses.

This attribute does not affect data accesses; the PPC405GP does not perform speculative loads or stores.

After any reset, all SGR bits are set to 1, marking all storage as guarded. For best performance, system software should clear the guarded attribute of appropriate regions as soon as possible. If MSR[IR] = 1, the G attribute comes from the TLB entry. Attempting to execute from a guarded region in translate mode causes an instruction storage interrupt. See “Instruction Storage Interrupt” on page 10-38 for more information.

6.8.1.5 Storage User-defined 0 Register (SU0R)

The Storage User-defined 0 Register (SU0R) controls the user-defined (U0) storage attribute for instruction and data accesses.

After any reset, all SU0R bits are set to 0.

6.8.1.6 Storage Little-Endian Register (SLER)

The SLER controls the E storage attribute for instruction and data accesses.

This attribute determines the byte ordering of storage. “Byte Ordering” on page 3-28 provides a detailed description of byte ordering in the IBM PowerPC Embedded Environment.

After any reset, all SLER bits are set to 0 (big endian).

Part III. PPC405GP System Operations

Chapter 7. Clocking

Clocking in the PPC405GP is highly configurable and supports a wide range of clock ratios on the internal and external buses.

Figure 7-1 illustrates the clocking options for the PPC405GP. A phase-locked loop (PLL) is the source for the CPU clock. Generated clock frequencies are integer ratios of this reference clock. Optional external clock inputs are shown where appropriate. The clocking inputs, which are highlighted in the figure (IICSCL is I/O), are described in more detail in Chapter 26, “Signal Summary.”

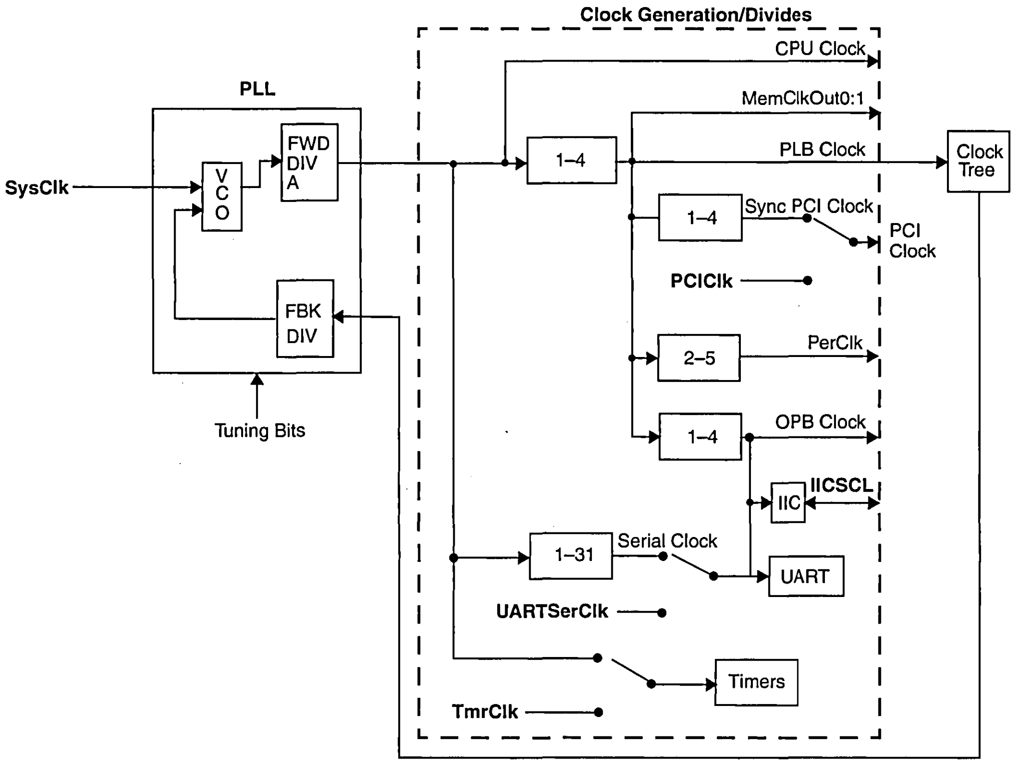


Figure 7-1. PPC405GP Clocking

7.1 PLL Overview

The PLL $V_{DD}A$ input to the PPC405GP requires 2.5 V ± 0.2 V, which should be supplied from a 2.5 V filtered board voltage.

The PLL operating range is controlled by forward divide and feedback divide ratios, tuning bits, and SysClk. The voltage controlled oscillator (VCO) in the PLL must operate at a range of 400–800 MHz. The value of the forward divide and feedback divide ratios affects the CPU frequency and the VCO operating frequency.

Several strapping values are required to correctly configure the PPC405GP. When setting the values, pay close attention to the following information to avoid accidentally configuring the controller in an unusable state.

For any acceptable SysClk input, the VCO frequency is set by the feedback and forward dividers, and the CPU to PLB divider, as shown in the following equation:

VCO = Reference Clock $\times$ *M*

where *M* = Feedback Divide $\times$ Forward Divide $\times$ CPU to PLB Divide

For example, with an input reference frequency of 33.33 MHz, feedback divider of 3, CPU to PLB divider of 2, and forward divider of 3, the VCO frequency at which the PLL stabilizes is 600 MHz.

Strap selection for PLL tuning depends upon the value of *M*, which can range from 6 to 32. The upper limit is based on a minimum input reference clock frequency of 25 MHz and a maximum VCO frequency of 800 MHz. The lower limit is based on a maximum input reference clock of 66 MHz and a minimum VCO frequency of 400MHz. *M* decreases as the reference clock frequency increases.

7.2 Input Reference Clock (SysClk)

The input reference clock, SysClk, must be between 25 MHz and 66 MHz for the PLL to achieve a stable lock. *Input clocks outside this range are not supported.* When synchronous PCI mode is selected, clock generation logic ensures that the internally generated PCI clock and SysClk are phase-aligned. This means that rising edges will coincide as long as the frequencies are the same. In most cases, the input reference clock is expected to be the same as the PCI clock provided to other PCI devices outside of the 405GP. Therefore, the phase alignment that automatically occurs ensures that the internally generated PCI clock matches the clock used by other PCI devices.

Clock generation logic also ensures that the internally generated peripheral clock (PerClk) is phase-aligned with SysClk, as long as both run at the same clock frequency. This is useful when an external device, such as an external master, cannot be synchronized with the PPC405GP PerClk.

7.3 External Clock Strapping Setup

The PPC405GP configures its clocking based on strapping resistors connected to module I/O pins. The state of the strapping pins is stored in the Chip Pin Strapping Register (CPC0_PSR) upon power-on or system reset. Software can use this read-only register to determine clock strapping values. Table 7-1 indicates which CPC0_PSR fields are assigned to clocking functions and how they are encoded. See Chapter 9, "Pin Strapping and Sharing" for a more information and a complete list of strapping values.

Table 7-1. Clock Strapping Values

Strap Option	Description
PLL Forward Divide Ratio CPC0_PSR[PFWD] 00 Bypass mode 01 Divide by 3 10 Divide by 4 11 Divide by 6	These bits indicate bypass mode or one of three valid PLL forward divide ratios. Bypass mode is useful when the PPC405GP is run with a low-frequency reference clock input that is too low for the PLL to lock. For example, if the PPC405GP is clocked at 1 MHz in an emulation system, the PLL cannot be used and should be placed in bypass mode. The other three strapping combinations select one of three valid PLL forward divide ratios: 3, 4, or 6. These ratios are the only acceptable ratios.
PLL Feedback Divide Ratio CPC0_PSR[PFBD] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four valid PLL feedback divide ratios: 1, 2, 3, or 4.
PLL Tuning CPC0_PSR[PT] 000 Choice 1; TUNE[5:0] = 010001 001 Choice 2; TUNE[5:0] = 010010 010 Choice 3; TUNE[5:0] = 010011 011 Choice 4; TUNE[5:0] = 010100 100 Choice 5; TUNE[5:0] = 010101 101 Choice 6; TUNE[5:0] = 010110 110 Choice 7; TUNE[5:0] = 010111 111 Choice 8; TUNE[5:0] = 100100	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL implemented in the PPC405GP. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs) with the PPC405GP, visit the technical documents area of the IBM PowerPC web site.
PLB Divide Ratio from CPU CPC0_PSR[PDC] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four PLB divide ratios from the CPU: 1, 2, 3, or 4.
OPB Divide Ratio from PLB CPC0_PSR[ODP] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four OPB divide ratios from the PLB: 1, 2, 3, or 4.

Table 7-1. Clock Strapping Values (continued)

Strap Option	Description
PCI Divide Ratio from PLB CPC0_PSR[PDP] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four PCI divide ratios from the PLB: 1, 2, 3 or 4.
Peripheral Bus Divide Ratio from PLB CPC0_PSR[EBDP] 00 Divide by 2 01 Divide by 3 10 Divide by 4 11 Divide by 5	These bits indicate one of four peripheral bus divide ratios from the PLB: 2, 3, 4, or 5.

Table 7-2 lists PLL tuning settings, which are controlled by the pin strappings reported by CPC0_PSR[PT]. The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL implemented in the PPC405GP. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs) with the PPC405GP, visit the technical documents area of the IBM PowerPC web site.

Table 7-2. PLL Tuning Settings

M Range	Recommended Choice	CPC0_PSR[PT] Value	Equivalent TUNE[5:0]
$6 \leq M \leq 7$	3	010	010011
$7 \leq M \leq 12$	5	100	010101
$12 \leq M \leq 32$	6	101	010110

7.4 Sample Clock Ratios

Table 7-3, Table 7-4, and Table 7-5 list all usable clocking ratio combinations that can be achieved using the strapping inputs for three different reference clock frequencies. VCO frequencies and *M* multipliers are calculated for each valid combination of forward divide (FWD), CPU to PLB divide (CPU/PLB), and feedback divide (FBK). The resulting clock frequencies for CPU and PLB are also shown. To simplify the table, PCI, OPB, external bus, and serial clock frequencies are not shown.

Ratios that result in unusable configurations are not listed. In some cases, the table indicates identical CPU and PLB frequencies. However, in such cases, the VCO frequency varies. When the table indicates a choice of VCO frequencies, select the highest frequency in the range of 400–800 MHz for best PLL performance.

Table 7-3. Possible Clocking Ratios for Reference Clock of 33.3MHz

Strapping Options			M	Calculated Values (MHz)		
FWD	CPU/PLB	FBK		VCO	CPU	PLB
By-pass[2]	1	x	—	—	refclk	refclk
	2	x	—	—	refclk	refclk/2
	3	x	—	—	refclk	refclk/3
	4	x	—	—	refclk	refclk/4
3	1	4	12	400	133	133
3	2	2	12	400	133	66
3	2	3	18	600	200	100
3	2	4	24	800	266	133
3	3	2	18	600	200	66
3	4	1	12	400	133	33
3	4	2	24	800	266	66
4	1	3	12	400	100	100
4	1	4	16	533	133	133
4	2	2	16	533	133	66
4	2	3	24	800	200	100
4	3	1	12	400	100	33
4	3	2	24	800	200	66
4	4	1	16	533	133	33
6	1	2	12	400	66	66
6	1	3	18	600	100	100
6	1	4	24	800	133	133
6	2	1	12	400	66	33
6	2	2	24	800	133	66
6	3	1	18	600	100	33
6	4	1	24	800	133	33

Note 1: x = don't care.

Note 2: By-pass mode is for hardware emulator use only. Contact your IBM representative for more information.

Note 3: Not all PPC405GP parts support the frequencies shown. Check *PPC405GP Datasheet* for supported CPU and PLB frequencies for a specific part number.

Table 7-4. Possible Clocking Ratios for Reference Clock of 25 MHz

Strapping Options				Calculated Values (MHz)		
FWD	CPU/PLB	FBK	M	VCO	CPU	PLB
By-pass[2]	1	x	—	—	refclk	refclk
	2	x	—	—	refclk	refclk/2
	3	x	—	—	refclk	refclk/3
	4	x	—	—	refclk	refclk/4
3	2	3	18	450	150	75
3	2	4	24	600	200	100
3	3	2	18	450	150	50
3	3	3	27	675	225	75
3	4	2	24	600	200	50
4	1	4	16	400	100	100
4	2	2	16	400	100	50
4	2	3	24	600	150	75
4	2	4	32	800	200	100
4	3	2	24	600	150	50
4	4	1	16	400	100	25
4	4	2	32	800	200	50
6	1	3	18	450	75	75
6	1	4	24	600	100	100
6	2	2	24	600	100	50
6	3	1	18	450	75	25
6	4	1	24	600	100	25

Note 1: x = don't care.

Note 2: By-pass mode is designed for hardware emulator use only. Contact your IBM representative for more information.

Note 3: Not all PPC405GP parts support the frequencies shown. Check *PPC405GP Datasheet* for supported CPU and PLB frequencies for a specific part number.

Table 7-5. Possible Clocking Ratios for Reference Clock of 41.6MHz

Strapping Options			M	Calculated Values (MHz)		
FWD	CPU/PLB	FBK		VCO	CPU	PLB
By-pass[2]	1	x	—	—	refclk	refclk
	2	x	—	—	refclk	refclk/2
	3	x	—	—	refclk	refclk/3
	4	x	—	—	refclk	refclk/4
3	2	2	12	500	166	83
3	2	3	18	750	250	125
3	3	2	18	750	250	83
3	4	1	12	500	166	41
4	1	3	12	500	125	125
4	2	2	16	666	166	83
4	3	1	12	500	125	41
4	4	1	16	666	166	41
6	1	2	12	500	83	83
6	1	3	18	750	125	125
6	2	1	12	500	83	41
6	3	1	18	750	125	41

Note 1: x = don't care.

Note 2: By-pass mode is designed for hardware emulator use only. Contact your IBM representative for more information.

Note 3: Not all PPC405GP parts support the frequencies shown. Check *PPC405GP Datasheet* for supported CPU and PLB frequencies for a specific part number.

7.5 PCI Clocking

The PPC405GP PCI interface can run synchronously or asynchronously relative to the on-chip PLB. The state at reset of the PCI Asynchronous Mode Enable (PAME) strapping pin selects synchronous PCI mode or asynchronous PCI mode. The state of the PAME pin is reported in the PAME field of the CPC0_PSR. For information about the strapping pins, refer to *PowerPC 405GP Embedded Processor Data Sheet*, which is available from your IBM representative and in the Technical Library on the IBM Microelectronics web site (www.chips.ibm.com).

Lower PCI latency through the PCI logic occurs in synchronous PCI mode, which should be used for host-bridge applications in which the PCI bus frequency is less than or equal to 33 Mhz, and the PLB frequency is an integer multiple of the PCI frequency. PPC405GP applications for higher PCI frequencies, or for PCI adapters, should use asynchronous PCI mode.

7.5.1 PCI Clocks

The following clocks are related to the PCI logic:

- The on-chip PLB clock
- The on-chip synchronous PCI clock
- An external asynchronous PCI clock, PCIClk

In synchronous PCI mode, the asynchronous PCI clock input is ignored and the on-chip PCI logic runs synchronously at the selected PLB:PCI divide ratio. In synchronous mode, the derived PCI clock must equal SysClk.

In asynchronous PCI mode, the asynchronous PCI clock attaches to logic that interfaces with the PCI bus. The synchronous clock must always be provided to the PCI logic, even when the asynchronous PCI clock is used. Table 7-6 describes the relationships between the synchronous PCI clock, the asynchronous PCI clock, and the PLB clock.

In asynchronous PCI mode, the synchronous PCI clock must meet certain requirements. The following equation describes the relationship that must be maintained between the asynchronous PCI clock and synchronous PCI clock. Select an appropriate PCI:PLB ratio to maintain the relationship:

$$AsyncPCIclock \leq SyncPCIclock \leq ((2 \times AsyncPCIclock) + 1\,\text{MHz})$$

Table 7-6 lists supported and commonly used combinations of synchronous and asynchronous PCI clocks. In general, higher synchronous PCI clock frequencies provides better performance, while lower synchronous PCI clock frequencies minimize power consumption.

Table 7-6. Example Synchronous PCI Clock Frequencies in Asynchronous Mode

Asynchronous PCI Frequency	Synchronous PCI Frequency	PLB Frequency	Sync PCI:PLB Ratio
20 MHz	33.3 MHz	133.3 MHz	1:4
	33.3 MHz	100 MHz	1:3
	27.6 MHz	83.3 MHz	1:3
33.3 MHz	44.4 MHz	133.3 MHz	1:3
	33.3 MHz	133.3 MHz	1:4
	50 MHz	100 MHz	1:2
	41.6 MHz	83.3 MHz	1:2
40 MHz	44.4 MHz	133.3 MHz	1:3
	50 MHz	100 MHz	1:2
	41.6 MHz	83.3 MHz	1:2
66.6 MHz	66.6 MHz	133.3 MHz	1:2
	100 MHz	100 MHz	1:1
	83.3 MHz	83.3 MHz	1:1

7.5.2 PCI Adapter Applications

Because various systems run PCI expansion buses at different PCI frequencies, several PCI clock frequencies may need to be supported when the PPC405GP is used in PCI adapters. Using the PCI clock as a PPC405GP system clock input, as required for synchronous PCI mode, can adversely affect PPC405GP CPU performance. In some cases, the PPC405GP might not function.

For example, if the PPC405GP is used on a PCI adapter and the PCI expansion bus frequency is 33MHz, the PPC405GP strapping pins can be configured to run the chip at 200 MHz. However, if the same strapping configuration is used and the PPC405GP is used in a system in which the PCI expansion bus frequency is 25 MHZ (an application supported by the PCI specification), the PPC405GP would run suboptimally at 150 MHz. Worse problems would be encountered if the PCI frequency were less than 25 Mhz (the PPC405GP PLL locking frequency). In this application, the PPC405GP might not function. For these reasons, asynchronous PCI mode is recommended when the PPC405GP is used in PCI adapter applications.

Asynchronous PCI mode uses an externally provided PCI clock that does not interact with an on-chip PLL, so there is no lower frequency limit imposed by loss of PLL lock. However, the requirements resulting from the relationship between the synchronous and asynchronous PCI clocks must still be satisfied.

Note: Satisfying the equation in “PCI Clocks” on page 7-8 presents a potential problem. The strapping pin selection needed to set an acceptable synchronous PCI clock for a 33 MHz asynchronous PCI clock differs from the strapping pin selection for a 66 MHz asynchronous PCI clock. External logic is required to detect the state of the M66 pin on the PCI adapter interface and select appropriate PPC405GP strapping pins during system reset.

7.6 Serial Port Clocking

The two PPC405GP UARTs (serial ports) can be clocked individually, either from an external serial clock or from an internally generated serial clock. The internally generated clock is derived from the CPU clock, and is CPU/n, where n ranges from 1 to 32. The divisor *n* is programmed by setting a value of 0 to 31 in CPC0_CR0[UDIV] (see “Chip Control Register 0 (CPC0_CR0)” on page 7-11). Subsequently, the serial clock input to the UART is further divided in the UART to generate the desired serial data rate (baud rate).

Refer to Chapter 21, “Serial Port Operations,” for more information about choosing CPU clock and serial input clock divisors.

7.7 Clocking Registers

Table 7-7 summarizes the Device Control Registers (DCRs) that control clocking in the PPC405GP.

Table 7-7. Clocking Control Registers

Register	Address	R/W	Description
CPC0_PLLMR	0x0B0	R/O	PLL Mode Register
CPC0_CR0	0x0B1	R/W	Chip Control Register 0

CPC0_CR0 selects the internal serial clock source. The clocking registers are read and written using the **mtdcr** and **mfdcr** instructions.

7.7.1 PLL Mode Register (CPC0_PLLMR)

The read-only CPC0_PLLMR contains a number of PLL and clock divisor values.

CPC0_PLLMR fields are based on the external strapping options.

The CPC0_PLLMR is big endian. However, values returned for the CPC0_PLLMR[FWDV, FBDV, TUN] fields are little endian to enable easier comparison of these bits with the PLL documentation in *ASIC SA-12E Databook* (available from your IBM representative), which uses little endian formats.

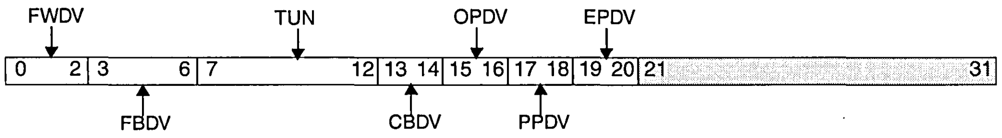


Figure 7-2. PLL Mode Register (CPC0_PLLMR)

0:2	FWDV	Forward Divisor 000 Forward divisor is 8. 001 Forward divisor is 7. 010 Forward divisor is 6. 011 Forward divisor is 5. 100 Forward divisor is 4. 101 Forward divisor is 3. 110 Forward divisor is 2. 111 Forward divisor is 1.	PLLOUTA Value: 50 MHz–100 MHz 58 MHz–114 MHz 66 MHz–134 MHz 80 MHz–160 MHz 100 MHz–200 MHz 133 MHz–267 MHz 200 MHz–400 MHz 400 MHz–800 MHz **Note:** PLLOUTA is the CPU clock in PPC405GP.
3:6	FBDV	Feedback Divisor 0000 Feedback divisor is 16. 0001 Feedback divisor is 1. 0010 Feedback divisor is 2. 0011 Feedback divisor is 3. 0100 Feedback divisor is 4. 0101 Feedback divisor is 5. 0110 Feedback divisor is 6. 0111 Feedback divisor is 7. 1000 Feedback divisor is 8. 1001 Feedback divisor is 9. 1010 Feedback divisor is 10. 1011 Feedback divisor is 11. 1100 Feedback divisor is 12. 1101 Feedback divisor is 13. 1110 Feedback divisor is 14. 1111 Feedback divisor is 15.	

7:12	TUN	TUNE[5:0] Field	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL implemented in the PPC405GP. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs) with the PPC405GP, visit the technical documents area of the IBM PowerPC web site.
13:14	CBDV	CPU:PLB Frequency Divisor 00 CPU:PLB divisor is 1 01 CPU:PLB divisor is 2 10 CPU:PLB divisor is 3 11 CPU:PLB divisor is 4	
15:16	OPDV	OPB–PLB Frequency Divisor 00 OPB–PLB divisor is 1 01 OPB–PLB divisor is 2 10 OPB–PLB divisor is 3 11 OPB–PLB divisor is 4	
17:18	PPDV	PCI–PLB Frequency Divisor 00 PCI–PLB divisor is 1 01 PCI–PLB divisor is 2 10 PCI–PLB divisor is 3 11 PCI–PLB divisor is 4	See “PCI Clocks” on page 7-8 for details regarding asynchronous PCI clocking and how it relates to synchronous clocking.
19:20	EPDV	External Bus–PLB Frequency Divisor 00 External bus–PLB divisor ratio is 2:1 01 External bus–PLB divisor ratio is 3:1 10 External bus–PLB divisor ratio is 4:1 11 External bus–PLB divisor ratio is 5:1	
21:31	▨	Reserved	

7.7.2 Chip Control Register 0 (CPC0_CR0)

Only CPC0_CR0 fields related to clocking are shown in Figure 7-3. CPC0_CR0$_{0:23}$, which control GPIO and UART functions, are summarized in “CPC0_CR0” on page 25-17.

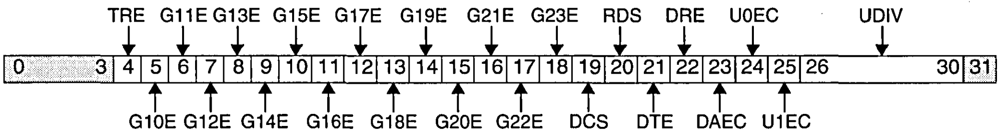


Figure 7-3. Chip Control Register 0 (CPC0_CR0)

0:3	▨	Reserved	
4	TRE	CPU Trace Enable 0 GPIO1-9 are enabled 1 GPIO1-9 are disabled	Trace interface cannot be used when GPIO is enabled.
5	G10E	GPIO 10 Enable 0 Enable $\overline{\text{PerCS1}}$ as a chip select 1 Enable $\overline{\text{PerCS1}}$ as GPIO10	
6	G11E	GPIO 11 Enable 0 Enable $\overline{\text{PerCS2}}$ as a chip select 1 Enable $\overline{\text{PerCS2}}$ as GPIO11	
7	G12E	GPIO 12 Enable 0 Enable $\overline{\text{PerCS3}}$ as a chip select 1 Enable $\overline{\text{PerCS3}}$ as GPIO12	
8	G13E	GPIO 13 Enable 0 Enable $\overline{\text{PerCS4}}$ as a chip select 1 Enable $\overline{\text{PerCS4}}$ as GPIO13	
9	G14E	GPIO 14 Enable 0 Enable $\overline{\text{PerCS5}}$ as a chip select 1 Enable $\overline{\text{PerCS5}}$ as GPIO14	
10	G15E	GPIO 15 Enable 0 Enable $\overline{\text{PerCS6}}$ as a chip select 1 Enable $\overline{\text{PerCS6}}$ as GPIO15	
11	G16E	GPIO 16 Enable 0 Enable $\overline{\text{PerCS7}}$ as a chip select 1 Enable $\overline{\text{PerCS7}}$ as GPIO16	
12	G17E	GPIO 17 Enable 0 Enable interrupt IRQ0 as an interrupt 1 Enable interrupt IRQ0 as GPIO17	The purpose of GPIO_17_EN through GPIO_23_EN is to isolate the interrupt controller from activity on a shared pin when that pin is being used as a GPIO. For instance, when G17E is set to a 1, IRQ0 at the UIC will always be forced to a zero. Note: Setting G17E to a 0 will not prevent GPIO channel 17 (if configured as an output) from creating contention with the off-chip source of the IRQ input. Therefore, be sure to leave the shared GPIO channel disabled when using the pin as an interrupt input.

13	G18E	GPIO 18 Enable 0 Enable interrupt IRQ1 as an interrupt 1 Enable interrupt IRQ1 as GPIO18
14	G19E	GPIO 19 Enable 0 Enable interrupt IRQ2 as an interrupt 1 Enable interrupt IRQ2 as GPIO19
15	G20E	GPIO 20 Enable 0 Enable interrupt IRQ3 as an interrupt 1 Enable interrupt IRQ3 as GPIO20
16	G21E	GPIO 21 Enable 0 Enable interrupt IRQ4 as an interrupt 1 Enable interrupt IRQ4 as GPIO21
17	G22E	GPIO 22 Enable 0 Enable interrupt IRQ5 as an interrupt 1 Enable interrupt IRQ5 as GPIO22
18	G23E	GPIO 23 Enable 0 Enable interrupt IRQ6 as an interrupt 1 Enable interrupt IRQ6 as GPIO23
19	DCS	DSR/CTS select 0 DSR is selected. 1 CTS is selected.
20	RDS	RTS/DTR select 0 RTS is selected. 1 DTR is selected.
21	DTE	DMA Transmit Enable for UART0 0 DMA transmit channel is disabled. 1 DMA transmit channel is enabled.
22	DRE	DMA Receive Enable for UART0 0 DMA receive channel is disabled. 1 DMA receive channel is enabled.
23	DAEC	DMA Allow Enable Clear for UART0 0 DTE and DRE for UART0 are not cleared when the UART receives a corresponding terminal count. 1 DTE and DRE for UART0 are cleared when the UART receives a corresponding terminal count.
24	U0EC	Select External Clock for UART0 0 UART0 uses the internally derived serial clock 1 UART0 uses the external serial clock input
25	U1EC	Select External Clock for UART1 0 UART1 uses the internally derived serial clock 1 UART1 uses the external serial clock input

26:30	UDIV	UART Internal Clock Divisor 00000 Divide by 1 00001 Divide by 2 00010 Divide by 3 . . . 11110 Divide by 31 11111 Divide by 32	UDIV specifies the divisor of the CPU clock frequency used to derive a UART serial clock frequency. For example, if the CPU is running at 200MHz, a divider value of 20 sets the serial clock frequency to 10MHz. **Note:** The maximum serial clock frequency is less than 1/2 × OPB frequency
31	▨	Reserved	

Chapter 8. Reset and Initialization

This chapter describes reset operations, the initial state of the PPC405GP processor core after a reset, and an example of the initialization code required to begin executing application code. Initialization of external system components or system-specific chip facilities must also be performed, in addition to the basic initialization described in this chapter.

Reset operations affect the PPC405GP at power on time as well as during normal operation, if programmed to do so. To understand how these operations work it is necessary to first understand the signal pins involved as well as the terminology of core, chip and system resets.

8.1 Reset Signals

The PPC405GP provides two reset signals, $\overline{\text{SysReset}}$ and $\overline{\text{ExtReset}}$. $\overline{\text{SysReset}}$ is bidirectional and $\overline{\text{ExtReset}}$ is an output.

When the $\overline{\text{SysReset}}$ signal is an input (asserted by an off-chip device), such as during power on reset (POR), the chip responds by performing a system reset as described in a following section. An external assertion of $\overline{\text{SysReset}}$ is not extended by an assertion of the open drain bidirectional $\overline{\text{SysReset}}$ driver.

As an output, the PPC405GP asserts the $\overline{\text{SysReset}}$ signal when a system reset request is detected. The $\overline{\text{SysReset}}$ open drain driver is activated and the signal is driven low for 8192 SysClk periods. This enables the PPC405GP to reset itself and other devices attached to the same reset network.

The $\overline{\text{ExtReset}}$ signal is used by synchronous peripheral devices served by the PerClk external bus clock, such as ROM and external bus masters. During chip and system resets, $\overline{\text{ExtReset}}$ is asserted until the PerClk signal is stable and all internal resets are released.

8.2 Reset Types

Three types of reset, each with different scope, are possible in the PPC405GP. A core reset affects only the processor core. Chip resets affect the processor core and all on-chip peripherals. System resets affect the processor core, all on-chip peripherals, and any off-chip devices connected to the PPC405GP reset net. Refer to chapters describing the on-chip peripherals for detailed information about their reset behavior.

8.2.1 Core Reset

A core reset results in a reset of the processor core. No other on-chip logic is affected. Clocking logic, outside the processor core, detects the core reset request and asserts the reset input to the processor core for a period of 16 processor core clock cycles.

8.2.2 Chip Reset

A chip reset results in the reset of the processor core and on-chip peripherals. Clocking logic detects the request for a chip reset and asserts the reset input to the processor core for a period of 16 processor core clock cycles. Subsequently, PLL locking is performed as described for a system reset.

During chip reset, the $\overline{\text{ExtReset}}$ signal is driven low to ensure the reset of synchronous devices that use the external bus clock signal, PerClk.

8.2.3 System Reset

A system reset results in a reset of all PPC405GP logic, including the processor core, internal phase-locked loop (PLL), and on-chip peripherals. A system reset can be initiated externally or internally. External system resets are initiated by the assertion of the $\overline{\text{SysReset}}$ signal for at least 16 SysClk cycles. Internal system resets are initiated by either the processor core or PCI power management logic.

When a system reset is requested internally, the bidirectional open drain $\overline{\text{SysReset}}$ signal is asserted to enable other chips to be reset at the same time. In this case, the $\overline{\text{SysReset}}$ signal is driven low for 8192 SysClk cycles, resulting in a System reset of the PPC405GP chip and all other devices attached to the reset network connected to $\overline{\text{SysReset}}$.

After the $\overline{\text{SysReset}}$ signal is deasserted, the PLL begins its locking process, which also requires 8192 SysClk cycles. When 64 SysClk cycles remain, the internal PPC405GP clocks (OPB, PCI, EXT, and Serial) begin toggling. The PLB and CPU clocks toggle while $\overline{\text{SysReset}}$ is asserted, and during the PLL locking process. When the PLL lock timer expires, internal resets are released, and the processor core begins its initial instruction fetch.

During system reset, the $\overline{\text{ExtReset}}$ signal is driven low to ensure the reset of synchronous devices that use the external bus clock signal, PerClk.

8.3 PCI Power Management Initiated Resets

An external PCI master can write the Power Management Control/Status Register (PCIC0_PMCSR) to request a change from the D3hot PCI power management state to the D0 state. The on-chip PCI logic always accepts such a write and assumes that requested state changes from D3hot are always to D0. After receiving such a write, the PCI logic asserts a signal that immediately results in an internally requested system reset.

8.4 Processor Initiated Resets

The processor core in the PPC405GP can request three types of processor resets: core, chip, and system. Each type of reset can be generated by a JTAG debug tool, by the second expiration of the watchdog timer, or by writing a non-zero value to the Reset (RST) field of the Debug Control Register 0 (DBCR0).

The effects of core and chip resets on the processor core are identical. To determine which reset type occurred, the most-recent reset (MRR) field of the Debug Status Register (DBSR) can be examined.

8.5 Processor State After Reset

After a reset, the contents of the Machine State Register (MSR) and the Special Purpose Registers (SPRs) control the initial processor state. The contents of Device Control Registers (DCRs) control the initial states of on-chip devices. Chapter 25, "Register Summary," contains descriptions of the registers.

In general, the contents of SPRs are undefined after a reset. Reset initializes the minimum number of SPR fields required for allow successful instruction fetching. "Contents of Special Purpose Registers after Reset" on page 8-4 describes these initial values. System software fully configures the processor.

"Machine State Register Contents after Reset" on page 8-3 describes the MSR contents.

The MCI field of the Exception Syndrome Register (ESR) is cleared so that it can be determined if there has been a machine check during initialization, before machine check exceptions are enabled.

Two SPRs contain status on the type of reset that has occurred. The Debug Status Register (DBSR) contains the most recent reset type. The Timer Status Register (TSR) contains the most recent watchdog reset.

8.6 Processor Register Contents After Reset

After a reset, the contents of the SPRs control the initial processor state. The initial register contents vary with the type of reset that occurred.

Chapter 25, "Register Summary," contains descriptions of the registers referred to in Table 8-1 through Table 8-3.

8.6.1 Machine State Register Contents after Reset

After all resets, all fields of the Machine State Register (MSR) contain zeros. Table 8-1 shows how this affects chip operation.

Table 8-1. MSR Contents after Reset

Register	Field	Core Reset	Chip Reset	System Reset	Comment
MSR	WE	0	0	0	Wait state disabled
	CE	0	0	0	Critical interrupts disabled
	EE	0	0	0	External interrupts disabled
	PR	0	0	0	Supervisor mode
	ME	0	0	0	Machine check exceptions disabled
	DWE	0	0	0	Debug wait mode disabled
	DE	0	0	0	Debug interrupts disabled
	DR	0	0	0	Instruction translation disabled
	IR	0	0	0	Data translation disabled

8.6.2 Contents of Special Purpose Registers after Reset

In general, the contents of Special Purpose Registers (SPRs) are undefined after a core, chip, or system reset. Some SPRs retain the contents they had before a reset occurred.

Table 8-2 shows the contents of SPRs that are defined or unchanged after core, chip, and system resets.

Table 8-2. SPR Contents After Reset

Register	Bits/Fields	Core Reset	Chip Reset	System Reset	Comment
DBCR0	EDM	0	0	0	External debug mode disabled
	RST	00	00	00	No reset action.
DBCR1	0:31	0x00000000	0x00000000	0x00000000	Instruction, data compares disabled
DBSR	MRR	01	10	11	Most recent reset
DCCR	S0:S31	0x00000000	0x00000000	0x00000000	Data cache disabled
ESR	0:31	0x00000000	0x00000000	0x00000000	No exception syndromes
ICCR	S0:S31	0x00000000	0x00000000	0x00000000	Instruction cache disabled
PVR	0:31				Processor version
SGR	G0:G31	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	Storage is guarded
SLER	S0:S31	0x00000000	0x00000000	0x00000000	Storage is big endian
SU0R	K0:K31	0x00000000	0x00000000	0x00000000	Storage is uncompressed

8.7 DCR Contents after Reset

DCR reset values are unaffected by core resets and are generally identical for chip and system resets.

Table 8-3. DCR Contents After Reset

Register	Bits	Reset Value	Comment
Chip Control			
CP0_PSR	0:31		At POR, CPC0_PSR fields are set to the strapping values of the corresponding pins. CPC0_PSR is read-only.
CPC0_CR0	0:31	0x0000003C	
CPC0_CR1	0:31	0x2B0DB800	
CPC0_JTAGID	0:31		Refer to *PPC405GP Embedded Processor Data Sheet* for the value of this read-only register.
CPC0_PLLMR	0:31		At POR, CPC0_PLLMR fields are set to strapping values of the corresponding pins. CPC0_PLLMR is read-only.
Clock and Power Management (CPM)			
CPC0_ER	0:31	0xFFFF8000	CPC0_ER$_{0:16}$ return 1, CPC0_ER$_{17:31}$ return 0.

Table 8-3. DCR Contents After Reset (continued)

Register	Bits	Reset Value	Comment
CPC0_FR	0:31	0x00000000	
CPC0_SR	0:31	0x00000000	
Decompression Controller			
DCP0_ADDR0	0:31	0x00000000	
DCP0_ADDR1	0:31	0x00000000	
DCP0_MEMBEAR	0:31	0x00000000	
DCP0_CFG	0:31	0x00000001	
DCP0_ESR	0:31	0x00000000	
DCP0_ID	0:31	0x0000504D	
DCP0_ITOR0	0:31	0x00000000	
DCP0_ITOR1	0:31	0x00000000	
DCP0_ITOR2	0:31	0x00000000	
DCP0_ITOR3	0:31	0x00000000	
DCP0_PLBBEAR	0:31	0x00000000	
DCP0_RAM0–DCP0_RAM3FF	1KB	Undefined	
DCP0_VER	0:31	0x00000200	
Direct Memory Access (DMA)			
DMA0_CR0	0:31	0x00000000	
DMA0_CR1	0:31	0x00000000	
DMA0_CR2	0:31	0x00000000	
DMA0_CR3	0:31	0x00000000	
DMA0_CT0	0:31	0x00000000	
DMA0_CT1	0:31	0x00000000	
DMA0_CT2	0:31	0x00000000	
DMA0_CT3	0:31	0x00000000	
DMA0_DA0	0:31	0x00000000	
DMA0_DA1	0:31	0x00000000	
DMA0_DA2	0:31	0x00000000	
DMA0_DA3	0:31	0x00000000	
DMA0_POL	0:31	0x00000000	
DMA0_SA0	0:31	0x00000000	
DMA0_SA1	0:31	0x00000000	
DMA0_SA2	0:31	0x00000000	

Table 8-3. DCR Contents After Reset (continued)

Register	Bits	Reset Value	Comment
DMA0_SA3	0:31	0x00000000	
DMA0_SG0	0:31	0x00000000	
DMA0_SG1	0:31	0x00000000	
DMA0_SG2	0:31	0x00000000	
DMA0_SG3	0:31	0x00000000	
DMA0_SGC	0:31	0x00000000	
DMA0_SLP	0:31	0x7C000000	
DMA0_SR	0:31	0x00000000	
External Bus Controller (EBC)			
EBC0_B0AP	0:31	0x7F8FFE80	Slowest possible bus timings.
EBC0_B0CR	0:31	0xFFE28000	2MB read-only bank.
EBC0_B1AP	0:31	0x00000000	
EBC0_B1CR	0:31	0x00000000	
EBC0_B2AP	0:31	0x00000000	
EBC0_B2CR	0:31	0x00000000	
EBC0_B3AP	0:31	0x00000000	
EBC0_B3CR	0:31	0x00000000	
EBC0_B4AP	0:31	0x00000000	
EBC0_B4CR	0:31	0x00000000	
EBC0_B5AP	0:31	0x00000000	
EBC0_B5CR	0:31	0x00000000	
EBC0_B6AP	0:31	0x00000000	
EBC0_B6CR	0:31	0x00000000	
EBC0_B7AP	0:31	0x00000000	
EBC0_B7CR	0:31	0x00000000	
EBC0_BEAR	0:31	0x00000000	
EBC0_BESR0	0:31	0x00000000	
EBC0_BESR1	0:31	0x00000000	
EBC0_CFG	0:31	0x00000000	
Indirect Addressing Registers			
DCP0_CFGADDR		0x00000000	
DCP0_CFGDATA		0x00000000	
EBC0_CFGADDR		0x00000000	

Table 8-3. DCR Contents After Reset (continued)

Register	Bits	Reset Value	Comment
EBC0_CFGDATA		0x00000000	
SDRAM0_CFGADDR		0x00000000	
SDRAM0_CFGDATA		0x00000000	
Media Access Layer (MAL)			
MAL0_CFG	0:31	0x0007C000	
MAL0_ESR	0:31	0x00000000	
MAL0_IER	0:31	0x00000000	
MAL0_RCBS0	0:31	Undefined	
MAL0_RXCASR	0:31	0x00000000	
MAL0_RXCTP0R	0:31	Undefined	
MAL0_RXDEIR	0:31	0x00000000	
MAL0_RXEOBISR	0:31	0x00000000	
MAL0_TXCASR	0:31	0x00000000	
MAL0_TXCTP0R	0:31	Undefined	
MAL0_TXCTP1R	0:31	Undefined	
MAL0_TXDEIR	0:31	0x00000000	
MAL0_TXEOBISR	0:31	0x00000000	
On-Chip Buses			
PLB0_ACR	0:31	0x00000000	
PLB0_BEAR	0:31	Undefined	
POB0_BEAR	0:31	Undefined	
POB0_BESR0	0:31	0x00000000	
POB0_BESR1	0:31	0x00000000	
On-Chip Memory			
OCM0_DSARC	0:31	Undefined	
OCM0_DSCNTL	0:31	0x40000000	
OCM0_ISARC	0:31	Undefined	
OCM0_ISCNTL	0:31	0x40000000	
SDRAM Controller			
SDRAM0_B0CR	0:31	0x00000000	
SDRAM0_B1CR	0:31	0x00000000	
SDRAM0_B2CR	0:31	0x00000000	
SDRAM0_B3CR	0:31	0x00000000	

Table 8-3. DCR Contents After Reset (continued)

Register	Bits	Reset Value	Comment
SDRAM0_BEAR	0:31	0x00000000	
SDRAM0_BESR0	0:31	0x00000000	
SDRAM0_BESR1	0:31	0x00000000	
SDRAM0_CFG	0:31	0x00000000	
SDRAM0_ECCCFG	0:31	0x00000000	
SDRAM0_ECCESR	0:31	0x00000000	
SDRAM0_PMIT	0:31	0x07C00000	
SDRAM0_RTR	0:31	0x05F00000	
SDRAM0_TR	0:31	0x00854009	
Universal Interrupt Controller (UIC)			
UIC0_CR		Undefined	
UIC0_ER		0x00000000	
UIC0_MSR		Undefined	
UIC0_PR		Undefined	
UIC0_SR		Undefined	
UIC0_TR		Undefined	
UIC0_VCR		Undefined	
UIC0_VR		Undefined	

8.8 MMIO Register Contents After Reset

MMIO registers are unaffected by core resets, and are generally identical for chip and system resets.

Table 8-4. MMIO Register Contents After Reset

Register	Bits	Reset Value	Comment
Ethernet (EMAC)			
EMAC0_GAHT1	0:31	0x00000000	
EMAC0_GAHT2	0:31	0x00000000	
EMAC0_GAHT3	0:31	0x00000000	
EMAC0_GAHT4	0:31	0x00000000	
EMAC0_IAHR	0:31	0x00000000	
EMAC0_IAHT1	0:31	0x00000000	
EMAC0_IAHT2	0:31	0x00000000	
EMAC0_IAHT3	0:31	0x00000000	

Table 8-4. MMIO Register Contents After Reset (continued)

Register	Bits	Reset Value	Comment
EMAC0_IAHT4	0:31	0x00000000	
EMAC0_IALR	0:31	0x00000000	
EMAC0_IPGVR	0:31	0x00000004	
EMAC0_ISER	0:31	0x00000000	
EMAC0_ISR	0:31	0x00000000	
EMAC0_LSAH	0:31	0x00000000	
EMAC0_LSAL	0:31	0x00000000	
EMAC0_MR0	0:31	0xC0000000	
EMAC0_MR1	0:31	0x00000000	
EMAC0_PTR	0:31	0x0000FFFF	
EMAC0_RMR	0:31	0x00000000	
EMAC0_RWMR	0:31	0x04001000	
EMAC0_STACR	0:31	0x00008000	
EMAC0_TMR0	0:31	0x00000000	
EMAC0_TMR1	0:31	0x380F0000	
EMAC0_TRTR	0:31	0x00000000	
EMAC0_VTCI	0:31	0x00000000	
EMAC0_VTPID	0:31	0x00008808	
General Purpose I/O (GPIO)			
GPIO0_IR	0:31	Undefined	Read-only; follows the GPIO_In input.
GPIO0_ODR	0:31	0x00000000	
GPIO0_OR	0:31	0x00000000	
GPIO0_TCR	0:31	0x00000000	
Inter-Integrated Circuit (IIC)			
IIC0_CLKDIV	0:7	0x00	
IIC0_CNTL	0:7	0x00	
IIC0_DIRECTCNTL	0:7	0x0F	
IIC0_EXTSTS	0:7	0x00	
IIC0_HMADR	0:7	Undefined	
IIC0_HSADR	0:7	Undefined	
IIC0_INTRMSK	0:7	0x00	
IIC0_LMADR	0:7	Undefined	
IIC0_LSADR	0:7	Undefined	

Table 8-4. MMIO Register Contents After Reset (continued)

Register	Bits	Reset Value	Comment
IIC0_MDBUF	0:7	0x00	
IIC0_MDCNTL	0:7	0x00	
IIC0_SDBUF	0:7	0x00	
IIC0_STS	0:7	0x00	
IIC0_XFRCNT	0:7	0x00	
IIC0_XTCNTLSS	0:7	0x00	
OPB Arbiter			
OPBA0_CR	0:31	0x00000000	
OPBA0_PR	0:31	0x00011011	
PCI Bridge			
PCIC0_BASECC	0:7	0x06	
PCIC0_BIST	0:7	0x00	
PCIC0_BRDGOPT1	0:15	0xFF60	
PCIC0_BRDGOPT2	0:15	0x0100	
PCIC0_CACHELS	0:7	0x00	
PCIC0_CAP	0:7	0x00	
PCIC0_CAPID	0:7	0x00	
PCIC0_CFGADDR	0:31	0x00000000	
PCIC0_CFGDATA	0:31	0x00000000	
PCIC0_CLS	0:23	0x060000	
PCIC0_CMD	0:15	0x0000	
PCIC0_DEVID	0:15	0x0156	
PCIC0_ERREN	0:7	0x00	
PCIC0_ERRSTS	0:7	0x00	
PCIC0_HDTYPE	0:7	0x00	
PCIC0_ICS	0:7	0x00	
PCIC0_INTCLS	0:7	0x00	
PCIC0_INTLN	0:7	0x00	
PCIC0_INTPN	0:7	0x01	
PCIC0_LATTIM	0:7	0x07	
PCIC0_MAXLTNCY	0:7	0x00	
PCIC0_MINGNT	0:7	0x00	
PCIC0_NEXTIPTR	0:7	0x00	

Table 8-4. MMIO Register Contents After Reset (continued)

Register	Bits	Reset Value	Comment
PCIC0_PLBBEAR	0:31	0x00000000	
PCIC0_PLBBESR0	0:31	0x00000000	
PCIC0_PLBBESR1	0:31	0x00000000	
PCIC0_PMC	0:15	0x0202	
PCIC0_PMCSR	0:15	0x0000	
PCIC0_PMCSRBSE	0:7	0x00	
PCIC0_PMSCRR	0:7	0x10	
PCIC0_PTM1BAR	0:31	0x00000008	
PCIC0_PTM2BAR	0:31	0x00000000	
PCIC0_REVID	0:7	0x00	Value corresponds to revision level.
PCIC0_SBSYSID	0:15	0x0000	
PCIC0_SBSYSVID	0:15	0x0000	
PCIC0_STATUS	0:15	0x0210	
PCIC0_SUBCLS	0:7	0x00	
PCIC0_VENDID	0:15	0x1014	
PCIL0_PMM0LA	0:31	0xFFFE0000	Value if strapped for PCI boot at reset.
PCIL0_PMM0MA	0:31	0xFFFE0001	Value if strapped for PCI boot at reset.
PCIL0_PMM0PCIHA	0:31	0x00000000	Value if strapped for PCI boot at reset.
PCIL0_PMM0PCILA	0:31	0xFFFE0000	Value if strapped for PCI boot at reset.
PCIL0_PMM1LA	0:31	Undefined	
PCIL0_PMM1MA	0:31	0x00000000	
PCIL0_PMM1PCIHA	0:31	Undefined	
PCIL0_PMM1PCILA	0:31	Undefined	
PCIL0_PMM2LA	0:31	Undefined	
PCIL0_PMM2MA	0:31	0x00000000	
PCIL0_PMM2PCIHA	0:31	Undefined	
PCIL0_PMM2PCILA	0:31	Undefined	
PCIL0_PTM1LA	0:31	Undefined	
PCIL0_PTM1MS	0:31	Undefined	
PCIL0_PTM2LA	0:31	Undefined	
PCIL0_PTM2MS	0:31	Undefined	
PCIC0_PLBBEAR2	0:31	0x00000000	
PCIC0_PLBBEAR3	0:31	0x00000000	

Table 8-4. **MMIO Register Contents After Reset (continued)**

Register	Bits	Reset Value	Comment
Serial Port (UART)			
UART0_DLL	0:7	0x00000000	
UART0_DLM	0:7	0x00000000	
UART0_FCR	0:7	0x00000000	
UART0_IER	0:7	0x00000000	
UART0_IIR	0:7	0x00000001	
UART0_LCR	0:7	0x00000000	
UART0_LSR	0:7	0x01100000	
UART0_MCR	0:7	0x00000000	
UART0_MSR	0:7	0xxxxx0000	UART0_MSR$_{0:3}$ are driven by $\overline{\text{UART0\_DCD}}$, $\overline{\text{UART0\_RI}}$, $\overline{\text{UART1\_DSR}}$[$\overline{\text{UART1\_CTS}}$], and $\overline{\text{UART1 \_RTS}}$[$\overline{\text{UART1\_DTR}}$].
UART0_RBR	0:7	0x00000000	
UART0_SCR	0:7	Undefined	
UART0_THR	0:7	0x00000000	
UART1_DLL	0:7	0x00000000	
UART1_DLM	0:7	0x00000000	
UART1_FCR	0:7	0x00000000	
UART1_IER	0:7	0x00000000	
UART1_IIR	0:7	0x00000001	
UART1_LCR	0:7	0x00000000	
UART1_LSR	0:7	0x01100000	
UART1_MCR	0:7	0x00000000	
UART1_MSR	0:7	Undefined	
UART1_RBR	0:7	Undefined	
UART1_SCR	0:7	Undefined	
UART1_THR	0:7	0x00000000	

8.9 PPC405GP Chip Initialization

The on-chip memory (OCM) controller and the universal Interrupt controller (UIC) require initialization for best performance (in the case of OCM) and proper operation (in the case of the UIC). The UART controller may require initialization, depending upon the application.

Other peripheral devices can be initialized as appropriate for the system design.

8.9.1 OCM Initialization

The following information applies if OCM is to be accessed.

If instruction-side OCM is to be accessed, the OCM Instruction-Side Address Range Compare Register (OCM0_ISARC) must be initialized to locate the instruction-side OCM address space in the PPC405GP address map. If data-side OCM is to be accessed, the OCM Data-Side Address Range Compare Register (OCM0_DSARC) must be initialized to locate the data-side OCM address space into the PPC405GP address map. "OCM Addressing" on page 5-1 provides details of the instruction-side OCM and data-side OCM address spaces. See "Memory Organization and Addressing" on page 3-1 for information about the PPC405GP memory map.

8.9.1.1 Initializing Instruction-Side OCM

The following procedure describes the steps to be taken If instruction-side OCM is to be accessed.

1. Set the ISEN field of the OCM Instruction-Side Control Register to disable instruction-side OCM accesses (OCM0_ISCNTL[ISEN] = 0).
2. To ensure that interrupts do not interfere with the instruction cache array invalidation performed in step 4, set MSR[EE] = 0 and MSR[CE] to mask interrupts. This prevents a potential problem caused by dirty cache addresses that would not be fetched from the cache because they are marked as non-cachable.
3. Mark the address region to be programmed as OCM address space as non-cachable.
4. Invalidate the instruction cache array to ensure that no addresses to be programmed as OCM addresses are in the cache. The **iccci** instruction invalidates the instruction cache array.
5. Modify the value in OCM0_ISARC.
6. Set OCM0_ISCNTL[ISEN] = 1 to enable instruction-side OCM accesses. Also, set OCM0_ISCNTL[ISTCM] = 0.)

OCM0_ISCNTL[ISTCM] should be initialized to 0 to take the instruction-side OCM controller out of two cycle mode, the default mode after a reset. This enables instruction-side fetches to complete in a single cycle, providing the same performance as cache hits. See "OCM Instruction-Side Control Register (OCM0_ISCNTL)" on page 5-6 for details.

8.9.1.2 Initializing Data-Side OCM

The following procedure describes the steps to be taken if data-side OCM is used to load the initial contents of instruction-side OCM, or if data-side OCM is to be accessed.

1. Set the DSEN field of the Data-Side Control Register to disable data-side OCM accesses (OCM0_DSCNTL[DSEN] = 0).
2. To ensure that interrupts do not interfere with the data cache array invalidation performed in step 4, set MSR[EE] = 0 and MSR[CE] to mask interrupts. This prevents a potential problem caused by dirty cache addresses that would not be fetched from the cache because they are marked as non-cachable.
3. Mark the address region that is to be programmed as OCM address space as non-cachable.
4. Invalidate the data cache array to ensure that no addresses to be programmed as OCM addresses are in the cache. Use a sequence of **dcbf** instructions to invalidate the data cache.

5. Modify the value in OCM0_DSARC.

6. Set OCM0_DSCNTL[DSEN] = 1 to enable data-side OCM accesses.

OCM0_DSCNTL[DOF] should remain at its reset value of 1. See "OCM Data-Side Control Register (OCM0_DSCNTL)" on page 5-7 for details.

8.9.2 UIC Initialization

The following information does not provide all initialization information for the UIC. Some initialization details are application-dependent.

The polarity and sensitivity of the on-chip interrupts must be initialized for proper chip operation. The fields controlling on-chip interrupts in the UIC Polarity Register (UIC0_PR$_{0:18}$) must be set to 1. See "UIC Polarity Register (UIC0_PR)" on page 10-10 for details. The fields controlling on-chip interrupts in the UIC Trigger Register (UIC0_TR$_{0:2}$ and UIC0_TR$_{4:18}$) must be set to 0. The field controlling the external master interrupt trigger (UIC0_TR$_3$) must be set to 1. See "UIC Trigger Register (UIC0_TR)" on page 10-13 for details.

8.9.3 UART Initialization

Bits 19 and 20 of Chip Control Register 0 (CPC0_CR0) control which of two modem control signal pairs, Data Set Ready (DSR) and Data Terminal Ready (DTR) or Clear to Send (CTS) and Ready to Send (RSR), are used. The signal pairs are implemented using the multiplexed pins $\overline{\text{UART1\_DSR}}[\overline{\text{UART1\_CTS}}]$, controlled by CPC0_CR0$_{19}$, and $\overline{\text{UART1\_RTS}}[\overline{\text{UART1\_DTR}}]$, controlled by CPC0_CR0$_{20}$. After reset, CPC0_CR0$_{19}$ or CPC0_CR0$_{20}$ must be changed to achieve a required typical pairing.

8.10 PPC405GP Initial Processor Sequencing

After any reset, the processor core fetches the word at address 0xFFFFFFFC and attempts to execute it. Because the only memory configured immediately after reset is the upper 2MB region (0xFFE00000–0xFFFFFFFF), the instruction at 0xFFFFFFFC must be a branch instruction.

Because the processor is initially in big endian mode, initialization code must be in big endian format until the endian storage attribute for the addressed region is changed, or until code branches to a region defined as little endian storage.

Before a reset operation begins, the system must provide non-volatile memory, or memory initialized by some mechanism external to the processor. This memory must be located at address 0xFFFFFFFC in the upper 2MB region. This memory can be attached to the external bus controller (EBC) or to the system PCI bus. For memory attached to the EBC, the upper 2MB bank configuration after reset is 256 wait states, three cycles of address to chip select delay, three cycles of chip select to output enable delay, and seven cycles of hold time. The bus width (8-, 16-, or 32-bit) is controlled by the ROM width strapping signals. See "Pin Strapping" on page 9-1 for details. See "Local Processor Boot from PCI Memory" on page 17-64 for information about memory attached to the system PCI bus.

8.11 Initialization Requirements

When any reset is performed, the processor is initialized to a minimum configuration to start executing initialization code. Initialization code is necessary to complete the processor and system configuration.

The initialization code example in this section performs the configuration tasks required to prepare the PPC405GP to boot an operating system or run an application program.

Some portions of the initialization code work with system components that are beyond the scope of this manual.

Initialization code should perform the following tasks to configure the processor resources.

To improve instruction fetching performance: initialize the SGR appropriately for guarded or unguarded storage. Since all storage is initially guarded and speculative fetching is inhibited to guarded storage, reprogramming the SGR will improve performance for unguarded regions.

1. Configure the following storage attribute control registers, if necessary:
   - Initialize the SLER to configure storage byte ordering.
   - Initialize the SU0R to configure storage compression.
2. Before executing instructions as cachable:
   - Invalidate the instruction cache.
   - Initialize the ICCR to configure instruction cachability.
3. Before using storage access instructions:
   - Invalidate the data cache.
   - Initialize CRRO to determine if a store miss results in a line fill (SWOA).
   - Initialize the DCWR to select copy-back or write-through caching.
   - Initialize the DCCR to configure data cachability.
4. Before allowing interrupts (synchronous or asynchronous):
   - Initialize the EVPR to point to vector table.
   - Provide vector table with branches to interrupt handlers.
5. Before enabling asynchronous interrupts:
   - Initialize timer facilities.
   - Initialize MSR to enable appropriate interrupts.
6. Initialize other processor features, such as the MMU, debug, and trace.
7. Initialize non-processor resources.
   - Initialize system memory as required by the operating system or application code.
   - Initialize off-chip system facilities.
8. Start the execution of operating system or application code.

8.12 Initialization Code Example

The following initialization code illustrates the steps that should be taken to initialize the processor before an operating system or user programs begin execution. The example is presented in pseudo-code; function calls are named similarly to PPC405GP mnemonics where appropriate.

```
[/* ---------------------------------------------------------------------------- */
/*    PPC405GP Initialization Pseudo Code                                        */
/* ----------------------------------------------------------------------------- */
@0xFFFFFFFC:                       /* initial instruction fetch from 0xFFFFFFFC   */
  ba(init_code);                   /* branch to initialization code               */

@init_code:

  /*--------------------------------------------------------------------------- */
  /* Configure guarded attribute for performance.                               */
  /*--------------------------------------------------------------------------- */

     mtspr(SGR, guarded_attribute);

  /*--------------------------------------------------------------------------- */
  /* Configure endianness and compression.                                      */
  /*--------------------------------------------------------------------------- */

     mtspr(SLER, endianness);
     mtspr(SU0R, compression_attribute);

  /*------------------------------------------------------- */
  /* Invalidate the instruction cache and enable cachability */
  /*------------------------------------------------------- */

  iccci;                           /* invalidate i-cache */
     mtspr(ICCR, i_cache_cachability);                  /* enable I-cache*/
  isync;

  /*--------------------------------------------------------------------------- */
  /* Invalidate the data cache and enable cachability                           */
  /*--------------------------------------------------------------------------- */

  address = 0;                     /* start at first line                         */
  for (line = 0; line <m_lines; line++)  /* D-cache has m_lines congruence classes */
  {
     dccci(address);               /* invalidate congruence class                 */
     address += 32;                /* point to the next congruence class          */
  }
  mtspr(CCR0, store-miss_line-fill);
  mtspr(DCWR, copy-back_write-thru);
  mtspr(DCCR, d_cache_cachability);                /* enable D-cache              */
  isync;

  /*--------------------------------------------------------------------------- */
  /* Prepare system for synchronous interrupts.                                 */
  /*--------------------------------------------------------------------------- */

  mtspr(EVPR, prefix_addr);        /* initialize exception vector prefix          */
```

```
/* Initialize vector table and interrupt handlers if not already done */

/* ------------------------------------------------------------------ */
/* Prepare system for asynchronous interrupts.                         */
/* ------------------------------------------------------------------ */

/* Initialize and configure timer facilities                           */

mtspr(PIT, 0);                /* clear PIT so no PIT indication after TSR cleared*/
mtspr(TSR, 0xFFFFFFFF);       /* clear TSR                                      */
mtspr(TCR, timer_enable);     /* enable desired timers                          */
mtspr(TBL, 0);                /* reset time base low first to avoid ripple      */
mtspr(TBU, time_base_u);      /* set time base, hi first to catch possible ripple */
mtspr(TBL, time_base_l);      /* set time base, low                             */
mtspr(PIT, pit_count);        /* set desired PIT count                          */

/* Initialize the MSR                                                  */

/*  Exceptions must be enabled immediately after timer facilities to avoid missing a */
/*  timer exception.                                                   */
/*                                                                     */
/* The MSR also controls privileged/user mode, translation, and the wait state.  */
/* These must be initialized by the operating system or application code.        */
/* If enabling translation, code must initialize the TLB.              */
/* ------------------------------------------------------------------ */

mtmsr(machine_state);

/* ------------------------------------------------------------------ */
/* Initialization of other processor facilities should be performed at this time. */
/* ------------------------------------------------------------------ */

/* ------------------------------------------------------------------ */
/* Initialization of non-processor facilities should be performed at this time.  */
/* ------------------------------------------------------------------ */

/* ------------------------------------------------------------------ */
/* Branch to operating system or application code can occur at this time.        */
/* ------------------------------------------------------------------ */
```

Chapter 9. Pin Strapping and Sharing

9.1 Pin Strapping

When power is applied to the PPC405GP, a start-up process is initiated in which internal functions are initialized. Some of these functions have optional choices. Which of the options are used for initialization is determined by the way a specific set of I/O pins (balls) are conditioned. The conditioning is achieved using external pull-up (indicated as 1) or pull-down (indicated as 0) resistors connected to the pins.

While the $\overline{\text{SysReset}}$ input signal is low (system reset), the state of the I/O pins is read to enable default initial conditions before PPC405GP start-up. The actual capture instant is the nearest SysClk clock edge before the deassertion of $\overline{\text{SysReset}}$. The state of the pins as read is stored in the Chip Pin Strapping Register (CPC0_PSR) shown in Figure 9-1. Refer to *PowerPC 405GP Embedded Processor Data Sheet*, which describes the strapping pins.

9.1.1 Chip Pin Strapping Register (CPC0_PSR)

CPC0_PSR contains the state of the strapping pins as read during system reset.

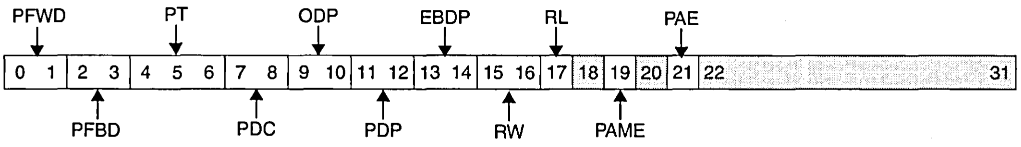


Figure 9-1. Chip Pin Strapping Register (CPC0_PSR)

0:1	PFWD	PLL Forward Divider 00 Bypass Mode 01 Divide by 3 10 Divide by 4 11 Divide by 6
2:3	PFBD	PLL Feedback Divider 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4

4:6	PT	PLL Tuning 000 Choice 1; TUNE[5:0] = 010001 001 Choice 2; TUNE[5:0] = 010010 010 Choice 3; TUNE[5:0] = 010011 011 Choice 4; TUNE[5:0] = 010100 100 Choice 5; TUNE[5:0] = 010101 101 Choice 6; TUNE[5:0] = 010110 110 Choice 7; TUNE[5:0] = 010111 111 Choice 8; TUNE[5:0] = 100100	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL implemented in the PPC405GP. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs) with the PPC405GP, visit the technical documents area of the IBM PowerPC web site.
7:8	PDC	PLB Divider from CPU 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
9:10	ODP	OPB Divider from PLB 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
11:12	PDP	PCI Divider from PLB 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
13:14	EBDP	External Bus Divider from PLB 00 Divide By 2 01 Divide By 3 10 Divide By 4 11 Divide By 5	
15:16	RW	ROM Width 00 8-bit ROM 01 16-bit ROM 10 32-bit ROM 11 Reserved	
17	RL	ROM Location 0 405GP Peripheral Attach 1 405GP PCI Attach	
18		Reserved	
19	PAME	PCI Asynchronous Mode Enable 0 Synchronous PCI Mode 1 Asynchronous Mode	
20		Reserved	
21	PAE	PCI Arbiter Enable 0 Internal Arbiter Disabled 1 Internal Arbiter Enabled	
22:31		Reserved	

9.2 Pin Sharing

The PPC405GP uses pin (ball) multiplexing (sharing) to reduce the total pin requirement without significantly reducing functionality. Some of the pins that are multiplexed require DCR register programming to configure the pin for the desired function. It is expected that in an application, a particular pin is programmed to serve one function. While nothing prevents changing the function of a pin during operation, most applications configure a pin once at power-on reset (POR). Table 9-1 lists the multiplexed PPC405GP signals and indicates the DCR bit that controls the pin. The default signal appears first and the alternate signal is in brackets.

Table 9-1. Multiplexed Pins

Signal	DCR Bit	Description
GPIO1[TS1E]	CPC0_CR0[TRE]	Set of GPIO pins that can be reconfigured for use as the CPU Trace interface.
GPIO2[TS2E]		
GPIO3[TS1O]		
GPIO4[TS2O]		
GPIO5[TS3]		
GPIO6[TS4]		
GPIO7[TS5]		
GPIO8[TS6]		
GPIO9[TrcClk]		
$\overline{\text{PerCS1}}$[GPIO10]	CPC0_CR0[G10E]	Peripheral Chip selects that can be reconfigured for use as GPIOs.
$\overline{\text{PerCS2}}$[GPIO11]	CPC0_CR0[G11E]	
$\overline{\text{PerCS3}}$[GPIO12]	CPC0_CR0[G12E]	
$\overline{\text{PerCS4}}$[GPIO13]	CPC0_CR0[G13E]	
$\overline{\text{PerCS5}}$[GPIO14]	CPC0_CR0[G14E]	
$\overline{\text{PerCS6}}$[GPIO15]	CPC0_CR0[G15E]	
$\overline{\text{PerCS7}}$[GPIO16]	CPC0_CR0[G16E]	
IRQ0[GPIO17]	CPC0_CR0[G17E]	External Interrupts that can be reconfigured for use as GPIOs.
IRQ1[GPIO18]	CPC0_CR0[G18E]	
IRQ2[GPIO19]	CPC0_CR0[G19E]	
IRQ3[GPIO20]	CPC0_CR0[G20E]	
IRQ4[GPIO21]	CPC0_CR0[G21E]	
IRQ5[GPIO22]	CPC0_CR0[G22E]	
IRQ6[GPIO23]	CPC0_CR0[G23E]	

Table 9-1. Multiplexed Pins (continued)

Signal	DCR Bit	Description
$\overline{\text{UART1\_DSR}}$[$\overline{\text{UART1\_CTS}}$][1]	CPC0_CR0[DCS]	UART1 DSR pin that can be reconfigured for use as UART1 CTS.
$\overline{\text{UART1\_RTS}}$[$\overline{\text{UART1\_DTR}}$][1]	CPC0_CR0[RDS]	UART1 RTS pin that can be reconfigured for use as UART1 DTR.
$\overline{\text{PCIINT}}$[$\overline{\text{PerWE}}$]	CPC0_CR1[PCIPW]	PCI Interrupt output that can be reconfigured for use by peripherals as a Write Byte Enable (logical AND of the four $\overline{\text{PerWBE0:3}}$ write byte enables).
$\overline{\text{PCIReq0}}$[$\overline{\text{Gnt}}$]	CPC0_PSR[PAE]	$\overline{\text{PCIReq0}}$ when internal arbiter is used or $\overline{\text{Gnt}}$ when external arbiter is used.
$\overline{\text{PCIGnt0}}$[$\overline{\text{Req}}$]	CPC0_PSR[PAE]	$\overline{\text{PCIGnt0}}$ when internal arbiter is used or $\overline{\text{Req}}$ when external arbiter is used.
Note: Typically DSR and DTR are paired and CTS and RTS are paired. With the current multiplex defaults one of the pins must be changed during initialization to achieve the typical pairings.		

Chapter 10. Interrupt Controller Operations

The PPC405GP contains a universal interrupt controller (UIC) that provides all necessary control, status, and communication between the various internal and external interrupt sources and the processor core.

10.1 UIC Overview

The UIC supports 19 internal interrupts and 7 external interrupts. Status reporting (using the UIC Status Register [UIC0_SR]) is provided to ensure that systems software can determine the current and interrupting state of the system and respond appropriately. Software can generate interrupts to simplify software development and for diagnostics.

The interrupts can be programmed, using the UIC Critical Register (UIC0_CR), to generate either a critical or a non-critical interrupt signal to the processor core.

The privileged **mtdcr** and **mfdcr** instructions, which are used by system software, are used to read and write the UIC registers.

An optional critical interrupt vector generator can reduce interrupt handling latency for critical interrupts. Vector calculation is described in detail in "UIC Vector Register (UIC0_VR)" on page 10-18.

10.2 UIC Features

- Support for 19 internal and 7 external interrupts
- Support for asychronous level- or edge-sensitive interrupt types
- Programmable polarity for all interrupt types
- Programmable critical/non-critical interrupt selection for each interrupt bit
- Prioritized critical interrupt vector generation
- A UIC Status Register (UIC0_SR) providing the following information:
  - Current state of interrupts
  - Current state of all enabled interrupts (those masked using the UIC Enable Register (UIC0_ER))

10.3 UIC Interrupt Assignments

The UIC supports various internal and external interrupt sources as shown in Table 10-1.

Table 10-1. UIC Interrupt Assignments

Interrupt	Polarity	Sensitivity	Interrupt Source
0	High	Level	UART0
1	High	Level	UART1
2	High	Level	IIC

Table 10-1. UIC Interrupt Assignments (continued)

Interrupt	Polarity	Sensitivity	Interrupt Source
3	High	Edge	External Master
4	High	Level	PCI
5	High	Level	DMA Channel 0
6	High	Level	DMA Channel 1
7	High	Level	DMA Channel 2
8	High	Level	DMA Channel 3
9	High	Level	Ethernet Wake Up
10	High	Level	MAL System Error (SERR)
11	High	Level	MAL TX End of Buffer (TXEOB0)
12	High	Level	MAL RX End of Buffer (RXEOB)
13	High	Level	MAL TX Descriptor Error (TXDE)
14	High	Level	MAL RX Descriptor Error (RXDE)
15	High	Level	Ethernet
16	High	Level	External PCI SERR
17	High	Level	ECC Correctable Error
18	High	Level	PCI Power Management
19	Reserved		
20			
21			
22			
23			
24			
25	Programmable	Programmable	External IRQ 0
26	Programmable	Programmable	External IRQ 1
27	Programmable	Programmable	External IRQ 2
28	Programmable	Programmable	External IRQ 3
29	Programmable	Programmable	External IRQ 4
30	Programmable	Programmable	External IRQ 5
31	Programmable	Programmable	External IRQ 6

10.4 Interrupt Programmability

The on-chip interrupts (interrupts 0–18) and the external IRQs (interrupts 25–31) are programmable. However, the polarity and sensitivity of the on-chip interrupts must be programmed as shown in Table 10-1, using the UIC Polarity Register (UIC0_PR) and UIC Trigger Register (UIC0_TR), respectively.

10.5 UIC Registers

The UIC includes the Device Control Registers (DCRs) listed in Table 10-2.

The registers are accessed using the **mfdcr** and **mtdcr** instructions.

Table 10-2. UIC DCRs

Mnemonic	Register	Address	Access	Page
UIC0_SR	UIC Status Register	0x0C0	Read/Clear	10-3
UIC0_ER	UIC Enable Register	0x0C2	R/W	10-6
UIC0_CR	UIC Critical Register	0x0C3	R/W	10-8
UIC0_PR	UIC Polarity Register	0x0C4	R/W	10-10
UIC0_TR	UIC Trigger Register	0x0C5	R/W	10-13
UIC0_MSR	UIC Masked Status Register	0x0C6	Read-only	10-16
UIC0_VR	UIC Vector Register	0x0C7	Write-only	10-19
UIC0_VCR	UIC Vector Configuration Register	0x0C8	Read-only	10-18

10.5.1 UIC Status Register (UIC0_SR)

To report interrupt status, the UIC0_SR fields capture and hold internal and external interrupts until the fields are intentionally reset. To reset a field, write 1 to the field.

The values of other UIC registers do not affect UIC0_SR fields.

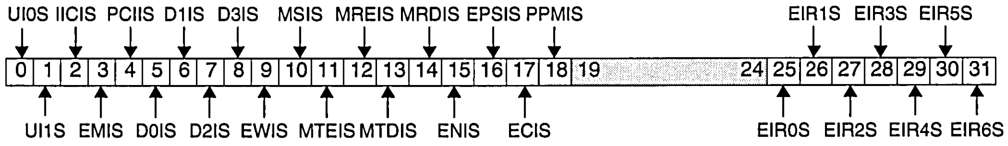


Figure 10-1. UIC Status Register (UIC0_SR)

0	U0IS	UART0 Interrupt Status 0 A UART0 interrupt has not occurred. 1 A UART0 interrupt occurred.
1	U1IS	UART1 Interrupt Status 0 A UART1 interrupt has not occurred. 1 A UART1 interrupt occurred.

2	IICIS	IIC Interrupt Status 0 An IIC interrupt has not occurred. 1 An IIC interrupt occurred.
3	EMIS	External Master Interrupt Status 0 An external master interrupt has not occurred. 1 An external master interrupt occurred.
4	PCIIS	PCI Interrupt Status 0 A PCI interrupt has not occurred. 1 A PCI interrupt occurred.
5	D0IS	DMA Channel 0 Interrupt Status 0 A DMA channel 0 interrupt has not occurred. 1 A DMA channel 0 interrupt occurred.
6	D1IS	DMA Channel 1 Interrupt Status 0 A DMA channel 1 interrupt has not occurred. 1 A DMA channel 1 interrupt occurred.
7	D2IS	DMA Channel 2 Interrupt Status 0 A DMA channel 2 interrupt has not occurred. 1 A DMA channel 2 interrupt occurred.
8	D3IS	DMA Channel 3 Interrupt Status 0 A DMA channel 3 interrupt has not occurred. 1 A DMA channel 3 interrupt occurred.
9	EWIS	Ethernet Wake-up Interrupt Status 0 An Ethernet wake-up interrupt has not occurred. 1 An Ethernet wake-up interrupt occurred.
10	MSIS	MAL SERR Interrupt Status 0 A MAL SERR interrupt has not occurred. 1 A MAL SERR interrupt occurred.
11	MTEIS	MAL TX EOB Interrupt Status 0 A MAL TX EOB interrupt has not occurred. 1 A MAL TX EOB interrupt occurred.
12	MREIS	MAL RX EOB Interrupt Status 0 A MAL RX EOB interrupt has not occurred. 1 A MAL RX EOB interrupt occurred.
13	MTDIS	MAL TX DE Interrupt Status 0 A MAL TX DE interrupt has not occurred. 1 A MAL TX DE interrupt occurred.
14	MRDIS	MAL RX DE Interrupt Status 0 A MAL RX DE interrupt has not occurred. 1 A MAL RX DE interrupt occurred.

15	EIS	Ethernet Interrupt Status 0 An Ethernet interrupt has not occurred. 1 An Ethernet interrupt occurred.
16	EPSIS	External PCI SERR Interrupt Status 0 An external PCI SERR interrupt has not occurred. 1 An external PCI SERR interrupt occurred.
17	ECIS	ECC Correctable Error Interrupt Status 0 An ECC correctable error interrupt did not occur. 1 An ECC correctable error interrupt occurred.
18	PPMIS	PCI Power Management Interrupt Status 0 A PCI power management interrupt did not occur. 1 A PCI power management interrupt occurred.
19:24	▨	Reserved
25	EIR0S	External IRQ 0 Status 0 An external IRQ 0 interrupt has not occurred. 1 An external IRQ 0 interrupt occurred.
26	EIR1S	External IRQ 1 Status 0 An external IRQ 1 interrupt has not occurred. 1 An external IRQ 1 interrupt occurred.
27	EIR2S	External IRQ 2 Status 0 An external IRQ 2 interrupt has not occurred. 1 An external IRQ 2 interrupt occurred.
28	EIR3S	External IRQ 3 Status 0 An external IRQ 3 interrupt has not occurred. 1 An external IRQ 3 interrupt occurred.
29	EIR4S	External IRQ 4 Status 0 An external IRQ 4 interrupt has not occurred. 1 An external IRQ 4 interrupt occurred.
30	EIR5S	External IRQ 5 Status 0 An external IRQ 5 interrupt has not occurred. 1 An external IRQ 5 interrupt occurred.
31	EIR6S	External IRQ 6 Status 0 An external IRQ 6 interrupt has not occurred. 1 An external IRQ 6 interrupt occurred.

10.5.2 UIC Enable Register (UIC0_ER)

The fields of the UIC0_ER, which correspond to the fields of the UIC0_SR, enable or disable the reporting of the corresponding fields of the UIC0_SR.

If a UIC0_ER field is set to 1, the corresponding field of the UIC0_SR generates a critical or non-critical interrupt signal to the processor core, if the UIC0_SR field is set to 1. If a UIC0_ER field is set to 0, the corresponding field of the UIC0_SR does not generate a critical or non-critical interrupt signal to the processor core, regardless of the setting of the UIC0_SR field. The critical and non-critical interrupt signals in the processor core are controlled by fields in the Machine State Register (MSR).

The class of generated signals (critical or non-critical) is controlled by the UIC0_CR.

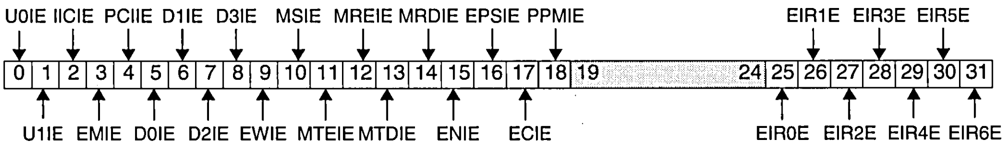


Figure 10-2. UIC Enable Register (UIC0_ER)

0	U0IE	UART0 Interrupt Enable 0 UART0 interrupt is disabled. 1 UART0 interrupt is enabled.
1	U1IE	UART1 Interrupt Enable 0 UART1 interrupt is disabled. 1 UART1 interrupt is enabled.
2	IICIE	IIC Interrupt Enable 0 IIC interrupt is disabled. 1 IIC interrupt is enabled.
3	EMIE	External Master Interrupt Enable 100 External master interrupt is disabled. 1 0xxExternal master interrupt is enabled.
4	PCIIE	PCI Interrupt Enable 0 PCI interrupt is disabled. 1 PCI interrupt is enabled.
5	D0IE	DMA Channel 0 Interrupt Enable 0 DMA channel 0 interrupt is disabled. 1 DMA channel 0 interrupt is enabled.
6	D1IE	DMA Channel 1 Interrupt Enable 0 DMA channel 1 interrupt is disabled. 1 DMA channel 1 interrupt is enabled.
7	D2IE	DMA Channel 2 Interrupt Enable 0 DMA channel 2 interrupt is disabled. 1 DMA channel 2 interrupt is enabled.

8	D3IE	DMA Channel 3 Interrupt Enable 0 DMA channel 3 interrupt is disabled. 1 DMA channel 3 interrupt is enabled.
9	EWIE	Ethernet Wake-up Interrupt Enable 0 Ethernet wake-up interrupt is disabled. 1 Ethernet wake-up interrupt is enabled.
10	MSIE	MAL SERR Interrupt Enable 0 MAL SERR interrupt is disabled. 1 MAL SERR interrupt is enabled.
11	MTEIE	MAL TX EOB Interrupt Enable 0 MAL TX EOB interrupt is disabled. 1 MAL TX EOB interrupt is enabled.
12	MREIE	MAL RX EOB Interrupt Enable 0 MAL RX EOB interrupt is disabled. 1 MAL RX EOB interrupt is enabled.
13	MTDIE	MAL TX DE Interrupt Enable 0 MAL TX DE interrupt is disabled. 1 MAL TX DE interrupt is enabled.
14	MRDIE	MAL RX DE Interrupt Enable 0 MAL RX DE interrupt is disabled. 1 MAL RX DE interrupt is enabled.
15	EIE	Ethernet Interrupt Enable 0 An Ethernet interrupt is disabled. 1 An Ethernet interrupt is enabled.
16	EPSIE	External PCI SERR Interrupt Enable 0 External PCI SERR interrupt is disabled. 1 External PCI SERR interrupt is enabled.
17	ECIE	ECC Correctable Error Interrupt Enable 0 ECC correctable error interrupt is disabled. 1 ECC correctable error interrupt is enabled.
18	PPMI	PCI Power management Interrupt Enable 0 PCI power management interrupt is disabled. 1 PCI power management interrupt is enabled.
19:24		Reserved
25	EIR0E	External IRQ 0 Enable 0 An external IRQ 0 interrupt is disabled. 1 An external IRQ 0 interrupt is enabled.
26	EIR1E	External IRQ 1 Enable 0 An external IRQ 1 interrupt is disabled. 1 An external IRQ 1 interrupt is enabled.
27	EIR2E	External IRQ 2 Enable 0 An external IRQ 2 interrupt is disabled. 1 An external IRQ 2 interrupt is enabled.

28	EIR3E	External IRQ 3 Enable 0 An external IRQ 3 interrupt is disabled. 1 An external IRQ 3 interrupt is enabled.
29	EIR4E	External IRQ 4 Enable 0 An external IRQ 4 interrupt is disabled. 1 An external IRQ 4 interrupt is enabled.
30	EIR5E	External IRQ 5 Enable 0 An external IRQ 5 interrupt is disabled. 1 An external IRQ 5 interrupt is enabled.
31	EIR6E	External IRQ 6 Enable 0 An external IRQ 6 interrupt is disabled. 1 An external IRQ 6 interrupt is enabled.

10.5.3 UIC Critical Register (UIC0_CR)

The fields of the UIC0_CR, which correspond to the fields of the UIC0_SR and UIC0_ER, determine whether an interrupt captured in the corresponding fields of the UIC0_SR generates a non-critical or critical interrupt, if the interrupts are enabled in the corresponding fields of the UIC0_ER. The processor handles critical interrupts when MSR[EE] = 1 and non-critical interrupts when MSR[CE]=1.

If a UIC0_CR field is set to 0, an enabled interrupt (captured in the corresponding field of the UIC0_SR and enabled in the corresponding field of the UIC0_ER) generates a non-critical interrupt signal to the processor core. If a UIC0_CR field is a 1, a critical interrupt signal is generated.

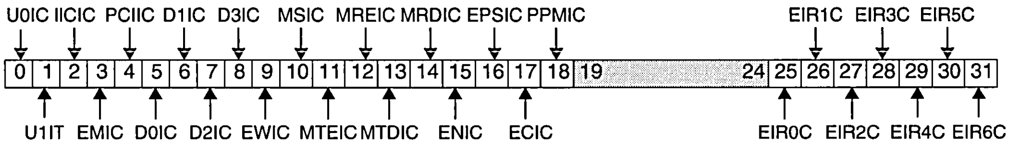


Figure 10-3. UIC Critical Register (UIC0_CR)

0	U0IC	UART0 Interrupt Class 0 UART0 interrupt is non-critical. 1 UART0 interrupt is critical.
1	U1IC	UART1 Interrupt Class 0 UART1 interrupt is non-critical. 1 UART1 interrupt is critical.
2	IICIC	IIC Interrupt Class 0 IIC interrupt is non-critical. 1 IIC interrupt is critical.
3	EMIC	External Master Interrupt Class 0 External master interrupt is non-critical. 1 External master interrupt is critical.
4	PCIIC	PCI Interrupt Class 0 PCI interrupt is non-critical. 1 PCI interrupt is critical.

5	D0IC	DMA Channel 0 Interrupt Class 0 DMA channel 0 interrupt is non-critical. 1 DMA channel 0 interrupt is critical.
6	D1IC	DMA Channel 1 Interrupt Class 0 DMA channel 1 interrupt is non-critical. 1 DMA channel 1 interrupt is critical.
7	D2IC	DMA Channel 2 Interrupt Class 0 DMA channel 2 interrupt is non-critical. 1 DMA channel 2 interrupt is critical.
8	D3IC	DMA Channel 3 Interrupt Class 0 DMA channel 3 interrupt is non-critical. 1 DMA channel 3 interrupt is critical.
9	EWIC	Ethernet Wake-up Interrupt Class 0 Ethernet wake-up interrupt is non-critical. 1 Ethernet wake-up interrupt is critical.
10	MSIC	MAL SERR Interrupt Class 0 MAL SERR interrupt is non-critical. 1 MAL SERR interrupt is critical.
11	MTEIC	MAL TX EOB Interrupt Class 0 MAL TX EOB interrupt is non-critical. 1 MAL TX EOB interrupt is critical.
12	MREIC	MAL RX EOB Interrupt Class 0 MAL RX EOB interrupt is non-critical. 1 MAL RX EOB interrupt is critical.
13	MTDIC	MAL TX DE Interrupt Class 0 MAL TX DE interrupt is non-critical. 1 MAL TX DE interrupt is critical.
14	MRDIC	MAL RX DE Interrupt Class 0 MAL RX DE interrupt is non-critical. 1 MAL RX DE interrupt is critical.
15	EIC	Ethernet Interrupt Class 0 An Ethernet interrupt is non-critical. 1 An Ethernet interrupt is critical.
16	EPSIC	External PCI SERR Interrupt Class 0 External PCI SERR interrupt is non-critical. 1 External PCI SERR interrupt is critical.
17	ECIC	ECC Correctable Error Interrupt Class 0 ECC correctable error interrupt is non-critical. 1 ECC correctable error interrupt is critical.
18	PPMIC	PCI Power management Interrupt Class 0 PCI power management interrupt is non-critical. 1 PCI power management interrupt is critical.
19:24		Reserved

25	EIR0C	External IRQ 0 Class 0 An external IRQ 0 interrupt is non-critical. 1 An external IRQ 0 interrupt is critical.
26	EIR1C	External IRQ 1 Class 0 An external IRQ 1 interrupt is non-critical. 1 An external IRQ 1 interrupt is critical.
27	EIR2C	External IRQ 2 Class 0 An external IRQ 2 interrupt is non-critical. 1 An external IRQ 2 interrupt is critical.
28	EIR3C	External IRQ 3 Class 0 An external IRQ 3 interrupt is non-critical. 1 An external IRQ 3 interrupt is critical.
29	EIR4C	External IRQ 4 Class 0 An external IRQ 4 interrupt is non-critical. 1 An external IRQ 4 interrupt is critical.
30	EIR5C	External IRQ 5 Class 0 An external IRQ 5 interrupt is non-critical. 1 An external IRQ 5 interrupt is critical.
31	EIR6C	External IRQ 6 Class 0 An external IRQ 6 interrupt is non-critical. 1 An external IRQ 6 interrupt is critical.

10.5.4 UIC Polarity Register (UIC0_PR)

The fields of the UIC0_PR, which correspond to the fields of the UIC0_SR, determine whether the corresponding fields in the UIC0_SR have a positive or negative polarity.

For level-sensitive interrupts, a 0 in a UIC0_PR field causes the corresponding interrupt to be negative active. A 1 in a UIC0_PR field causes the corresponding interrupt to be positive active.

For edge-sensitive interrupts, a 0 in a UIC0_PR field causes the corresponding interrupt to be detected on a falling edge (as polarity changes from 1 to 0). A 1 in a UIC0_PR field causes the corresponding interrupt to be detected on a rising edge (as polarity changes from 0 to 1).

Because the on-chip interrupts (those controlled by UIC0_PR$_{0:18}$) have positive polarity, the associated fields must be set to 1.

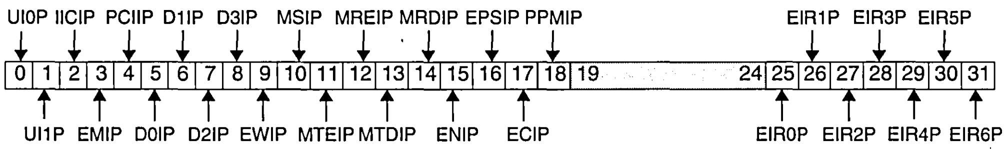


Figure 10-4. UIC Polarity Register (UIC0_PR)

0	U0IP	UART0 Interrupt Polarity 0 UART0 interrupt has negative polarity. 1 UART0 interrupt has positive polarity.	Must be set to 1.
1	U1IP	UART1 Interrupt Polarity 0 UART1 interrupt has negative polarity. 1 UART1 interrupt has positive polarity.	Must be set to 1.
2	IICIP	IIC Interrupt Polarity 0 IIC interrupt has negative polarity. 1 IIC interrupt has positive polarity.	Must be set to 1.
3	EMIP	External Master Interrupt Polarity 0 External master interrupt has negative polarity. 1 External master interrupt has positive polarity.	Must be set to 1.
4	PCIIP	PCI Interrupt Polarity 0 PCI interrupt has negative polarity. 1 PCI interrupt has positive polarity.	Must be set to 1.
5	D0IP	DMA Channel 0 Interrupt Polarity 0 DMA channel 0 interrupt has negative polarity. 1 DMA channel 0 interrupt has positive polarity.	Must be set to 1.
6	D1IP	DMA Channel 1 Interrupt Polarity 0 DMA channel 1 interrupt has negative polarity. 1 DMA channel 1 interrupt has positive polarity.	Must be set to 1.
7	D2IP	DMA Channel 2 Interrupt Polarity 0 DMA channel 2 interrupt has negative polarity. 1 DMA channel 2 interrupt has positive polarity.	Must be set to 1.
8	D3IP	DMA Channel 3 Interrupt Polarity 0 DMA channel 3 interrupt has negative polarity. 1 DMA channel 3 interrupt has positive polarity.	Must be set to 1.
9	EWIP	Ethernet Wake-up Interrupt Polarity 0 Ethernet wake-up interrupt has negative polarity. 1 Ethernet wake-up interrupt has positive polarity.	Must be set to 1.
10	MSIP	MAL SERR Interrupt Polarity 0 MAL SERR interrupt has negative polarity. 1 MAL SERR interrupt has positive polarity.	Must be set to 1.
11	MTEIP	MAL TX EOB Interrupt Polarity 0 MAL TX EOB interrupt has negative polarity. 1 MAL TX EOB interrupt has positive polarity.	Must be set to 1.

12	MREIP	MAL RX EOB Interrupt Polarity 0 MAL RX EOB interrupt has negative polarity. 1 MAL RX EOB interrupt has positive polarity.	Must be set to 1.
13	MTDIP	MAL TX DE Interrupt Polarity 0 MAL TX DE interrupt has negative polarity. 1 MAL TX DE interrupt has positive polarity.	Must be set to 1.
14	MRDIP	MAL RX DE Interrupt Polarity 0 MAL RX DE interrupt has negative polarity. 1 MAL RX DE interrupt has positive polarity.	Must be set to 1.
15	EIP	Ethernet Interrupt Polarity 0 An Ethernet interrupt has negative polarity. 1 An Ethernet interrupt has positive polarity.	Must be set to 1.
16	EPSIP	External PCI SERR Interrupt Polarity 0 External PCI SERR interrupt has negative polarity. 1 External PCI SERR interrupt has positive polarity.	Must be set to 1.
17	ECIP	ECC Correctable Error Interrupt Polarity 0 ECC correctable error interrupt has negative polarity. 1 ECC correctable error interrupt has positive polarity.	Must be set to 1.
18	PPMIC	PCI Power management Interrupt Class 0 PCI power management interrupt has negative polarity. 1 PCI power management interrupt has positive polarity.	Must be set to 1.
19:24		Reserved	
25	EIR0P	External IRQ 0 Polarity 0 An external IRQ 0 interrupt has negative polarity. 1 An external IRQ 0 interrupt has positive polarity.	
26	EIR1P	External IRQ 1 Polarity 0 An external IRQ 1 interrupt has negative polarity. 1 An external IRQ 1 interrupt has positive polarity.	
27	EIR2P	External IRQ 2 Polarity 0 An external IRQ 2 interrupt has negative polarity. 1 An external IRQ 2 interrupt has positive polarity.	
28	EIR3P	External IRQ 3 Polarity 0 An external IRQ 3 interrupt has negative polarity. 1 An external IRQ 3 interrupt has positive polarity.	
29	EIR4P	External IRQ 4 Polarity 0 An external IRQ 4 interrupt has negative polarity. 1 An external IRQ 4 interrupt has positive polarity.	
30	EIR5P	External IRQ 5 Polarity 0 An external IRQ 5 interrupt has negative polarity. 1 An external IRQ 5 interrupt has positive polarity.	

31	EIR6P	External IRQ 6 Polarity 0 An external IRQ 6 interrupt has negative polarity. 1 An external IRQ 6 interrupt has positive polarity.

10.5.5 UIC Trigger Register (UIC0_TR)

The fields of the UIC0_TR, which correspond to the fields of the UIC0_SR, determine whether corresponding fields in the UIC0_SR are edge-sensitive or level-sensitive.

Edge-sensitive interrupts are triggered depending on whether the associated interrupt signal is rising or falling (changing from 0 to 1 or 1 to 0, respectively). Whether a rising or falling edge causes the trigger is controlled by bits in the UIC0_PR.

Level-sensitive interrupts are triggered depending on whether the associated interrupt signal is high (1) or low (0).

If a UIC0_TR field is 0, the associated interrupt is level-sensitive. If the UIC0_TR field is 1, the interrupt is edge-sensitive. Because the on-chip interrupts (those controlled by UIC0_TR$_{0:2,\ 4:18}$) are level-sensitive, the associated fields must be set to 0. The external master interrupt trigger (controlled by UIC0_TR$_3$) is edge-sensitive; this field must be set to 1.

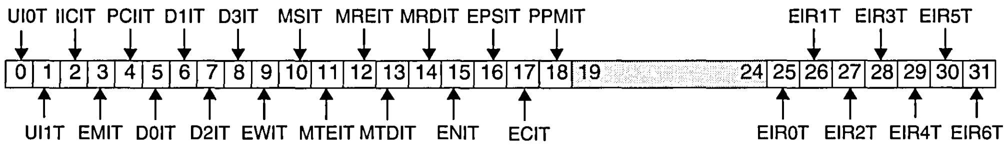


Figure 10-5. UIC Trigger Register (UIC0_TR)

0	U0IT	UART0 Interrupt Trigger 0 UART0 interrupt is level-sensitive. 1 UART0 interrupt is edge-sensitive.	Must be set to 0.
1	U1IT	UART1 Interrupt Trigger 0 UART1 interrupt is level-sensitive. 1 UART1 interrupt is edge-sensitive.	Must be set to 0.
2	IICIT	IIC Interrupt Trigger 0 IIC interrupt is level-sensitive. 1 IIC interrupt is edge-sensitive.	Must be set to 0.
3	EMIT	External Master Interrupt Trigger 0 External master interrupt is level-sensitive. 1 External master interrupt is edge-sensitive.	Must be set to 1.
4	PCIIT	PCI Interrupt Trigger 0 PCI interrupt is level-sensitive. 1 PCI interrupt is edge-sensitive.	Must be set to 0.

5	D0IT	DMA Channel 0 Interrupt Trigger 0 DMA channel 0 interrupt is level-sensitive. 1 DMA channel 0 interrupt is edge-sensitive.	Must be set to 0.
6	D1IT	DMA Channel 1 Interrupt Trigger 0 DMA channel 1 interrupt is level-sensitive. 1 DMA channel 1 interrupt is edge-sensitive.	Must be set to 0.
7	D2IT	DMA Channel 2 Interrupt Trigger 0 DMA channel 2 interrupt is level-sensitive. 1 DMA channel 2 interrupt is edge-sensitive.	Must be set to 0.
8	D3IT	DMA Channel 3 Interrupt Trigger 0 DMA channel 3 interrupt is level-sensitive. 1 DMA channel 3 interrupt is edge-sensitive.	Must be set to 0.
9	EWIT	Ethernet Wake-up Interrupt Trigger 0 Ethernet wake-up interrupt is level-sensitive. 1 Ethernet wake-up interrupt is edge-sensitive.	Must be set to 0.
10	MSIT	MAL SERR Interrupt Trigger 0 MAL SERR interrupt is level-sensitive. 1 MAL SERR interrupt is edge-sensitive.	Must be set to 0.
11	MTEIT	MAL TX EOB Interrupt Trigger 0 MAL TX EOB interrupt is level-sensitive. 1 MAL TX EOB interrupt is edge-sensitive.	Must be set to 0.
12	MREIT	MAL RX EOB Interrupt Trigger 0 MAL RX EOB interrupt is level-sensitive. 1 MAL RX EOB interrupt is edge-sensitive.	Must be set to 0.
13	MTDIT	MAL TX DE Interrupt Trigger 0 MAL TX DE interrupt is level-sensitive. 1 MAL TX DE interrupt is edge-sensitive.	Must be set to 0.
14	MRDIT	MAL RX DE Interrupt Trigger 0 MAL RX DE interrupt is level-sensitive. 1 MAL RX DE interrupt is edge-sensitive.	Must be set to 0.
15	EIT	Ethernet Interrupt Trigger 0 An Ethernet interrupt is level-sensitive. 1 An Ethernet interrupt is edge-sensitive.	Must be set to 0.
16	EPSIT	External PCI SERR Interrupt Trigger 0 External PCI SERR interrupt is level-sensitive. 1 External PCI SERR interrupt is edge-sensitive.	Must be set to 0.

17	ECIT	ECC Correctable Error Interrupt Trigger 0 ECC correctable error interrupt is level-sensitive. 1 ECC correctable error interrupt is edge-sensitive.	Must be set to 0.
18	PPMIT	PCI Power management Interrupt Trigger 0 PCI power management interrupt is level-sensitive. 1 PCI power management interrupt is edge-sensitive.	Must be set to 0.
19:24		Reserved	
25	EIR0T	External IRQ 0 Trigger 0 An external IRQ 0 interrupt is level-sensitive. 1 An external IRQ 0 interrupt is edge-sensitive.	
26	EIR1T	External IRQ 1 Trigger 0 An external IRQ 1 interrupt is level-sensitive. 1 An external IRQ 1 interrupt is edge-sensitive.	
27	EIR2T	External IRQ 2 Trigger 0 An external IRQ 2 interrupt is level-sensitive. 1 An external IRQ 2 interrupt is edge-sensitive.	
28	EIR3T	External IRQ 3 Trigger 0 An external IRQ 3 interrupt is level-sensitive. 1 An external IRQ 3 interrupt is edge-sensitive.	
29	EIR4T	External IRQ 4 Trigger 0 An external IRQ 4 interrupt is level-sensitive. 1 An external IRQ 4 interrupt is edge-sensitive.	
30	EIR5T	External IRQ 5 Trigger 0 An external IRQ 5 interrupt is level-sensitive. 1 An external IRQ 5 interrupt is edge-sensitive.	
31	EIR6T	External IRQ 6 Trigger 0 An external IRQ 6 interrupt is level-sensitive. 1 An external IRQ 6 interrupt is edge-sensitive.	

10.5.6 UIC Masked Status Register (UIC0_MSR)

This read-only register contains the result of masking the UIC0_SR with the UIC0_ER. Reading this register, instead of the actual UIC0_SR, eliminates the need for software to read and apply the enable mask to the contents of the UIC0_SR to determine which enabled interrupt fields are active.

If an interrupt is configured as level-sensitive, and a clear is attempted on the UIC0_SR, the UIC0_SR field is not cleared if the incoming interrupt signal is at the asserted polarity. The interrupt signal must be reset before the UIC0_SR can be successfully cleared.

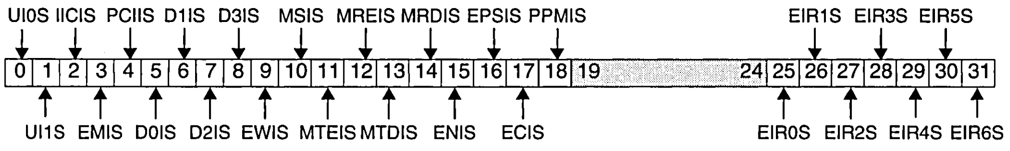


Figure 10-6. UIC Masked Status Register (UIC0_MSR)

0	U0IS	UART0 Masked Interrupt Status 0 A UART0 interrupt has not occurred. 1 A UART0 interrupt occurred.
1	U1IS	UART1 Masked Interrupt Status 0 A UART1 interrupt has not occurred. 1 A UART1 interrupt occurred.
2	IICIS	IIC Masked Interrupt Status 0 An IIC interrupt has not occurred. 1 An IIC interrupt occurred.
3	EMIS	External Master Masked Interrupt Status 0 An external master interrupt has not occurred. 1 An external master interrupt occurred.
4	PCIIS	PCI Masked Interrupt Status 0 A PCI interrupt has not occurred. 1 A PCI interrupt occurred.
5	D0IS	DMA Channel 0 Masked Interrupt Status 0 A DMA channel 0 interrupt has not occurred. 1 A DMA channel 0 interrupt occurred.
6	D1IS	DMA Channel 1 Masked Interrupt Status 0 A DMA channel 1 interrupt has not occurred. 1 A DMA channel 1 interrupt occurred.
7	D2IS	DMA Channel 2 Masked Interrupt Status 0 A DMA channel 2 interrupt has not occurred. 1 A DMA channel 2 interrupt occurred.

8	D3IS	DMA Channel 3 Masked Interrupt Status 0 A DMA channel 3 interrupt has not occurred. 1 A DMA channel 3 interrupt occurred.
9	EWIS	Ethernet Wake-up Masked Interrupt Status 0 An Ethernet wake-up interrupt has not occurred. 1 An Ethernet wake-up interrupt occurred.
10	MSIS	MAL SERR Masked Interrupt Status 0 A MAL SERR interrupt has not occurred. 1 A MAL SERR interrupt occurred.
11	MTEIS	MAL TX EOB Masked Interrupt Status 0 A MAL TX EOB interrupt has not occurred. 1 A MAL TX EOB interrupt occurred.
12	MREIS	MAL RX EOB Masked Interrupt Status 0 A MAL RX EOB interrupt has not occurred. 1 A MAL RX EOB interrupt occurred.
13	MTDIS	MAL TX DE Masked Interrupt Status 0 A MAL TX DE interrupt has not occurred. 1 A MAL TX DE interrupt occurred.
14	MRDIS	MAL RX DE Masked Interrupt Status 0 A MAL RX DE interrupt has not occurred. 1 A MAL RX DE interrupt occurred.
15	EIS	Ethernet Masked Interrupt Status 0 An Ethernet interrupt has not occurred. 1 An Ethernet interrupt occurred.
16	EPSIE	External PCI SERR Masked Interrupt Status 0 An external PCI SERR interrupt has not occurred. 1 An external PCI SERR interrupt occurred.
17	ECIS	ECC Correctable Error Masked Interrupt Status 0 An ECC correctable error interrupt did not occur. 1 An ECC correctable error interrupt occurred.
18	PPMIS	PCI Power Management Masked Interrupt Status 0 A PCI power management interrupt did not occur. 1 A PCI power management interrupt occurred.
19:24	▨	Reserved

25	EIR0E	External IRQ 0 Masked Status 0 An external IRQ 0 interrupt has not occurred. 1 An external IRQ 0 interrupt occurred.
26	EIR1S	External IRQ 1 Masked Status 0 An external IRQ 1 interrupt has not occurred. 1 An external IRQ 1 interrupt occurred.
27	EIR2S	External IRQ 2 Masked Status 0 An external IRQ 2 interrupt has not occurred. 1 An external IRQ 2 interrupt occurred.
28	EIR3S	External IRQ 3 Masked Status 0 An external IRQ 3 interrupt has not occurred. 1 An external IRQ 3 interrupt occurred.
29	EIR4S	External IRQ 4 Masked Status 0 An external IRQ 4 interrupt has not occurred. 1 An external IRQ 4 interrupt occurred.
30	EIR5S	External IRQ 5 Masked Status 0 An external IRQ 5 interrupt has not occurred. 1 An external IRQ 5 interrupt occurred.
31	EIR6S	External IRQ 6 Masked Status 0 An external IRQ 6 interrupt has not occurred. 1 An external IRQ 6 interrupt occurred.

10.5.7 UIC Vector Configuration Register (UIC0_VCR)

The write-only UIC0_VCR enables software control of interrupt vector generation for critical interrupts. UIC0_VCR contains an address, used as an interrupt vector base address, and specifies interrupt ordering priority. Vector generation is not performed for non-critical interrupts.

UIC0_VCR[VBA] can contain either the base address for an interrupt handler vector table or the base address for the interrupt handler associated with each interrupt. The actual interrupt vector (the address of the interrupt handler that services the interrupt) is generated in the UIC0_VR, using UIC0_VCR[VBA]. Vector generation is described in “UIC Vector Register (UIC0_VR)” on page 10-19. Because the two lowest-order bits of an interrupt handler address are assumed to be 00 to ensure word alignment, 30 bits are sufficient to form the base address.

A general interrupt handler uses the vector to access a table of interrupt vectors. Each interrupt vector table entry contains the address of an interrupt handler for a specific interrupt. Alternatively, UIC0_VCR[VBA] can directly address the interrupt handlers for specific interrupts, which in memory are separated by an offset calculated in UIC0_VR.

UIC0_VCR[PRO] controls whether the interrupt associated with UIC0_SR[0] or UIC0_SR[31] has the highest priority. If UIC0_VCR[PRO] = 0, the interrupt associated with UIC0_SR[0] has the highest priority; if UIC0_VCR[PRO] = 1, the interrupt associated with UIC0_SR[31] has the highest priority.

The bit closest to the highest priority field that is programmed in the UIC0_CR as a interrupt has the second highest priority. Priority decreases across the UIC0_SR to the end opposite the highest priority field.

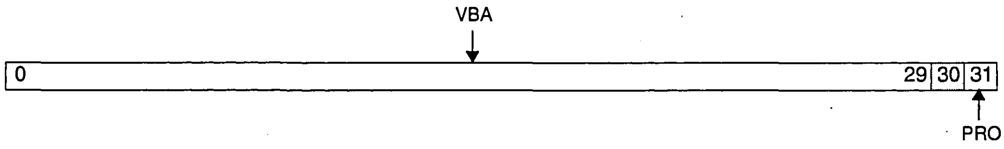


Figure 10-7. UIC Vector Configuration Register (UIC0_VCR)

0:29	VBA	Vector Base Address
30		Reserved
31	PRO	Priority Ordering 0 UIC0_SR[0] is the highest priority interrupt. 1 UIC0_SR[31] is the highest priority interrupt. Note: Vector generation is not performed for non-critical interrupts.

10.5.8 UIC Vector Register (UIC0_VR)

The read-only UIC0_VR contains an interrupt vector that can reduce interrupt handling latency for critical interrupts. Vector generation logic adds an offset to UIC0_VCR[VBA], and the sum is returned in the UIC0_VR. Vectors are not computed for non-critical interrupts.

The interrupt vector is based on the field position of the current highest priority, enabled, active, critical interrupt relative to the highest priority interrupt in the UIC0_SR. The generated vectors can be programmed to point directly to the interrupt handlers.

Programming Note: Regardless of the programming of UIC0_VCR and UIC0_VR registers, the processor always vectors to EVPR[0:16] || 0x100 when a critical interrupt occurs.

The interrupt vector offset is based on the bit position of the current highest priority, enabled, active, critical interrupt relative to the highest priority interrupt in the UIC0_SR. The offset has a fixed value of 512 per bit. The main critical interrupt handler can interpret the vector returned by UIC0_VR as the address of the interrupt handler for that interrupt, assuming the routine is 512 bytes or smaller. Alternatively, the main critical interrupt handler can interpret the vector as a look-up table entry for the address of the interrupt handler for that interrupt.

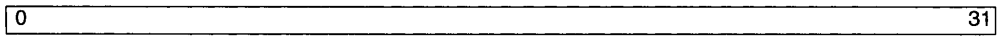


Figure 10-8. UIC Vector Register (UIC0_VR)

0:31		Interrupt Vector

The following example illustrates the generation of a UIC0_VR vector for external interrupt request IRQ2.

For the example, assume that UIC0_VCR[PRO] = 1, so that UIC0_SR[EIR6S] (UIC0_SR$_{31}$)has the highest interrupt priority, and that UIC0_SR[EIR2S] (UIC0_SR$_{27}$) is the current highest priority, enabled, active, critical interrupt. To generate the vector for the interrupt associated with UIC0_SR[EIR2S], internal logic multiplies the difference between the highest priority interrupt bit and the active enabled priority interrupt bit by 512. The interrupt vector offset is therefore $(31 - 27) \times 512 = 4 \times 512$. This offset is added to the base address in UIC0_VCR[VBA], and the UIC0_VR returns UIC0_VCR[VBA] + (4×512).

10.5.8.1 Using the Value in UIC0_VR as a Vector Address or Entry Table Lookup

If an interrupt handler is 512 bytes or smaller, system software can interpret the value returned in the UIC0_VR as an address. In this case, when the interrupt is received, the UIC0_VR is read and software simply jumps to the address represented by the UIC0_VR value. Alternatively, the routine can be at a different address, and system software can treat the value of the UIC0_VR as a pointer, storing the interrupt handler address in the UIC0_VR during system initialization. In this case, when the interrupt is handled, software must read the UIC0_VR, read the entry at the UIC0_VR value, and jump to the entry. Hardware has no knowledge of the method is used, which is determined by system software.

10.5.8.2 Vector Generation Scenarios

For the following sequence, assume that the interrupts are enabled and critical (vectors are not generated for disabled or non-critical interrupts). The sequence illustrates several scenarios for vector generation.

1. An intermediate priority interrupt goes active; its vector is stored in UIC0_VR.
2. A low priority interrupt goes active; UIC0_VR is unchanged.
3. Software reads the vector; UIC0_VR is unchanged.
4. Software resets the intermediate priority interrupt; UIC0_VR contains the vector for the low priority interrupt.
5. A high priority interrupt goes active; UIC0_VR contains the vector for the high priority interrupt.
6. Software resets the high priority interrupt; UIC0_VR contains the vector for the low priority interrupt.

7. Software resets the UIC0_ER field for the low priority interrupt, disabling it; UIC0_VR contains 0x00000000.
8. UIC0_CR is reprogrammed to make the low priority interrupt non-critical and UIC0_ER is reprogrammed to re-enable the low priority interrupt; UIC0_VR continues to contain 0x00000000.

10.6 Interrupt Handling in the Processor Core

An *interrupt* is the action in which the processor saves its old context (MSR and instruction pointer) and begins execution at a pre-determined interrupt-handler address, with a modified MSR. *Exceptions* are events which, if enabled, cause the processor to take an interrupt. Exceptions are generated by signals from internal and external peripherals, instructions, internal timer facilities, debug events, or error conditions.

Table 10-4, "Interrupt Vector Offsets," on page 10-27, lists the interrupts handled by the PPC405GP in the order of interrupt vector offsets. Detailed descriptions of each interrupt follow, in the same order. Table 10-4 also provides an index to the descriptions.

Several registers support interrupt handling and control. "General Interrupt Handling Registers" on page 10-27 describes the general interrupt handling registers:

- Data Exception Address Register (DEAR)
- Exception Syndrome Register (ESR)
- Exception Vector Prefix Register (EVPR)
- Machine State Register (MSR)
- Save/Restore Registers (SRR0–SRR3)

10.7 Architectural Definitions and Behavior

Precise interrupts are those for which the instruction pointer saved by the interrupt must be either the address of the excepting instruction or the address of the next sequential instruction. *Imprecise* interrupts are those for which it is possible (but not required) for the saved instruction pointer to be something else, possibly prohibiting guaranteed software recovery.

Note that "precise" and "imprecise" are defined assuming that the interrupts are unmasked (enabled to occur) when the associated exception occurs. Consider an exception that would cause a precise interrupt, if the interrupt was enabled at the time of the exception, but that occurs while the interrupt is masked. Some exceptions of this type can cause the interrupt to occur later, immediately upon its enabling. In such a case, the interrupt is not considered precise with respect to the enabling instruction, but imprecise ("delayed precise") with respect to the cause of the exception.

Asynchronous interrupts are caused by events which are independent of instruction execution. All asynchronous interrupts are precise, and the following rules apply:

1. All instructions prior to the one whose address is reported to the interrupt handling routine (in the save/restore register) have completed execution. However, some storage accesses generated by these preceding instructions may not have completed.
2. No subsequent instruction has begun execution, including the instruction whose address is reported to the interrupt handling routine.
3. The instruction having its address reported to the interrupt handler may appear not to have begun execution, or may have partially completed.

Synchronous interrupts are caused directly by the execution (or attempted execution) of instructions. Synchronous interrupts can be either precise or imprecise.

For synchronous precise interrupts, the following rules apply:

1. The save/restore register addresses either the instruction causing the exception or the next sequential instruction. Which instruction is addressed is determined by the interrupt type and status bits.
2. All instructions preceding the instruction causing the exception have completed execution. However, some storage accesses generated by these preceding instructions may not have completed.
3. The instruction causing the exception may appear not to have begun execution (except for causing the exception), may have partially completed, or may have completed, depending on the interrupt type.
4. No subsequent instruction has begun execution.

The PPC405GP does not implement any imprecise interrupts. Refer to *IBM PowerPC Embedded Environment* for an architectural description of imprecise interrupts.

Machine check interrupts are a special case typically caused by some kind of hardware or storage subsystem failure, or by an attempt to access an invalid address. A machine check can be indirectly caused by the execution of an instruction, but not recognized or reported until long after the processor has executed past the instruction that caused the machine check. As such, machine check interrupts cannot properly be thought of as synchronous, nor as precise or imprecise. For machine checks, the following general rules apply:

1. No instruction following the one whose address is reported to the machine check handler in the save/restore register has begun execution.
2. The instruction whose address is reported to the machine check handler in the save/restore register, and all previous instructions, may or may not have completed successfully. All previous instructions that would ever complete have completed, within the context existing before the machine check interrupt. No further interrupt (other than possible additional machine checks) can occur as a result of those instructions.

10.8 Behavior of the PPC405GP Implementation

All interrupts, except for machine checks, are handled precisely. Precise handling implies that the address of the excepting instruction (for synchronous exceptions other than the system call exception), or the address of the next instruction to be executed (asynchronous exceptions and the system call exception), is passed to an interrupt handling routine. Precise handling also implies that all instructions that precede the instruction whose address is reported to the interrupt handling routine have executed and that no subsequent instruction has begun execution. The specific instruction whose address is reported may not have begun execution or may have partially completed, as specified for each precise interrupt type.

Synchronous precise interrupts include most debug event interrupts, program interrupts, instruction and data storage interrupts, TLB miss interrupts, system call interrupts, and alignment interrupts.

Asynchronous precise interrupts include the critical and noncritical external interrupts, and can be caused by on-chip peripherals, timer facility interrupts, and some debug events.

In the PPC405GP, machine checks are handled as critical interrupts (see “Critical and Noncritical Interrupts” on page 10-26). If a machine check is associated with an instruction fetch, the critical interrupt save/restore register contains the address of the instruction being fetched when the machine check occurred.

The synchronism of instruction-side machine checks (errors that occur while attempting to fetch an instruction from external memory) requires further explanation. Fetch requests to cachable memory that miss in the instruction cache unit (ICU) cause an instruction cache line fill (eight words). If any instructions (words) in the fetched line are associated with an exception, an interrupt occurs upon attempted execution and the cache line is invalidated.

It is improper to declare an exception when an erroneous word is passed to the fetcher; the address could be the result of an incorrect speculative access. It is quite likely that no attempt will be made to execute an instruction from the erroneous address. An instruction-side machine check interrupt occurs only when execution is attempted. If an exception occurs, execution is suppressed, SRR2 contains the erroneous address, and the ESR indicates that an instruction-side machine check occurred. Although such an interrupt is clearly asynchronous to the erroneous memory access, it is handled synchronously with respect to the attempted execution from the erroneous address.

Except for machine checks, all PPC405GP interrupts are handled precisely:

- The address of the excepting instruction (for synchronous exceptions, other than the system call exception) or the address of the next sequential instruction (for asynchronous exceptions and the system call exception) is passed to the interrupt handling routine.
- All instructions that precede the instruction whose address is reported to the interrupt handling routine have completed execution and that no subsequent instruction has begun execution. The specific instruction whose address is reported might not have begun execution or might have partially completed, as specified for each interrupt type.

10.9 Interrupt Handling Priorities

The PPC405GP processor core handles only one interrupt at a time. Multiple simultaneous interrupts are handled in the priority order shown in Table 10-3 (assuming, of course, that the interrupt types are enabled).

Multiple interrupts can exist simultaneously, each of which requires the generation of an interrupt. The architecture does not provide for simultaneously reporting more than one interrupt of the same class (critical or non-critical). Therefore, interrupts are ordered with respect to each other. A masking mechanism is available for certain persistent interrupt types.

When an interrupt type is masked, and an event causes an exception which would normally generate an interrupt of that type, the exception *persists* as a *status* bit in a register. However, no interrupt is generated. Later, if the interrupt type is enabled (unmasked), and the exception status has not been cleared by software, the interrupt due to the original exception event is finally generated.

All asynchronous interrupt types can be masked. In addition, certain synchronous interrupt types can be masked.

Table 10-3. Interrupt Handling Priorities

Priority	Interrupt Type	Critical or Noncritical	Causing Conditions
1	Machine check—data	Critical	External bus error during data-side access
2	Debug—IAC	Critical	IAC debug event (in internal debug mode)
3	Machine check—instruction	Critical	Attempted execution of instruction for which an external bus error occurred during fetch
4	Debug—EXC, UDE	Critical	EXC or UDE debug event (in internal debug mode)
5	Critical interrupt input	Critical	Active level on the critical interrupt input
6	Watchdog timer—first time-out	Critical	Posting of an enabled first time-out of the watchdog timer in the TSR
7	Instruction TLB Miss	Noncritical	Attempted execution of an instruction at an address and process ID for which a valid matching entry was not found in the TLB
8	Instruction storage — ZPR[Z*n*] = 00	Noncritical	Instruction translation is active, execution access to the translated address is not permitted because ZPR[Z*n*] = 00 in user mode, and an attempt is made to execute the instruction
9	Instruction storage — TLB_entry[EX] = 0	Noncritical	Instruction translation is active, execution access to the translated address is not permitted because TLB_entry[EX] = 0, and an attempt is made to execute the instruction
	Instruction storage — TLB_entry[G] = 1	Noncritical	Instruction translation is active, the page is marked guarded, and an attempt is made to execute the instruction
10	Program	Noncritical	Attempted execution of illegal instructions, TRAP instruction, privileged instruction in problem state
	System call	Noncritical	Execution of the **sc** instruction
11	Data TLB miss	Noncritical	Valid matching entry for the effective address and process ID of an attempted data access is not found in the TLB
12	Data storage— ZPR[Z*n*] = 00	Noncritical	Data translation is active and data-side access to the translated address is not permitted because ZPR[Z*n*] = 00 in user mode
13	Data storage— TLB_entry[WR] = 0	Noncritical	Data translation is active and write access to the translated address is not permitted because TLB_entry[WR] = 0
	Data storage— TLB_entry[U0] = 1 or SU0R[U*n*] = 1	Noncritical	Data translation is active and write access to the translated address is not permitted because TLB_entry[U0] = 1 or SU0R[U*n*] = 1
14	Alignment	Noncritical	**dcbz** to non-cachable address or write-through storage; non-word aligned **dcread**, **lwarx**, and **stwcx**, as described in Table 10-13
15	Debug—BT, DAC, DVC, IC, TIE	Critical	BT, DAC, DVC, IC, TIE debug event (in internal debug mode)

Table 10-3. Interrupt Handling Priorities (continued)

Priority	Interrupt Type	Critical or Noncritical	Causing Conditions
16	External interrupt input	Noncritical	Interrupts from the external interrupt (external to the processor core) input
17	Fixed Interval Timer (FIT)	Noncritical	Posting of an enabled FIT interrupt in the TSR
18	Programmable Interval Timer (PIT)	Noncritical	Posting of an enabled PIT interrupt in the TSR

10.10 Critical and Noncritical Interrupts

The PPC405GP processes interrupts as noncritical and critical. Twelve interrupts are defined as *noncritical*: data storage, instruction storage, an active external interrupt input, alignment, program, system call, programmable interval timer (PIT), fixed interval timer (FIT), data TLB miss, and instruction TLB miss. Five interrupts are defined as *critical*: machine check interrupts (instruction- and data-side), debug interrupts (any of the seven types), interrupts caused by an active critical interrupt input, and the first time-out from the watchdog timer.

When a *noncritical* interrupt is taken, Save/Restore Register 0 (SRR0) is written with the address of the excepting instruction (most synchronous interrupts) or the next sequential instruction to be processed (asynchronous interrupts and system call).

If the PPC405GP was executing a multicycle instruction (multiply, divide, or cache operation), the instruction is terminated and its address is written in SRR0.

Aligned scalar loads/stores that are interrupted do not appear on the PLB. An aligned scalar load/store cannot be interrupted after it is requested on the PLB, so the Guarded (G) storage attribute does not need to prevent the interruption of an aligned scalar load/store.

To enhance performance, the DCU can respond to non-cachable load requests by retrieving a line instead of a word. This is controlled by CCR0[LWL]. Note, however, that If CCR0[LWL] = 1, and the target non-cachable region is also marked as guarded (the G storage attribute is set to 1), that the DCU will request on the PLB only those bytes requested by the CPU.

Load/store multiples, load/store string, and misaligned scalar loads/stores that cross a word boundary can be interrupted and restarted upon return from the interrupt handler.

When load instructions terminate, the addressing registers are not updated. This ensures that the instructions can be restarted; if the addressing registers were in the range of registers to be loaded, this would be an invalid form in any event. Some target registers of a load instruction may have been written by the time of the interrupt; when the instruction restarts, the registers will simply be written again. Similarly, some of the target memory of a store instruction may have been written, and is written again when the instruction restarts.

Save/Restore Register 1 (SRR1) is written with the contents of the MSR; the MSR is then updated to reflect the new machine context. The new MSR contents take effect beginning with the first instruction of the interrupt handling routine.

Interrupt handling routine instructions are fetched at an address determined by the interrupt type. The address of the interrupt handling routine is formed by concatenating the 16 high-order bits of the

EVPR and the interrupt vector offset. (A user must initialize the EVPR contents at power-up using an **mtspr** instruction.)

Table 10-4 shows the interrupt vector offsets for the interrupt types. Note that there can be multiple sources of the same interrupt type; interrupts of the same type are mapped to the same interrupt vector, regardless of source. In such cases, the interrupt handling routine must examine status registers to determine the exact source of the interrupt.

At the end of the interrupt handling routine, execution of an **rfi** instruction forces the contents of SRR0 and SRR1 to be written to the program counter and the MSR, respectively. Execution then begins at the address in the program counter.

Critical interrupts are processed similarly. When a critical interrupt is taken, Save/Restore Register 2 (SRR2) and Save/Restore Register 3 (SRR3) hold the next sequential address to be processed when returning from the interrupt, and the contents of the MSR, respectively. At the end of the critical interrupt handling routine, execution of an **rfci** instruction writes the contents of SRR2 and SRR3 into the program counter and the MSR, respectively.

Table 10-4. Interrupt Vector Offsets

Offset	Interrupt Type	Interrupt Class	Category	Page
0x0100	Critical input interrupt	Asynchronous precise	Critical	10-34
0x0200	Machine check—data	—	Critical	10-35
	Machine check—instruction	—	Critical	10-35
0x0300	Data storage interrupt—MSR[DR]=1 and ZPR[Z*n*] = 0 or TLB_entry[WR] = 0 or TLB_entry[U0] = 1 or SU0R[U*n*] = 1	Synchronous precise	Noncritical	10-36
0x0400	Instruction storage interrupt	Synchronous precise	Noncritical	10-38
0x0500	External interrupt (external to the processor core)	Asynchronous precise	Noncritical	10-38
0x0600	Alignment	Synchronous precise	Noncritical	10-39
0x0700	Program	Synchronous precise	Noncritical	10-40
0x0C00	System Call	Synchronous precise	Noncritical	10-41
0x1000	PIT	Asynchronous precise	Noncritical	10-41
0x1010	FIT	Asynchronous precise	Noncritical	10-42
0x1020	Watchdog timer	Asynchronous precise	Critical	10-43
0x1100	Data TLB miss	Synchronous precise	Noncritical	10-43
0x1200	Instruction TLB miss	Synchronous precise	Noncritical	10-44
0x2000	Debug—BT, DAC, DVC, IAC, IC, TIE	Synchronous precise	Critical	10-44
	Debug—EXC, UDE	Asynchronous precise	Critical	

10.11 General Interrupt Handling Registers

The general interrupt handling registers are the Machine State Register (MSR), SRR0–SRR3, the Exception Vector Prefix Register (EVPR), the Exception Syndrome Register (ESR), and the Data Exception Address Register (DEAR).

10.11.1 Machine State Register (MSR)

The MSR is a 32-bit register that holds the current context of the PPC405GP. When a noncritical interrupt is taken, the MSR contents are written to SRR1; when a critical interrupt is taken, the MSR contents are written to SRR3. When an **rfi** or **rfci** instruction executes, the contents of the MSR are read from SRR1 or SRR3, respectively.

Programming Note: The **rfi** and **rfci** instructions can alter reserved MSR fields.

The MSR contents can be read into a general purpose register (GPRs) using an **mfmsr** instruction. The contents of a GPR can be written to the MSR using an **mtmsr** instruction. The MSR[EE] bit may be set/cleared atomically using the **wrtee** or **wrteei** instructions.

Figure 10-9 shows the MSR bit definitions and describes the function of each bit.

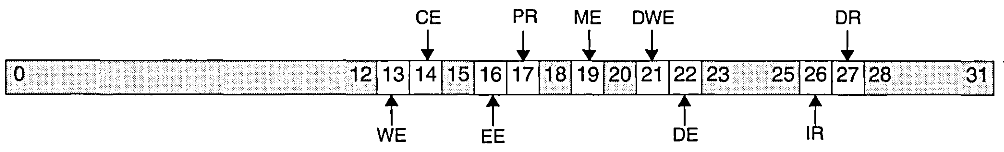


Figure 10-9. Machine State Register (MSR)

0:12		Reserved	
13	WE	Wait State Enable 0 The processor is not in the wait state. 1 The processor is in the wait state.	If MSR[WE] = 1, the processor remains in the wait state until an interrupt is taken, a reset occurs, or an external debug tool clears WE.
14	CE	Critical Interrupt Enable 0 Critical interrupts are disabled. 1 Critical interrupts are enabled.	Controls the critical interrupt input and watchdog timer first time-out interrupts.
15		Reserved	
16	EE	External Interrupt Enable 0 Asynchronous interrupts (external to the processor core) are disabled. 1 Asynchronous interrupts are enabled.	Controls the non-critical external interrupt input, PIT, and FIT interrupts.
17	PR	Problem State 0 Supervisor state (all instructions allowed). 1 Problem state (some instructions not allowed).	
18		Reserved	
19	ME	Machine Check Enable 0 Machine check interrupts are disabled. 1 Machine check interrupts are enabled.	
20		Reserved	

21	DWE	Debug Wait Enable 0 Debug wait mode is disabled. 1 Debug wait mode is enabled.
22	DE	Debug Interrupts Enable 0 Debug interrupts are disabled. 1 Debug interrupts are enabled.
23:25		Reserved
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.
27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.
28:31		Reserved

10.11.2 Save/Restore Registers 0 and 1 (SRR0–SRR1)

SRR0 and SRR1 are 32-bit registers that hold the interrupted machine context when a noncritical interrupt is processed. On interrupt, SRR0 is set to the current or next instruction address and the contents of the MSR are written to SRR1. When an **rfi** instruction is executed at the end of the interrupt handler, the program counter and the MSR are restored from SRR0 and SRR1, respectively.

The contents of SRR0 and SRR1 can be written into GPRs using the **mfspr** instruction. The contents of GPRs can be written to SRR0 and SRR1 using the **mtspr** instruction.

Figure 10-10 shows the bit definitions for SRR0.

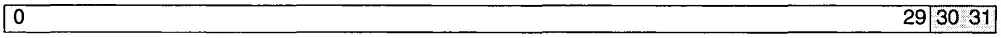


Figure 10-10. Save/Restore Register 0 (SRR0)

0:29		SRR0 receives an instruction address when a non-critical interrupt is taken; the Program Counter is restored from SRR0 when **rfi** executes.
30:31		Reserved

Figure 10-11 shows the bit definitions for SRR1.

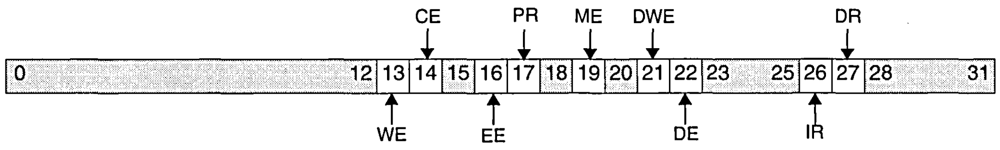


Figure 10-11. Save/Restore Register 1 (SRR1)

0:31	SRR1 receives a copy of the MSR when an interrupt is taken; the MSR is restored from SRR1 when **rfi** executes.

10.11.3 Save/Restore Registers 2 and 3 (SRR2–SRR3)

SRR2 and SRR3 are 32-bit registers that hold the interrupted machine context when a critical interrupt is processed. On interrupt, SRR2 is set to the current or next instruction address and the contents of the MSR are written to SRR3. When an **rfci** instruction is executed at the end of the interrupt handler, the program counter and the MSR are restored from SRR2 and SRR3, respectively.

The contents of SRR2 and SRR3 can be written to GPRs using the **mfspr** instruction. The contents of GPRs can be written to SRR2 and SRR3 using the **mtspr** instruction.

Figure 10-12 shows the bit definitions for SRR2.

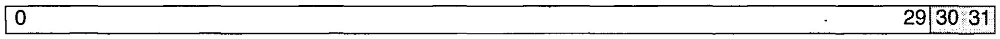


Figure 10-12. Save/Restore Register 2 (SRR2)

0:29		SRR2 receives an instruction address when a critical interrupt is taken; the Program Counter is restored from SRR2 when **rfci** executes.
30:31	▨	Reserved

Figure 10-13 shows the bit definitions for SRR3.

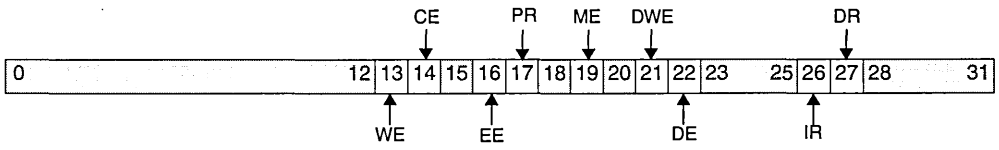


Figure 10-13. Save/Restore Register 3 (SRR3)

0:31	SRR3 receives a copy of the MSR when a critical interrupt is taken; the MSR is restored from SRR3 when **rfci** executes.

Because critical interrupts do not automatically clear MSR[ME], SRR2 and SRR3 can be corrupted by a machine check interrupt, if the machine check occurs while SRR2 and SRR3 contain valid data that has not yet been saved by the critical interrupt handler.

10.11.4 Exception Vector Prefix Register (EVPR)

The EVPR is a 32-bit register whose high-order 16 bits contain the prefix for the address of an interrupt handling routine. The 16-bit interrupt vector offsets (shown in Table 10-4, "Interrupt Vector Offsets," on page 10-27) are concatenated to the right of the high-order 16 bits of the EVPR to form the 32-bit address of an interrupt handling routine.

The contents of the EVPR can be written to a GPR using the **mfspr** instruction. The contents of a GPR can be written to EVPR using the **mtspr** instruction.

Figure 10-14 shows the EVPR bit definitions.

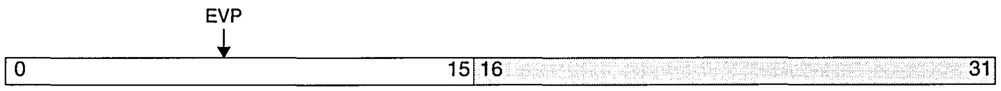


Figure 10-14. Exception Vector Prefix Register (EVPR)

0:15	EVP	Exception Vector Prefix
16:31	▨	Reserved

10.11.5 Exception Syndrome Register (ESR)

The ESR is a 32-bit register whose bits help to specify the exact cause of various synchronous interrupts. These interrupts include instruction and data side machine checks, data storage interrupts, and program interrupts, instruction storage interrupts, and data TLB miss interrupts.

"Instruction Machine Check Handling" on page 10-35 describes instruction machine checks. "Data Storage Interrupt" on page 10-36 describes data storage interrupts. "Program Interrupt" on page 10-40 describes program interrupts.

Although interrupt handling routines are not required to reset the ESR, it is recommended that instruction machine check handlers reset the ESR; "Instruction Machine Check Handling" on page 10-35 describes why such resets are recommended.

The contents of the ESR can be written to a GPR using the **mfspr** instruction. The contents of a GPR can be written to the ESR using the **mtspr** instruction.

Figure 10-15 shows the ESR bit definitions.

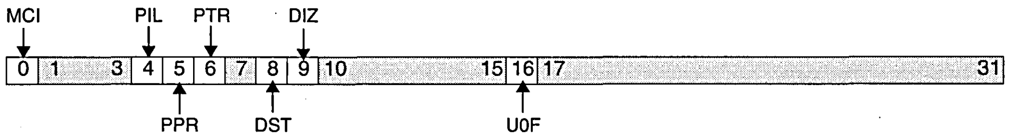


Figure 10-15. Exception Syndrome Register (ESR)

0	MCI	Machine check—instruction 0 Instruction machine check did not occur. 1 Instruction machine check occurred.
1:3	▨	Reserved
4	PIL	Program interrupt—illegal 0 Illegal Instruction error did not occur. 1 Illegal Instruction error occurred.
5	PPR	Program interrupt—privileged 0 Privileged instruction error did not occur. 1 Privileged instruction error occurred.
6	PTR	Program interrupt—trap 0 Trap with successful compare did not occur. 1 Trap with successful compare occurred.
7	▨	Reserved
8	DST	Data storage interrupt—store fault 0 Excepting instruction was not a store. 1 Excepting instruction was a store (includes **dcbi**, **dcbz**, and **dccci**).
9	DIZ	Data/instruction storage interrupt—zone fault 0 Excepting condition was not a zone fault. 1 Excepting condition was a zone fault.
10:15	▨	Reserved

16	U0F	Data storage interrupt—U0 fault 0 Excepting instruction did not cause a U0 fault. 1 Excepting instruction did cause a U0 fault.
17:31		Reserved

In general, ESR bits are set to indicate the type of precise interrupt that occurred; other bits are cleared. However, the machine check—instruction (ESR[MCI]) bit behaves differently. Because instruction-side machine checks can occur without an interrupt being taken (if MSR[ME] = 0), ESR[MCI] can be set even while other ESR-setting interrupts (program, data storage, DTLB-miss) occurring. Thus, data storage and program interrupts leave ESR[MCI] unchanged, clear all other ESR bits, and set the bits associated with any data storage or program interrupts occurred. Enabled instruction-side machine checks (MSR[ME] = 1) set ESR[MCI] and clear the data storage and program interrupt bits.

If a machine check—instruction interrupt occurs but is disabled (MSR[ME] = 0), it sets ESR[MCI] but leaves the data storage and program interrupt bits alone. If a machine check—instruction interrupt occurs while MSR[ME] = 0, *and* the instruction upon which the machine check—instruction interrupt is occurring also is some other kind of ESR-setting instruction (program, data storage, DTLB-miss, or instruction storage interrupt), ESR[MCI] is set to indicate that a machine check—instruction interrupt occurred; the other ESR bits are set or cleared to indicate the other interrupt. These scenarios are summarized in Table 10-5

Table 10-5. ESR Alteration by Various Interrupts

Scenario	ESR$_{4:}$	ESR$_{8:9, 16}$
Program interrupt without machine check interrupt	Set to type	Cleared
Enabled MCI	Cleared	Cleared
Disabled MCI, no others	Unchanged	Unchanged
Disabled MCI and program interrupt	Set to type	Cleared

Table 10-6. ESR Alteration by Various Interrupts

Scenario	ECR[MCI]	ESR$_{4:}$	ESR$_{8:9, 16}$
Program interrupt	Unchanged	Set to type	Cleared
Data storage interrupt	Unchanged	Cleared	Set to Type
Data TLB miss interrupt	Unchanged	Cleared	Cleared
Machine check—instruction	Set to 1	Cleared	Cleared
Disabled MCI, no others	Unchanged	Unchanged	Unchanged
Disabled MCI and program interrupt	Unchanged	Set to type	Cleared

10.11.6 Data Exception Address Register (DEAR)

The DEAR is a 32-bit register that contains the address of the access for which one of the following synchronous precise errors occurred: alignment error, data TLB miss, or data storage interrupt.

The contents of the DEAR can be written to a GPR using the **mfspr** instruction. The contents of a GPR can be written to the DEAR using the **mtspr** instruction.

Figure 10-16 shows the DEAR bit definitions.

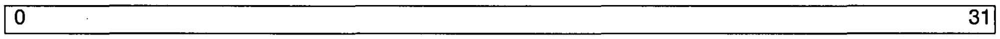


Figure 10-16. Data Exception Address Register (DEAR)

0:31		Address of Data Error (synchronous)

10.12 Critical Input Interrupts

The UICCR can be programmed so that any UIC interrupt can be presented as a critical interrupt input to the processor core. See "UIC Trigger Register (UIC0_TR)" on page 10-13 for details. Critical interrupts are recognized only if enabled by MSR[CE].

MSR[CE] also enables the watchdog timer first-time-out interrupt. However, the watchdog interrupt has a different interrupt vector than the critical pin interrupt. See "Watchdog Timer Interrupt" on page 10-43.

After detecting a critical interrupt, if no synchronous precise interrupts are outstanding, the PPC405GP immediately takes the critical interrupt and writes the address of the next instruction to be executed in SRR2. Simultaneously, the contents of the MSR are saved in SRR3. MSR[CE] is reset to 0 to prevent another critical interrupt or the watchdog timer first time-out interrupt from interrupting the critical interrupt handler before SRR2 and SRR3 get saved. MSR[DE] is reset to 0 to disable debug interrupts during the critical interrupt handler.

The MSR is also written with the values shown in Table 10-7, "Register Settings during Critical Input Interrupts," on page 10-35. The high-order 16 bits of the program counter are then loaded with the contents of the EVPR and the low-order 16 bits of the program counter are loaded with 0x0100. Interrupt processing begins at the address in the program counter.

Inside the interrupt handling routine, after the contents of SRR2/SRR3 are saved, critical interrupts can be enabled again by setting MSR[CE] = 1.

Executing an **rfci** instruction restores the program counter from SRR2 and the MSR from SRR3, and execution resumes at the address in the program counter.

Table 10-7. Register Settings during Critical Input Interrupts

SRR2	Written with the address of the next instruction to be executed
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, DWE, DE,IR, DR←0 ME← unchanged
PC	EVPR[0:15] ll 0x0100

10.13 Machine Check Interrupts

When an external bus error occurs on an instruction fetch, and execution of that instruction is subsequently attempted, a machine check—instruction interrupt occurs.

When an external bus error occurs while attempting data accesses, a machine check—data interrupt occurs.

When an instruction-side machine check interrupt occurs, the PPC405GP stores the address of the excepting instruction in SRR2. When a data-side machine check occurs, the PPC405GP stores the address of the next sequential instruction in SRR2. Simultaneously, for all machine check interrupts, the contents of the MSR are loaded into SRR3.

The MSR Machine Check Enable bit (MSR[ME]) is reset to 0 to disable another machine check from interrupting the machine check interrupt handling routine. The other MSR bits are loaded with the values shown in Table 10-8, "Register Settings during Machine Check—Instruction Interrupts," on page 10-36 and Table 10-9, "Register Settings during Machine Check—Data Interrupts," on page 10-36. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0200. Interrupt processing begins at the new address in the program counter.

Executing an **rfci** instruction restores the program counter from SRR2 and the MSR from SRR3, and execution resumes at the address in the program counter.

10.13.1 Instruction Machine Check Handling

When a machine check occurs on an instruction fetch, *and execution of that instruction is subsequently attempted*, a machine check—instruction interrupt occurs. If enabled by MSR[ME], the processor reports the machine check—instruction interrupt by vectoring to the machine check handler (EVPR[0:15] ll 0x0200), setting ESR[MCI]. Note that only a bus error can cause a machine check—instruction interrupt. Taking the vector automatically clears MSR[ME] and the other MSR fields.

Note that it is improper to declare a machine check—instruction interrupt when the instruction is fetched, because the address is possibly the result of an incorrect speculation by the fetcher. It is quite likely that no attempt will be made to execute an instruction from the erroneous address. The interrupt will occur only if execution of the instruction is subsequently attempted.

When a machine check occurs on an instruction fetch, the erroneous instruction is never validated in the instruction cache unit (ICU). Fetch requests to cachable memory that miss in the ICU cause an instruction cache line fill (eight words). If any words in the fetched line are associated with an error, an interrupt occurs upon attempted execution and the cache line is invalidated. If any word in the line is in error, the cache line is invalidated after the line fill.

ESR[MCI] is set, even if MSR[ME] = 0. This means that if a machine check—instruction interrupt occurs while running in code in which MSR[ME] is disabled, the machine check—instruction interrupt is recorded in the ESR, but no interrupt occurs. Software running with MSR[ME] disabled can sample ESR[MCI] to determine whether at least one machine check—instruction interrupt occurred during the disabled execution.

If a new machine check—instruction interrupt occurs after MSR[ME] is enabled again, the new machine check—instruction interrupt is recorded in ESR[MCI] and the machine check—instruction interrupt handler is invoked. However, enabling MSR[ME] again does *not* cause a machine Check interrupt to occur simply due to the presence of ESR[MCI] indicating that a machine check—instruction interrupt occurred while MSR[ME] was disabled. The machine check—instruction interrupt must occur while MSR[ME] is enabled for the machine check interrupt to be taken. Software should, in general, clear the ESR bits before returning from a machine check interrupt to avoid any ambiguity when handling subsequent machine check interrupts.

Table 10-8. Register Settings during Machine Check—Instruction Interrupts

SRR2	Written with the address that caused the machine check.
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, ME, DWE, DE,IR, DR←0
PC	EVPR[0:15] II 0x0200
ESR	MCI ← 1 All other bits are cleared.

10.13.2 Data Machine Check Handling

When a machine check occurs on an data access, a machine check—data interrupt occurs. To determine the cause of a machine check, examine the various error reporting registers of the external PLB slaves.

Table 10-9. Register Settings during Machine Check—Data Interrupts

SRR2	Written with the address of the next sequential instruction.
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, ME, DWE, DE, IR, DR←0
PC	EVPR[0:15] II 0x0200

10.14 Data Storage Interrupt

The data storage interrupt occurs when the desired access to the effective address is not permitted for any of the following reasons:

- A U0 fault: any store to an EA with the U0 storage attribute set and CCR0[U0XE] = 1

- In the problem state with data translation enabled:
  - A *zone fault*, which is any user-mode storage access (data load, store, **icbi**, **dcbz**, **dcbst**, or **dcbf**) with an effective address with (ZPR field) = 00. (**dcbt** and **dcbtst** will no-op in this situation, rather than cause an interrupt. The instructions **dcbi**, **dccci**, **icbt**, and **iccci**, being privileged, cannot cause zone fault data storage interrupts.)
  - Data store or **dcbz** to an effective address with the WR bit clear and (ZPR field) ≠ 11. (The privileged instructions **dcbi** and **dccci** are treated as "stores," but will cause privileged program interrupts, rather than data storage interrupts.)
- In the supervisor state with data translation enabled:
  - Data store, **dcbi**, **dcbz**, or **dccci** to an effective address with the WR bit clear and (ZPR field) other than 11 or 10.

  Programming Note: The **icbi**, **icbt**, and **iccci** instructions are treated as loads from the addressed byte with respect to address translation and protection. Instruction cache operations use MSR[DR], not MSR[IR], to determine translation of their operands. Instruction storage interrupts and Instruction-side TLB Miss Interrupts are associated with the *fetching* of instructions, not with the execution of instructions. Data storage interrupts and data TLB miss interrupts are associated with the *execution* of instruction cache operations.

When a data storage interrupt is detected, the PPC405GP suppresses the instruction causing the interrupt and writes the instruction address in SRR0. The Data Exception Address Register (DEAR) is loaded with the data address that caused the access violation. ESR bits are loaded as shown in Table 10-10, "Register Settings during Data Storage Interrupts," on page 10-37 to provide further information about the error. The current contents of the MSR are loaded into SRR1, and MSR bits are then loaded with the values shown in Table 10-10.

The high-order 16 bits of the program counter are then loaded with the contents of the EVPR and the low-order 16 bits of the program counter are loaded with 0x0300. Interrupt processing begins at the new address in the program counter. Executing the return from interrupt instruction (**rfi**) restores the contents of the program counter and the MSR from SRR0 and SRR1, respectively, and the PPC405GP resumes execution at the new program counter address.

For instructions that can simultaneously generate program interrupts (privileged instructions executed in Problem State) and data storage interrupts, the program interrupt has priority.

Table 10-10. Register Settings during Data Storage Interrupts

SRR0	Written with the EA of the instruction causing the data storage interrupt
SRR1	Written with the value of the MSR at the time of the interrupt
MSR	WE, EE, PR, DWE, IR, DR←0 CE, ME, DE ← unchanged
PC	EVPR[0:15] ll 0x0300
DEAR	Written with the EA of the failed access
ESR	DST ← 1 if excepting operation is a store (includes **dcbi** and **dcbz)** DIZ ← 1 if access failure caused by a zone protection fault (ZPR[Z*n*] = 00 in user mode) U0F ← 1 if access failure caused by a U0 fault (the U0 storage attribute is set and CCR0[U0XE] = 1) MCI ← unchanged All other bits are cleared.

10.15 Instruction Storage Interrupt

The instruction storage interrupt is generated when instruction translation is active and execution is attempted for an instruction whose fetch access to the effective address is not permitted for any of the following reasons:

- In Problem State:
  - Instruction fetch from an effective address with (ZPR field) = 00.
  - Instruction fetch from an effective address with the EX bit clear and (ZPR field) ≠ 11.
  - Instruction fetch from an effective address contained within a Guarded region (G=1).
- In Supervisor State:
  - Instruction fetch from an effective address with the EX bit clear and (ZPR field) other than 11 or 10.
  - Instruction fetch from an effective address contained within a Guarded region (G=1).

SRR0 will save the address of the instruction causing the instruction storage interrupt.

ESR is set to indicate the following conditions:

- If ESR[DIZ] = 1, the excepting condition was a zone fault: the attempted execution of an instruction address fetched in user-mode with (ZPR field) = 00.
- If ESR[DIZ] = 0, then the excepting condition was either EX = 0 or G = 1.

The interrupt is precise with respect to the attempted execution of the instruction. Program flow vectors to EVPR[0:15] || 0x0400.

The following registers are modified to the specified values:

Table 10-11. Register Settings during Instruction Storage Interrupts

SRR0	Set to the EA of the instruction for which execute access was not permitted
SRR1	Set to the value of the MSR at the time of the interrupt
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0400
ESR	DIZ ← 1If access failure due to a zone protection fault (ZPR[Z*n*] = 00 in user mode) **Note:** If ESR[DIZ] is not set, the interrupt occurred because TBL_entry[EX] was clear in an otherwise accessible zone, or because of an instruction fetch from a storage region marked as guarded. See "Exception Syndrome Register (ESR)" on page 10-31 for details of ESR operation. MCI ← unchanged All other bits are cleared.

10.16 External Interrupt

External interrupts (external to the processor core) are triggered by active levels for non-critical interrupts in the UIC. All external interrupting events are presented to the processor as a single external interrupt. External interrupts are enabled or disabled by MSR[EE].

Programming Note: MSR[EE] also enables PIT and FIT interrupts. However, after timer interrupts, control passes to different interrupt vectors than for the interrupts discussed in the preceding paragraph. Therefore, these timer interrupts are described in "Programmable Interval Timer (PIT) Interrupt" on page 10-41 and "Fixed Interval Timer (FIT) Interrupt" on page 10-42.

10.16.1 External Interrupt Handling

When MSR[EE] = 1 (external interrupts are enabled), a noncritical external interrupt occurs, and this interrupt is the highest priority interrupt condition, the processor immediately writes the address of the next sequential instruction into SRR0. Simultaneously, the contents of the MSR are saved in SRR1.

When the processor takes a noncritical external interrupt, MSR[EE] is set to 0. This disables other external interrupts from interrupting the interrupt handler before SRR0 and SRR1 are saved. The MSR is also written with the other values shown in Table 10-12, "Register Settings during External Interrupts," on page 10-39. The high-order 16 bits of the program counter are written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0500. Interrupt processing begins at the address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 10-12. Register Settings during External Interrupts

SRR0	Written with the address of the next sequential instruction
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0500

10.17 Alignment Interrupt

Alignment interrupts are caused by **dcbz** instructions to non-cachable or write-through storage and. Table 10-13 summarizes the instructions and conditions causing alignment interrupts.

Table 10-13. Alignment Interrupt Summary

Instructions Causing Alignment Interrupts	Conditions
dcbz	EA in non-cachable or write-through storage
dcread, lwarx, stwcx.	EA not word-aligned

Execution of an instruction causing an alignment interrupt is prohibited from completing. SRR0 is written with the address of that instruction and the current contents of the MSR are saved into SRR1. The DEAR is written with the address that caused the alignment error. The MSR bits are written with the values shown in Table 10-14, "Register Settings during Alignment Interrupts," on page 10-40. The high-order 16 bits of the program counter are written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0600. Interrupt processing begins at the new address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter

Alignment interrupts cannot be disabled. To avoid overwrites of SRR0 and SRR1 by alignment interrupts that occur within a handler, interrupt handlers should save these registers as soon as possible.

Table 10-14. Register Settings during Alignment Interrupts

SRR0	Written with the address of the instruction causing the alignment interrupt
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] II 0x0600
DEAR	Written with the address that caused the alignment violation

10.18 Program Interrupt

Program interrupts are caused by attempting to execute:

- An illegal instruction
- A privileged instruction while in the problem state
- Executing a trap instruction with conditions satisfied

The ESR bits that differentiate these situations are listed and described in Table 10-15. When a program interrupt occurs, the appropriate bit is set and the others are cleared. These interrupts are not maskable.

Table 10-15. ESR Usage for Program Interrupts

Bits	Interrupts	Cause
ESR[PIL]	Illegal instruction	Opcode not recognized
ESR[PPR]	Privileged instruction	Attempt to use a privileged instruction in the problem state
ESR[PTR]	Trap	Excepting instruction is a trap

The program interrupt handler does not need to reset the ESR.

When one of the following occurs, the PPC405GP does not execute the instruction, but writes the address of the excepting instruction into SRR0:

- Attempted execution of a privileged instruction in problem state
- Attempted execution of an illegal instruction (including memory management instructions when memory management is disabled

Trap instructions can be used as a program interrupt or a debug event, or both (see "Debug Events" on page 12-16 for information about debug events). When a trap instruction is detected as a program interrupt, the PPC405GP writes the address of the trap instruction into SRR0. See **tw** on page 24-190 and **twi** on page 24-193 (both in Chapter 24, "Instruction Set") for a detailed discussion of the behavior of trap instructions with various interrupts enabled.

After any program interrupt, the contents of the MSR are written into SRR1 and the MSR bits are written with the values shown in Table 10-16. The high-order 16 bits of the program counter are written with the contents of the EVPR; the low-order 16 bits of the program counter are written with 0x0700. Interrupt processing begins at the new address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 10-16. Register Settings during Program Interrupts

SRR0	Written with the address of the excepting instruction
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] II 0x0700
ESR	Written with the type of program interrupt. (See Table 10-15) MCI ← unchanged All other bits are cleared.

10.19 System Call Interrupt

System call interrupts occur when a **sc** instruction is executed. The PPC405GP writes the address of the instruction following the **sc** into SRR0. The contents of the MSR are written into SRR1 and the MSR bits are written with the values shown in Table 10-17. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0C00. Interrupt processing begins at the new address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 10-17. Register Settings during System Call Interrupts

SRR0	Written with the address of the instruction following the **sc** instruction
SRR1	Written with the contents of the MSR
MSR	AP, APE, WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] II 0x0C00

10.20 Programmable Interval Timer (PIT) Interrupt

For a discussion of the PPC405GP timer facilities, see Chapter 11, "Timer Facilities." The PIT is described in "Programmable Interval Timer (PIT)" on page 11-4.

If the PIT interrupt is enabled by TCR[PIE] and MSR[EE], the PPC405GP initiates a PIT interrupt after detecting a time-out from the PIT. Time-out is detected when, at the beginning of a clock cycle, TSR[PIS] = 1. (This occurs on the cycle after the PIT decrements on a PIT count of 1.) The PPC405GP immediately takes the interrupt. The address of the next sequential instruction is saved in SRR0; simultaneously, the contents of the MSR are written into SRR1 and the MSR is written with the values shown in Table 10-18. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x1000. Interrupt processing begins at the address in the program counter.

To clear a PIT interrupt, the interrupt handling routine must clear the PIT interrupt bit, TSR[PIS]. Clearing is performed by writing a word to TSR, using an **mtspr** instruction, that has 1 in bit positions to be cleared and 0 in all other bit positions. The data written to the TSR is not direct data, but a mask; a 1 clears the bit and 0 has no effect.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 10-18. Register Settings during Programmable Interval Timer Interrupts

SRR0	Written with the address of the next instruction to be executed
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] ll 0x1000
TSR	PIS ← 1

10.21 Fixed Interval Timer (FIT) Interrupt

For a discussion of the PPC405GP timer facilities, see Chapter 11, "Timer Facilities." The FIT is described in "Fixed Interval Timer (FIT) Interrupt" on page 10-42.

If the FIT interrupt is enabled by TCR[FIE] and MSR[EE], the PPC405GP initiates a FIT interrupt after detecting a time-out from the FIT. Time-out is detected when, at the beginning of a clock cycle, TSR[FIS] = 1. (This occurs on the second cycle after the 0 → 1 transition of the appropriate time-base bit.) The PPC405GP immediately takes the interrupt. The address of the next sequential instruction is written into SRR0; simultaneously, the contents of the MSR are written into SRR1 and the MSR is written with the values shown in Table 10-19. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x1010. Interrupt processing begins at the address in the program counter.

To clear a FIT interrupt, the interrupt handling routine must clear the FIT interrupt bit, TSR[FIS]. Clearing is performed by writing a word to TSR, using an **mtspr** instruction, that has 1 in any bit positions to be cleared and 0 in all other bit positions. The data written to the TSR is not direct data, but a mask; a 1 clears a bit and 0 has no effect.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 10-19. Register Settings during Fixed Interval Timer Interrupts

SRR0	Written with the address of the next sequential instruction
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] ll 0x1010
TSR	FIS ← 1

10.22 Watchdog Timer Interrupt

For a general description of the PPC405GP timer facilities, see Chapter 11, "Timer Facilities." The watchdog timer (WDT) is described in "Watchdog Timer" on page 11-6.

If the WDT interrupt is enabled by TCR[WIE] and MSR[CE], the PPC405GP initiates a WDT interrupt after detecting the first WDT time-out. First time-out is detected when, at the beginning of a clock cycle, TSR[WIS] = 1. (This occurs on the second cycle after the 0→1 transition of the appropriate time-base bit while TSR[ENW] = 1 and TSR[WIS] = 0.) The PPC405GP immediately takes the interrupt. The address of the next sequential instruction is saved in SRR2; simultaneously, the contents of the MSR are written into SRR3 and the MSR is written with the values shown in Table 10-20. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x1020. Interrupt processing begins at the address in the program counter.

To clear the WDT interrupt, the interrupt handling routine must clear the WDT interrupt bit TSR[WIS]. Clearing is done by writing a word to TSR (using **mtspr**), with a 1 in any bit position that is to be cleared and 0 in all other bit positions. The data written to the status register is not direct data, but a mask; a 1 causes the bit to be cleared, and a 0 has no effect.

Executing the return from critical interrupt instruction (**rfci**) restores the contents of the program counter and the MSR from SRR2 and SRR3, respectively, and the PPC405GP resumes execution at the contents of the program counter.

Table 10-20. Register Settings during Watchdog Timer Interrupts

SRR2	Written with the address of the next sequential instruction
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, DE, DWE, IR, DR ← 0 ME ← unchanged
PC	EVPR[0:15] ‖ 0x1020
TSR	WIS ← 1

10.23 Data TLB Miss Interrupt

The data TLB miss interrupt is generated if data translation is enabled and a valid TLB entry matching the EA and PID is not present. The address of the instruction generating the untranslatable effective data address is saved in SRR0. In addition, the hardware also saves the data address (that missed in the TLB) in the DEAR.

The ESR is set to indicate whether the excepting operation was a store (includes **dcbz**, **dcbi**, **dccci**).

The interrupt is precise. Program flow vectors to EVPR[0:15] ‖ 0x1100.

The following registers are modified to the values specified in Table 10-21.

Table 10-21. Register Settings during Data TLB Miss Interrupts

SRR0	Set to the address of the instruction generating the effective address for which no valid translation exists.
SRR1	Set to the value of the MSR at the time of the interrupt

Table 10-21. Register Settings during Data TLB Miss Interrupts (continued)

MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] ll 0x1100
DEAR	Set to the effective address of the failed access
ESR	DST ← 1 if excepting operation is a store operation (includes **dcbi, dcbz,** and **dccci**). MCI ← unchanged All other bits are cleared.

Programming Note: Data TLB miss interrupts can happen whenever data translation is enabled. Therefore, ensure that SRR0 and SRR1 are saved before enabling translation in an interrupt handler.

10.24 Instruction TLB Miss Interrupt

The instruction TLB miss interrupt is generated if instruction translation is enabled and execution is attempted for an instruction for which a valid TLB entry matching the EA and PID for the instruction fetch is not present. The instruction whose fetch caused the TLB miss is saved in SRR0.

The interrupt is precise with respect to the attempted execution of the instruction. Program flow vectors to EVPR[0:15 ll 0x1200.

The following are modified to the values specified in Table 10-22

Table 10-22. Register Settings during Instruction TLB Miss Interrupts

SRR0	Set to the address of the instruction for which no valid translation exists.
SRR1	Set to the value of the MSR at the time of the interrupt
MSR	AP, APE, WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] ll 0x1200

Programming Note: Instruction TLB miss interrupts can happen whenever instruction translation is active. Therefore, insure that SRR0 and SRR1 are saved before enabling translation in an interrupt handler.

10.25 Debug Interrupt

Debug interrupts can be either *synchronous* or *asynchronous.* These debug events generate synchronous interrupts: branch taken (BT), data address compare (DAC), data value compare (DVC), instruction address compare (IAC), instruction completion (IC), and trap instruction (TIE). The exception (EXC) and unconditional (UDE) debug events generate asynchronous interrupts. See “Debug Events” on page 12-16 for more information about debug events.

For debug events, SRR2 is written with an address, which varies with the type of debug event, as shown in Table 10-23.

Table 10-23. SRR2 during Debug Interrupts

Debug Event	Address Saved in SRR2
BT DAC IAC TIE	Address of the instruction causing the event
DVC IC	Address of the instruction *following* the instruction that causing the event
EXC	Interrupt vector address of the initial exception that caused the exception debug event
UDE	Address of next instruction to be executed at time of UDE

SRR3 is written with the contents of the MSR and the MSR is written with the values shown in Table 10-24, "Register Settings during Debug Interrupts," on page 10-45. The high-order 16 bits of the program counter are then written with the contents of the EVPR; the low-order 16 bits of the program counter are written with 0x2000. Interrupt processing begins at the address in the program counter.

Executing an **rfci** instruction restores the program counter from SRR2 and the MSR from SRR3, and execution resumes at the address in the program counter.

Table 10-24. Register Settings during Debug Interrupts

SRR2	Written with an address as described in Table 10-23
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, DE, DWE, IR, DR ← 0 ME ← unchanged
PC	EVPR[0:15] ll 0x2000
DBSR	Set to indicate type of debug event.

Chapter 11. Timer Facilities

The PPC405GP provides four timer facilities: a time base, a Programmable Interval Timer (PIT), a fixed interval timer (FIT), and a watchdog timer. The PIT is a Special Purpose Register (SPR). These facilities, which are driven by the same base clock, can, among other things, be used for:

- Time-of-day functions
- Data logging functions
- Peripherals requiring periodic service
- Periodic task switching

Additionally, the watchdog timer can help a system to recover from faulty hardware or software.

Figure 11-1 shows the relationship of the timers and the clock source to the time base.

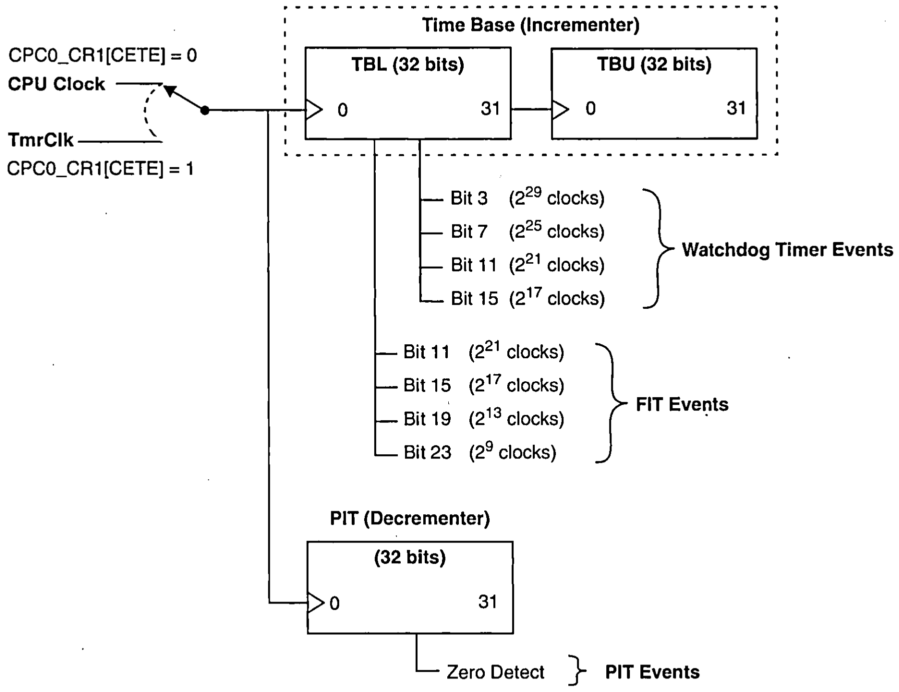


Figure 11-1. Relationship of Timer Facilities to the Time Base

11.1 Time Base

The PPC405GP implements a 64-bit time base as required in *The PowerPC Architecture*. The time base, which increments once during each period of the source clock, provides a time reference. The source clock is either the CPU clock or an external clock connected to the TmrClk input. The CETE field of Chip Control Register 1 (CPC0_CR1[CETE]) selects the clock source.

Read access to the time base is through the **mftb** instruction. **mftb** provides user-mode read-only access to the time base. The TBR numbers (0x10C and 0x10D; TBL and TBU, respectively) that specify the time base registers to **mftb** are not SPR numbers. However, the PowerPC Architecture allows an implementation to handle **mftb** as **mfspr**. Accordingly, these register numbers cannot be used for other SPRs. PowerPC compilers cannot use **mftb** with register numbers other than those specified in the PowerPC Architecture as read-access time base registers (0x10C and 0x10D).

Write access to the time base, using **mtspr**, is privileged. Different register numbers are used for read access and write access. Writing the time base is accomplished by using SPR 0x11C and SPR 0x11D (TBL and TBU, respectively) as operands for **mtspr**.

The period of the 64-bit time base is approximately 2925 years for a 200 MHz clock source. The time base does not generate interrupts, even when it wraps. For most applications, the time base is set once at system reset and only read thereafter. Note that the FIT and the watchdog timer (discussed below) are driven by 0→1 transitions of bits from the TBL. Transitions caused by software alteration of TBL have the same effect as transitions caused by normal incrementing of the time base.

Figure 11-2 illustrates the TBL.

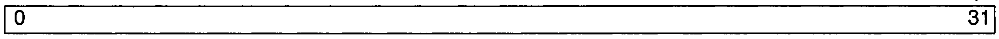


Figure 11-2. Time Base Lower (TBL)

0:31		Time Base Lower	Current count; low-order 32 bits of time base.

Figure 11-3 illustrates the TBU.

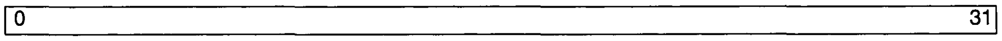


Figure 11-3. Time Base Upper (TBU)

0:31		Time Base Upper	Current count, high-order 32 bits of time base.

Table 11-1 summarizes the TBRs, instructions used to access the TBRs, and access restrictions.

Table 11-1. Time Base Access

	Instructions	**Register Number**	**Access Restrictions**
TBU Upper 32 bits	mftbu RT *Extended mnemonic for* **mftb RT,TBU**	0x10D	Read-only
	mttbu RS *Extended mnemonic for* **mtspr TBU,RS**	0x11D	Privileged; write-only
TBL Lower 32 bits	mftb RT *Extended mnemonic for* **mftb RT,TBL**	0x10C	Read-only
	mttbl *Extended mnemonic for* **mtspr TBL,RS**	0x11C	Privileged; write-only

11.1.1 Reading the Time Base

The following code provides an example of reading the time base. **mftb** moves the low-order 32 bits of the time base to a GPR; **mftbu** moves the high-order 32 bits of the time base to a second GPR.

```
loop:
        mftbu Rx                # load from TBU
        mftb  Ry                # load from TBL
        mftbu Rz                # load from TBU
        cmpw Rz, Rx             # see if old = new
        bne   loop              # loop/reread if rollover occurred
```

The comparison and loop ensure that a consistent pair of values is obtained.

11.1.2 Writing the Time Base

The following code provides an example of writing the time base. Writing the time base is privileged. **mttbl** moves the contents of a GPR to the low-order 32 bits of the time base; **mttbu** moves the contents of a second GPR to the high-order 32 bits of the time base.

```
lwz     Rx, upper                       # load 64-bit time base value into Rx and Ry
lwz     Ry, lower
li      Rz, 0
mttbl   Rz                              # force TBL to 0 to avoid rollover while writing TBU
mttbu   Rx                              # set TBU
mttbl   Ry                              # set TBL
```

11.2 Programmable Interval Timer (PIT)

The PIT is a 32-bit SPR that decrements at the same rate as the time base. The PIT is read and written using **mfspr** and **mtspr**, respectively. Writing to the PIT also simultaneously writes to a hidden reload register. Reading the PIT using **mfspr** returns the current PIT contents; the hidden reload register cannot be read. When a non-zero value is written to the PIT, it begins to decrement. A PIT event occurs when a decrement occurs on a PIT count of 1. When a PIT event occurs, the following occurs:

1. If the PIT is in auto-reload mode (the ARE field of the Timer Control Register (TCR) is 1), the PIT is loaded with the last value an **mtspr** wrote to the PIT. A decrement from a PIT count of 1 immediately causes a reload; no intermediate PIT content of 0 occurs.

   If the PIT is not in auto-reload mode (TCR[ARE] = 0), a decrement from a PIT count of 1 simply causes a PIT content of 0.
2. TSR[PIS] is set to 1.
3. If enabled (TCR[PIE] = 1 and the EE field of the Machine State Register (MSR) is 1), a PIT interrupt is taken. See "Programmable Interval Timer (PIT) Interrupt" on page 10-41 for details of register behavior during a PIT interrupt.

The interrupt handler should use software to reset the PIS field of the Timer Status Register (TSR). This is done by using **mtspr** to write a word to the TSR having a 1 in TSR[PIS] and any other bits to be cleared, and a 0 in all other bits. The data written to the TSR is not direct data, but a mask. A 1 clears a bit; a 0 has no effect.

Using **mtspr** to force the PIT to 0 *does not* cause a PIT interrupt. However, decrementing that was ongoing at the instant of the **mtspr** instruction can cause the appearance of an interrupt. To eliminate the PIT as a source of interrupts, write a 0 to TCR[PIE], the PIT interrupt enable bit.

To eliminate all PIT activity:

1. Write a 0 to TCR[PIE]. This prevents PIT activity from causing interrupts.
2. Write a 0 to TCR[ARE]. This disables the PIT auto-reload feature.
3. Write zeroes to the PIT to halt PIT decrementing. Although this action does not cause a pit PIT interrupt to become pending, a near-simultaneous decrement to 0 might have done so.
4. Write a 1 to TSR[PIS] (PIT Interrupt Status bit). This clears TSR[PIS] to 0 (see "Timer Status Register (TSR)" on page 11-8). This also clears any pending PIT interrupt. Because the PIT stops decrementing, no further PIT events are possible.

If the auto-reload feature is disabled (TCR[ARE] = 0) when the PIT decrements to 0, the PIT remains 0 until software uses **mtspr** to reload it.

After a reset, TCR[ARE] = 0, which disables the auto-reload feature.

Figure 11-4 illustrates the PIT.

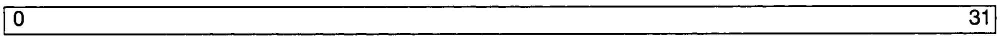


Figure 11-4. Programmable Interval Timer (PIT)

0:31		Programmed interval remaining	Number of clocks remaining until the PIT event

11.2.1 Fixed Interval Timer (FIT)

The FIT provides timer interrupts having a repeatable period. The FIT is functionally similar to an auto-reload PIT, except that only a smaller fixed selection of interrupt periods are available.

The FIT exception occurs on 0→1 transitions of selected bits from the time base, as shown in Table 11-2.

Table 11-2. FIT Controls

TCR[FP]	TBL Bit	Period (Time Base Clocks)	Period (200 Mhz Clock)
0, 0	23	2^9 clocks	2.56 µsec
0, 1	19	2^{13} clocks	40.96 µsec
1, 0	15	2^{17} clocks	0.655 msec
1, 1	11	2^{21} clocks	10.49 msec

The TSR[FIS] field logs a FIT exception as a pending interrupt. A FIT interrupt occurs if TCR[FIE] and MSR[EE] are enabled at the time of the FIT exception. “Fixed Interval Timer (FIT) Interrupt” on page 10-42 describes register settings during a FIT interrupt.

The interrupt handler should reset TSR[FIS]. This is done by using **mtspr** to write a word to the TSR having a 1 in TSR[FIS] and any other bits to be cleared, and a 0 in all other bits. The data written to the TSR is not direct data, but a mask. A 1 clears a bit and a 0 has no effect.

11.3 Watchdog Timer

The watchdog timer aids system recovery from software or hardware faults.

A watchdog timeout occurs on 0→1 transitions of a selected bit from the time base, as shown in the following table.

Table 11-3. Watchdog Timer Controls

TCR[WP]	TBL Bit	Period (Time Base Clocks)	Period (200 MHz Clock)
0,0	15	2^{17} clocks	0.655 msec
0,1	11	2^{21} clocks	10.49 msec
1,0	7	2^{25} clocks	0.168 sec
1,1	3	2^{29} clocks	2.684 sec

If a watchdog timeout occurs while TSR[WIS] = 0 and TSR[ENW] = 1, a watchdog interrupt occurs if the interrupt is enabled by TCR[WIE] and MSR[CE]. "Watchdog Timer" on page 11-6 describes register behavior during a watchdog interrupt.

The interrupt handler should reset the TSR[WIS] bit. This is done by using **mtspr** to write a word to the TSR having a 1 in TSR[WIS] and any other bits to be cleared, and a 0 in all other bits. The data written to the TSR is not direct data, but a mask. A 1 clears a bit and a 0 has no effect.

If a watchdog timeout occurs while TSR[WIS] = 1 and TSR[ENW] = 1, a hardware reset occurs if enabled by a non-zero value of TCR[WRC]. In other words, a reset can occur if a watchdog timeout occurs while a previous watchdog timeout is pending. The assumption is that TSR[WIS] was not cleared because the processor could not execute the watchdog handler, leaving reset as the only way to restart the system. Note that after TCR[WRC] is set to a non-zero value, it cannot be reset by software. This prevents errant software from disabling the watchdog timer reset capability. After a reset, the initial value of TCR[WRC] = 00.

Figure 11-5 describes the watchdog state machine. In the figure, numbers in parentheses refer to descriptions of operating modes that follow the table.

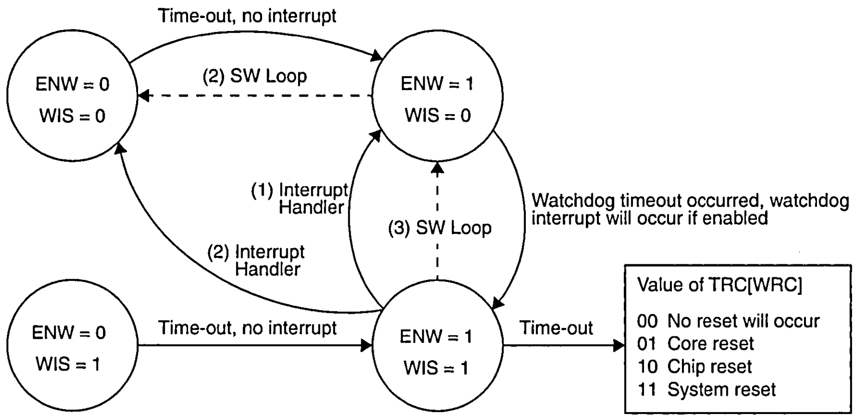


Figure 11-5. Watchdog Timer State Machine

Enable Next Watchdog TSR[ENW]	Watchdog Timer Status TSR[WIS]	Action When Timer Interval Expires
0	0	Set TSR[ENW] = 1.
0	1	Set TSR[ENW] = 1.
1	0	Set TSR[WIS] = 1. If TCR[WIE] = 1 and MSR[CE] = 1, then interrupt.
1	1	Cause the watchdog reset action specified by TCR[WRC]. On reset, copy current TCR[WRC] to TSR[WRS] and clear TCR[WRC], disabling the watchdog timer.

The controls described in Figure 11-5 imply three different ways of using the watchdog timer. The modes assume that TCR[WRC] was set to allow processor reset by the watchdog timer:

1. Always take a pending watchdog interrupt, and never attempt to prevent its occurrence. (This mode is described in the preceding text.)

   a. Clear TSR[WIS] in the watchdog timer handler.

   b. Never use TSR[ENW].

2. Always take a pending watchdog interrupt, but avoid it whenever possible by delaying a reset until a second watchdog timer occurs.

   This assumes that a recurring code loop of known maximum duration exists outside the interrupt handlers, or that a FIT interrupt handler is operational. One of these mechanisms clears TSR[ENW] more frequently than the watchdog period.

   a. Clear TSR[ENW] to 0 in loop or in FIT interrupt handler.

To clear TSR[ENW], use **mtspr** to write a 1 to TSR[ENW] (and to any other bits that are to be cleared), with 0 in all other bit locations.

b. Clear TSR[WIS] in watchdog timer handler.

It is not expected that a watchdog interrupt will occur every time, but only if an exceptionally high execution load delays clearing of TSR[ENW] in the usual time frame.

3. Never take a watchdog interrupt.

This assumes that a recurring code loop of reliable duration exists outside the interrupt handlers, or that a FIT interrupt handler is operational. This method only guarantees one watchdog timeout period before a reset occurs.

a. Clear TSR[WIS] in the loop or in FIT handler.

b. Never use TSR[ENW] but have it set.

11.4 Timer Status Register (TSR)

The TSR can be accessed for read or write-to-clear.

Status registers are generally set by hardware and read and cleared by software. The **mfspr** instruction reads the TSR. Clearing the TSR is performed by writing a word to the TSR, using **mtspr,** having a 1 in all fields to be cleared and a 0 in all other fields. The data written to the TSR is not direct data, but a mask. A 1 clears the field and a 0 has no effect.

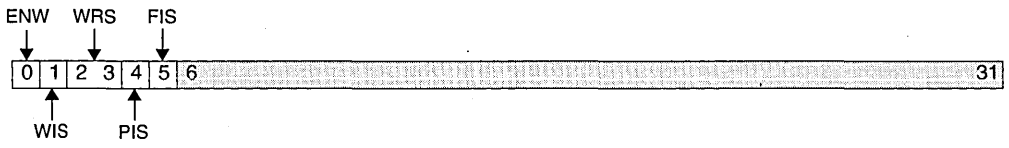


Figure 11-6. Timer Status Register (TSR)

0	ENW	Enable Next Watchdog 0 Action on next watchdog event is to set TSR[ENW] = 1. 1 Action on next watchdog event is governed by TSR[WIS].	Software must reset TSR[ENW] = 0 after each watchdog timer event.
1	WIS	Watchdog Interrupt Status 0 No Watchdog interrupt is pending. 1 Watchdog interrupt is pending.	
2:3	WRS	Watchdog Reset Status 00 No Watchdog reset has occurred. 01 Core reset was forced by the watchdog. 10 Chip reset was forced by the watchdog. 11 System reset was forced by the watchdog.	
4	PIS	PIT Interrupt Status 0 No PIT interrupt is pending. 1 PIT interrupt is pending.	

5	FIS	FIT Interrupt Status 0 No FIT interrupt is pending. 1 FIT interrupt is pending.
6:31		Reserved

11.5 Timer Control Register (TCR)

The TCR controls PIT, FIT, and watchdog timer operation.

The TCR[WRC] field is cleared to 0 by all processor resets. (Chapter 8, “Reset and Initialization,” describes the types of processor reset.) This field is set only by software. However, hardware does not allow software to clear the field after it is set. After software writes a 1 to a bit in the field, that bit remains a 1 until any reset occurs. This prevents errant code from disabling the watchdog timer reset function.

All processor resets clear TCR[ARE] to 0, disabling the auto-reload feature of the PIT.

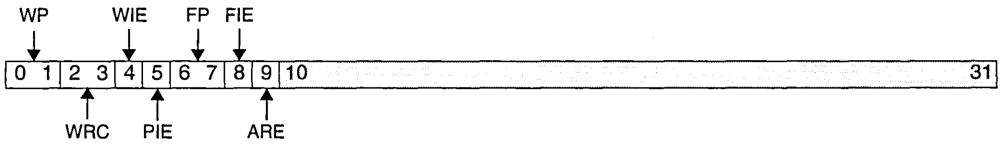


Figure 11-7. Timer Control Register (TCR)

0:1	WP	Watchdog Period 00 2^{17} clocks 01 2^{21} clocks 10 2^{25} clocks 11 2^{29} clocks	
2:3	WRC	Watchdog Reset Control 00 No Watchdog reset will occur. 01 Core reset will be forced by the Watchdog. 10 Chip reset will be forced by the Watchdog. 11 System reset will be forced by the Watchdog.	TCR[WRC] resets to 00. This field can be set by software, but cannot be cleared by software, except by a software-induced reset.
4	WIE	Watchdog Interrupt Enable 0 Disable watchdog interrupt. 1 Enable watchdog interrupt.	
5	PIE	PIT Interrupt Enable 0 Disable PIT interrupt. 1 Enable PIT interrupt.	

6:7	FP	FIT Period 00 2^9 clocks 01 2^{13} clocks 10 2^{17} clocks 11 2^{21} clocks	
8	FIE	FIT Interrupt Enable 0 Disable FIT interrupt. 1 Enable FIT interrupt.	
9	ARE	Auto Reload Enable 0 Disable auto reload. 1 Enable auto reload.	Disables on reset.
10:31		Reserved	

Chapter 12. Debugging

The debug facilities of the PPC405GP include support for debug modes for debugging during hardware and software development, and debug events that allow developers to control the debug process. Debug registers control the debug modes and debug events. The debug registers are accessed through software running on the processor or through a JTAG debug port. The debug interface is the JTAG debug port. The JTAG debug port can also be used for board test.

The debug modes, events, controls, and interface provide a powerful combination of debug facilities for a wide range of hardware and software development tools.

12.1 Development Tool Support

The RISCWatch product from IBM is an example of a development tool that uses the external debug mode, debug events, and the JTAG debug port to implement a hardware and software development tool. The RISCTrace™ feature of RISCWatch is an example of a development tool that uses the real-time instruction trace capability of the PPC405GP.

12.2 Debug Interfaces

The PPC405GP provides JTAG and trace interfaces to support hardware and software test and debug. Typically, the JTAG interface connects to a debug port external to the PPC405GP; the debug port is typically connected to a JTAG connector on a processor board.

The trace interface connects to a trace port, also external to the PPC405GP, that is typically connected to a trace connector on the processor board.

12.3 IEEE 1149.1 Test Access Port (JTAG Debug Port)

The IEEE 1149.1 Test Access Port (TAP), commonly called the JTAG (Joint Test Action Group) debug port, is an architectural standard described in IEEE Std 1149.1–1990, *IEEE Standard Test Access Port and Boundary Scan Architecture*. The standard describes a method for accessing internal chip facilities using a four- or five-signal interface.

The JTAG debug port, originally designed to support scan-based board testing, is enhanced to support the attachment of debug tools. The enhancements, which comply with the IEEE 1149.1 specifications for vendor-specific extensions, are compatible with standard JTAG hardware for boundary-scan system testing.

JTAG Signals	The JTAG debug port implements the four required JTAG signals: TCK, TMS, TDI, and TDO, and the optional $\overline{\text{TRST}}$ signal.
JTAG Clock Requirements	The frequency of the TCK signal can range from DC to one-half of the internal chip clock frequency.
JTAG Reset Requirements	The JTAG debug port logic is reset at the same time as a system reset. Upon receiving $\overline{\text{TRST}}$, the JTAG debug port returns to the Test-Logic Reset state.

12.4 JTAG Connector

A 16-pin male 2x8 header connector is suggested as the JTAG debug port connector. This connector definition matches the requirements of the RISCWatch debugger from IBM. The connector is shown in Figure 12-1 and the signals are shown in Table 12-1. The connector should be placed as close as possible to the chip to ensure signal integrity.

Note that position 14 does not contain a pin.

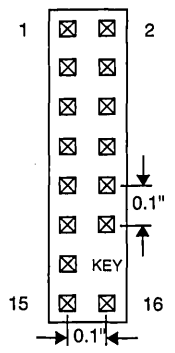


Figure 12-1. JTAG Connector Physical Layout (Top View)

Table 12-1. JTAG Connector Signals

Pin	I/O	Signal	Description
1	O	TDO	JTAG Test Data Out
2		No connect (NC)	Reserved
3	I	TDI[1]	JTAG Test Data In
4		$\overline{\text{TRST}}$	JTAG Reset
5		NC	Reserved
6		+POWER[2]	Processor I/O Voltage
7	I	TCK[3]	JTAG Test Clock
8		NC	Reserved
9	I	TMS[1]	JTAG Test Mode Select
10		NC	Reserved
11	I	HALT[3]	Processor Halt
12		NC	Reserved
13		NC	Reserved
14		Key	The pin at this position should be removed.
15		NC	Reserved
16		GND	Ground

1. A 10K ohm pullup resistor should be connected to this signal to reduce chip power consumption. The pullup resistor is not required.
2. The +POWER signal, sourced from the target development board, indicates whether the I/O voltage at which the processor is operating. This signal does not supply power to the RISCWatch hardware or to the processor. The active level on this signal can be +5V or +3.3V (note that the PPC405GP can have either +5V or +3.3V I/O, but the processor itself must be powered by +3.3V). A series resistor (1K ohm or less) should be used to provide short circuit current-limiting protection.
3. A 10K ohm pullup resistor must be connected to these signals to ensure proper chip operation when these inputs are not used.

12.4.1 JTAG Instructions

The JTAG debug port provides the standard *extest*, *idcode*, *sample/preload*, and *bypass* instructions and the optional *highz* and *clamp* instructions. Invalid instructions behave as the *bypass* instruction.

Table 12-2. JTAG Instructions

Instruction	Code	Comments
Extest	1111000	IEEE 1149.1 standard.
▨	1111001	Reserved.
Sample/Preload	1111010	IEEE 1149.1 standard.
IDCode	1111011	IEEE 1149.1 standard.
Private	xxxx100	Private instructions
HighZ	1111101	IEEE 1149.1a-1993 optional
Clamp	1111110	IEEE 1149.1a-1993 optional
Bypass	1111111	IEEE 1149.1 standard.

12.4.2 JTAG Boundary Scan

Boundary Scan Description Language (BSDL), IEEE 1149.1b-1994, is a supplement to IEEE 1149.1-1990 and IEEE 1149.1a-1993 *Standard Test Access Port and Boundary-Scan Architecture.* BSDL, a subset of the IEEE 1076-1993 Standard VHSIC Hardware Description Language (VHDL), allows a rigorous description of testability features in components which comply with the standard. BSDL is used by automated test pattern generation tools for package interconnect tests and by electronic design automation (EDA) tools for synthesized test logic and verification. BSDL supports robust extensions that can be used for internal test generation and to write software for hardware debug and diagnostics.

The primary components of BSDL include the logical port description, the physical pin map, the instruction set, and the boundary register description.

The logical port description assigns symbolic names to the pins of a chip. Each pin has a logical type of in, out, inout, buffer, or linkage that defines the logical direction of signal flow.

The physical pin map correlates the logical ports of the chip to the physical pins of a specific package. A BSDL description can have several physical pin maps; each map is given a unique name.

Instruction set statements describe the bit patterns that must be shifted into the Instruction Register to place the chip in the various test modes defined by the standard. Instruction set statements also support descriptions of instructions that are unique to the chip.

The boundary register description lists each cell or shift stage of the Boundary Register. Each cell has a unique number: the cell numbered 0 is the closest to the Test Data Out (TDO) pin; the cell with the highest number is closest to the Test Data In (TDI) pin. Each cell contains additional information, including: cell type, logical port associated with the cell, logical function of the cell, safe value, control cell number, disable value, and result value.

12.4.3 JTAG Implementation

PPC405GP JTAG interface I/Os (TDI, TDO, TMs, TCK, and $\overline{\text{TRST}}$) are 5V tolerant and do not contain internal pull up resistors.

The optional JTAG instructions, *idcode* and *highz*, offer additional JTAG functionality. The *idcode* instruction returns the PPC405GP JTAG ID, which is unique for each chip version. The *highz* instruction disables all chip outputs regardless of whether they are included in the JTAG boundary scan chain.

The PPC405GP provides boundary scan structures on most I/O signals. However, the following signals are excluded because of speed and functional considerations:

- DrvrInh1
- DrvrInh2
- PciClk
- RcvrInh
- TestEn

12.4.4 JTAG ID Register (CPC0_JTAGID)

CPC0_JTAGID is a Device Control Register that enables manufacturing, part number, and version information to be determined through the TAP. The **mfdcr** instruction is used to read this register.

Refer to *PowerPC 405GP Embedded Processor Data Sheet* for the values of the CPC0_JTAGID fields.

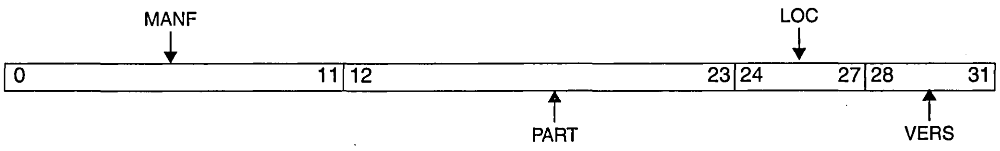


Figure 12-2. JTAG ID Register (CPC0_JTAGID)

0:11	MANF	Manufacturer Identifier
12:23	PART	Part Number
24:27	LOC	Developer Location
28:31	VERS	Version

12.5 Trace Port

The PPC405GP implements a trace status interface to support the tracing of code running in real-time. This interface enables the connection of an external trace tool, such as RISCWatch, and allows for user-extended trace functions. A software tool with trace capability, such as RISCWatch with RISCTrace, can use the data collected from this port to trace code running on the processor. The result is a trace of the code executed, including code executed out of the instruction cache if it was enabled. Information on trace capabilities, how trace works, and how to connect the external trace tool is available in *RISCWatch Debugger User's Guide*.

A 20-pin male 2x10 header connector is recommended for connecting to the trace status port of the PPC405GP. The connector, shown in Figure 12-3, and the signal descriptions in Table 12-3 match the requirements of RISCTrace, when used with the RISCWatch processor probe with RISCTrace option.

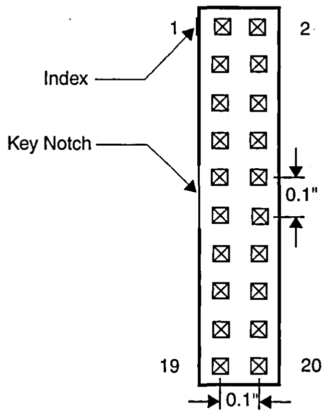


Figure 12-3. RISCTrace Header (Top View)

Table 12-3 describes the assignment of the RISCTrace signals and the processor core outputs to the header pins:

Table 12-3. RISCTrace Header Pin Description

Pin	Signal Name	Pin	Signal Name
1	No connect	11	No connect
2	No connect	12	TS1o (odd)
3	TrcClk	13	TS2o
4	No connect	14	TS1e (even)
5	No connect	15	TS2e
6	No connect	16	TS3
7	No connect	17	TS4
8	No connect	18	TS5
9	No connect	19	TS6

Table 12-3. RISCTrace Header Pin Description (continued)

Pin	Signal Name	Pin	Signal Name
10	No connect	20	GND

12.6 Debug Modes

The PPC405GP supports the following debug modes, each of which supports a type of debug tool or debug task commonly used in embedded systems development:

- Internal debug mode, which supports ROM monitors
- External debug mode, which supports JTAG debuggers
- Debug wait mode, which supports processor stopping or stepping for JTAG debuggers while servicing interrupts
- Real-time trace mode, which supports trigger events for real-time tracing

Internal and external debug modes can be enabled simultaneously. Both modes are controlled by fields in Debug Control Register 0 (DBCR0). Real-time trace mode is available only if internal, external, and debug wait modes are disabled.

12.6.1 Internal Debug Mode

Internal debug mode provides access to architected processor resources and supports setting hardware and software breakpoints and monitoring processor status. In this mode, debug events generate debug interrupts, which can interrupt normal program flow so that monitor software can collect processor status and alter processor resources.

Internal debug mode relies on exception handling software at a dedicated interrupt vector and an external communications path to debug software problems. This mode, used while the processor executes instructions, enables debugging of operating system or application programs.

In this mode, debugger software is accessed through a communications port, such as a serial port, external to the processor core.

To enable internal debug mode, the Debug Control Register 0 (DBCR0) field IDM is set to 1 (DBCR0[IDM] = 1). To enable debug interrupts, MSR[DE] = 1. A debug interrupt occurs on a debug event only if DBCR0[IDM] = 1 and MSR[DE] = 1.

12.6.2 External Debug Mode

External debug mode provides access to architected processor resources and supports stopping, starting, and stepping the processor, setting hardware and software breakpoints, and monitoring processor status. In this mode, debug events cause the processor to become architecturally frozen. While the processor is frozen, normal instruction execution stops and architected processor resources can be accessed and altered. External bus activity continues in external debug mode.

The JTAG mechanism can pass instructions to the processor for execution, allowing a JTAG debugger to display and alter processor resources, including memory.

The JTAG mechanism prevents the occurrence of a privileged exception when a privileged instruction is executed while the processor is in user mode.

Storage access control by a memory management unit (MMU) remains in effect while in external debug mode; the debugger may need to modify MSR or TLB values to access protected memory.

Because external debug mode relies only on internal processor resources, it can be used to debug system hardware and software.

In this mode, access to the processor is through the JTAG debug port.

To enable external debug mode, DBCR0[EDM] = 1. To enable debug interrupts, MSR[DE] = 1. A debug interrupt occurs on a debug event only if DBCR0[EDM] = 1 and MSR[DE] = 1.

12.6.3 Debug Wait Mode

In debug wait mode, debug events cause the PPC405GP to enter a state in which interrupts can be serviced while the processor appears to be stopped.

Debug wait mode provides access to architected processor resources in a manner similar to external debug mode, except that debug wait mode allows the servicing of interrupt handlers. It supports stopping, starting, and stepping the processor, setting hardware and software breakpoints, and monitoring processor status. In this mode, if a debug event caused the processor to become architecturally frozen, an interrupt causes the processor to run an interrupt handler and return to the architecturally frozen state upon returning from the interrupt handler. While the processor is frozen, normal instruction execution stops and architected processor resources can be accessed and altered. External bus activity continues in debug wait mode.

The processor enters debug wait mode when internal and external debug modes are disabled (DBCR0[IDM, EDM] = 0), debug wait mode is enabled (MSR[DWE] = 1), debug wait is enabled by the JTAG debugger, and a debug event occurs.

For example, while the PPC405GP is in debug wait mode, an external device might generate an interrupt that requires immediate service. The PPC405GP can service the interrupt (vector to an interrupt handler and execute the interrupt handler code) and return to the previous stopped state.

Debug wait mode relies only on internal processor resources, so it can be used to debug both system hardware and software problems. This mode can also be used for software development on systems without a control program, or to debug control program problems.

In this mode, access to the processor is through the JTAG debug port.

12.6.4 Real-time Trace Debug Mode

Real-time trace debug mode supports the generation of trigger events for tracing the instruction stream being executed out of the instruction cache in real-time. In this mode, debug events can be used to control the collection of trace information through the use of trigger event generation. The broadcast of trace information is independent of the use of debug events as trigger events.This mode does not alter the processor performance.

A trace event occurs when internal and external debug modes are disabled (DBCR0[IDM, EDM] = 0) and a debug events occurs.

When a trace event occurs, a trace device can capture trace signals that provide the instruction trace information. Most trace events generated from debug events are blocked when internal debug, external debug, or debug wait modes are enabled

12.7 Processor Control

The PPC405GP provides the following debug functions for processor control. Not all facilities are available in all debug modes.

Instruction Step	The processor is stepped one instruction at a time, while stopped, using the JTAG debug port.
Instruction Stuff	While the processor is stopped, instructions can be stuffed into the processor and executed using the JTAG debug port.
Halt	The processor can be stopped by activating an external halt signal on an external event, such as a logic analyzer trigger. This signal freezes the processor architecturally. While frozen, normal instruction execution stops and architected processor resources can be accessed and altered using the JTAG debug port. Normal execution resumes when the halt signal is deactivated.
Stop	The processor can be stopped using the JTAG debug port. Activating a stop causes the processor to become architecturally frozen. While frozen, normal instruction execution stops and the architected processor resources can be accessed and altered using the JTAG debug port.
Reset	An external reset signal, the JTAG debug port, or DBCR0 can request core, chip, and system resets.
Debug Events	A debug event triggers a debug operation. The operation depends on the debug mode. For more information and a list of debug events, see "Debug Events" on page 12-16.
Freeze Timers	The JTAG debug port or DBCR0 can control timer resources. The timers can be enabled to run, freeze always, or freeze on a debug event.
Trap Instructions	The trap instructions **tw** and **twi** can be used, with debug events, to implement software breakpoints.

12.8 Processor Status

The processor execution status, exception status, and most recent reset can be monitored.

Execution Status	The JTAG debug port can monitor processor execution status to determine whether the processor is stopped, waiting, or running.
Exception Status	The JTAG debug port can monitor the status of pending synchronous exceptions.
Most Recent Reset	The JTAG debug port or an **mfspr** instruction can be used to read the Debug Status Register (DBSR) to determine the type of the most recent reset.

12.9 Debug Registers

Several debug registers, available to debug tools running on the processor, are not intended for use by application code. Debug tools control debug resources such as debug events. Application code that uses debug resources can cause the debug tools to fail, as well as other unexpected results, such as program hangs and processor resets.

Application code should not use the debug resources, including the debug registers.

12.9.1 Debug Control Registers

The debug control registers (DBCR0 and DBCR1) can enable and configure debug events, reset the processor, control timer operation during debug events, enable debug interrupts, and set the processor debug mode.

12.9.1.1 Debug Control Register 0 (DBCR0)

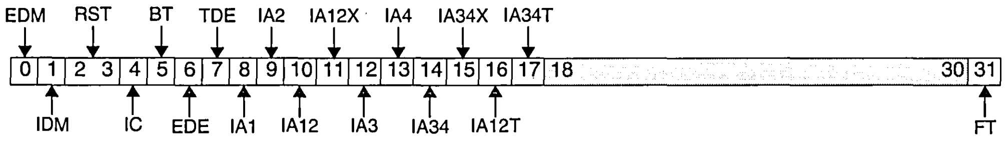


Figure 12-4. Debug Control Register 0 (DBCR0)

0	EDM	External Debug Mode 0 Disabled 1 Enabled	
1	IDM	Internal Debug Mode 0 Disabled 1 Enabled	
2:3	RST	Reset 00 No action 01 Core reset 10 Chip reset 11 System reset	Causes a processor reset request when set by software.
		Attention: Writing 01, 10, or 11 to this field causes a processor reset request.	
4	IC	Instruction Completion Debug Event 0 Disabled 1 Enabled	
5	BT	Branch Taken Debug Event 0 Disabled 1 Enabled	
6	EDE	Exception Debug Event 0 Disabled 1 Enabled	
7	TDE	Trap Debug Event 0 Disabled 1 Enabled	
8	IA1	IAC 1 Debug Event 0 Disabled 1 Enabled	
9	IA2	IAC 2 Debug Event 0 Disabled 1 Enabled	

10	IA12	Instruction Address Range Compare 1–2 0 Disabled 1 Enabled	Registers IAC1 and IAC2 define an address range used for IAC address comparisons.
11	IA12X	Enable Instruction Address Exclusive Range Compare 1–2 0 Inclusive 1 Exclusive	Selects the range defined by IAC1 and IAC2 to be inclusive or exclusive.
12	IA3	IAC 3 Debug Event 0 Disabled 1 Enabled	
13	IA4	IAC 4 Debug Event 0 Disabled 1 Enabled	
14	IA34	Instruction Address Range Compare 3–4 0 Disabled 1 Enabled	Registers IAC3 and IAC4 define an address range used for IAC address comparisons.
15	IA34X	Instruction Address Exclusive Range Compare 3–4 0 Inclusive 1 Exclusive	Selects range defined by IAC3 and IAC4 to be inclusive or exclusive.
16	IA12T	Instruction Address Range Compare 1-2 Toggle 0 Disabled 1 Enable	Toggles range 12 inclusive, exclusive DBCR[IA12X] on debug event.
17	IA34T	Instruction Address Range Compare 3–4 Toggle 0 Disabled 1 Enable	Toggles range 34 inclusive, exclusive DBCR[IA34X] on debug event.
18:30	▨	Reserved	
31	FT	Freeze timers on debug event 0 Timers not frozen 1 Timers frozen	

12.9.1.2 Debug Control Register1 (DBCR1)

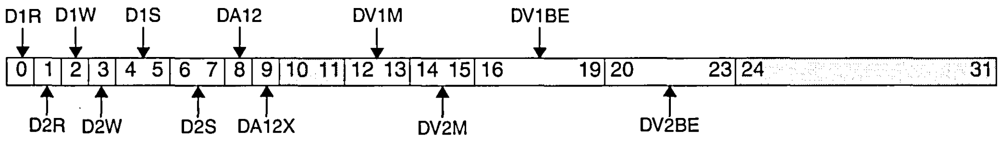


Figure 12-5. Debug Control Register 1 (DBCR1)

0	D1R	DAC1 Read Debug Event 0 Disabled 1 Enabled	
1	D2R	DAC 2 Read Debug Event 0 Disabled 1 Enabled	
2	D1W	DAC 1 Write Debug Event 0 Disabled 1 Enabled	
3	D2W	DAC 2 Write Debug Event 0 Disabled 1 Enabled	
4:5	D1S	DAC 1 Size 00 Compare all bits 01 Ignore lsb (least significant bit) 10 Ignore two lsbs 11 Ignore five lsbs	Address bits used in the compare: Byte address Halfword address Word address Cache line (8-word) address
6:7	D2S	DAC 2 Size 00 Compare all bits 01 Ignore lsb (least significant bit) 10 Ignore two lsbs 11 Ignore five lsbs	Address bits used in the compare: Byte address Halfword address Word address Cache line (8-word) address
8	DA12	Enable Data Address Range Compare 1:2 0 Disabled 1 Enabled	Registers DAC1 and DAC2 define an address range used for DAC address comparisons
9	DA12X	Data Address Exclusive Range Compare 1:2 0 Inclusive 1 Exclusive	Selects range defined by DAC1 and DAC2 to be inclusive or exclusive
10:11	▨	Reserved	

12:13	DV1M	Data Value Compare 1 Mode 00 Undefined	Type of data comparison used:
		01 AND	All bytes selected by DBCR1[DV1BE] must compare to the appropriate bytes of DVC1.
		10 OR	One of the bytes selected by DBCR1[DV1BE] must compare to the appropriate bytes of DVC1.
		11 AND-OR	The upper halfword or lower halfword must compare to the appropriate halfword in DVC1. When performing halfword compares set DBCR1[DV1BE] = 0011, 1100, or 1111.
14:15	DV2M	Data Value Compare 2 Mode 00 Undefined	Type of data comparison used
		01 AND	All bytes selected by DBCR1[DV2BE] must compare to the appropriate bytes of DVC2.
		10 OR	One of the bytes selected by DBCR1[DV2BE] must compare to the appropriate bytes of DVC2.
		11 AND-OR	The upper halfword or lower halfword must compare to the appropriate halfword in DVC2. When performing halfword compares set DBCR1[DV2BE] = 0011, 1100, or 1111.
16:19	DV1BE	Data Value Compare 1 Byte 0 Disabled 1 Enabled	Selects which data bytes to use in data value comparison
20:23	DV2BE	Data Value Compare 2 Byte 0 Disabled 1 Enabled	Selects which data bytes to use in data value comparison
24:31		Reserved	

12.9.2 Debug Status Register (DBSR)

The DBSR contains status on debug events and the most recent reset; the status is obtained by reading the DBSR. The status bits are normally set by debug events or by any of the three reset types.

Clearing DBSR fields is performed by writing a word to the DBSR, using the **mtdbsr** extended mnemonic, having a 1 in all bit positions to be cleared and a 0 in the all other bit positions. The data written to the DBSR is not direct data, but a mask. A 1 clears the bit and a 0 has no effect.

Application code should not use the DBSR.

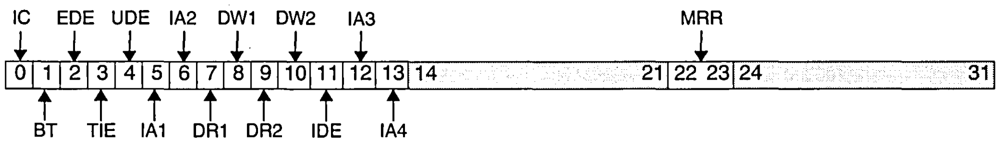


Figure 12-6. Debug Status Register (DBSR)

0	IC	Instruction Completion Debug Event 0 Event did not occur 1 Event occurred
1	BT	Branch Taken Debug Event 0 Event did not occur 1 Event occurred
2	EDE	Exception Debug Event 0 Event did not occur 1 Event occurred
3	TIE	Trap Instruction Debug Event 0 Event did not occur 1 Event occurred
4	UDE	Unconditional Debug Event 0 Event did not occur 1 Event occurred
5	IA1	IAC1 Debug Event 0 Event did not occur 1 Event occurred
6	IA2	IAC2 Debug Event 0 Event did not occur 1 Event occurred
7	DR1	DAC1 Read Debug Event 0 Event did not occur 1 Event occurred
8	DW1	DAC1 Write Debug Event 0 Event did not occur 1 Event occurred
9	DR2	DAC2 Read Debug Event 0 Event did not occur 1 Event occurred
10	DW2	DAC2 Write Debug Event 0 Event did not occur 1 Event occurred

11	IDE	Imprecise Debug Event 0 No circumstance that would cause a debug event (if MSR[DE] = 1) occurred 1 A debug event would have occurred, but debug exceptions were disabled (MSR[DE] = 1)	
12	IA3	IAC3 Debug Event 0 Event did not occur 1 Event occurred	
13	IA4	IAC4 Debug Event 0 Event did not occur 1 Event occurred	
14:21		Reserved	
22:23	MRR	Most Recent Reset 00 No reset has occurred since last cleared by software. 01 Core reset 10 Chip reset 11 System reset	This field is set to a value, indicating the type of reset, when a reset occurs.
24:31		Reserved	

12.9.3 Instruction Address Compare Registers (IAC1–IAC4)

The PPC405GP can take a debug event upon an attempt to execute an instruction from an address. The address, which must be word-aligned, is defined in an IAC register. The DBCR0[IA1, IA2] fields of DBCR0 controls the instruction address compare (IAC) debug event.

0 29	30 31

Figure 12-7. Instruction Address Compare Registers (IAC1–IAC4)

0:29		Instruction Address Compare word address	Omit two low-order bits of complete address.
30:31		Reserved	

12.9.4 Data Address Compare Registers (DAC1–DAC2)

The PPC405GP can take a debug event upon storage or cache references to addresses specified in the DAC registers. The specified addresses in the DAC registers are EAs of operands of storage references or cache instructions.The fields DBCR1[D1R], [D2R] and DBCR[D1W], [D2W] control the DAC-read and DAC-write debug events, respectively.

Addresses in the DAC registers specify exact byte EAs for DAC debug events. However, one may want to take a debug event on any byte within a halfword (ignore the least significant bit (LSb) of the DAC), on any byte within a word (ignore the two LSbs of DAC), or on any byte within eight words (ignore four LSbs of DAC). DBCR1[D1S, D2S] control the addressing options.

Errors related to execution of storage reference or cache instructions prevent DAC debug events.

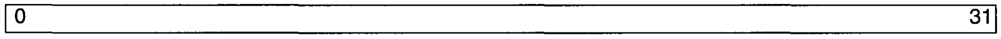


Figure 12-8. Data Address Compare Registers (DAC1–DAC2)

0:31		Data Address Compare (DAC) byte address	DBCR0[D1S] determines which address bits are examined.

12.9.5 Data Value Compare Registers (DVC1–DVC2)

The PPC405GP can take a debug event upon storage or cache references to addresses specified in the DAC registers, that also require the data at that address to match the value specified in the DVC registers. The data address compare for a DVC events works the same as for a DAC event. Cache operations do not cause DVC events. If the data at the address specified matches the value in the corresponding DVC register a DVC event will occur. The fields DBCR1[DV1M, DV2M] control how the data value are compared.

Errors related to execution of storage reference or cache instructions prevent DVC debug events.

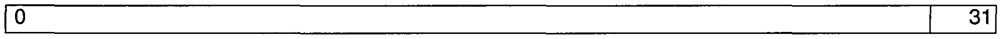


Figure 12-9. Data Value Compare Registers (DVC1–DVC2)

0:31		Data Value to Compare

12.9.6 Debug Events

Debug events, enabled and configured by DBCR0 and DBCR1 and recorded in the DBSR, cause debug operations. A debug event occurs when an event listed in Table 12-4, "Debug Events," on page 12-16 is detected. The debug operation is performed after the debug event.

In internal debug mode, the processor generates a debug interrupt when a debug event occurs. In external debug mode, the processor stops when a debug event occurs. When internal and external debug mode are both enabled, the processor stops on a debug event with the debug interrupt pending. When external and internal debug mode are both disabled, and debug wait mode is enabled the processor stops, but can be restarted by an interrupt. When all debug modes are disabled, debug events are recorded in the DBSR, but no action is taken.

Table 12-4 lists the debug events and the related fields in DBCR0, DBCR1, and DBSR. DBCR0 and DBCR1 enable the debugs events, and the DBSR fields report their occurrence.

Table 12-4. Debug Events

Event	Enabling DBCR0, DBCR1 Fields	Reporting DBSR Fields	Description
Instruction Completion	IC	IC	Occurs after completion of an instruction.
Branch Taken	BT	BT	Occurs before execution of a branch instruction determined to be taken.
Exception Taken	EDE	EXC	Occurs after an exception.
Trap Instruction	TDE	TIE	Occurs before execution of a trap instruction where the conditions are such that the trap will occur.
Unconditional	UDE	UDE	Occurs immediately upon being set by the JTAG debug port.
Instruction Address Compare	IA1, IA2, IA3, IA4, IA12, IA12X, IA12T, IA34, IA34X, IA34T	IA1, IA2, IA3, IA4	Occurs before execution of an instruction at an address that matches an address defined by the Instruction Address Compare Registers (IAC1–IAC4).
Data Address Compare	D1R, D1W, D1S, D2R, D2W, D2S, DA12, DA12X	DR2,DW2	Occurs before execution of an instruction that accesses a data address that matches the contents of the specified DAC register.
Data Value Compare	DV1M, DV2M, DV1BE, DV2BE	DR1, DW1	Occurs after execution of an instruction that accesses a data address for which a DAC occurs, and for which the value at the address matches the value in the specified DVC register.
Imprecise		IDE	Indicates that another debug event occurred while MSR[DE] = 0

12.9.7 Instruction Complete Debug Event

This debug event occurs after the completion of an instruction. If DBCR0[IDM] = 1, DBCR0[EDM] = 0 and MSR[DE] =0 this debug event is disabled.

12.9.8 Branch Taken Debug Event

This debug event occurs before execution of a branch instruction determined to be taken. If DBCR0[IDM] = 1, DBCR0[EDM] = 0 and MSR[DE] =0 this debug event is disabled.

12.9.9 Exception Taken Debug Event

This debug event occurs after an exception. Exception debug events always include the non-critical class of exceptions. When DBCR0[IDM] = 1 and DBCR0[EDM] = 0 the critical exceptions are not included.

12.9.10 Trap Taken Debug Event

This debug event occurs before execution of a trap instruction where the conditions are such that the trap will occur. When trap is enabled for a debug event, external debug mode is enabled, internal debug mode is enabled with MSR[DE] enabled, or debug wait mode is enabled, a trap instruction will not cause a program exception.

12.9.11 Unconditional Debug Event

This debug event occurs immediately upon being set by the JTAG debug port.

12.9.12 IAC Debug Event

This debug event occurs before execution of an instruction at an address that matches an address defined by the Instruction Address Compare Registers (IAC1–IAC4). DBCR0[IA1, IA2, IA3, IA4] enable IAC debug events IAC can be defined as an exact address comparison to one of the IAC*n* registers or on a range of addresses to compare defined by a pair of IAC*n* registers.

12.9.12.1 IAC Exact Address Compare

In this mode each IAC*n* register specifies an exact address to compare. These are enabled by setting DBCR0[IA*n*] = 1 and disabling IAC range compare (DBCR0[IA12X] = 0 for IAC1 and IAC2 and DBCR0[IA23X] = 0 for IAC3 and IAC4). The corresponding DBSR[IA*n*] bit displays the results of the debug event.

12.9.12.2 IAC Range Address Compare

In this mode a pair of IAC*n* registers are used to define a range of addresses to compare:

Range 1:2 corresponds to IAC1 and IAC2
Range 3:4 corresponds to IAC3 and IAC4

To enable Range 1:2, DBCR0[IA12] = 1 and DBCR0[IA1] or DBCR0[IA2] =1. An IAC event will be seen on the DBSR[IA*n*] field that corresponds to the enabled DBCR0[IA*n*] field. If DBCR0[IA1] and DBCR0[IA2] are enabled, the results of the event are reported on both DBSR fields. Setting DBCR0[IA12] =1 prohibits IAC1 and IAC2 from being used for exact address compares.

To enable Range 3:4, DBCR0[IA34] = 1 and DBCR0[IA3] or DBCR0[IA4] =1. An IAC event will be seen on the DBSR[IA*n*] field that corresponds to the enabled DBCR0[IA*n*] field. If DBCR0[IA3] and DBCR0[IA4] are enabled, the results of the event will be reported on both DBSR fields. Setting DBCR0[IA34] =1 prohibits IAC3 and IAC4 from being used for exact address compares.

Ranges can be defined as inclusive, as shown in the preceding examples, or exclusive, using DBCR0[IA12X] (corresponding to range 1:2) and DBCR0[IA34X] (corresponding to range 3:4), as follows:

DBCR0[IA12] = 1: Range 1:2 = IAC1 ≤ range < IAC2.
DBCR0[IA12X] = 1: Range 1:2 = Range low < IAC1 or IAC2 ≤ Range high
DBCR0[IA34] = 1: Range 3:4 = IAC3 ≤ range < IAC4.
DBCR0[IA34X] = 1: Range 3:4 = Range low < IAC3 or IAC4 ≤ Range high

Figure 12-10 shows the range selected in an inclusive IAC range address compare. Note that the address in IAC1 is considered part of the range, but the address in IAC2 is not, as shown in the preceding examples. The thick lines indicate that the indicated address is included in the compare results.

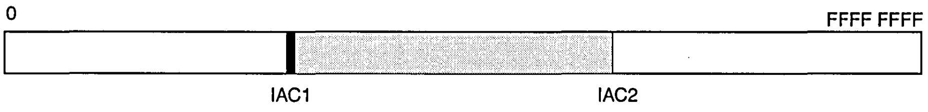


Figure 12-10. Inclusive IAC Range Address Compares

Figure 12-11 shows the range selected in an inclusive IAC range address compare. Note that the address in IAC1 is not considered part of the range, but the address in IAC2 is, along with the highest memory address, as shown in the preceding examples.

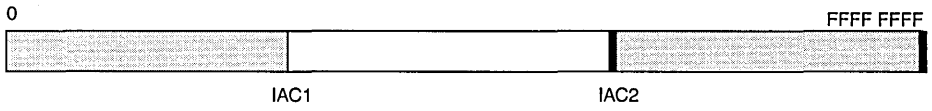


Figure 12-11. Exclusive IAC Range Address Compares

To toggle the range from inclusive to exclusive or from exclusive to inclusive on a IAC range debug event, DBCR0[IA12T] (corresponding to range 1:2) and DBCR0[IA34T] (corresponding to range 3:4) are used. If these fields are set, the DBCR0[IA12X] or DBCR0[IA34X] fields toggle on an IAC debug event, changing the defined range.

When a toggle is enabled (DBCR0[IA12T] for range 1:2 or DBCR0[IA34T] = 1 for range 3:4), and DBCR0[IDM] =1, DBCR0[EDM] = 0, and MSR[DE] = 0, IAC range comparisons for the corresponding toggle field are disabled.

12.9.13 DAC Debug Event

This debug event occurs before execution of an instruction that accesses a data address that matches the contents of the specified DAC register. DBCR1[D1R, D2R, D1W, D2W] enable DAC debug events for address comparisons on DAC1 and DAC2 for read instructions, DAC2 for read instructions, DAC1 for write instructions, DAC2 for write instructions respectively. Loads are reads and stores are writes. DAC can be defined(DBCR1[D1R, D2R])as an exact address comparison to one of the DACn registers or a range of addresses to compare defined by DAC1 and DAC2 registers.

12.9.13.1 DAC Exact Address Compare

In this mode, each DAC*n* register specifies an exact address to compare. Thes registers are enabled by setting one or more of DBCR1[D1R,D2R,D1W,D2W] = 1, and disabling DAC range compare DBCR1[DA12X] = 0. The corresponding DBSR[DR1,DR2,DW1,DW2] field displays the results of a DAC debug event.

The address for a DAC is the effective address (EA) of a storage reference instruction. EAs are always generated within a single aligned word of memory. Unaligned load and store, strings, and multiples generate multiple EAs to be used in DAC comparisons.

Data address compare (DAC) debug events can be set to react to any byte in a larger block of memory, in addition to reacting to a byte address match. The DAC Compare Size fields (DBCR1[D1S, D2S]) allow DAC debug events to react to byte, halfword, word, or 8-word line address by ignoring a number of LSBs in the EA.

DAC 1 Size	
00 Compare all bits	Byte address
01 Ignore LSB (least significant bit)	Halfword address
10 Ignore two LSBs	Word address
11 Ignore five LSBs	Cache line (8-word) address

The user must determine how the addresses of interest are accessed, relative to byte, halfword, word, string, and unaligned storage instructions, and adjust the DAC compare size field appropriately to cover the addresses of interest.

For example, suppose that a DAC debug event should react to byte 3 of a word-aligned target. A DAC set for exact compare would not recognize a reference to that byte by load/store word or load/store halfword instructions, because the byte address is not the EA of such instructions. In such a case, the D1S field must be set for a wider capture range (for example, to ignore the two least significant bits (LSBs) if word operations to the misaligned byte are to be detected). The wider capture range may result in excess debug events (events that are within the specified capture range, but reflect byte operations in addition to the desired byte). Such excess debug events must be handled by software.

While load/store string instructions are inherently byte addressed the processor will generate EAs containing the largest portion of an aligned word address as possible. It may not be possible to DAC on a specific individual byte using load/store string instructions.

12.9.13.2 DAC Range Address Compare

In this mode, the pair of DAC1 and DAC2 registers are used to define a range of addresses to compare.

To enable DAC range, DBCR1[DA12] = 1 and one or more of DBCR1[D1R,D2R,D1W,D2W] =1. The DAC event is seen on the DBSR[DR1,DR2,DW1,DW2] field that corresponds to the DBCR1[D1R,D2R,D1W,D2W] field that is enabled. For example, if DBCR1[D1R] and DBCR1[D2R] are enabled, the results of a DAC debug event are reported on DBSR[DR1, DR2]. Setting DBCR1[DA12] =1 prohibits DAC1 and DAC2 from being used for exact address compares.

Ranges are defined to be inclusive or exclusive, using the DBCR1[DA12X], as follows:

DBCR1[DA12] = 1: Range = DAC1 $\leq$ range < DAC2.
DBCR1[DA12X] = 1: Range = Range low < DAC1 or DAC2   Range high.

Figure 12-12 shows the range selected in an inclusive DAC range address compare. Note that the address in DAC1 is considered part of the range, but the address in DAC2 is not, as shown in the preceding examples. The thick lines indicate that the indicated address is included in the compare results.

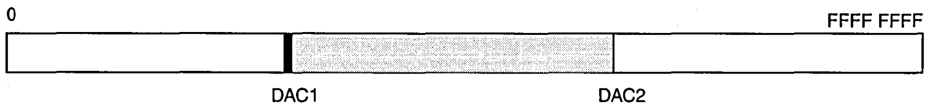


Figure 12-12. Inclusive DAC Range Address Compares

Figure 12-13 shows the range selected in an exclusive DAC range address compare. Note that the address in DAC1 is not considered part of the range, but the address in DAC2 is, along with the highest memory address, as shown in the preceding examples.

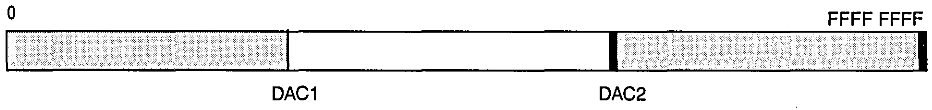


Figure 12-13. Exclusive DAC Range Address Compares

The DAC Compare Size fields (DBCR1[D1S, D2S]) are not used by DAC range comparisons.

12.9.13.3 Data Address Compare (DAC) Applied to Cache Instructions

Some cache instructions may cause DAC debug events. There are several special cases.

Table 12-5 summarizes possible DAC debug events by cache instruction:

Table 12-5. DAC Applied to Cache Instructions

Instruction	Possible DAC Debug Event	
	DAC-Read	DAC-Write
dcba	No	Yes
dcbf	No	Yes
dcbi	No	Yes
dcbst	No	Yes
dcbt	Yes	No
dcbz	No	Yes
dccci	No	No
dcread	No	No
dcbtst	Yes	No
icbi	Yes	No

Table 12-5. DAC Applied to Cache Instructions (continued)

Instruction	Possible DAC Debug Event	
	DAC-Read	DAC-Write
icbt	Yes	No
iccci	No	No
icread	No	No

Architecturally, the **dcbi** and **dcbz** instructions are "stores." These instructions can change data, or cause the loss of data by invalidating a dirty line. Therefore, they can cause DAC-write debug events.

The **dccci** instruction can also be considered a "store" because it can change data by invalidating a dirty line. However, **dccci** is not address-specific; it affects an entire congruence class regardless of the operand address of the instruction. Because it is not address-specific, **dccci** does not cause DAC-write debug events.

Architecturally, the **dcbt**, **dcbtst**, **dcbf**, and **dcbst** instructions are "loads." These instructions do not change data. Flushing or storing a cache line from the cache is not architecturally a "store" because a store had already updated the cache; the **dcbf** or **dcbst** instruction only updates the copy in main memory.

The **dcbt** and **dcbtst** instructions can cause DAC-read debug events regardless of cachability.

Although **dcbf** and **dcbst** are architecturally "loads," these instructions can create DAC-write (but not DAC-read) debug events. In a debug environment, the fact that external memory is being written is the event of interest.

Even though **dcread** and **dccci** are not address-specific (they affect a congruence class regardless of the instruction operand address), and are considered "loads," in the PPC405GP they do not cause DAC debug events.

All ICU operations (**icbi**, **icbt**, **iccci**, and **icread**) are architecturally treated as "loads." **icbi** and **icbt** cause DAC debug events. **iccci** and **icread** do not cause DAC debug events in the PPC405GP.

12.9.13.4 DAC Applied to String Instructions

An **stswx** instruction with a string length of 0 is a no-op. The **lswx** instruction with the string length equal to 0 does not alter the RT operand with undefined data, as allowed by the PowerPC Architecture. Neither **stswx** nor **lswx** with zero length causes a DAC debug event because storage is not accessed by these instructions.

12.9.14 Data Value Compare Debug Event

A data value compare (DVC) debug event can occur only after execution of a load or store instruction to an address that compares with the address in one of the DAC*n* registers and has a data value that matches the corresponding DVC*n* register. Therefore, a DVC debug event requires both the data address comparison and the data value comparison to be true. A DVCn debug event when enabled in the DBCR1 supercedes a DACn debug event since the DVCn and the DACn both use the same DACn register.

DVC1 debug events are enabled by setting the appropriate DAC enable DBCR1[D1R,D1W] to cause an address comparison and by setting anybit combination in the DBCR1[DV1BE]. DVC2 debug events are enabled by setting the appropriate DAC enable DBCR1[D2R,D2W] to cause an address comparison and by setting any bit combination in the DBCR1[DV1BE]. Each bit in DBCR1[DV1BE, DV2BE] correspondes to a byte in DVC1 and DVC2. Exact address compare and range address compare work the same for DVC as for a simple DAC.

DBSR[DR1] and DBSR[DW1] record status for DAC1 debug events. Which DBSR bit is set depends on the setting of DBCR1[D1R] and DBCR[D1W]. If DBCR1[D1R] = 1, DBSR[DR1] = 1, assuming that a DVC event occurred. Similarly, if DBCR1[D1W] = 1, DBSR[DW1] = 1, assuming that a DVC event occurred.

Similarly, DBSR[DR2] and DBSR[DW2] record status for DAC2 debug events. Which DBSR bit is set depends on the setting of DBCR1[D2R] and DBCR[D2W]. If DBCR1[D2R] = 1, DBSR[DR2] = 1, assuming that a DVC event occurred. Similarly, if DBCR1[D2W] = 1, DBSR[DW2] = 1, assuming that a DVC event occurred.

In the following example, a DVC1 event is enabled setting the DBCR1[D1R] = 1, DBCR1[D1W] = 1, DBCR1[DA12] = 0, and DBCR1[DV1BE ' 0000]. When the data address and data value match the DAC1 and DVC1 , a DVC1 event is recorded in DBSR[DR1] or DBSR[DW1], depending on whether the operation is a load (read) or a store (write). This example corresponds to the last line of Table 12-6.

In Table 12-6, *n* is 1 or 2, depending on whether the bits apply to DAC1, DAC2, DVC1, and DVC2 events. "Hold" indicates that the DBSR holds its value unless cleared by software. "RA" indicates that the operation is a read (load) and the data address compares (exact or range). "WA" indicates that the operation is a write (store) and the data address compares (exact or range). "RV" indicates that the operation is a read (load), the data address compares (exact or range), and the data value compares according to the DBCR1[DVC*n*].

Table 12-6. Setting of DBSR Bits for DAC and DVC Events

			DBCR1			DBSR	
DACn Event	**DVCn Enabled**	**DVCn Event**	**[DnR]**	**[DnW]**	**[DA12]**	**[DRn]**	**[DWn]**
0	—	—	—	—	—	Hold	Hold
—	—	—	0	0	—	Hold	Hold
1	0	—	0	1	—	Hold	WA
1	0	—	1	0	—	RA	Hold
1	0	—	1	1	—	RA	WA
1	1	0	—	—	—	Hold	Hold
1	1	1	0	1	—	Hold	WV
1	1	1	1	0	—	RV	Hold
1	1	1	1	1	—	RV	WV

The settings of DBCR1[DV1M] and DBCR1[DV2M] are more precisely defined in Table 12-7 (*n* enables the table to apply to DBCR1[DV1M, DV2M] and DBCR1[DV1BE, DV2BE]). DV*n*BE$_m$ indicates bytes selected (or not selected) for comparison in DBCR1[DV*n*BE].

When DBCR1[DV*n*M] = 01, the comparison is an AND; all bytes must compare to the appropriate bytes of DVC1.

When DBCR1[DV*n*M] = 10, the comparison is an OR; at least one of the selected bytes must compare to the appropriate bytes of DVC1.

When DBCR1[DV*n*M] = 11, the comparison is an AND-OR (halfword) comparison. This is intended for use when DBCR1[DV*n*BE] is set to 0011, 0111, or 1111. Other values of DBCR1[DV*n*BE] can be compared, but the results are more easily understood using the AND and OR comparisons. In Table 12-7, “not” is ¬, AND is ∧, and OR is ∨.

Table 12-7. Comparisons Based on DBCR1[DVnM]

DBCR1[DVnM] Setting	Operation	Comparison
00	—	Undefined
01	AND	$(\neg DVnBE_0 \vee (DVC1[byte\ 0] = data[byte\ 0])) \wedge$ $(\neg DVnBE_1 \vee (DVC1[byte\ 1] = data[byte\ 1])) \wedge$ $(\neg DVnBE_2 \vee (DVC1[byte\ 2] = data[byte\ 2])) \wedge$ $(\neg DVnBE_3 \vee (DVC1[byte\ 3] = data[byte\ 3]))$
10	OR	$(DVnBE_0 \wedge (DVC1[byte\ 0] = data[byte\ 0])) \vee$ $(DVnBE_1 \wedge (DVC1[byte\ 1] = data[byte\ 1])) \vee$ $(DVnBE_2 \wedge (DVC1[byte\ 2] = data[byte\ 2])) \vee$ $(DVnBE_3 \wedge (DVC1[byte\ 3] = data[byte\ 3]))$
11	AND-OR	$(DVnBE_0 \wedge (DVC1[byte\ 0] = data[byte\ 0])) \wedge$ $(DVnBE_1 \wedge (DVC1[byte\ 1] = data[byte\ 1])) \vee$ $(DVnBE_2 \wedge (DVC1[byte\ 2] = data[byte\ 2])) \wedge$ $(DVnBE_3 \wedge (DVC1[byte\ 3] = data[byte\ 1]))$

12.9.15 Imprecise Debug Event

The imprecise debug event is not an independent debug event, but indicates that a debug event occurred while MSR[DE] = 0. This is useful in internal debug mode if a debug event occurs while in a critical interrupt handler. On return from interrupt, a debug interrupt occurs if MSR[DE] = 1. If DBSR[IDE] = 1, the debug event causing the interrupt occurred sometime earlier, not immediately after a debug event.

Chapter 13. Clock and Power Management

The PPC405GP provides a clock and power management (CPM) controller that reduces power dissipation by stopping clocks in unused or dormant functional units. Use of the CPM controller requires careful programming and special consideration to avoid compromising system and functional unit integrity.

The CPM controller supports three different types of sleep interfaces to the functional units:

- In a CPM class 1 interface, the CPM_Sleep_N signal is asserted by the CPM controller when a register bit is set by software. The functional unit is unconditionally put to sleep. There is no other communication with the functional unit.
- In a CPM class 2 interface, the functional unit uses a combination of its internal state and external inputs to determine whether or not it can be put to sleep. If sleeping is permitted, the functional unit asserts the Sleep_Req signal to the CPM controller that responds by asserting CPM_Sleep_N if the enable for that unit is set. The CPM_Sleep_N signal to a class 2 unit is deasserted when the CPM controller enable bit for that unit is reset, or when the unit deasserts its Sleep_Req signal.
- The CPM class 3 interface has a CPM_SleepInit signal that is asserted by the CPM controller to request that a functional unit go to sleep. If the unit can sleep, it asserts the Sleep_Req signal to the CPM controller. The CPM_Sleep_N signal is then asserted by the CPM controller to shut off the class 3 clocks in the functional unit. The functional unit or the CPM controller can end the sleep state. If the CPM controller enable bit for the unit is reset, the CPM controller immediately deasserts CPM_SleepInit and CPM_Sleep_N.

13.1 CPM Registers

Table 13-1 lists the registers used to program the CPM controller.

Table 13-1. CPM Registers

Register Name	DCR Address	Access	Reset Value (0:16) (bits 17:31 reserved)
CPC0_ER	0x0B9	Read/Write	0x0000xxxx
CPC0_FR	0x0BA	Read/Write	0x0000xxxx
CPC0_SR	0x0B8	Read Only	0xFFFFxxxx

Each functional unit has one bit in each of CPC0_ER, CPC0_FR, and CPC0_SR. The bit assignment is the same in the three registers. Figure 13-1 shows the bit assignment and CPM class for each PPC405GP functional units.

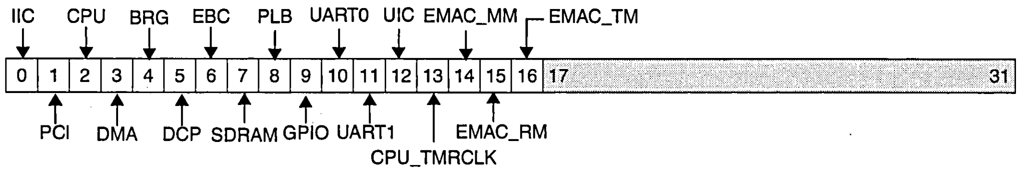


Figure 13-1. CPM Registers (CPC0_ER, CPC0_FR, CPC0_SR)

0	IIC	IIC Interface	Class 3
1	PCI	PCI Bridge	Class 3
2	CPU	PPC405GP Processor Core	Class 2
3	DMA	DMA Controller	Class 2
4	BRG	PLB to OPB Bridge	Class 2
5	DCP	CodePack	Class 2
6	EBC	ROM/SRAM Peripheral Controller	Class 2
7	SDRAM	SDRAM Memory Controller	Class 2
8	PLB	PLB Bus Arbiter	Class 2
9	GPIO	General Purpose Interrupt Controller	Class 1
10	UART0	Serial Port 0	Class 1
11	UART1	Serial Port 1	Class 1
12	UIC	Universal Interrupt Controller	Class 1
13	CPU_TMRCLK	CPU Timers	Class 1
14	EMAC_MM	Ethernet MM Unit	Class 1
15	EMAC_RM	Ethernet RM Unit	Class 1
16	EMAC_TM	Ethernet TM Unit	Class 1
17:31	▨	Reserved	

13.1.1 CPM Enable Register (CPC0_ER)

The CPC0_ER bits enable the process of putting a functional unit to sleep. The class of a unit determines how its interface signals are controlled when the bit associated with the unit is set to 1.

Class 1	When an associated CPC0_ER bit is set to 1, the CPM_Sleep_N signal to the class 1 unit is asserted. When the bit is set to 0, CPM_Sleep_N is deasserted. There are some additional considerations to avoid generating extraneous interrupts when waking the UIC. Before enabling sleep mode (setting DPC0_ER[UIC] to 1), save the contents of the UIC Masked Status Register (UIC_MSR) and UIC Enable Register (UIC0_ER), and disable all interrupts by setting UIC0_ER to 0. After exiting sleep mode, write the ones complement of the saved contents of the UIC_MSR to the UIC Status Register (UIC0_SR), and restore the state of the UIC0_ER.
Class 2	When an associated CPC0_ER bit is set to 1, and the Sleep_Req signal from the class 2 unit is asserted (the unit is requesting sleep state), CPM_Sleep_N to the class 2 unit is asserted. When the bit is set to 0, the CPM_Sleep_N signal is deasserted.
Class 3	When an associated CPC0_ER bit is set to 1, the CPM_SleepInit signal to the class 3 unit is asserted (the CPM controller is requesting permission to put the unit to sleep). When the class 3 unit activating the Sleep_Req in response, (the unit is giving permission to be put to sleep), CPM_Sleep_N signal to the class 3 unit is asserted. When the bit is set to 0, CPM_SleepInit and CPM_Sleep_N are deasserted.

13.1.2 CPM Force Register (CPC0_FR)

Setting a CPC0_FR bit forces assertion of the CPM_Sleep_N signal to the functional unit. For a class 1 unit, this is equivalent to setting the CPC0_ER bit associated with the unit. For class 2 or class 3 units, CPM_Sleep_N is asserted regardless of the state of the Sleep_Req signal coming from the unit.

13.1.3 CPM Status Register (CPC0_SR)

The read-only CPC0_SR shows the current state of all CPM_Sleep_N signals.

Chapter 14. Decompression Controller Operation

The decompression controller enables PPC405GP instructions to be stored in memory in a compressed form, thus, reducing memory requirements and overall system cost. Because the processor core cannot execute compressed instructions, hardware is provided to decompress the compressed instructions. The decompression controller decompresses instructions on-the-fly, as the processor core fetches instructions from memory. The decompression unit is located in the path between memory and the processor core.

When decompressed instructions are stored in the instruction cache unit (ICU), the performance penalty is minimized. Increased latency occurs only when instruction fetches miss in the cache. Average latency is further reduced by a buffer in the decompression hardware that holds more instructions than are requested by the CPU. If a subsequent fetch occurs to a nearby memory location that is already in the buffer, data is returned from the buffer; no memory cycle is required. In such cases, initial latency is less than a normal uncompressed fetch from memory.

Code to be stored in compressed form is compressed by software after compiling and before loading. Thus, code compression can be performed once by the application developer. The compressed code is in the load image.

The decompression controller, located between the processor core and memory, is transparent to the processor core. When the processor core performs an instruction fetch, the decompression controller works with the memory controller (SDRAM or external bus controller [EBC]) to provide a decompressed instruction stream to the CPU. The memory controllers deliver the compressed stream of instructions to the decompression controller, which decompresses the instructions and delivers the resulting instruction stream to the processor core.

14.1 Code Compression and Decompression

While code decompression is performed in hardware when the processor core fetches instructions in a compressed code address space, code compression is performed only once, between the compile and load steps in the usual programming sequence.

14.1.1 Code Compression

Compression software analyzes compiled code, finds the set of most frequently used instructions, and builds decode tables from those instructions.

The compression software splits each instruction, analyzes the upper and lower halfwords of the instruction to find the most common 16-bit patterns. The most frequently used patterns are tagged for insertion in a decode table. Infrequently used instructions are tagged as not compressed. The 16-bit patterns that match those in the decode table are stored in the compressed code image, along with uncompressed instructions. The decode table is loaded into the decompression controller before attempted execution of the compressed instruction streams.

After the decode table is built, a compiled code image is compressed, using a scheme employing a tag and an index. The decompression controller generates an address into the decode table to select the original 16-bit pattern.

The compression software stores compressed and uncompressed instructions in 64-byte blocks, called compression blocks. The number of bytes used to store compressed code varies in each compression block, depending on the encoding. Because the length resulting compressed block varies, the compression software generates an index table that translates addresses of uncompressed 64-byte blocks, called target instruction addresses (TIAs), to addresses of the compressed variable length blocks. The index table also contains information indicating, for each block, whether the compression block was compressed or left uncompressed. The index table is loaded into system memory with the compressed blocks of program code. The decompression controller accesses the index table when a fetch is made to compressed program space.

14.1.2 Code Decompression

The decompression controller is located between the processor core and the SDRAM and EBC memory controllers, isolating the memory controllers from the processor local bus (PLB). The decompression controller is transparent to the processor core and the memory controllers. The processor core side of the decompression controller is a PLB slave to the PLB bus, and the memory controller side is a PLB master to the memory controllers. Instruction-side requests for external memory go though the decompression controller, and are either intercepted for decompression or allowed to pass through unaltered.

The decompression controller must distinguish between compressed and uncompressed code space. To accomplish this, the processor core provides a signal with each instruction memory reference, indicating whether the reference is in compressed or uncompressed memory space. The CPU's U0 storage attribute is used to implement this signal. When address translation is enabled (the IR field of the Machine State Register (MSR) is 1), a corresponding U0 bit is implemented in translation lookaside buffer (TLB) entries in the memory management unit (MMU). When address translation is disabled (MSR[IR] = 0, the Storage User-defined 0 Register (SU0R) implements the U0 storage attribute for memory regions. The U0 storage attribute indicates whether a page is marked as compressed.

14.1.3 Instruction Fetches to Compressed Pages

The decompression controller intercepts instruction fetches to code in uncompressed pages, locates the compressed code in memory, and decompresses the instruction stream before forwarding it to the processor core. These steps are detailed in the following procedure:

1. Locate compressed code
   a. Calculate offset into index table.
   b. Retrieve index from index table.
   c. Calculate address of compression block using the index table entry.
   d. Retrieve compression block.
2. Decode instructions into the output buffer, if required.
3. Return instructions to the processor core.

The decompression controller contains a 64B output buffer that stores instruction words as they are decompressed. Because code is compressed and decompressed in 64B blocks and the CPU requests smaller line fills (32 bytes), some accesses to compressed instructions may already be available in the decompression controller output buffer, decompressed and ready for execution. In this case, line fill requests are serviced from the output buffer, without having to address external memory.

Step 2, above, indicates that compressed code is decoded "if required." Some compression blocks do not contain compressed code, even though they reside in compressed code space. The compression algorithm can infrequently result in blocks that are larger than the original code. In such cases, the compression software does not compress the instruction block and marks it as uncompressed. The decoder will not attempt to decompress such blocks.

14.1.4 Instruction Fetches to Uncompressed Pages

The decompression controller does not intercept instruction fetches to code in uncompressed memory space. These fetches are passed, unaltered, to the appropriate memory controller. The decompression controller also provides unaltered data, from the appropriate memory controller, back to the CPU in the case of a read. This transaction does not incur any additional latency.

14.1.5 Performance

The decompression controller is designed so that read/write requests to uncompressed memory spaces incur no additional latency over a system without a decompression controller.

The decompression controller adds additional latency to some read accesses to compressed memory space. The decompression controller requires additional cycles to locate compressed code, fetch it, decode the instructions, and place them in the output buffer. However, because instructions are encoded in 64-byte compression blocks, the decompression controller continues to decompress instructions until its 64-byte output buffer is full. In such cases, the decompression controller acts as a prefetch buffer, and subsequent fetches to instructions in the same 64-byte block actually incur less latency than if the decompression controller were not present.

More significant latencies occur when the processor core executes a branch out of the current 64-byte block. If the branch is to a TIA that is not at the beginning of a 64-byte block, the decompression controller must fetch and decode the block containing the TIA, byte by byte, until it reaches the desired instruction. The worst latencies occur when the TIA is outside of the 128-byte group, since the decompression core must then go through its entire decode process, including a refetch of the index table entry.

14.2 Decompression Controller Registers

The registers that control operation of the decompression controller, listed in Table 14-1, are indirectly accessed using the Decompression Controller Address Register (DCP0_CFGADDR) and Decompression Controller Data Register (DCP0_CFGDATA) listed in Table 14-2. The DCP0_CFGADDR and DCP0_CFGDATA are accessed using the **mfdcr** and **mtdcr** instructions.

Descriptions of the registers follow Table 14-2.

The DCP0_CFGADDR stores the address of the target register, while the DCP0_CFGDATA register is used to write or read the target register contents.

The following procedure accesses the decompression controller registers.

1. Write the offset from Table 14-2 to the Decompression Controller Address Register (DCP0_CFGADDR).
2. Read data from or write data to the Decompression Controller Data Register (DCP0_CFGDATA).

Table 14-1. DCRs Used to Access the Decompression Controller Registers

Register	DCR Address	Access	Description
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register

Table 14-2. Offsets for Decompression Controller Registers

Register	Offset	R/W	Description
DCP0_ITOR0	0x00	R/W	Index Table Origin Register 0
DCP0_ITOR1	0x01	R/W	Index Table Origin Register 1
DCP0_ITOR2	0x02	R/W	Index Table Origin Register 2
DCP0_ITOR3	0x03	R/W	Index Table Origin Register 3
DCP0_ADDR0	0x04	R/W	Address Decode Definition Register 0
DCP0_ADDR1	0x05	R/W	Address Decode Definition Register 1
DCP0_CFG	0x40	R/W	Decompression Controller Configuration Register
DCP0_ID	0x41	R	Decompression Controller ID Register
DCP0_VER	0x42	R	Decompression Controller Version Register
DCP0_PLBBEAR	0x50	R	Bus Error Address Register (PLB)
DCP0_MEMBEAR	0x51	R	Bus Error Address Register (EBC/SDRAM)
DCP0_ESR	0x52	R/Clear	Bus Error Status Register 0 (Masters 0–3)
DCP0_RAM0- DCP0_RAM3FF	0x400–0x7FF	R/W	SRAM/ROM Read/Write

14.2.1 Index Table Origin Registers (DCP0_ITOR0–DCP0_ITOR3)

DCP0_ITOR0–DCP0_ITOR3 each contain the high-order address bits of the index table for a compression region.

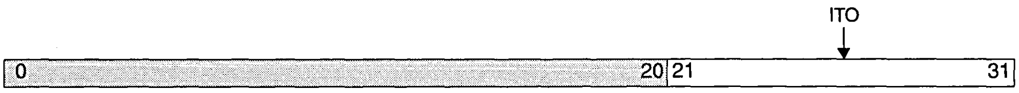


Figure 14-1. Decompression Index Table Origin Registers (DCP0_ITOR0–DCP0_ITOR3)

0:20	▨	Reserved	
21:31	ITO	Index Table Origin	High-order index table address bits

14.2.2 Decompression Address Decode Definition Registers (DCP0_ADDR0–DCP0_ADDR1)

DCP0_ADDR0–DCP0_ADDR1 each define an address space used by the decompression controller for compressed instructions, and for uncompressed accesses that are passed to a memory controller.

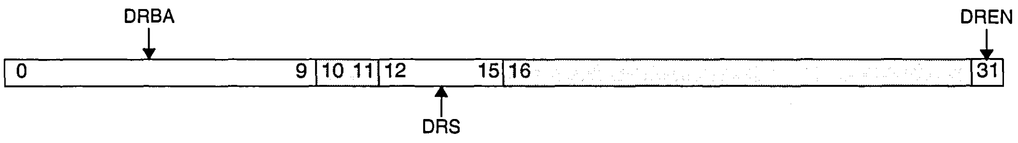


Figure 14-2. Decompression Address Decode Definition Registers (DCP0_ADDR0–DCP0_ADDR1)

0:9	DRBA	Decode Region Base Address High-order decode region address bits
10:11		Reserved
12:15	DRS	Decode Region Size 0000–0100 Reserved 0101 4MB 0110 8MB 0111 16MB 1000 32MB 1001 64MB 1010 128MB 1011 256MB 1100 512MB 1101 1GB 1110 2GB 1111 4GB
16:30		Reserved
31	DREN	Enable Decode Region 0 Decoding is disabled for region 1 Decoding is enabled for region.

14.2.3 Decompression Configuration Register (DCP0_CFG)

DCP0_CFG is used to configure the operation of the decompression controller.

DCP0_CFG[IKB] must be set to 0 to enable decompression. The value of this field determines whether the U0 storage attribute is recognized; the storage attribute must be recognized for decompression to occur.

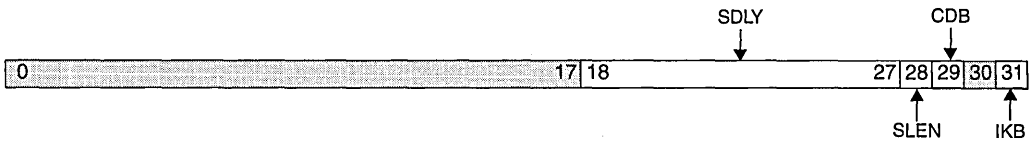


Figure 14-3. Decompression Controller Configuration Register (DCP0_CFG)

0:17	▨	Reserved	
18:27	SLDY	Sleep Delay	Sleep delay
28	SLEN	Sleep Enable 0 Disable sleep 1 Enable sleep	
29	CDB	Clear Decompression Buffer 0 Normal operation 1 Clear decompression buffer	Self-clearing; always reads 0
30	▨	Reserved	
31	IKB	Enable Decrompression 0 Decompression is enabled; U0 storage attribute is recognized 1 Decompression is disabled; U0 storage attribute is ignored	

14.2.4 Decompression Controller ID Register (DCP0_ID)

DCP0_ID contains the decompression controller ID.

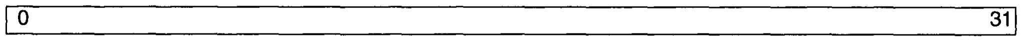


Figure 14-4. Decompression Controller ID Register (DCP0_ID)

0:31		Decompression Controller ID	Read-only, value is 0000504D

14.2.5 Decompression Controller Version Register (DCP0_VER)

DCP0_VER contains the decompression controller version.

Figure 14-5. Decompression Controller Version Register (DCP0_VER)

0:31		Decompression Controller Version	Read-only, value is 00000200

14.2.6 Decompression Controller PLB Error Address Register (DCP0_PLBBEAR)

On an error, DCP0_PLBBEAR contains the address on the PLB that caused the decompression controller to generate cycles to the SDRAM and EBC memory controllers.

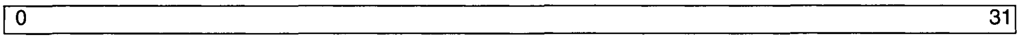


Figure 14-6. Decompression Controller PLB Error Address Register (DCP0_PLBBEAR)

0:31		Address of PLB Error

14.2.7 Decompression Controller Bus Error Address Register (DCP0_MEMBEAR)

On an error, DCP0_MEMBEAR contains the address, presented to the memory controllers, that caused an error in the decompression controller. The address was presented to the memory controllers for either an ITE fetch or a block burst request.

Figure 14-7. Decompression Controller Bus Error Address Register (DCP0_MEMBEAR)

0:31		Address of SDRAM or EBC Error

14.2.8 Decompression Controller Error Status Register 0 (DCP0_ESR)

DCP0_ESR reports error status.

Status information includes error type and master read/write status. DCP0_ESR0 field and DCP0_PLBBEAR or DCP0_MEMBEAR lock status is also reported.

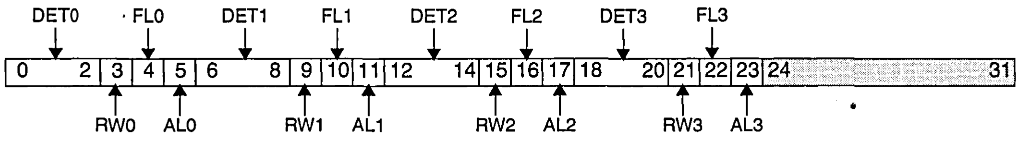


Figure 14-8. Decompression Controller Error Status Register 0 (DCP0_ESR)

0:2	DET0	Decompression Error Type for Master 0 000 Time-out during ITE fetch 001 Time-out during block fetch 010 Memory controller error during ITE fetch 011 Memory controller error during block fetch 100–111 Reserved	Master 0 is the processor core instruction cache unit (ICU).
3	RW0	Read/Write Status for Master 0 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
4	FL0	DCP0_ESR Field Lock for Master 0 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
5	AL0	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 0 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
6:8	DET1	Decompression Error Type for Master 1	See description for DCP0_ESR[DET0]. Master1 is the processor core data cache unit (DCU).
9	RW1	Read/Write Status for Master 1 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
10	FL1	DCP0_ESR Field Lock for Master 1 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
11	AL1	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 1 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.

12:14	DET2	Decompression Error Type for Master 2	See description for DCP0_ESR[DET0]. Master 2 is the external master.
15	RW2	Read/Write Status for Master 2 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
16	FL2	DCP0_ESR Field Lock for Master 2 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
17	AL2	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 2 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
18:20	DET3	Decompression Error Type for Master 3	See description for DCP0_ESR[DET0]. Master 3 is PCI.
21	RW3	Read/Write Status for Master 3 0 Error operation was a write 1 Error operation was a read	This implementation only report errors for compressed reads.
22	FL3	DCP0_ESR Field Lock for Master 3 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
23	AL3	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 3 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
24:31	▨	Reserved	

For more information, refer to *CodePack™ PowerPC Code Compression Utility User's Manual,* Version 3.0.

Part IV. PPC405GP External Interfaces

Chapter 15. SDRAM Controller

The SDRAM controller provides a 32-bit interface to SDRAM memory with optional Error Checking and Correction (ECC). The memory controller provides flexible, fully programmable timings for interfacing to a wide variety of SDRAM devices. It supports four physical banks of dual- or quad-bank SDRAMs, where each physical bank is from 4 MB to 256 MB in size. In addition, the SDRAM controller supports the use of registered SDRAMs.

The controller supports page mode operation with bank interleaving and maintains up to four open pages. To improve performance, the controller features separate 32-byte read and 128-byte write buffers. For applications where data integrity must be guaranteed, optional ECC provides standard single error correction and double error detection. Designers also have the opportunity to reduce system power by placing the SDRAM controller in sleep and/or self-refresh mode.

15.1 Interface Signals

In many systems the memory controller can directly interface to SDRAM, relieving designers from the need to buffer signals or add circuitry to phase adjust the SDRAM clock. However, since each application is different, a detailed timing analysis should always be performed. Please note that all of the signals on the SDRAM interface utilize industry standard bit and byte lane numbering. That is, MemData0 is the least significant bit and DQM0 is the enable for the MemData7:0.

Figure 15-1 illustrates the SDRAM signal I/Os.

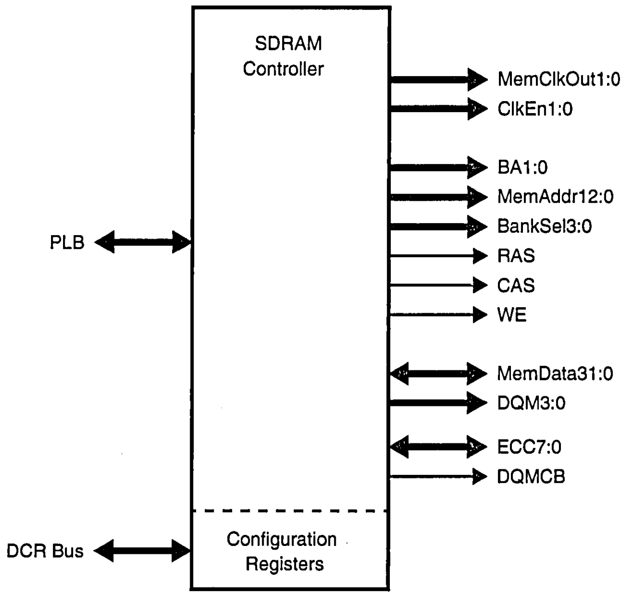


Figure 15-1. SDRAM Controller Signals

The usage and signal state after a PPC405GP reset for each of these signals is shown in Table 15-1 on page 15-2.

Table 15-1. SDRAM Signal Usage and State During/Following Reset

Signal	Reset State	Usage
MemClkOut1:0	Toggling[1]	Two copies of the SDRAM clock.
ClkEn1:0	1	SDRAM clock enable.
BA1:0	2'b0	Bank address. Used to select an internal SDRAM bank in dual- and quad-bank SDRAM devices.
MemAddr12:0	13b'0	Memory address. See Table 15.3.5, "Logical Address to Memory Address Mapping," on page 15-8 for additional details.
$\overline{\text{BankSel3:0}}$	4b'1	Bank Selects. Used to select between different physical banks of SDRAM memory.
$\overline{\text{RAS}}$	1	Row Address Strobe.
$\overline{\text{CAS}}$	1	Column Address Strobe.
$\overline{\text{WE}}$	1	Write Enable.
MemData31:0	High-Z	Data input/output. MemData31 is the most significant bit.
DQM3:0	1	Data Mask, an input mask for write accesses and an output enable during read operations. DQM3 applies to MemData31:24, DQM2 to MemData23:16, DQM1 to MemData15:8 and DQM0 to MemData7:0.
ECC7:0	High-Z	ECC check bits.
DQMCB	1	Data Mask for ECC check bits.

Note 1: During power-up, MemClkOut1:0 tracks SysClk until the internal PLL begins to lock. At that time MemClkOut1:0 transitions to the clock rate configured via the strapping resistors.

15.2 Accessing SDRAM Registers

After a system reset, software is required to configure and then enable the SDRAM controller. When this is complete, the SDRAM memory is ready for access by the processor, or any other master in the PPC405GP. Once the controller is operating, it is usually not necessarily for software to access the SDRAM configuration registers. The status registers, however, are useful for determining the state of the memory controller or querying information regarding any corrected or uncorrectable ECC errors that may have occurred.

All SDRAM configuration and status registers are accessed using the **mtdcr** and **mfdcr** PowerPC instructions. Access to these registers is performed using an indirect addressing method through the SDRAM0_CFGADDR and SDRAM0_CFGDATA registers.

Table 15-2. SDRAM Controller DCR Addresses

Register	DCR Address	Access	Description
SDRAM0_CFGADDR	0x010	R/W	SDRAM Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	SDRAM Controller Data Register

Table 15-3 lists the indirectly accessed SDRAM configuration and status registers.

Table 15-3. SDRAM Controller Configuration and Status Registers

Mnemonic	Address Offset	Access	Description	Page
SDRAM0_BESR0	0x00	R/Clear	Bus Error Syndrome Register 0	15-17
SDRAM0_BESR1	0x08	R/Clear	Bus Error Syndrome Register 1	15-18
SDRAM0_BEAR	0x10	R/W	Bus Error Address Register	15-17
SDRAM0_CFG	0x20	R/W	SDRAM Configuration	15-3
SDRAM0_STATUS	0x24	R	SDRAM Controller Status	15-5
SDRAM0_RTR	0x30	R/W	Refresh Timer Register	15-13
SDRAM0_PMIT	0x34	R/W	Power Management Idle Timer	15-20
SDRAM0_B0CR	0x40	R/W	Memory Bank 0 Configuration Register	15-6
SDRAM0_B1CR	0x44	R/W	Memory Bank 1 Configuration Register	15-6
SDRAM0_B2CR	0x48	R/W	Memory Bank 2 Configuration Register	15-6
SDRAM0_B3CR	0x4C	R/W	Memory Bank 3 Configuration Register	15-6
SDRAM0_TR	0x80	R/W	SDRAM Timing Register	15-9
SDRAM0_ECCCFG	0x94	R/W	ECC Configuration	15-14
SDRAM0_ECCESR	0x98	R	ECC Error Status	15-16

To read or write one of these SDRAM controller registers, software must first write the register's address offset into the SDRAM0_CFGADDR register. The target register can then be read or written through the SDRAM0_CFGDATA DCR address. The following PowerPC code illustrates this procedure by reading the SDRAM0_CFG register.

```
li        r3,SDRAM0_CFG               ! address offset of SDRAM0_CFG
mtdcr     SDRAM0_CFGADDR,r3           ! set offset address
mfdcr     r4,SDRAM0_CFGDATA           ! read value of SDRAM0_CFG
```

Programming Note: Reserved fields in SDRAM controller configuration registers must not change when the register is written. To modify bit fields within a register, read the register, use a mask to clear the target bits, OR in the new field value, and then write the result back.

15.3 SDRAM Controller Configuration and Status

Software must program a number of SDRAM control registers before the SDRAM controller can be started and memory accessed. At a minimum, this involves writing to the SDRAM Configuration Register (SDRAM0_CFG), SDRAM Timing Register (SDRAM0_TR) and one or more Memory Bank Configuration Registers (SDRAM0_BnCR).

15.3.1 Memory Controller Configuration Register (SDRAM0_CFG)

After a system reset the SDRAM controller is disabled and must be configured before memory transactions can occur. The Memory Controller Configuration Register (SDRAM0_CFG) serves to both enable the controller and select from various memory controller features. These features include enabling power management, ECC error checking, and registered SDRAM support. Each of these settings is global and applies to the entire SDRAM subsystem.

Prior to enabling the SDRAM controller by setting SDRAM0_CFG[DCE], the SDRAM0_TR, SDRAM0_RTR, SDRAM0_PMIT, and SDRAM0_BnCR registers must be configured. This is because once SDRAM0_CFG[DCE]=1 writing any of the listed SDRAM registers does not actually update the target register. Write access to SDRAM0_CFG is independent of SDRAM0_CFG[DCE]. However, software must ensure that the SDRAM controller is idle when updating SDRAM0_CFG[3:10]. This guarantees that the register update does not affect any in-progress SDRAM operations.

Before software enables the SDRAM controller by setting SDRAM0_CFG[DCE] it must ensure that the SDRAM power-on delay has been satisfied. For example, the IBM 16MB SDRAM requires a 100μs pause; the 64MB SDRAM requires a 200μs pause. Once enabled, the SDRAM controller automatically performs the following initialization procedure:

1. Issues the precharge command to all banks.
2. Waits SDRAM0_TR[PTA] cycles.
3. Performs eight $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh cycles, each separated by SDRAM0_TR[RFTA] clock cycles)
4. Issues the mode register write command to each bank.
5. Perform eight $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh cycles (each separated by SDRAM0_TR[RFTA] clock cycles).
6. Waits SDRAM0_TR[RFTA] clock cycles.

The SDRAM is then available for read and write access.

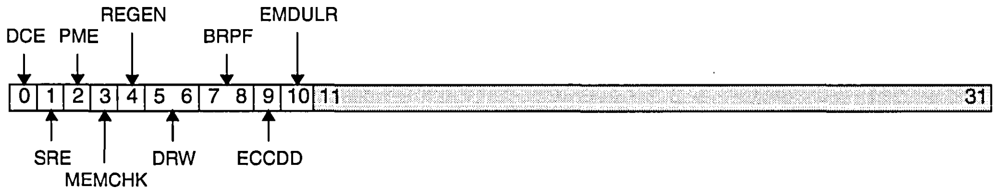


Figure 15-2. Memory Controller Configuration (SDRAM0_CFG)

0	DCE	SDRAM Controller Enable 0 Disable 1 Enable	All SDRAM controller configuration registers must be initialized and valid prior to setting DCE.
1	SRE	Self-Refresh Enable 0 Disable 1 Enable	See “Self-Refresh” on page 15-19.
2	PME	Power Management Enable 0 Disable 1 Enabled	See “Power Management” on page 15-20.
3	MEMCHK	Memory Data Error Checking 0 None 1 ECC	See “Error Checking and Correction (ECC)” on page 15-14.
4	REGEN	Registered Memory Enable 0 Disabled 1 Enabled	

5:6	DRW	SDRAM Width 00 32-bit 01 Reserved 10 Reserved 11 Reserved	Must be set to 2'b00.
7:8	BRPF	Burst Read Prefetch Granularity 00 Reserved 01 16 bytes 10 32 bytes 11 Reserved	Most applications should set with field to 2'b01.
9	ECCDD	ECC Driver Disable 0 Check bit data output on ECC7:0. 1 ECC7:0 are placed in high-Z state.	Regardless of whether ECC checking is enabled, SDRAM writes cause the PPC405GP to output check bit data on ECC7:0. If ECC is not used, setting ECCDD=1 disables these drivers.
10	EMDULR	Enable Memory Data Unless Read 0 MemData0:31 are high-Z unless a memory write is being performed. 1 MemData0:31 are driven unless a memory read is being performed.	
11:31	▨	Reserved	

15.3.2 Memory Controller Status (SDRAM0_STATUS)

The Memory Controller Status Register (SDRAM0_STATUS) allows software to determine the current state of the SDRAM controller. If SDRAM0_STATUS[MRSCMP]=0 the SDRAM is not accessible for either read or write operations. Similarly, the SDRAM is also inaccessible when the SDRAM is in self-refresh mode, SDRAM0_STATUS[SRSTATUS]=1.

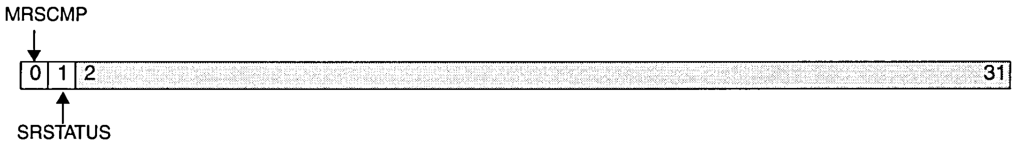


Figure 15-3. Memory Controller Status (SDRAM0_STATUS)

0	MRSCMP	Mode Register Set Complete 0 MRS not complete 1 MRS completed	Set to 1 when the SDRAM controller completes the Mode Register Set Command, which results from setting SDRAM0_CFG[DCE]. Clearing SDRAM0_CFG[DCE] causes this bit to clear in the following MemClkOut1:0 cycle.
1	SRSTATUS	Self-Refresh State 0 Not in Self-Refresh Mode 1 Self-Refresh Mode	See "Self-Refresh" on page 15-19.
2:31	▨	Reserved	

15.3.3 Memory Bank 0–3 Configuration (SDRAM0_B0CR–SDRAM0_B3CR)

These registers are used to configure and enable memory in each respective bank. Only SDRAM banks with SDRAM0_BnCR[BE]=1 are initialized when SDRAM0_CFG[DCE] is set to 1 and subsequently available for access. Since the SDRAM0_BnCR registers cannot be modified when SDRAM0_CFG[DCE]=1, adding or removing memory banks requires that the SDRAM controller be disabled and then reinitialized.

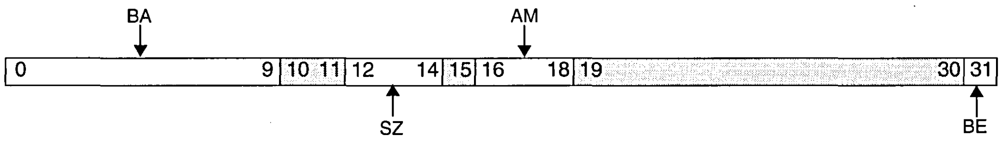


Figure 15-4. Memory Bank 0–3 Configuration Registers (SDRAM0_B0CR–SDRAM0_B3CR)

0:9	BA	Base Address	The base address must be aligned on a boundary that matches the size of the region as defined by the SZ field. For example, a 4MB region must begin on an address that is divisible by 4MB.
10:11	▨	Reserved	
12:14	SZ	Size 000 4M byte 001 8M byte 010 16M byte 011 32M byte 100 64M byte 101 128M byte 110 256M byte 111 Reserved	
15	▨	Reserved	
16:18	AM	Addressing Mode 000 Mode 1 001 Mode 2 010 Mode 3 011 Mode 4 100 Mode 5 101 Mode 6 110 Mode 7 111 Reserved	See "SDRAM Addressing Modes" on page 15-7.
19:30	▨	Reserved	
31	BE	Memory Bank Enable 0 Bank is disabled 1 Bank is enabled	

Table 15-4. SDRAM Addressing Modes

Addressing Mode	SDRAM Memory Organization
1	11 x 9 - 2 Bank 11 x 10 - 2 Bank
2	12 x 9 - 4 Bank 12 x 10 - 4 Bank
3	13 x 9 - 4 Bank 13 x 10 - 4 Bank 13 x 11 - 4 Bank
4	12 x 8 - 2 Bank 12 x 8 - 4 Bank 13 x 8 - 2 Bank
5	11 x 8 - 2 Bank 11 x 8 - 4 Bank
6	13 x 8 - 2 Bank 13 x 8 - 4 Bank
7	13 x 9 - 2 Bank 13 x 10 - 2 Bank

15.3.4 Page Management

The SDRAM controller supports page mode operation with bank interleaving and maintains up to four open pages in the memory subsystem. The SDRAM controller page management unit (PMU) tracks memory accesses (activate, read/write, precharge, and refresh) and maintains a directory of up to four open pages. All PMU entries are allocated and deallocated based on current and pending accesses. Allocated entries are checked against the address of the pending access, and a page hit occurs when a match exists. All PMU directory entries are deallocated when a $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh occurs.

Open pages can be spread across the system memory array on different bank selects ($\overline{\text{BankSeln}}$) or be contained in a single bank select, depending on the memory access sequences and the memory subsystem implementation. For a single bank memory subsystem, the number of open pages is limited to the number of internal banks associated with the SDRAM devices in that bank. For example, a single bank implementation consisting of SDRAMs with two internal banks can have two open pages. In this case, the maximum of two open pages is a limitation of the memory subsystem implementation, not the SDRAM controller.

The SDRAM page size for page hits varies, depending on the address mode programmed in SDRAM0_BnCR[AM]. Table 15-5, "SDRAM Page Size," on page 15-7 details the relationship of the address mode to the page size.

Table 15-5. SDRAM Page Size

Address Mode	Page Size
1,2,3,7	2 KB
4,5,6	1 KB

15.3.5 Logical Address to Memory Address Mapping

SDRAM memory requires that addresses be divided into row and column portions. The relationship between a logical address and the SDRAM row and column address is determined by the address mode programmed in SDRAM0_BnCR[AM] and is shown in Table 15-6.

Table 15-6. Logical Address Bit on BA1:0 and MemAddr12:0 Versus Addressing Mode.

	Bank Size	Address	BA		MemAddr												
Mode	Organization[1]	**Phase**	**1**	**0**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	8 MB	Row	7	9	7	9	10	11	12	13	14	15	16	17	18	19	20
	11 x 9 x 2	Column	7	9	7	9	AP	8	21	22	23	24	25	26	27	28	29
	16 MB	Row	7	9	7	9	10	11	12	13	14	15	16	17	18	19	20
	11 x 10 x 2	Column	7	9	7	9	AP	8	21	22	23	24	25	26	27	28	29
2	32 MB	Row	7	8	7	9	10	11	12	13	14	15	16	17	18	19	20
	12 x 9 x 4	Column	7	8	7	4	AP	6	21	22	23	24	25	26	27	28	29
	64 MB	Row	7	8	7	9	10	11	12	13	14	15	16	17	18	19	20
	12 x 10 x 4	Column	7	8	7	4	AP	6	21	22	23	24	25	26	27	28	29
3	64 MB	Row	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 9 x 4	Column	6	7	7	4	AP	5	21	22	23	24	25	26	27	28	29
	128 MB	Row	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 10 x 4	Column	6	7	7	4	AP	5	21	22	23	24	25	26	27	28	29
	256 MB	Row	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 11 x 4	Column	6	7	7	4[2]	AP	5	21	22	23	24	25	26	27	28	29
4	8/16 MB	Row	8	21	8	9	10	11	12	13	14	15	16	17	18	19	20
	12 x 8 x 2/4	Column	8	21	8	4	AP	6	21	22	23	24	25	26	27	28	29
5	4/8 MB	Row	9	21	9	21	10	11	12	13	14	15	16	17	18	19	20
	11 x 8 x 2/4	Column	9	21	9	21	AP	8	21	22	23	24	25	26	27	28	29
6	16/32 MB	Row	7	21	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 8 x 2/4	Column	7	21	8	9	AP	8	21	22	23	24	25	26	27	28	29
7	32 MB	Row	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 9 x 2	Column	7	7	7	4	AP	6	21	22	23	24	25	26	27	28	29
	64 MB	Row	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 10 x 2	Column	7	7	7	4	AP	6	21	22	23	24	25	26	27	28	29

Note 1: Memory organization is the number of rows x columns x internal banks.

Note 2: Column address bit 10 sent out on MemAddr11 for 13 x 11 x 4 parts.

Note 3: All read operations transfer 64-bits from SDRAM memory to the controller. Therefore, MemAddr0 is first driven with 0 and then 1. For data items narrower than 64-bits, the requested byte(s) are fulfilled from the 64-bit doubleword.

15.3.6 SDRAM Timing Register (SDRAM0_TR)

The SDRAM Timing Register sets the timing parameters for all SDRAM memory banks. This register must be written prior to setting SDRAM0_CFG[DCE]. If SDRAM0_CFG[DCE]=1, writes to this register appear to complete, but do not affect the contents of SDRAM0_TR.

If the SDRAM interface is operated in registered mode, (SDRAM0_CFG[REGE]=1) a programmed $\overline{\text{CAS}}$ latency of 2 clocks (SDRAM0_TR[CASL] = 2'b01) corresponds to a registered $\overline{\text{CAS}}$ latency of 3 clocks, and a programmed latency of 3 clocks (SDRAM0_TR[CASL] = 2'b10) corresponds to a registered $\overline{\text{CAS}}$ latency of 4 clocks. Programming the $\overline{\text{CAS}}$ latency to 4 clocks (SDRAM0_TR[CASL] = 2'b11), corresponding to a registered $\overline{\text{CAS}}$ latency of 5 clocks, is not supported.

See "Selected Timing Diagrams" on page 15-10 for timing diagrams illustrating how the fields in SDRAM0_TR affect the signalling on the SDRAM memory interface.

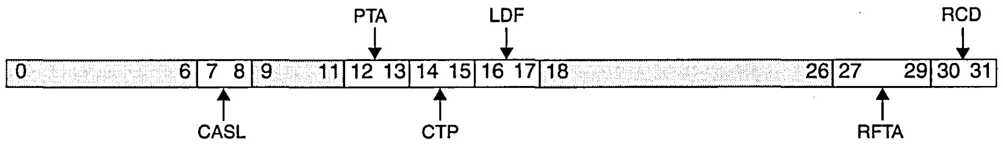


Figure 15-5. SDRAM Timing Register (SDRAM0_TR)

0:6	▨	Reserved
7:8	CASL	SDRAM $\overline{\text{CAS}}$ latency. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
9:11	▨	Reserved
12:13	PTA	SDRAM Precharge Command to next Activate Command minimum. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
14:15	CTP	SDRAM Read / Write Command to Precharge Command minimum. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
16:17	LDF	SDRAM Command Leadoff. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
18:26	▨	Reserved

27:29	RFTA	SDRAM $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ Refresh Command to next Activate Command minimum. 000 4 MemClkOut1:0 cycles 001 5 MemClkOut1:0 cycles 010 6 MemClkOut1:0 cycles 011 7 MemClkOut1:0 cycles 100 8 MemClkOut1:0 cycles 101 9 MemClkOut1:0 cycles 110 10 MemClkOut1:0 cycles 111 Reserved
30:31	RCD	SDRAM RAS to CAS Delay 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles

15.3.7 Selected Timing Diagrams

The SDRAM controller is capable of performing many different types of read and write operations. For example, while the CPU generally issues four doubleword line fills and line writes, the EBC external master interface can issue various types of burst transactions. Since the SDRAM controller is often servicing read and write requests from several masters, the exact sequence of operations on the external SDRAM interface cannot be predicted. Therefore, the timing diagrams in this section should be considered as examples of the signalling that can be observed on the SDRAM interface, and not the only types of transactions that occur.

The timing diagrams in this section are intended to illustrate cycle-based SDRAM programmable timing parameters only. As such, AC specific timing information should not be inferred from the timing diagrams. Instead, please refer to the PPC405GP data sheet for AC specifications.

Table 15-7 summarizes the SDRAM memory timing parameters used to annotate the waveforms. These parameters are set in the SDRAM Timing Register (SDRAM0_TR).

Table 15-7. SDRAM Memory Timing Parameters

Name	Function	Description
RCD	Activate to Read/Write Command	Minimum number of clock cycles from an activate command to a read or write command. Corresponds to SDRAM $\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ assertion delay.
RFTA	Refresh to Activate	Minimum number of clock cycles from a $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh command to the next activate command.
CTP	Command to Precharge	Minimum number of clock cycles from a read or write command to a precharge command.
PTA	Precharge to Active	Minimum number of clock cycles required to wait following a Precharge Command to issuing the next activate command.
CASL	$\overline{\text{CAS}}$ Latency	$\overline{\text{CAS}}$ access latency.
LDF	Command Leadoff Delay	Number of clock cycles from address/command assertion to bank select ($\overline{\text{BankSeln}}$) assertion.

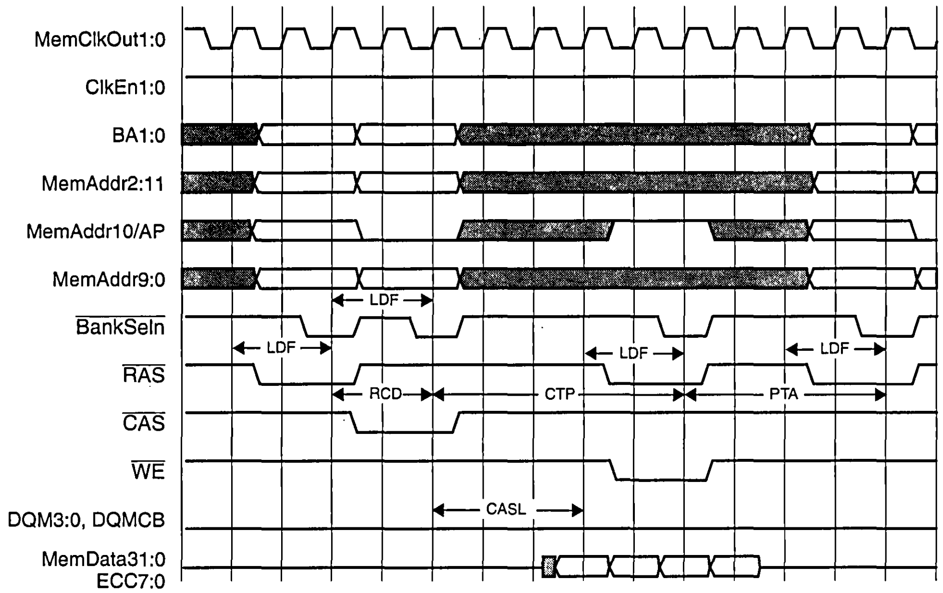


Figure 15-6. Activate, Four Word Read, Precharge, Activate

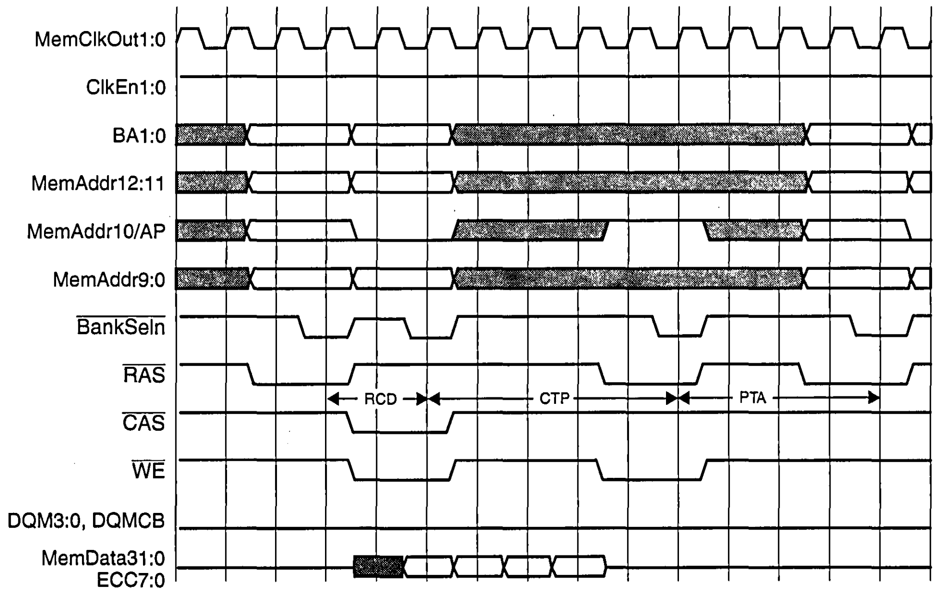


Figure 15-7. Activate, Four Word Write, Precharge, Activate

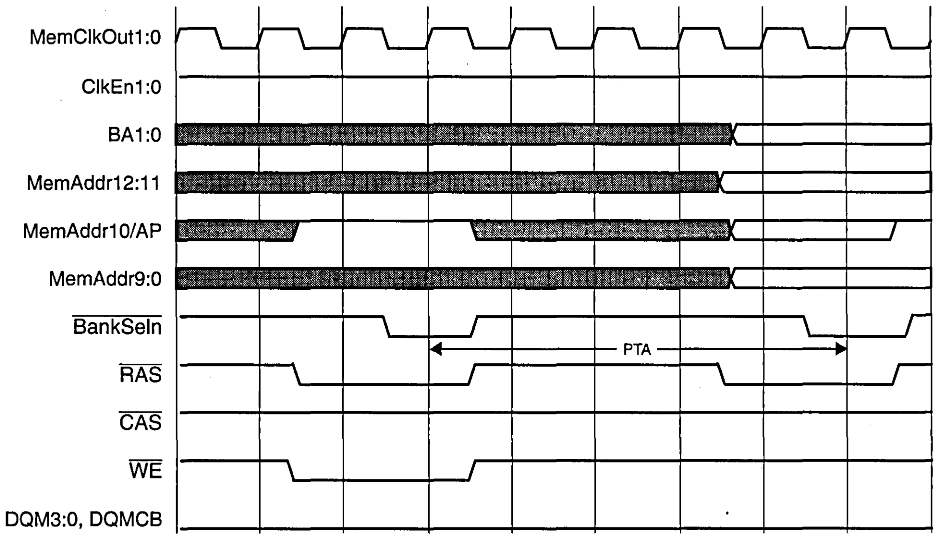


Figure 15-8. Precharge All, Activate

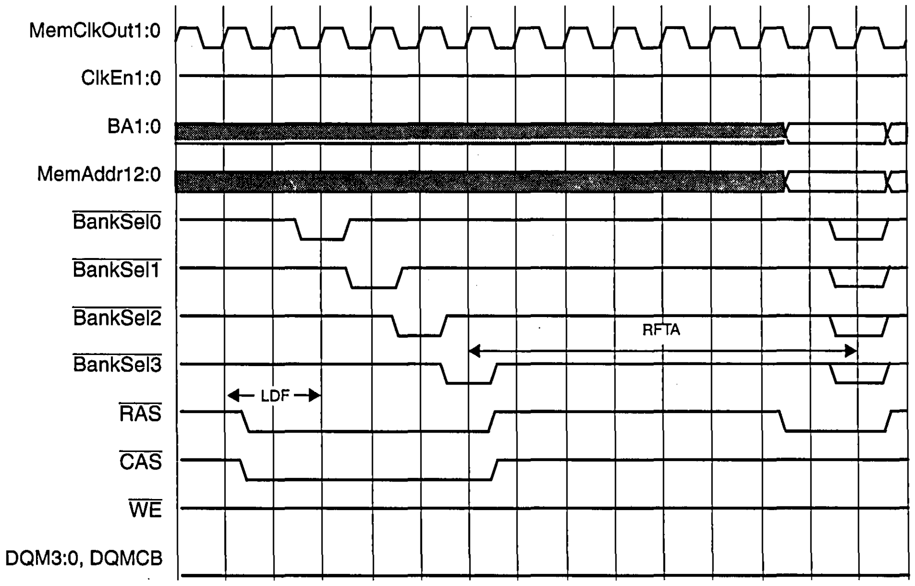


Figure 15-9. $\overline{\text{CAS}}$ Before $\overline{\text{RAS}}$ Refresh

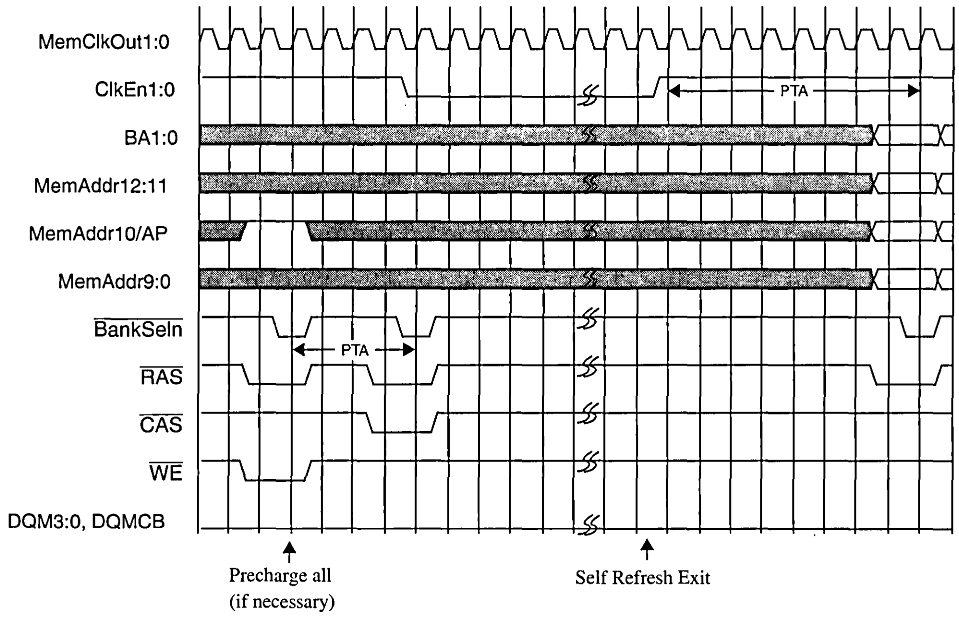


Figure 15-10. Self-Refresh Entry and Exit

15.3.8 Auto ($\overline{\text{CAS}}$ Before $\overline{\text{RAS}}$) Refresh

Refresh of odd memory banks is staggered from the refresh of even memory banks. Only enabled SDRAM banks (SDRAM0_BnCR[BE]=1) are initialized when the controller is enabled (SDRAM0_CFG[DCE] set to 1) and refreshed during normal operation. Once the memory controller is enabled and the initialization sequence has completed, the refresh mechanism starts automatically with refreshing of the memory continuing independent of SDRAM0_CFG[DCE].

Refresh requests are generated internally when the refresh timer expires. The refresh interval is programmable via the Refresh Timer Register (SDRAM0_RTR). During refresh, all SDRAM accesses are delayed until the refresh cycle completes.

15.3.9 Refresh Timer Register (SDRAM0_RTR)

The Refresh Timer Register determines the memory refresh rate for the SDRAM. The internal counter runs at the controller clock frequency, thus if MemClkOut1:0 is 100MHz, a value of 0x05F0 produces a refresh interval of 15.20 s (1520 x 10 ns = 15.20s). This register is progammable to accommodate other SDRAM clock frequencies.

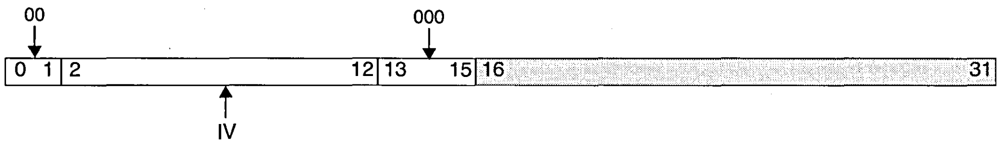


Figure 15-11. Refresh Timing Register (SDRAM0_RTR)

0:1		Always 0b00	
2:12	IV	Interval Programmable between 0b00000000000 and 0b11111111111 (that is, 0x0000 to 0x3FF8 for the complete 16-bit field).	Reset value is 0b00010111110 (that is, 0x05F0 for the complete 16-bit field)
13:15		Always 0b000	
16:31	▨	Reserved	

15.4 Error Checking and Correction (ECC)

Error Checking and Correction (ECC) corrects all single bit errors and detects all double bit errors when reading from SDRAM memory. In addition, ECC detects any and all errors which may exist in an aligned 4-bit nibble. As detailed in Table 15-8, "Additional Latency when using ECC," on page 15-14, the ECC function is transparent in terms of latency, except for partial write operations. On partial writes, a read-modify-write sequence, including bus turn-around, is required to generate the write check bits and store the resultant data.

Table 15-8. Additional Latency when using ECC

PLB Transaction	Added Latency
Read	None
Burst or full single beat write	None
Partial write	SDRAM0_TR[CASL] + 4 clocks

ECC is enabled for all SDRAM banks by setting SDRAM0_CFG[MEMCHK]=1 during the SDRAM initialization procedure. Software is then required to write each word of SDRAM to set the check bits to a valid state. When this is complete, ECC checking and correction is enabled on a per-bank basis through the SDRAM0_ECCCFG register.

15.4.1 ECC Configuration Register (SDRAM0_ECCCFG)

Write access to SDRAM0_ECCCFG is independent of SDRAM0_CFG[DCE]. Software must ensure that the SDRAM controller is idle when updating SDRAM0_ECCCFG. This guarantees that the register update does not affect any in-progress SDRAM operations.

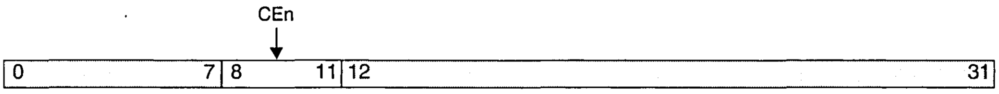


Figure 15-12. ECC Configuration Register (SDRAM0_ECCCFG)

0:7		Reserved	
8:11	CEn	ECC Correction Enable for bank n. 0 Disabled 1 Enabled	When CEn is set, ECC correction is enabled for bank n ($\overline{\text{BankSeln}}$). When cleared, the ECC logic ignores the check bits and passes read data along unmodified.
12:31		Reserved	

15.4.2 Correctable ECC Errors

During SDRAM memory read operations the ECC logic automatically detects and corrects any single bit error that occurs within each 32-bit word of SDRAM memory. This includes errors detected during the read portion of the read-modify-sequence sequence required for partial (less than 64-bit) SDRAM writes.

If a correctable ECC error occurs during a memory read the corrected data is returned to the requesting master. When a correctable error is detected during the read portion of a partial write the corrected data is combined with the write data and written back to memory with new ECC check bits. In both cases, the memory bank ($\overline{\text{BankSeln}}$) that caused the error is logged in SDRAM0_ECCESR[BKnE] and the byte lane that experienced the correctable error is identified in SDRAM0_ECCESR[BLnCE]. Furthermore, the correctable error bit (SDRAM0_ECCESR[CE]) is set causing an ECC Correctable Error interrupt to the Universal Interrupt Controller. The interrupt remains active until software clears SDRAM0_ECCESR[CE].

15.4.3 Uncorrectable ECC Errors

Uncorrectable ECC errors may occur during SDRAM read and SDRAM partial write operations. An uncorrectable error detected during a memory read results in the data from system memory (unchanged) being returned to the requesting master. An uncorrectable error detected on the read portion of a read-modify-write sequence for an SDRAM partial write results in the data from system memory (unchanged) being combined with the write data and written back to memory with new ECC check bits.

Whenever an uncorrectable ECC error occurs the errant memory address is captured in SDRAM0_BEAR. The error status for the master that initiated the memory operation is logged in either SDRAM0_BESR0 or SDRAM0_BESR1. The memory bank ($\overline{\text{BanknSel}}$) that experienced the error is recorded in SDRAM0_ECCESR[BKnE] and an uncorrectable error is flagged via SDRAM0_ECCESR[UE].

Since an uncorrectable error also results in an error signal to the master that initiated the transfer, other side effects may occur. For example, an instruction fetch with an uncorrectable error causes a machine check. In the case of a DMA transfer, the DMA channel stops and an error is logged.

15.4.4 Error Locking

The PCI Bridge and Media Access Layer (MAL) controllers may qualify their PLB transactions to the SDRAM Controller such that the information describing any errors that occur during these transfers becomes locked. When an error is locked, subsequent errors are not permitted to overwrite the information detailing the first error.

When a master requests error locking an error locks not only the SDRAM0_BESRn field for the master, but also SDRAM0_BEAR. These remain locked until software clears them. The SDRAM Controller has a SDRAM0_BESRn field for each PLB master containing two bits associated with error locking. One is the field lock bit and the other is the address lock bit. When an error is detected with locking enabled the field lock bit is set to a value of one. Setting the field lock bit prevents subsequent errors for this master from being logged and overwriting the contents of the field. In addition, the address lock bit is set if no other master has previously locked the SDRAM0_BEAR. Once the SDRAM0_BEAR is locked, no future errors from this or any master can update the SDRAM0_BEAR until software clears the lock bits. When software processes an error it should clear the error status and both lock bits at the same time.

15.4.5 ECC Error Status Register (SDRAM0_ECCESR)

The ECC Error Status Register (SDRAM0_ECCESR) tracks ECC related errors encountered during SDRAM memory accesses. Bits in SDRAM0_ECCESR are cleared by writing a 32-bit value to SDRAM0_ECCESR with a 1 in any bit position that is to be cleared and 0 in all other bit positions.

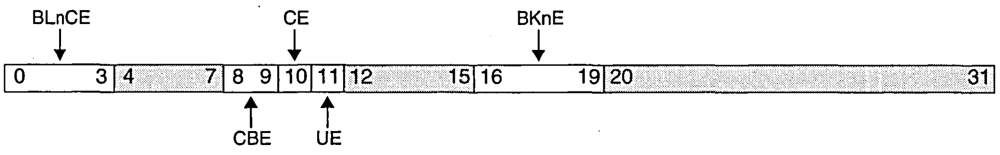


Figure 15-13. ECC Error Status Register (SDRAM0_ECCESR)

0:3	BLnCE	Byte Lane n Corrected Error 0 No error 1 Error occurred in byte lane n
4:7	▨	Reserved
8:9	CBE	Error Detected in Check bits 00 No error 01 Error in lower check bits 10 Error in upper check bits 11 Error in both sets of check bits
10	CE	Correctable Error
11	UE	Uncorrectable Error
12:15	▨	Reserved

16:19	BKnE	Bank n Error 0 No error 1 Error occurred in bank n
20:31		Reserved

15.4.6 Bus Error Address Register (SDRAM0_BEAR)

The SDRAM Bus Error Address Register (SDRAM0_BEAR) is a 32-bit register containing the address of the access where a correctable or uncorrectable ECC error occurred. If the master that initiated the transfer requested error locking, and the SDRAM0_BEAR is not already locked, the contents of SDRAM0_BEAR are locked until the lock bit in one of the Peripheral Bus Error Status Registers (SDRAM0_BESR0 or SDRAM0_BESR1) is cleared. The contents of the SDRAM0_BEAR are accessed indirectly through the SDRAM_CFGADDR and SDRAM0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

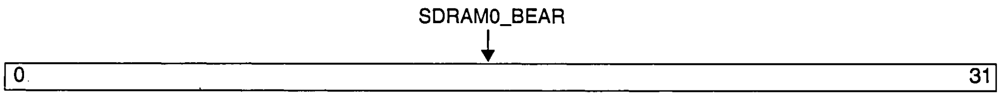


Figure 15-14. Bus Error Address Register (SDRAM0_BEAR)

0:31	SDRAM0_BEAR	Address of ECC Error.

15.4.7 Bus Error Syndrome Register 0 (SDRAM0_BESR0)

This register tracks errors encountered during CPU, external bus master and PCI accesses to SDRAM memory. Bits in SDRAM0_BESR0 are cleared by writing a 32-bit value to SDRAM0_BESR0 with a 1 in any bit position that is to be cleared and 0 in all other bit positions.

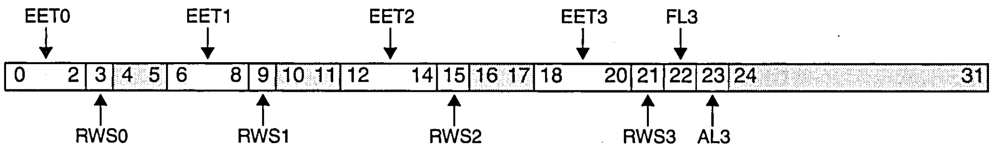


Figure 15-15. Bus Error Syndrome Register 0 (SDRAM0_BESR0)

0:2	EET0	Error type for master 0 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 0 is the processor instruction fetcher.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	

4:5		Reserved	
6:8	EET1	Error type for master 1 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 1 is the processor data side.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10:11		Reserved	
12:14	EET2	Error type for master 2 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 2 is the external bus master.
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16:17		Reserved	
18:20	EET3	Error type for master 3 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 3 is the PCI bridge.
21	RWS3	Read/write status for master 3 0 Error operation was a write operation 1 Error operation was a read operation	
22	FL3	Field lock for master 3 0 EET3 and RWS3 fields are unlocked 1 EET3 and RWS3 fields are locked	
23	AL3	SDRAM0_BEAR address lock for master 3 0 SDRAM0_BEAR address unlocked 1 SDRAM0_BEAR address locked	
24:31		Reserved	

15.4.8 Bus Error Syndrome Register 1 (SDRAM0_BESR1)

This register tracks errors encountered during MAL and DMA accesses to SDRAM memory. Bits in SDRAM0_BESR1 are cleared by writing a 32-bit value to SDRAM0_BESR1 with a 1 in any bit position that is to be cleared and 0 in all other bit positions.

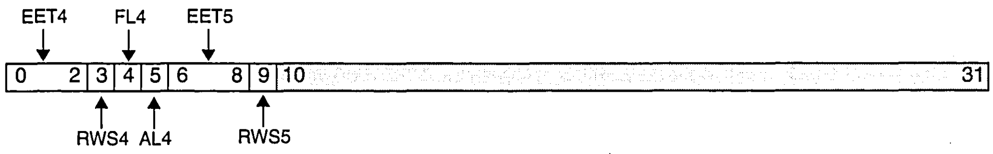


Figure 15-16. Bus Error Status Register 1 (SDRAM0_BESR1)

0:2	EET4	Error type for master 4 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 4 is the MAL.
3	RWS4	Read/write status for master 4 0 Error operation was a write operation 1 Error operation was a read operation	
4	FL4	Field lock for master 4 0 EET4 and RWS4 fields are unlocked 1 EET4 and RWS4 fields are locked	
5	AL4	SDRAM0_BEAR address lock for master 4 0 SDRAM0_BEAR address unlocked 1 SDRAM0_BEAR address locked	
6:8	EET5	Error type for master 5 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 5 is the DMA controller.
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation	
10:31		Reserved	

15.5 Self-Refresh

The SDRAM controller supports self-refresh operation for applications desiring lower power. When the SDRAM memory is placed in self-refresh mode it is no longer accessible for read or write accesses. Prior to placing the SDRAM controller in self-refresh mode all pending and previously queued requests targeting the SDRAM controller must be allowed to complete. Self-refresh entry is then initiated by setting SDRAM0_CFG[SRE]. When set, the SDRAM controller:

1. Completes the current SDRAM operation.
2. Issues precharge all commands to close all open pages.
3. Performs an auto-refresh cycle.
4. Enters self-refresh mode and sets SDRAM0_STATUS[SRSTATUS].

The SDRAM controller maintains the SDRAM in self-refresh mode, independent of any pending memory access requests, until SDRAM0_CFG[SRE] is cleared. Any attempt to read or write SDRAM memory during this time will stall the PLB.

Once SDRAM0_CFG[SRE] is cleared, the SDRAM controller performs the following:

1. Exits self-refresh mode.
2. Performs an auto-refresh cycle.
3. Clears SDRAM0_STATUS[SRSTATUS].

The SDRAM controller is then ready to service any memory request.

15.6 Power Management

The SDRAM controller provides a sleep mode where all SDRAM controller clocking is disabled with the exception of the SDRAM refresh logic and the power management wake-up logic. When the SDRAM controller is in sleep mode SDRAM refresh continues to preserve the contents of the memory and maintain the refresh interval.

15.6.1 Sleep Mode Entry

Sleep mode is enabled by setting SDRAM0_CFG[PME] and CPM0_ER[SDRAM]. Once sleep mode is enabled and the SDRAM controller has been idle for the number of cycles programmed in SDRAM0_PMIT, the SDRAM controller goes to sleep.

15.6.2 Power Management Idle Timer (SDRAM0_PMIT)

The SDRAM0_PMIT register determines the number for SDRAM clock (MemClkOut1:0) cycles that the controller must be idle before it asserts a sleep request when power management is enabled (SDRAM0_CFG[PME]=1). At system reset, SDRAM0_PMIT[CNT] is set to zero. This corresponds to a sleep request after 32 idle cycles.

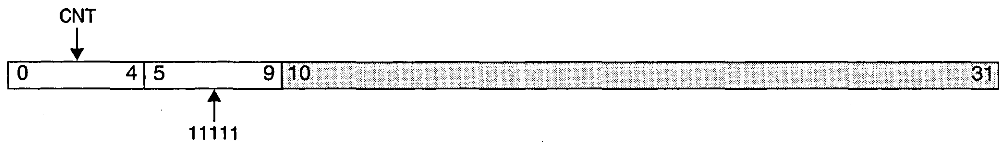


Figure 15-17. Power Management Idle Timer (SDRAM0_PMIT)

0:4	CNT	Count 0-31
5:9		Always 0b11111
10:31	▨	Reserved

15.6.3 Sleep Mode Exit

The power management wake-up logic monitors the PLB for SDRAM reads or writes from other masters. In addition, the wakeup logic also monitor the DCR bus for accesses to SDRAM configuration and status registers. If either a PLB or DCR operation targeting the SDRAM controller is detected the SDRAM controller wakes up. The wakeup process results in a two cycle additional latency to the pending operation.

Chapter 16. External Bus Controller

The PPC405GP External Bus controller (EBC) provides direct attachment for most SRAM/Flash type memory and peripheral devices. The interface minimizes the amount of external glue logic needed to communicate with memory and peripheral devices. This reduces the embedded system device count, circuit board area, and cost.

To eliminate off-chip address decoding, the EBC provides eight programmable chip selects that enable system designers to locate memory and peripherals within the PPC405GP memory map. Chip select, data bus, and associated control signal timings are programmable for both single and burst transfers. For peripherals with variable timing requirements the EBC supports device-paced transfers with optional bus-timeout. System design is further simplified through dynamic bus sizing which supports seamlessly attaching 8-, 16-, and 32-bit wide memories and peripherals. Whenever a size mismatch exists between a read or write operation and the externally attached device, the EBC automatically packs or unpacks data as appropriate.

In addition to the peripheral and memory interface, the EBC includes an external bus master (EBM) interface. Using the EBM, external masters arbitrate and gain access to the peripheral interface. Once an external master owns the peripheral interface it can read and write all PLB- and OPB-addressable memory, with the exception of devices controlled by the EBC. Typical destinations for EBM transactions are PCI address space and SDRAM memory. For EBC-attached peripherals and memory, the external master is required to directly control the target.

16.1 Interface Signals

Figure 16-1 on page 16-2 illustrates the signal I/O between the EBC and the external peripheral bus.

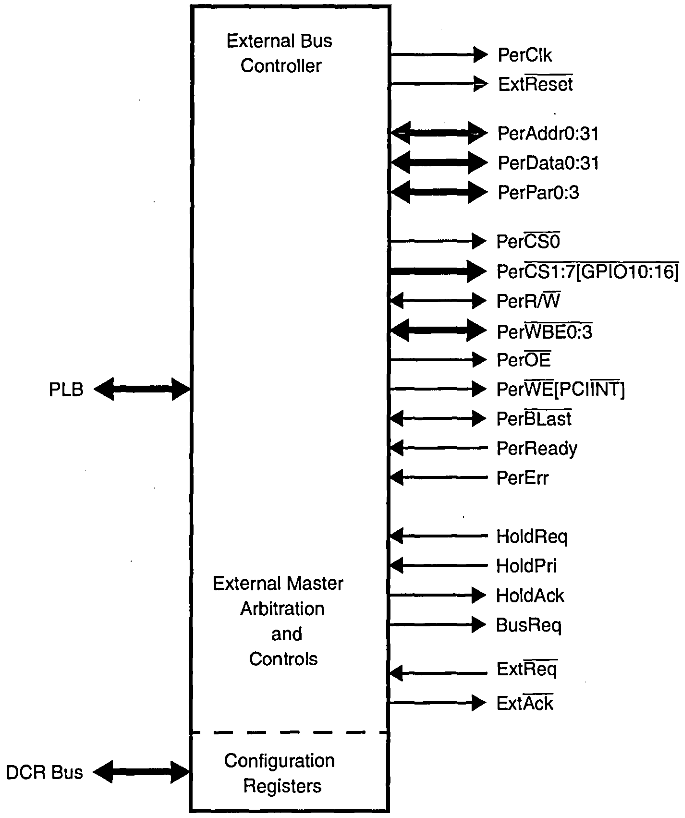


Figure 16-1. External Bus Controller Signals

The usage along with the state of these signals during and after a reset is as follows:

Table 16-1. EBC Signal Usage and State During/Following a Chip or System Reset

Signal	ExtReset=0	ExtReset=1	Usage
PerClk	See Note 1	Toggling	Peripheral bus clock. During an EBC transfer all EBC signal transitions and data sampling occurs synchronous to PerClk.
ExtReset	0	1	Peripheral reset for use by slaves and external bus masters.
PerAddr0:31	High-Z	Last Address	Peripheral address bus. PerAddr0 is the most significant bit.
PerData0:31	High-Z	High-Z	Peripheral data bus. PerData0 is the most significant bit.
PerPar0:3	High-Z	High-Z	Peripheral parity bus. The EBC implements odd parity.
PerCS0:7	High-Z	1	Chip selects. PerCS1:7 are multiplexed with GPIO10:16 and power up as actively driven chip selects. See "Pin Sharing" on page 9-3 for additional details.
PerR/W	High-Z	1	Read not write.

Table 16-1. **EBC Signal Usage and State During/Following a Chip or System Reset**

Signal	$\overline{\text{ExtReset}}$=0	$\overline{\text{ExtReset}}$=1	Usage
$\overline{\text{PerWBE0:3}}$	High-Z	1	Write byte enables or read/write byte enables.
$\overline{\text{PerOE}}$	High-Z	1	Output enable.
$\overline{\text{PerWE}}$	High-Z	High-Z	Write enable. $\overline{\text{PerWE}}$ is low whenever any bit in $\overline{\text{PerWBE0:3}}$ is low and PerR/$\overline{\text{W}}$ = 0. This signal is multiplexed with and defaults to $\overline{\text{PCIINT}}$ after a reset. $\overline{\text{PerWE}}$ is selected by setting CPC0_CR1[PCIPW].
$\overline{\text{PerBLast}}$	High-Z	1	Burst Last. Active during non-burst operations and the last transfer of a burst access.
PerReady	Input	Input	An input to allow external peripherals to perform device-paced transfers.
PerErr	Input	Input	Peripheral data error input.
HoldReq	Input	Input	Hold request, used by an external master to request ownership of the peripheral bus.
HoldPri	Input	Input	Hold priority, used by an external master to indicate the priority in effect for an external master bus tenure.
HoldAck	High-Z	0	Hold acknowledge, informs the external master that it has been granted ownership of the peripheral bus.
BusReq	0	0	Bus request, active when the EBC needs to regain control of the peripheral interface from an external master.
$\overline{\text{ExtReq}}$	Input	Input	External request, used by an external master to indicate that it is prepared to transfer data.
$\overline{\text{ExtAck}}$	High-Z	1	External acknowledge, used by the PPC405GP to indicate that an external master data transfer occurred.

Note 1: During a chip or system reset PerClk begins clocking 64 SysClk cycles prior to the time when the $\overline{\text{ExtReset}}$ output switches from 0 to 1.

16.1.1 Interfacing to Byte, Halfword and Word Devices

Figure 16-2 on page 16-4 illustrates how to interface byte, halfword, and word devices to the peripheral data bus. When devices are connected in this way the EBC supports burst transfers and automatically converts read and write operations to the data width of the external device. As shown in Figure 16-2, halfword devices should not connect to PerAddr31. Similarly, a 32-bit device does not require either PerAddr30 or PerAddr31. Instead, the active byte lanes should be inferred from $\overline{\text{PerWBE0:3}}$, the read/write byte enables.

When a large number of byte and halfword devices are attached to the peripheral data bus, the capacitive loading on byte lane 0 (and byte lane 1, if many halfword devices are used) will be much larger than the loading on byte 3, possibly resulting in unacceptable timing performance on byte 0 or byte 1.

If a bank register is configured as word-wide, then byte-wide devices may be attached to the bus in any byte lane (and accessed using byte loads and stores). Similarly, if a bank register is configured as word-wide, then halfword-wide devices may be attached to the bus in the byte 0/byte 1 lane, or in the byte 2/byte 3 lane, and accessed using halfword loads and stores. External logic may be required to develop additional control signals if the data bus is utilized in this manner.

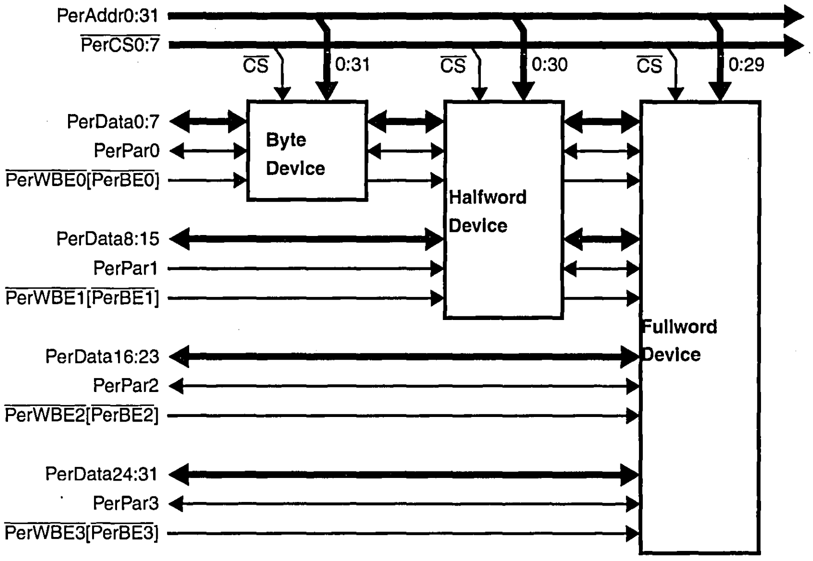


Figure 16-2. Attachment of Devices of Various Widths to the Peripheral Data Bus

16.1.2 Multiplexed I/Os

Seven of the chip select outputs, PerCS1:7, are multiplexed with GPIOs, while PerWE is multiplexed with PCIINT. The multiplexing of these I/Os occurs outside of the EBC logic, see "Pin Sharing" on page 9-3 for additional details. As a result, software can still configure and attempt to use a peripheral memory bank whose output is set up as a GPIO. Doing so causes an EBC transaction without an active chip select and the results are therefore undefined.

In the remainder of this chapter PerCS0:7 and PerWE are assumed to be configured as EBC outputs.

16.1.3 Driver Enables

As shown in Table 16-2, "Effect of Driver Enable Programming on EBC Signal States," on page 16-5, the output enables for the peripheral address, data and most of the EBC control signals are configurable. For systems that do not use an external master or where the external master does not directly control devices on the peripheral bus, setting EBC0_CFG[CSTC]=1 eliminates the need for pull-up resistors on PerCS0:7. Pullups are also unnecessary on the remainder of the EBC control signals when EBC0_CFG[EBTC]=1.

Both chip and system resets set EBC0_CFG[EBTC]=1 and EBC0_CFG[CSTC]=1. In most applications, clearing EBC0_CFG[CSTC] is not recommended. If EBC0_CFG[EBTC]=0, EBC control signals can transition from the active state to high-Z without first being driven inactive. To prevent this, all peripheral banks must be configured with at least one hold cycle, EBC0_BnAP[TH] > 0.

Table 16-2. Effect of Driver Enable Programming on EBC Signal States

EBC Operation	$\overline{\text{ExtReset}}$ **PerClk** $\overline{\text{PerWE}}$ **HoldAck** **BusReq** $\overline{\text{ExtAck}}$	$\overline{\text{PerCS0:7}}$	**PerAddr0:31** **PerR/$\overline{\text{W}}$** $\overline{\text{PerWBE0:3}}$ $\overline{\text{PerOE}}$ $\overline{\text{PerBLast}}$	**PerData0:31** **PerPar0:3**
Reset	High-Z	High-Z	High-Z	High-Z
Idle	Driven	EBC0_CFG[CSTC]	EBC0_CFG[EBTC]	EBC0_CFG[EBTC]
Read	Driven	Driven	Driven	High-Z
Write	Driven	Driven	Driven	Driven
External Master	Driven	EBC0_CFG[CSTC]	High-Z	High-Z

Note 1: If the EBC0_CFG bit is set, the signal is driven to the appropriate state during the indicated EBC operation. Otherwise, the I/O is High-Z.

16.2 Non-Burst Peripheral Bus Transactions

The timing of the $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$, and $\overline{\text{PerWBE0:3}}$ signals is programmable via the Peripheral Bank Access Parameter (EBC0_BnAP) registers. For non-burst transfers, the access parameter registers control the peripheral bus timing as follows:

- $\overline{\text{PerCSn}}$ becomes active 0-3 PerClk cycles (EBC0_BnAP[CSN]) after the address is driven.
- $\overline{\text{PerOE}}$ is driven low 0-3 PerClk cycles (EBC0_BnAP[OEN]) after $\overline{\text{PerCSn}}$ is active.
- $\overline{\text{PerBLast}}$ is active throughout the entire transfer and is driven high during the programmed hold time (EBC0_BnAP[TH]).
- $\overline{\text{PerWBE0:3}}$ can be either write byte enables or read and write enables.

  If EBC0_BnAP[BEM]=0, $\overline{\text{PerWBE0:3}}$ are write byte enables and:

  - $\overline{\text{PerWBE0:3}}$ goes active 0-3 (EBC0_BnAP[WBN]) PerClk cycles after $\overline{\text{PerCSn}}$ becomes active.
  - $\overline{\text{PerWBE0:3}}$ becomes inactive 0-3 (EBC0_BnAP[WBF]) PerClk cycles before $\overline{\text{PerCSn}}$ becomes inactive.

  If EBC0_BnAP[BEM]=1, $\overline{\text{PerWBE0:3}}$ are read/write byte enables and have timing identical to the peripheral address bus. In this case the EBC0_BnAP[WBN] and EBC0_BnAP[WBF] parameters are ignored.
- 1-256 PerClk cycles (EBC0_BnAP[TWT] + 1) after the address became valid:
  - If EBC0_CFG[CSTC]=1 or EBC0_BnAP[TH]>0, $\overline{\text{PerCSn}}$ is driven high.
  - If EBC0_CFG[CSTC]=0 and EBC0_BnAP[TH]=0, $\overline{\text{PerCSn}}$ transitions directly from logic 0 to the high-impedance state.
- The parameters TWT, CSN, OEN, WBN, and WBF in EBC0_BnAP are not independent. For non-burst configured banks it is required that TWT $\geq$ CSN + MAX(OEN,WBN) + WBF.

- The hold time, EBC0_BnAP[TH], is programmable from 0 to 7 PerClk cycles. During the hold time the peripheral address bus remains driven with the last address and all control signals are actively driven high. If the operation was a write, the peripheral data bus continues driving the last data value.
- There is no guarantee of dead cycles between transfers on the peripheral interface. If there are back-to-back transfers to the same memory bank and the number of hold cycles is programmed to zero (EBC0_BnAP[TH]=0) and EBC0_BnAP[CSN]=0, then:
  - $\overline{\text{PerCSn}}$ may not go inactive between the back-to-back transfers.
  - If EBC0_BnAP[OEN]=0, $\overline{\text{PerOE}}$ may not become inactive between the two transfers.
  - If EBC0_BnAP[WBN]=0 and EBC0_BnAP[WBF]=0, $\overline{\text{PerWBE0:3}}$ may not go inactive between the back-to-back transfers.

16.2.1 Single Read Transfer

Figure 16-3 shows the peripheral interface timing for a single read transfer from a non-burst enabled (EBC0_BnAP[BME]=0) bank. The transaction begins with the address being driven. Since this is a single transfer, $\overline{\text{PerBLast}}$ is also driven active along with the address. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on $\overline{\text{PerWBE0:3}}$. $\overline{\text{PerCSn}}$ then becomes active EBC0_BnAP[CSN] cycles after the address, while $\overline{\text{PerOE}}$ goes low EBC0_BnAP[OEN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until EBC0_BnAP[TWT]+1 cycles have elapsed since the start of the transaction and then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PAR]=1) the parity is also read at this time. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles.

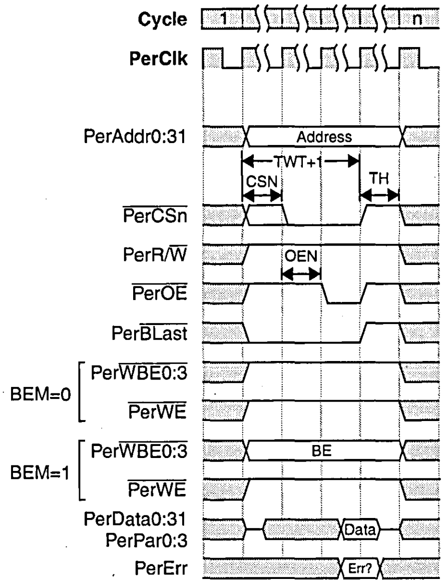


Figure 16-3. Single Read Transfer

16.2.2 Single Write Transfer

Figure 16-4 shows the peripheral interface timing for a single write transfer to a non-burst enabled (EBC0_BnAP[BME]=0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address. At this point the signalling sequence depends on whether or not byte enable mode is enabled for the bank.

- If EBC0_BnAP[BEM]=0, byte enable mode is disabled and the PerWBE0:3 are write byte enables. The appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn. The EBC then waits until (EBC0_BnAP[TWT] - EBC0_BnAP[WBF] + 1) cycles have elapsed since the start of the transaction, then drives all the PerWBE0:3 inactive.
- If EBC0_BnAP[BEM]=1, the PerWBE0:3 lines are byte enables and have the same timing as the peripheral address bus.

After EBC0_BnAP[TWT+1] cycles elapse from the start of transfer, PerCSn and PerBLast are driven high. The EBC then waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

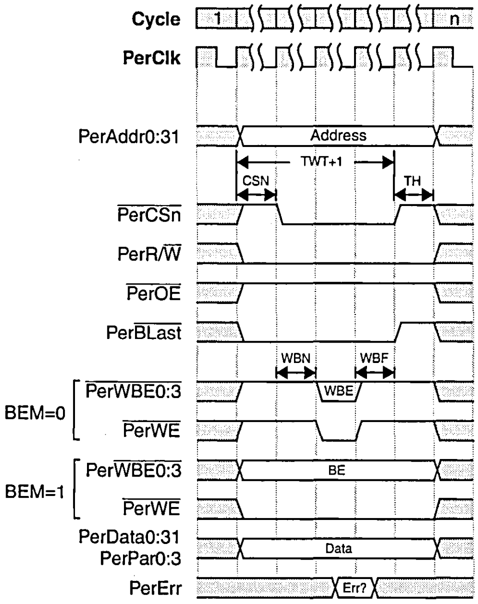


Figure 16-4. Single Write Transfer

16.3 Burst Transactions

Bursting is controlled on a per-bank basis by the Burst Mode Enable bit in the EBC0_BnAP registers. When enabled (EBC0_BnAP[BME]=1) this mode activates bursting for all cache line fills and flushes, PLB burst transfers to the EBC, and all packing and unpacking operations. When bursting is enabled:

- $\overline{\text{PerCSn}}$ becomes active 0-3 (EBC0_BnAP[CSN]) PerClk cycles after the address becomes valid.
- $\overline{\text{PerCSn is no longer actively driven:}}$
  - 1-32 (EBC0_BnAP[FWT]+1) PerClk cycles after the address becomes valid when a single transfer occurs to a burst-enabled bank.
  - 1-8 (EBC0_BnAP[BWT]+1) PerClk cycles after the last address becomes valid during a burst:
    - If EBC0_CFG[CSTC]=1 or EBC0_BnAP[TH]>0, $\overline{\text{PerCSn}}$ is driven high.
    - If EBC0_CFG[CSTC]=0 and EBC0_BnAP[TH]=0, $\overline{\text{PerCSn}}$ transitions directly from logic 0 to the high-impedance state.
- $\overline{\text{During read operations PerOE}}$ is driven low 0-3 (EBC0_BnAP[OEN]) PerClk cycles after $\overline{\text{PerCSn}}$ is active. $\overline{\text{PerOE}}$ goes inactive when $\overline{\text{PerCSn}}$ goes inactive.
- For bursts, the EBC drives a new address (EBC0_BnAP[FWT]+1) + N*(EBC0_BnAP[BWT]+1) cycles after the start of the transaction, where N = 0, 1, 2, ...
- Addresses during a burst may "wrap." For example, cache line fills are processed critical word first.
- During write operations, the write data is driven concurrent with each address.
- $\overline{\text{PerWBE0:3}}$ can be either write byte enables or read and write enables.

  If EBC0_BnAP[BEM]=0, $\overline{\text{PerWBE0:3}}$ are write byte enables and:
  - For the first transfer of a burst, or a single transfer to a burst enabled bank, the appropriate write byte enables go low 0-3 (EBC0_BnAP[WBN]) cycles after $\overline{\text{PerCSn}}$ becomes active. The EBC then waits until EBC0_BnAP[FWT] - EBC0_BnAP[WBF] + 1 cycles have elapsed since the start of the transaction and drives $\overline{\text{PerWBE0:3}}$ inactive.
  - The remaining transfers of the burst are similar, except that $\overline{\text{PerWBE0:3}}$ becomes active EBC0_BnAP[WBN] cycles after each new address is driven on the interface. The $\overline{\text{PerWBE0:3}}$ remain low for (EBC0_BnAP[BWT] + 1) - EBC0_BnAP[WBN] - EBC0_BnAP[WBF] cycles.

  If EBC0_BnAP[BEM]=1, $\overline{\text{PerWBE0:3}}$ are byte enables that have timing identical to the peripheral address bus. In this case the EBC0_BnAP[WBN] and EBC0_BnAP[WBF] parameters are ignored.
- $\overline{\text{PerBLast}}$ is active throughout the entire last (or only) transfer of a burst operation and is deactivated during the programmed hold time (EBC0_BnAP[TH]).
- Access bank parameters CSN and OEN apply to the first (or only) transfer of a burst, while WBN and WBF apply to all transfers. It is required that FWT $\geq$ CSN + MAX(OEN,WBN) + WBF and BWT $\geq$ WBN+WBF.
- Hold time (EBC0_BnAP[TH]) is programmable from 0 to 7 cycles. During the hold time, the peripheral address bus remains driven and all control signals are driven inactive. If the operation was a write, the peripheral data bus continues driving the write data.
- There is no guarantee of dead cycles between transfers on the peripheral interface. If there are back-to-back transfers to the same memory bank and the number of hold cycles is programmed to zero (EBC0_BnAP[TH]=0) and EBC0_BnAP[CSN]=0, then:

- $\overline{\text{PerCSn}}$ may not go inactive between the back-to-back transfers.
- If EBC0_BnAP[OEN]=0, $\overline{\text{PerOE}}$ may not become inactive between the two transfers.
- If EBC0_BnAP[WBN]=0 and EBC0_BnAP[WBF]=0, $\overline{\text{PerWBE0:3}}$ may not go inactive between the back-to-back transfers.

16.3.1 Burst Read Transfer

Figure 16-5 shows the peripheral interface timing for a burst read transfer from a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on $\overline{\text{PerWBE0:3}}$. $\overline{\text{PerCSn}}$ then becomes active EBC0_BnAP[CSN] cycles after the address, while $\overline{\text{PerOE}}$ goes low EBC0_BnAP[OEN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PEN]=1) the parity is also read at this same time.

The next address of the burst is then driven and after EBC0_BnAP[BWT]+1 cycles the EBC performs the next read. The remaining items in the burst are read in the same manner, except that $\overline{\text{PerBLast}}$ is active during the last data element. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

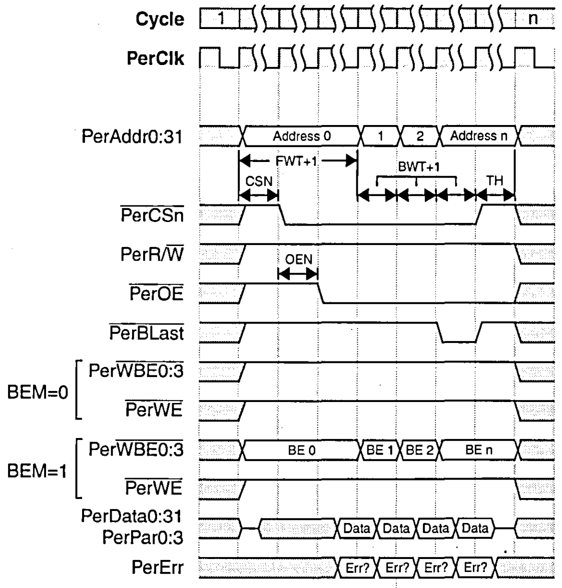


Figure 16-5. Burst Read Transfer

16.3.2 Burst Write Transfer

Figure 16-6 shows the peripheral interface timing for a burst write transfer to burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. At this point the signalling sequence depends on whether byte enable mode is enabled for the bank.

- If EBC0_BnAP[BEM]=0, byte enable mode is disabled and $\overline{\text{PerWBE0:3}}$ are write byte enables. In this case, the appropriate write byte enables go low EBC0_BnAP[WBN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until (EBC0_BnAP[FWT] +1) - EBC0_BnAP[WBF] cycles have elapsed since the start of the transaction and drives $\overline{\text{PerWBE0:3}}$ inactive. EBC0_BnAP[WBF] cycles are then allowed to elapse after which the address and data are output for the second element in the burst. As shown in Figure 16-6, the EBC transfers the subsequent data items in a similar manner.
- If EBC0_BnAP[BEM]=1, the $\overline{\text{PerWBE0:3}}$ lines are byte enables and have the same timing as the peripheral address bus. In this configuration external logic may be necessary to latch write data at the appropriate times.

$\overline{\text{PerBLast}}$ goes low at the beginning of the last transfer to indicate that the burst is ending. The EBC then drives $\overline{\text{PerCSn}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

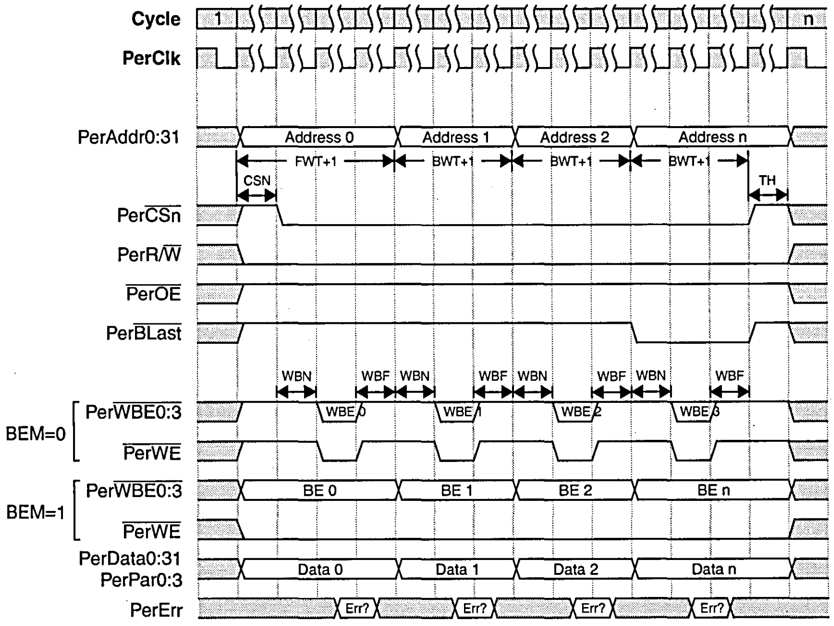


Figure 16-6. Burst Write Transfer

16.4 Device-Paced Transfers

For device-paced transfers, the EBC provides two distinct modes: Sample On Ready enabled and Sample On Ready disabled. The selection of these modes is controlled on a per-bank basis by EBC0_BnAP[SOR]. When Sample On Ready is enabled (EBC0_BnAP[SOR] = 1) data is transferred on the PerClk rising edge where PerReady is sampled active. When Sampling On Ready is disabled (EBC0_BnAP[SOR] = 0), PerReady sampled active causes the data transfer to occur in the next cycle, which results in an additional cycle of wait time.

The ready signal (PerReady) is an input which allows the insertion of externally generated (device-paced) wait states. PerReady is monitored only when EBC0_BnAP[RE]=1.

- For burst disabled banks (EBC0_BnAP[BME] = 0) sampling of the PerReady input starts EBC0_BnAP[TWT] cycles after the beginning of the transfer. Wait states are inserted and sampling continues once per cycle until either PerReady is high when sampled or a timeout occurs.
- For burst enabled banks (EBC0_BnAP[BME] = 1) sampling of the PerReady input starts EBC0_BnAP[FWT] PerClk cycles after the beginning of the first transfer of a burst, and EBC0_BnAP[BWT] cycles after the beginning of subsequent transfers of the burst. Sampling continues once per cycle until either PerReady is sampled high or a timeout occurs.
- When EBC0_BnAP[SOR] = 1 data transfer occurs in the same cycle where PerReady is sampled active. In contrast, if EBC0_BnAP[SOR]=0 the data transfer occurs in the next cycle.
- When EBC0_BnAP[SOR] = 1, if the hold time is set to zero, EBC0_BnAP[TH] = 0, the programmed hold time is ignored and the EBC performs the transaction with one hold cycle.
- When EBC0_BnAP[RE] = 1, the Write Byte Enable Off parameter must be programmed to zero, EBC0_BnAP[WBF] = 0.

The EBC may be programmed to wait only a limited time for PerReady to become active, or it may be programmed for unlimited wait. If EBC0_CFG[PTD] = 1, timeouts are disabled and the EBC waits indefinitely for an active PerReady.

If EBC0_CFG[PTD] = 0, device-paced timeouts are enabled and the EBC only waits for the number of PerClk cycles programmed in EBC0_CFG[RTC]. The timeout counter is reset whenever the peripheral address changes. In this manner each data element within a device-paced burst transaction is treated separately for the purposes of determining whether a timeout error occurs. If PerReady does not become active before the timeout counter reaches the value programmed into EBC0_CFG[RTC], the transfer is aborted and an error is signalled. See “Error Reporting” on page 16-29 for details on how timeout errors are logged.

16.4.1 Device-Paced Single Read Transfer

Figure 16-7 shows the peripheral interface timing for a device-paced single read transfer from a burst disabled (EBC0_BnAP[BME]=0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn.

The EBC then waits until EBC0_BnAP[TWT]+1 cycles have elapsed since the start of the transaction and then begins sampling PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

Once PerReady is sampled active if Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle. The EBC then samples the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PEN]=1) the parity is also read at this time. The EBC then drives PerCSn, PerOE and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending EBC transfers to occur.

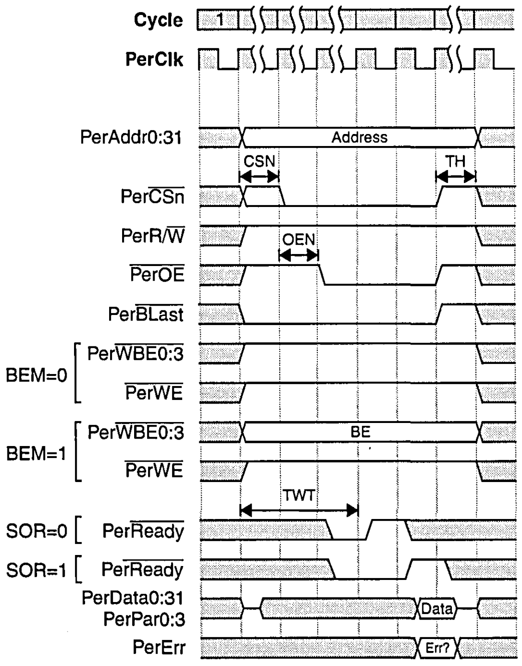


Figure 16-7. Device-Paced Single Read Transfer

16.4.2 Device-Paced Single Write Transfer

Figure 16-8 shows the peripheral interface timing for a device-paced single write transfer from a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. Since this is a single transfer, $\overline{\text{PerBLast}}$ is also driven active along with the address. At this point the signalling sequence depends on whether byte enable mode is enabled for the particular bank.

- If EBC0_BnAP[BEM]=0, byte enable mode is disabled and the $\overline{\text{PerWBE0:3}}$ are write byte enables. The appropriate write byte enables go low EBC0_BnAP[WBN] cycles after $\overline{\text{PerCSn}}$ went low. $\overline{\text{PerWBE0:3}}$ return high on the same PerClk edge that the write data is transferred (see below).
- If EBC0_BnAP[BEM]=1, the $\overline{\text{PerWBE0:3}}$ lines are byte enables and have the same timing as the peripheral address bus.

The EBC then waits until EBC0_BnAP[TWT]+1 cycles have elapsed since the start of the transaction and then begins sample PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle. At this point, the write transfer occurs and the EBC reads the peripheral error input, PerErr. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles.

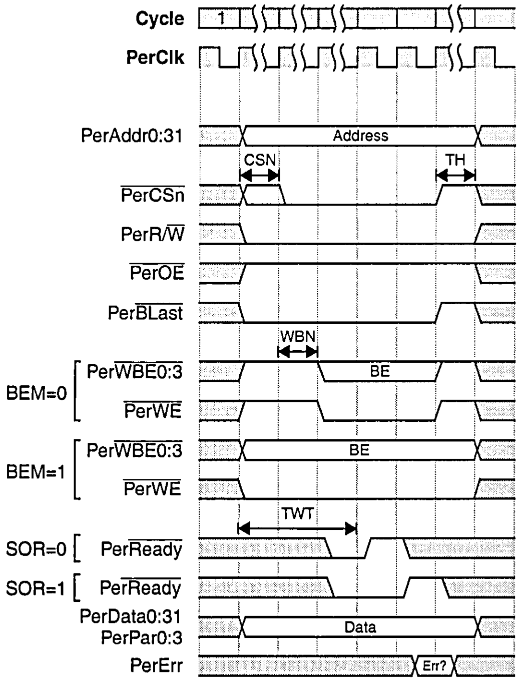


Figure 16-8. Device-Paced Single Write Transfer

16.4.3 Device-Paced Burst Read Transfer

Figure 16-9 shows the peripheral interface timing for a device-paced burst read transfer from a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on $\overline{\text{PerWBE0:3}}$. $\overline{\text{PerCSn}}$ then becomes active EBC0_BnAP[CSN] cycles after the address, while $\overline{\text{PerOE}}$ goes low EBC0_BnAP[OEN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and begins sampling PerReady.

If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle before sampling read data. The EBC then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PEN]=1) the parity is also read.

The next address of the burst is then driven and after EBC0_BnAP[BWT]+1 cycles the EBC waits for PerReady as described above. The remaining items in the burst are read in this same manner, except that $\overline{\text{PerBLast}}$ is active during the last data element. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

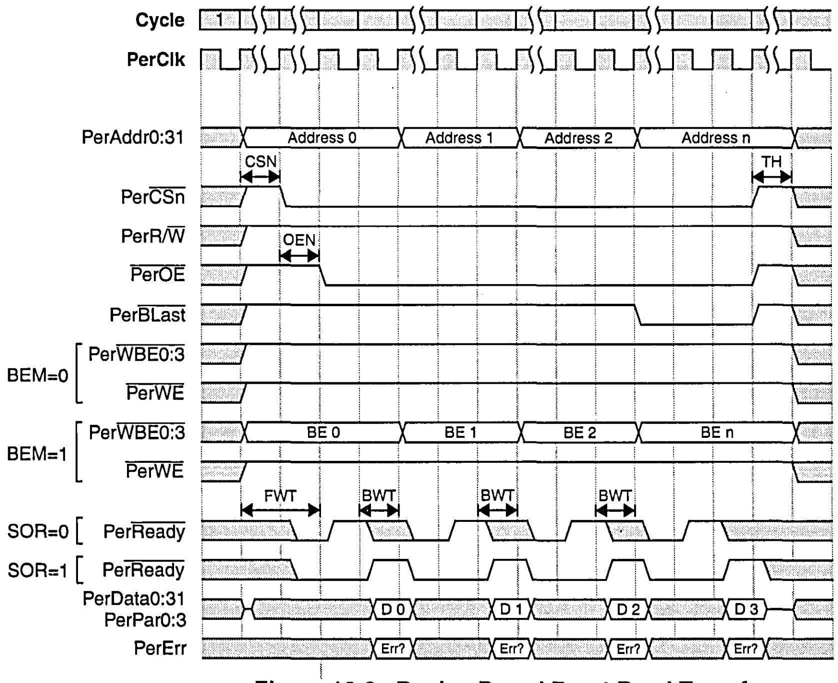


Figure 16-9. Device-Paced Burst Read Transfer

16.4.4 Device-Paced Burst Write Transfer

Figure 16-10 shows the peripheral interface timing for a device-paced burst write transfer to a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. $\overline{\text{PerCSn}}$ then becomes active EBC0_BnAP[CSN] cycles after the address. At this point the signalling sequence depends on whether or not byte enable mode is enabled for the bank.

- If byte enable mode is disabled (EBC0_BnAP[BEM]=0) $\overline{\text{PerWBE0:3}}$ are write byte enables. In this case the appropriate write byte enables go low EBC0_BnAP[WBN] cycles after $\overline{\text{PerCSn}}$ becomes active for the first element in a burst and EBC0_BnAP[WBN] cycles after each new address for the remainder of the burst. If EBC0_BnAP[WBN]<>0, $\overline{\text{PerWBE0:3}}$ is driven inactive on the same PerClk edge that write data is transferred (see below). Otherwise, $\overline{\text{PerWBE0:3}}$ remains low for all data elements in the burst.
- If EBC0_BnAP[BEM]=1, $\overline{\text{PerWBE0:3}}$ are byte enables and have the same timing as PerAddr0:31.

The EBC then waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and begins sampling PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle. At this point the write transfer occurs and the EBC reads the peripheral error input, PerErr.

The next address of the burst is then driven and after EBC0_BnAP[BWT]+1 cycles the EBC waits for PerReady as described above. The remaining items in the burst are transferred in this same manner, except that $\overline{\text{PerBLast}}$ is active for the last data element. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

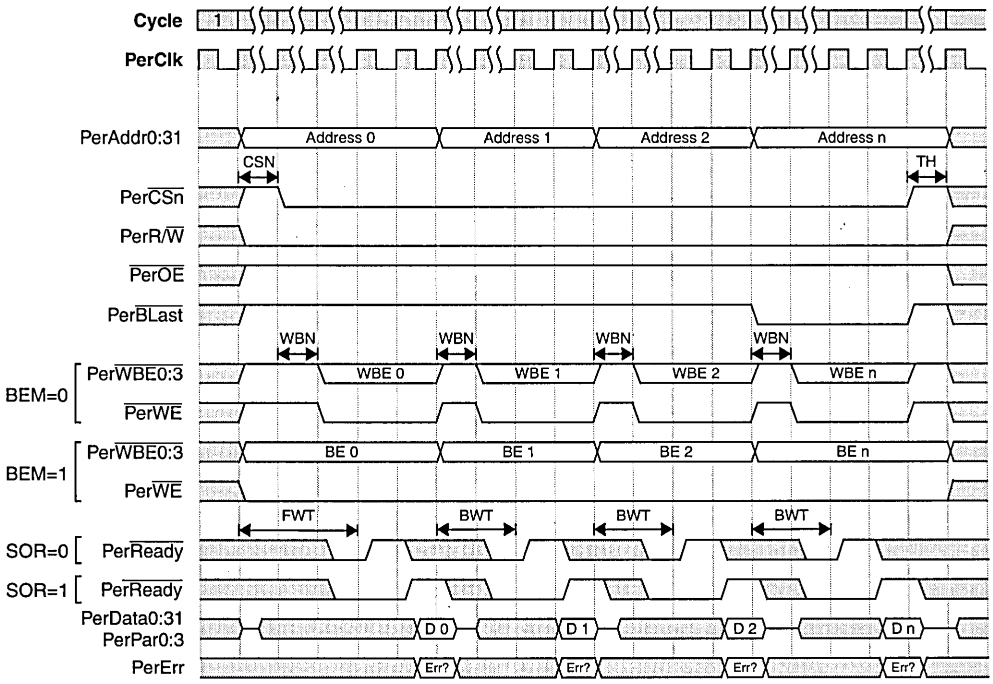


Figure 16-10. Device-Paced Burst Write Transfer

16.5 External Bus Master Interface

The EBC includes an External Bus Master (EBM) interface supporting a shared bus protocol which allows an EBM to gain control of the peripheral bus. Once an external master has been granted access to the peripheral interface it can read and write all PLB- and OPB-addressable memory, with the exception of devices controlled by the EBC. Typical destinations for EBM transactions are PCI address space and SDRAM memory. For EBC-attached peripherals and memory, the external master is required to directly control the target.

Figure 16-11 shows a sample interconnection of the EBC, one SRAM bank, and one external bus master. While only one SRAM bank is shown, the bus master could access all eight of the SRAM banks. Also, with the appropriate arbitration logic, multiple bus masters may be used in a system.

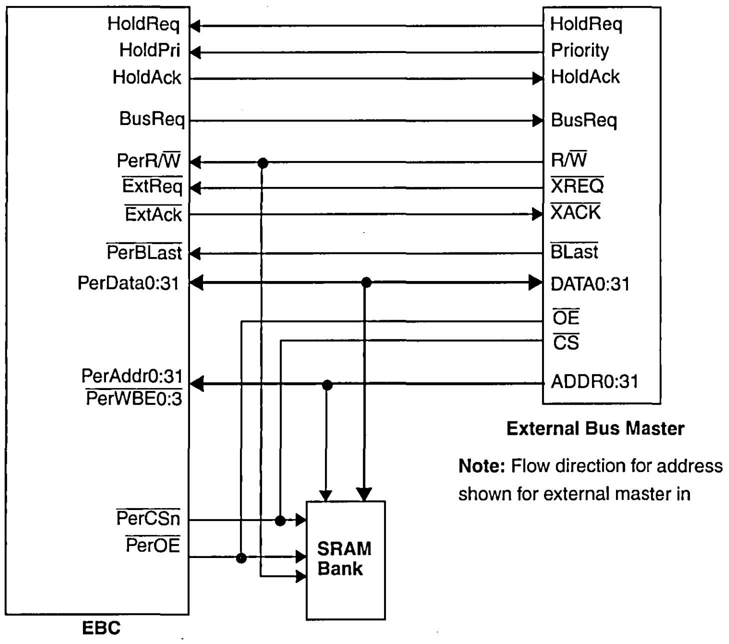


Figure 16-11. Sample External Bus Master System

The EBM interface includes both arbitration and data transfer functions. The arbiter grants the peripheral bus to either the EBC or an external master, while the datapath logic implements an external master to PLB bridge function. This bridge function translates the EBM interface protocol to that required by the on-chip PLB bus.

16.5.1 Arbitration

To gain control of the peripheral bus, the external bus master places an active level on the HoldReq input and drives the HoldPri signal to the desired value (see Figure 16-12 on page 16-20 for an illustrative waveform). The HoldPri input selects the priority of the external master's transactions relative to other EBC operations. Internal to the PPC405GP all reads and writes that target the EBC are assigned the priority 0b10.

If HoldPri=0 the external master priority is set to EBC0_CFG[EMPL], whereas HoldPri=1 sets the priority to EBC0_CFG[EMPH]. Table 16-3 details the only unique ways of programming the external master priority fields in EBC0_CFG, along with the arbiter's response.

Table 16-3. External Master Arbitration.

EMPL	EMPH	HoldPri	When HoldReq=1 HoldAck Becomes Active	When HoldAck=1 BusReq Goes Active
0b11	0b11	X	After any active EBC operation completes.	Never.
0b00	0b11	0	If no EBC transfers are active or pending.	If an EBC transfer is pending.
		1	After any active EBC operation completes.	Never.
0b00	0b00	X	If no EBC transfers are active or pending.	If an EBC transfer is pending.

When an external bus master requests the bus by driving HoldAck=1, the EBC finishes any transfer in progress (except for bursts) before arbitrating between any pending PLB request and the external master request. If the external master presents a higher priority request during a PLB burst, the burst is terminated. Note that processor cache reads and writes are PLB line operations, not PLB bursts, and are never interrupted.

As detailed in Table 16-3, the arbitration logic drives HoldAck active either when the current EBC transaction completes or when there are no EBC transfers pending. Once the external master has been granted the bus, it may either directly control devices on the peripheral bus or read and write PLB- and OPB-addressable memory. If the EBC detects a pending PLB request when the external master owns the peripheral bus (HoldAck=1), BusReq may be asserted to signal the external master that the PPC405GP wants to regain ownership of the peripheral bus. Table 16-3 lists the cases where a PLB request causes BusReq to go active. To ensure fairness and optimize bus utilization, an external master that receives an active BusReq should complete its current transaction and then relinquish ownership of the peripheral bus by driving HoldReq inactive. This is important since the EBC cannot reclaim ownership of the external bus until the external master negates HoldReq.

Table 16-4, "Signal States During Hold Acknowledge (HoldAck=1)," on page 16-18 details the usage of the EBC I/Os when an external master has been granted ownership of the peripheral bus. The usage statements in this table for signals other than HoldPri, HoldReq, HoldAck and BusReq do not apply if an external master directly controls a device on the peripheral bus.

Table 16-4. Signal States During Hold Acknowledge (HoldAck=1)

Signal Name	State	Usage
HoldPri	Input	Requested priority for external master tenure. HoldPri must not change state when HoldReq=1.
HoldReq	Input (=1)	External master must maintain its request when it owns the bus.
HoldAck	Output (=1)	External master is receiving an active bus grant.
BusReq	Output	See Table 16-3 on Page 16-18.
ExtReq	Input	Indicates the external master is ready to transfer data.
ExtAck	Output	Indicates to the external master that a data transfer occurred.

Table 16-4. Signal States During Hold Acknowledge (HoldAck=1)

Signal Name	State	Usage
PerAddr0:31	Input	Requested address from external master.
$\overline{\text{PerWBE0:3}}$	Input	Selects the requested byte(s) for reads and writes.
PerR/$\overline{\text{W}}$	Input	Determines if the operation is a read or write.
$\overline{\text{PerBLast}}$	Input	Active during single transfers and the last transfer of a burst.
PerData0:31	I/O	Read and write data.
$\overline{\text{PerOE}}$	High-Z	Unused.
$\overline{\text{PerWE}}$	1	Unused.
PerErr	High-Z	Unused.
PerPar0:3	High-Z	Unused.

16.5.2 Transaction Overview

The EBM interface supports direct attachment of 8-, 16-, and 32-bit masters. By programming the width of the external master into EBC0_CFG[EMS] the interface accepts write data and provides read data at the appropriate width for the master. The EBM interface includes a 32-byte data buffer, used for both read and write operations between the EBM and PLB- and OPB-mapped memory locations. While write operations only use the buffer during bursts, all reads prefetch one doubleword and burst reads prefetch EBC0_CFG[BPR] doublewords from the source memory into the buffer. This prefetched data remains in the buffer until either a write operation is performed or a read is requested to a different 32-byte block of memory.

To provide the best possible performance, the EBM interface supports both single and burst transactions. Single read transfers result in the EBM reading and buffering a 64-bit doubleword from the requested memory address. The requested read data is then serviced from within this doubleword. If the next operation on the EBM interface is a read and targets this same doubleword, it is serviced directly from the buffer. Burst reads are similar, except that the EBM prefetches four doublewords beginning with the requested word.

Single write transfers result in a separate PLB transaction for each data item. To improve performance, burst writes are gathered in the 32-byte buffer and forwarded in a single PLB transaction to the target memory.

16.5.3 Single Read and Single Write Transfers

Figure 16-12 illustrates external master bus arbitration along with a single read and signal write transfer. An external master requests ownership of the peripheral bus by driving HoldReq active along with the desired priority on HoldPri. Observe that HoldPri must be held at constant value throughout the entire external master tenure. After two or more PerClk cycles the arbiter will grant the peripheral bus to the external master. The delay from when the external master asserts HoldReq to when HoldAck becomes active is variable and depends on any EBC transaction that may be in progress or pending, the level on HoldPri, and the programming in EBC0_CFG[EMPL] and EBC0_CFG[EMPH].

Once the external master is granted the peripheral bus (HoldAck=1), it may either directly control a device on the EBC or issue read and write transactions to the external master interface. This waveform and the ones that follow apply only to the later case. Additionally, cycles shown with breaks in the clock may not be present or may extend for multiple cycles.

To perform a single read operation the external master must:

- Place the desired address on PerAddr0:31.
- Indicate the requested data byte(s) on $\overline{\text{PerWBE0:3}}$.
- Drive PerR/$\overline{\text{W}}$ high.
- Assert $\overline{\text{PerBLast}}$ to mark this as a single transfer.
- Request the transfer by driving and holding $\overline{\text{ExtReq}}$ low.

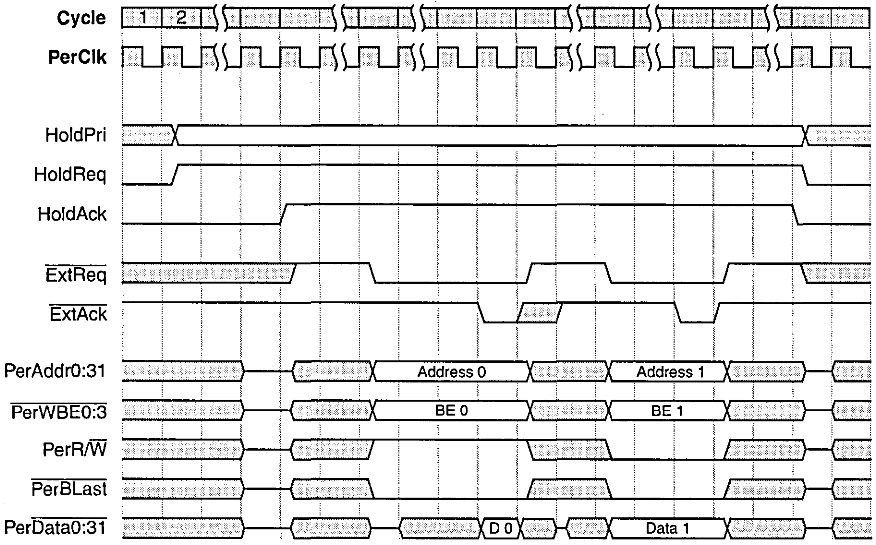


Figure 16-12. External Master Arbitration, Single Read and Single Write

The EBM interface then converts this read request into a PLB transaction and reads the target memory location. $\overline{\text{ExtAck}}$ then goes low when read data is available on PerData. Note that the external master must not remove $\overline{\text{ExtReq}}$ until the cycle after $\overline{\text{ExtAck}}$ becomes active. In addition, $\overline{\text{ExtReq}}$ must be high for at least one cycle between all external master transactions.

Write transfers are similar to reads except that the write data must be provided along with the address and PerR/$\overline{\text{W}}$ is low to indicate a write. As with reads, $\overline{\text{ExtReq}}$ must be held until the cycle after $\overline{\text{ExtAck}}$ is received.

16.5.4 Burst Read Transfer

Burst reads are preferred when accessing sequential addresses as they provide much better performance. Figure 16-13 illustrates an external master burst read transaction. A burst read differs from a single read in that $\overline{\text{PerBLast}}$ is held inactive for all but the last transfer of the burst. In addition, the EBM only requires the address of the first burst element. Following each $\overline{\text{ExtAck}}$ the EBM uses the size of the master configured in EBC0_CFG[EMS] to increment its internal address counter as appropriate.

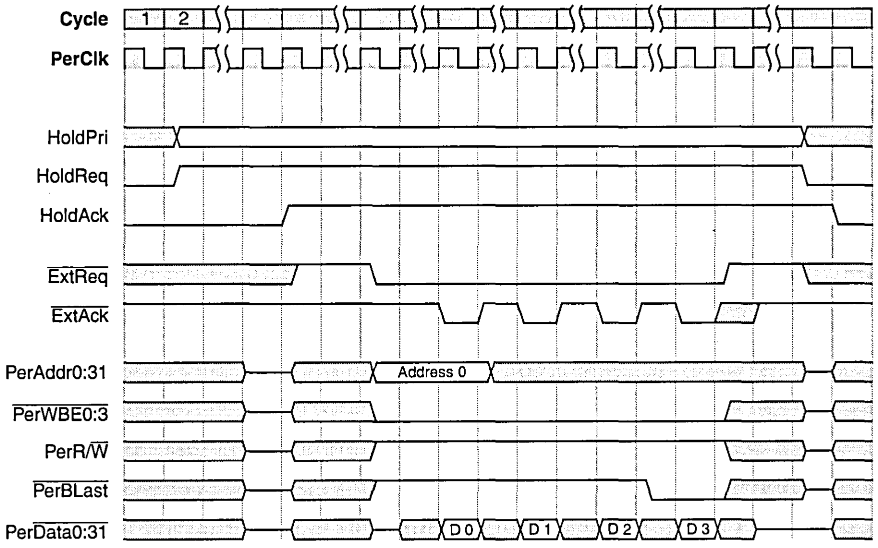


Figure 16-13. External Master Burst Read

16.5.5 Burst Write Transfer

Burst writes are preferred when accessing sequential addresses as they provide much better performance. Figure 16-14 illustrates an external master burst write transaction. A burst write differs from a single write in that $\overline{\text{PerBLast}}$ is held inactive for all but the last transfer of the burst. In addition, the EBM only requires the address of the first burst element. Following each $\overline{\text{ExtAck}}$ the EBM uses the size of the master in EBC0_CFG[EMS] to increment its internal address counter as appropriate.

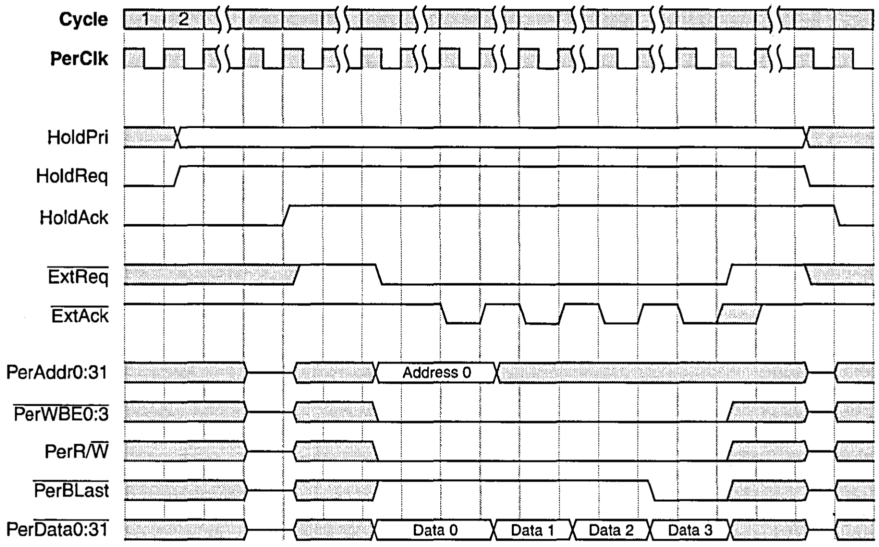


Figure 16-14. External Master Burst Write

16.5.6 External Master Error Interrupts

The EBM can generate an interrupt if a PLB error is encountered while read or writing data. As example, an SDRAM uncorrectable ECC error will cause and EBM interrupt. Please refer to the UIC Chapter for additional information regarding interrupts.

16.6 EBC Registers

All EBC configuration and status registers are accessed using the PowerPC **mtdcr** and **mfdcr** instructions. Access to these registers is performed using an indirect addressing method through the EBC0_CFGADDR and EBC0_CFGDATA registers.

Table 16-5. EBC DCR Addresses

Register	DCR Address	Access	Description
EBC0_CFGADDR	0x012	R/W	External Bus Controller Address Register
EBC0_CFGDATA	0x013	R/W	External Bus Controller Data Register

Table 16-6 lists the indirectly accessed EBC configuration and status registers.

Table 16-6. External Bus Configuration and Status Registers

Mnemonic	Address Offset	Access	Description	Page
EBC0_B0CR–EBC0_B7CR	0x00–0x07	R/W	Peripheral Bank Configuration Registers	16-25
EBC0_B0AP–EBC0_B7AP	0x10–0x17	R/W	Peripheral Bank Access Parameters	16-26
EBC0_BEAR	0x20	R	Peripheral Bus Error Address Register	16-29
EBC0_BESR0	0x21	R/W	Peripheral Bus Error Status Register 0	16-30
EBC0_BESR1	0x22	R/W	Peripheral Bus Error Status Register 1	16-32
EBC0_CFG	0x23	R/W	EBC Configuration Register	16-23

To access one of these registers, software must first write the address offset into the EBC0_CFGADDR register. The target register can then be read or written through the EBC0_CFGDATA DCR address. The following PowerPC code illustrates this procedure by writing the EBC0_B0CR register and then reading back the written value.

```
li        r3,EBC0_B0CR              ! address offset
lis       r4,<config upper>         ! upper 16-bits of configuration data
ori       r4,r4,<config lower>      ! lower 16-bits of configuration data
mtdcr     EBC0_CFGADDR,r3           ! set offset addr
mtdcr     EBC0_CFGDATA,r4           ! write config data
mfdcr     r5,EBC0_CFGDATA           ! read back config data
```

16.6.1 EBC Configuration Register (EBC0_CFG)

The contents of EBC0_CFG are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

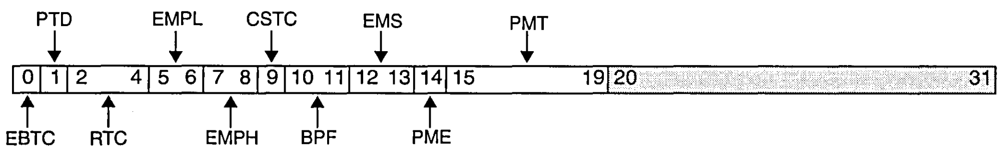


Figure 16-15. EBC Configuration Register (EBC0_CFG)

0	EBTC	External Bus Three-State Control 0 Address, data and control signals are high-Z between EBC transfers. 1 Between EBC transfers the peripheral data bus, address bus and control signals are driven.	Default after reset is EBTC=1. See "Effect of Driver Enable Programming on EBC Signal States" on page 16-5.
1	PTD	Device-Paced Time-out Disable 0 Enabled time-outs 1 Disable time-outs	If PTD=1, the EBC waits indefinitely for assertion of PerReady during device-paced accesses.
2:4	RTC	Ready Timeout Count 000 16 PerClk cycles 001 32 PerClk cycles 010 64 PerClk cycles 011 128 PerClk cycles 100 256 PerClk cycles 101 512 PerClk cycles 110 1024 PerClk cycles 111 2048 PerClk cycles	When PTD=0, the number of cycles from PerAddr0:31 changing until a time-out error occurs.
5:6	EMPL	External Master Priority Low 00 Low 01 Medium low 10 Medium high 11 High	The PLB priority for external master initiated transfers when the external master hold priority input is low (HoldPri=0).
7:8	EMPH	External Master Priority High 00 Low 01 Medium low 10 Medium high 11 High	The PLB priority for external master initiated transfers when the external master hold priority input is high (HoldPri=1).
9	CSTC	Chip Select Three-state Control 0 $\overline{\text{PerCS0:7}}$ are high-Z between EBC transfers and when an external master is active (HoldAck=1) 1 $\overline{\text{PerCS0:7}}$ are always driven.	Default after reset is CSTC=1. See "Effect of Driver Enable Programming on EBC Signal States" on page 16-5.
10:11	BPF	Burst Prefetch 00 Prefetch 1 doubleword 01 Prefetch 2 doublewords 10 Prefetch 4 doublewords 11 Reserved	Controls the amount of data prefetching when the EBC is servicing a PLB burst read. For most applications set this field to 0b00.

12:13	EMS	External Master Size 00 8-bit 01 16-bit 10 32-bit 11 No external master attached	Width of the attached external master.
14	PME	Power Management Enable 0 Disabled 1 Enabled	
15:19	PMT	Power Management Timer 0-31	The EBC makes a sleep request to the Clock and Power Management unit when PME=1 and the EBC has been idle for 32*PMT PerClk cycles.
20:31		Reserved	

16.6.2 Peripheral Bank Configuration Registers (EBC0_BnCR)

These registers must be configured to enable memory in each respective bank. Boot ROM must be attached to bank 0 if installed.

If a boot ROM is present, the bank 0 starting address register is loaded with a value of 0xFFE, and the bank 0 size register is loaded with a value of 0b001 (2MB) immediately following $\overline{\text{SysReset}}$ inactive.

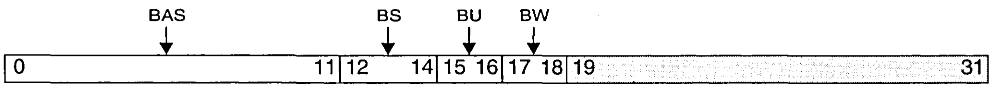


Figure 16-16. Peripheral Bank Configuration Registers (EBC0_BnCR)

0:11	BAS	Base Address Select	Specifies the bank starting address, which must be a multiple of the bank size.
12:14	BS	Bank Size 000 1 MB bank 001 2 MB bank 010 4 MB bank 011 8 MB bank 100 16 MB bank 101 32 MB bank 110 64 MB bank 111 128 MB bank	
15:16	BU	Bank Usage 00 Disabled 01 Read-only 10 Write-only 11 Read/Write	Specifies the type of accesses allowed for the bank. A protect error occurs if a write is attempted to a read-only bank or a read from a write-only bank.

17:18	BW	Bus Width 00 8-bit bus 01 16-bit bus 10 32-bit bus 11 Reserved
19:31		Reserved

- **BAS (Base Address Select, bits 0:11)** – Sets the base address for a peripheral device. The bank starting address must be a multiple of the bank size programmed in the BS field. The BAS field is compared to bits 0:11 of the address. If the address is within the range of a BAS field, the associated bank is enabled for the transaction.

  Multiple bank registers may be inadvertently programmed with the same base address or as overlapping banks. An attempt to use such overlapping banks is recorded in EBC0_BESR0 or EBC0_BESR1 as a configuration error and no external access occurs. This error may result in a machine check exception if the requesting master is the CPU. If the error occurred during a DMA access, the DMA may signal an interrupt to the PPC405GP through the UIC.
- **BS (Bank Size, bits 12:14)** – Sets the number of bytes which the bank may access, beginning with the base address set in the BAS field.
- BU (Bank Usage, bits 15:16) – Protects banks of physical devices from read or write accesses.

  When a write access is attempted to an address within the range of the BAS field, and the bank is designated as read-only, a protection error occurs. Also, when a read access is attempted to an address within the range of the BAS field, and the bank is designated as write-only, a protection error occurs. The address of the attempted access is logged in EBC0_BEAR and type of error is logged in either EBC0_BESR0 or EBC0_BESR1.
- BW (Bus Width, bits 17:18) – Controls the width of region accesses. If the BW field is 0b00, the region is configured for an 8-bit data bus; 0b01 indicates a 16-bit data bus and 0b10 indicates a 32-bit data bus. If devices are attached to the data bus as shown in Figure 16.1.1 on page 16-3, the EBC automatically packs read data and unpacks write data when a data transfer size mismatch exists.

16.6.3 Peripheral Bank Access Parameters (EBC0_BnAP)

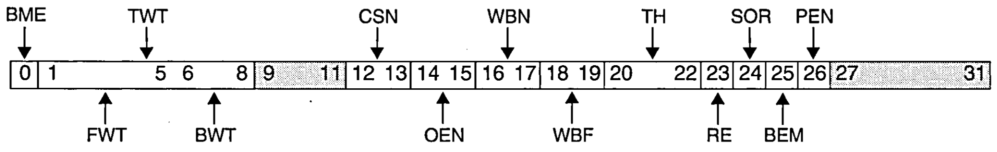


Figure 16-17. Peripheral Bank Access Parameters (EBC0_BnAP)

0	BME	Burst Mode Enable 0 Bursting is disabled 1 Bursting is enabled	
1:8	TWT	Transfer Wait 0-255 PerClk cycles	Wait states on all transfers when BME=0.

1:5	FWT	First Wait 0-31 PerClk cycles	If BME=1, number of wait states on the first transfer of a burst.
6:8	BWT	Burst Wait 0-7 PerClk cycles	If BME=1, number of wait states on non-first transfers of a burst.
9:11		Reserved	
12:13	CSN	Chip Select On Timing 0-3 PerClk cycles	Number of cycles from peripheral address driven to $\overline{\text{PerCSn}}$ low.
14:15	OEN	Output Enable On Timing 0-3 PerClk cycles	Number of cycles from $\overline{\text{PerCSn}}$ low to $\overline{\text{PerOE}}$ low.
16:17	WBN	Write Byte Enable On Timing 0-3 PerClk cycles	If BEM=0, number of cycles from $\overline{\text{PerCSn}}$ low to $\overline{\text{PerWBE0:3}}$ active.
18:19	WBF	Write Byte Enable Off Timing 0-3 PerClk cycles	If BEM=0 and RE=0, number of cycles $\overline{\text{PerWBEn}}$ becomes inactive prior to $\overline{\text{PerCSn}}$ inactive.
20:22	TH	Transfer Hold 0-7 PerClk cycles	Contains the number of hold cycles inserted at the end of a transfer.
23	RE	Ready Enable 0 PerReady is disabled 1 PerReady is enabled	
24	SOR	Sample on Ready 0 Data transfer occurs one PerClk cycle after PerReady is sampled active 1 Data transfer occurs in the same PerClk cycle that PerReady becomes active	
25	BEM	Byte Enable Mode 0 $\overline{\text{PerWBE0:3}}$ are only active for write cycles 1 $\overline{\text{PerWBE0:3}}$ are active for read and write cycles	If BEM=0, $\overline{\text{PerWBE0:3}}$ timing is controlled by WBN and WBF. If BEM=1, $\overline{\text{PerWBE0:3}}$ has the same timing as PerAddr0:31.
26	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	The EBC implements odd parity.
27:31		Reserved	

- BME (Burst Mode Enable, bit 0) – Controls bursting for cache line fills and flushes, PLB burst transfers and all packing and unpacking operations. If BME=1, bursting is enabled. When bursting is enabled the parameters Chip Select On (CSN), Output Enable On (OEN), and First Wait (FWT) apply only to the first transfer, while Burst Wait (BWT) and Write Byte Enable On (WBN) apply during all remaining transfers of the burst.
- TWT (Transfer Wait, bits 1:8) – Specifies the number of wait states taken by each transfer to the bank. The number of cycles from address valid to the deassertion of $\overline{\text{PerCSn}}$ is (1 + TWT), where $0 \le \text{TWT} \le 255$. This field is used for non-burst transfers (field BME = 0).
- FWT (First Wait, bits 1:5) – Specifies the number of wait states to be taken by the first access to the bank during a burst transfer (field BME = 1). During a burst the number of cycles from the first address valid to the second address is (1 + FWT), where $0 \le \text{FWT} \le 31$.

- BWT (Burst Wait, bits 6:8) – Specifies the number of wait states to be taken by accesses beyond the first during a burst transfer (field BME = 1). On burst accesses except for the last, the number of cycles from address valid to the next valid address on each burst access is (1 + BWT), where $0 \le \text{BWT} \le 7$. On the last burst access, the number of cycles from address valid to the deassertion of $\overline{\text{PerCSn}}$ is (1 + BWT), where $0 \le \text{BWT} \le 7$.
- CSN (Chip Select On Timing, bits 12:13) – Specifies the chip select turn on delay relative to the address. $\overline{\text{PerCSn}}$ may turn on coincident with the address or be delayed by 1, 2, or 3 PerClk cycles.
- OEN (Output Enable On Timing, bits 14:15) – Specifies when the output enable signal, $\overline{\text{PerOE}}$, is asserted for read operations relative to the chip select signal. If 0, $\overline{\text{PerOE}}$ is asserted coincident with the chip select. If 1, 2 or 3, $\overline{\text{PerOE}}$ is delayed by 1, 2, or 3 PerClk cycles.
- WBN (Write Byte Enable On Timing, bits 16:17) – Specifies when the write byte enables, $\overline{\text{PerWBE0:3}}$, are asserted relative to the chip select signal. If 0, then $\overline{\text{PerWBE0:3}}$ turns on coincident with the chip select. If 1, 2, or 3, then $\overline{\text{PerWBE0:3}}$ is delayed 1, 2, or 3 PerClk cycles from the chip select.
- WBF (Write Byte Enable Off Timing, bits 18:19) – Specifies when the write byte enables are deasserted, relative to the deassertion of the chip select signal. If WBF=0, $\overline{\text{PerWBE0:3}}$ goes high coincident with the chip select signal. If WBF is 1, 2, or 3, then $\overline{\text{PerWBE0:3}}$ turns off 1, 2, or 3 PerClk cycles before the turn-off of the chip select signal.

  Programming Note: It is an error to set WBF > BWT. Moreover, for device-paced transfers (EBC0_BnAP[RE]=1) WBF must be set to zero.
- TH (Transfer Hold, bits 20:22) – Specifies the number of PerClk cycles (0 through 7) that the peripheral bus is held idle after the deassertion of $\overline{\text{PerCSn}}$. During these cycles, the address bus and data bus are active and $\text{PerR}/\overline{\text{W}}$ is valid. During the hold time, chip select, output enable, and write byte enables are inactive. If Ready Mode is used (RE=1) along with Sample on Ready (SOR=1) TH must be set to at least 1.
- RE (Ready Enable, bit 23) – Controls the use of the PerReady input signal. If RE=0, the PerReady input is ignored and no additional wait states are inserted into bus transactions. If RE=1, the PerReady input is examined after the wait period expires; additional wait states are inserted if the PerReady input is 0. The maximum number of wait states in each transaction is determined by the settings in the Device-Paced Timeout Disable (PTD) and Ready Timeout Counter (RTC) fields in EBC0_CFG. If EBC0_CFG[PTD]=0, the ready timeout function is disabled and the PPC405GP waits indefinitely until PerReady=1. If EBC0_CFG[PTD]=1, the PPC405GP waits the number of cycle indicated by EBC0_CFG[RTC] cycles for PerReady to become active. If PerReady does not become active in the allotted time, the address of the error is logged in EBC0_BEAR and the type of error is captured in either EBC0_BESR0 or EBC0_BESR1.
- **SOR (Sample Ready, bit 24)** – Controls the location of the data transfer cycle with respect to the PerReady input. If SOR=1 the data transfer occurs on the same PerClk edge that PerReady is sampled active, whereas if SOR=0 the data transfer occurs one cycle later.
- **BEM (Byte Enable Mode, bit 25)** – Controls whether $\overline{\text{PerWBE0:3}}$ is active during writes or for both reads and writes.
- **PEN (Parity Enable Mode, bit 26)** – Enables parity generation and checking.

16.7 Error Reporting

The EBC monitors four kinds of errors when performing read and write transfers. Of these four, bank protect and external bus errors are always checked, while timeout and read parity error checking must be enabled via DCR-mapped configuration registers.

- **Protect Error** – Requested read or write operation violates the bank usage programmed in EBC0_BnCR[BU]. For example, write attempt to read-only bank. In all cases, no external bus activity occurs.
- **External Bus Error** – The PerErr input was sampled active during the data transfer cycle of a read or write operation. The associated data is read or written as usual.
- **Timeout Error** – This error is possible during memory operations when both PerReady sampling is enabled, EBC0_BnAP[RE]=1, and device paced timeouts are enabled, EBC0_CFG[PTD]=0. Whenever the peripheral address bus changes the EBC begins counting PerClk cycles. If the count reaches the value represented by EBC0_CFG[RTC] a timeout error occurs. Note that timeout errors are not possible during the peripheral portion of DMA transfers.
- **Parity Error** – Indicates that the parity calculated for the read data did not match the parity read. Parity generation and checking is enabled for memory operations by setting EBC0_BnAP[PEN]=1 and for DMA peripheral transfers by programming DMA0_CRn[PCE]=1.

  When the EBC slave detects one of the above errors it reports the error condition to the PLB master that initiated the transfer. The EBC also logs the type of error into EBC0_BESR0 or EBC0_BESR1 and the address of the error in EBC0_BEAR.

16.7.1 Error Locking

The PCI Bridge and Media Access Layer (MAL) controllers may qualify their PLB transactions to the EBC such that the information describing any errors that occur during these transfers becomes locked. When an error is locked, subsequent errors are not permitted to overwrite the information detailing the first error.

When a master requests error locking an error locks not only the EBC0_BESRn field for the master, but also the EBC0_BEAR. These remain locked until software clears them. For each PLB master that supports error locking the EBC has a EBC0_BESRn field containing two bits associated with error locking. One is the field lock bit and the other is the address lock bit. When an error is detected with locking enabled the field lock bit is set to a value of one. Setting the field lock bit prevents subsequent errors for this master from being logged and overwriting the contents of the field. In addition, the address lock bit is set if no other master has previously locked the EBC0_BEAR. Once the EBC0_BEAR is locked, no future errors from this or any master can update the EBC0_BEAR until software clears the lock bits. When software processes an error it should clear the error status and both lock bits at the same time.

16.7.2 Peripheral Bus Error Address Register (EBC0_BEAR)

The Peripheral Bus Error Address Register (EBC0_BEAR) is a 32-bit register containing the address of the access where a data bus error occurred. If the master that initiated the transfer requested error locking, and the EBC0_BEAR is not already locked, the contents of EBC0_BEAR are locked until the lock bit in one of the Peripheral Bus Error Status Registers (EBC0_BESR0 or EBC0_BESR1) is cleared. The contents of the EBC0_BEAR are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

Precise address capture in the EBC0_BEAR when a parity error occurs only applies to devices with at least one cycle of hold time: EBC0_BnAP[TH] > 0. This is because parity errors are not calculated until the cycle after the data was valid on the external bus, and the EBC0_BEAR is loaded with the address on the bus during that cycle. If the device timings do not include at least one hold cycle, the address may transition to the next address in a burst or the address of a new transaction before the EBC0_BEAR latches the value.

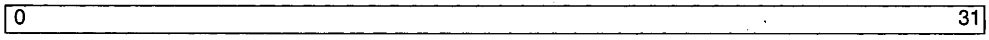


Figure 16-18. Peripheral Bus Error Address Register (EBC0_BEAR)

0:31		Address of Bus Error (asynchronous)

16.7.3 Peripheral Bus Error Status Register 0 (EBC0_BESR0)

The Peripheral Bus Error Status Register 0 (EBC0_BESR0) records the occurrence and type of errors for transactions attempted on behalf of the CPU, EBC External Bus Master and PCI Bridge. The contents of EBC0_BESR0 are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

It is possible to have both a parity error and a bus error during the same data transfer. If this occurs, the bus error is detected first and EBC0_BESR0 and EBC0_BEAR are updated. In the next cycle the parity error is detected and, if error locking is not enabled, logged in EBC0_BESR0.

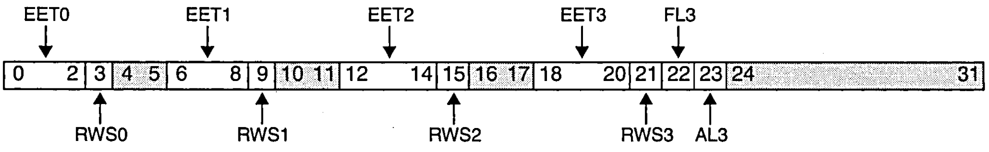


Figure 16-19. Peripheral Bus Error Status Register 0 (EBC0_BESR0)

0:2	EET0	Error type for master 0 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 0 is the processor instruction fetcher.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	

4:5		Reserved	
6:8	EET1	Error type for master 1 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 1 is the processor data side.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10:11		Reserved	
12:14	EET2	Error type for master 2 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 2 is the external bus master.
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16:17		Reserved	
18:20	EET3	Error type for master 3 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 3 is the PCI bridge.
21	RWS3	Read/write status for master 3 0 Error operation was a write operation 1 Error operation was a read operation	
22	FL3	Field lock for master 3 0 EET3 and RWS3 fields are unlocked 1 EET3 and RWS3 fields are locked	
23	AL3	EBC0_BEAR address lock for master 3 0 EBC0_BEAR address unlocked 1 EBC0_BEAR address locked	
24:31		Reserved	

16.7.4 Peripheral Bus Error Status Register 1 (EBC0_BESR1)

The Peripheral Bus Error Status Register 1 (EBC0_BESR1) records the occurrence and type of errors for transactions attempted on behalf of the MAL and DMA controllers. The contents of EBC0_BESR1 are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

It is possible to have both a parity error and a bus error during the same data transfer. If this occurs the bus error is detected first and EBC0_BESR1 and EBC0_BEAR are updated. In the next cycle the parity error is detected and, if error locking is not enabled, logged in EBC0_BESR1.

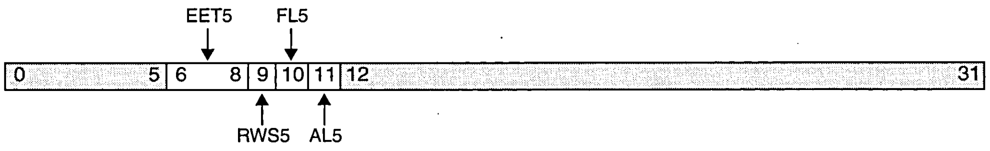


Figure 16-20. Peripheral Bus Error Status Register 1 (EBC0_BESR1)

0:2	EET4	Error type for master 4 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 4 is the MAL.
3	RWS4	Read/write status for master 4 0 Error operation was a write operation 1 Error operation was a read operation	
4	FL4	Field lock for master 4 0 EET4 and RWS4 fields are unlocked 1 EET4 and RWS4 fields are locked	
5	AL4	EBC0_BEAR address lock for master 4 0 EBC0_BEAR address unlocked 1 EBC0_BEAR address locked	
6:8	EET5	Error type for master 5 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 5 is the DMA controller.
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation	

10	FL5	Field lock for master 5 0 EET5 and RWS5 fields are unlocked 1 EET5 and RWS5 fields are locked
11	AL5	EBC0_BEAR address lock for master 5 0 EBC0_BEAR address unlocked 1 EBC0_BEAR address locked
12:31		Reserved

Chapter 17. PCI Interface

17.1 PCI Overview

The peripheral component interconnect (PCI) interface and bridge (referred to as PCI bridge in this chapter) provides a means for connecting PCI-compliant devices to the on-chip bus architecture of the PPC405GP chip. The PCI bridge complies with *PCI Specification*, Version 2.2. The PCI bridge is bidirectional in that it allows PPC405GP PLB masters to access PCI targets off-chip. It also allows PCI masters to access PLB slave devices such as the SDRAM controller. The PCI bridge contains an arbiter which can optionally be used for host applications.

The PCI bridge is configurable by an external PCI agent, allowing it to be used in target adapter applications. The PCI bridge contains address mapping register sets to provide address mapping for both transaction directions. See Figure 17-3 on page 17-5 for a graphic overview of the PCI bridge.

Agents on the PLB are referred to as masters or slaves. Agents on the PCI are referred to as targets or slaves.

17.1.1 PCI Bridge Features

- PLB bus frequency up to 100 MHz
- PCI bus frequency up to 66 MHz (asynchronous) or 33 MHz (synchronous)
- Asynchronous clocking between PLB and PCI buses (optional)
- Supports 1:1, 2:1, 3:1, and 4:1 clock ratios from PLB to PCI
- 32-bit PCI Address/Data Bus
- Power Management
- Buffering:
  - PCI target 64-byte write post buffer
  - PCI target 96-byte read prefetch buffer
  - PLB slave 32-byte write post buffer
  - PLB slave 64-byte read prefetch buffer
- Error tracking/status
- PCI arbitration function (optional)
- Supports PCI target-side configuration
- Supports processor access to all PCI address spaces:
  - Single-beat PCI I/O reads and writes
  - PCI memory single-beat and prefetch-burst reads and single-beat writes
  - Single-beat PCI configuration reads and writes (type 0 and type 1)
  - PCI interrupt acknowledge

17.1.2 PCI Bridge Block Diagram

Figure 17-1 shows the PCI bridge block diagram.

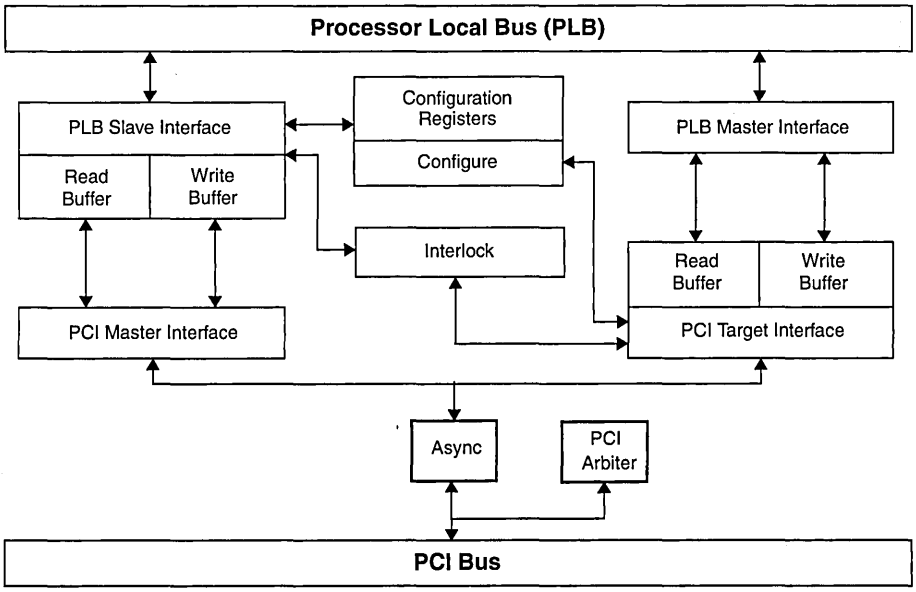


Figure 17-1. PCI Bridge Block Diagram

17.1.3 Byte Ordering

The PCI bridge configuration register address space must be treated as little endian, as required by *PCI Specification*, Version 2.2. In most cases data memory areas in PCI address space will be configured and used in little endian format. To provide for this, PCI configuration space and memory map regions should be defined as little endian memory space by means of the corresponding entry in the PPC405GP CPU's MMU or by means of the appropriate memory region bit in the Storage Little-Endian Register (SLER) if the MMU is not being used. Because the endianness attribute in the SLER can only be applied to 128MB memory regions, this method of defining little endian memory space for PCI must be carefully considered in defining a system memory map.

Byte ordering and management of little endian memory space from a PowerPC CPU point of view is described in detail in "Byte Ordering" on page 3-28. PowerPC architecture and CoreConnect bus architecture both use a bit naming convention in which the most significant bit (msb) name incorporates the numeral 0 and the least significant bit (lsb name for a 32-bit vector incorporates the numeral 31. Table 17-1 shows the correspondence of address bit-naming conventions for PowerPC, CoreConnect PLB, and PCI interface.

Table 17-1. PowerPC, CoreConnect PLB, and PCI Address Bit-Naming Conventions

Functional Unit/Interface	Word Address	Byte Address
PPC405GP Processor Core Address	A0:29	A30:31
CoreConnect — PLB Address Bus	PLB_ABus0:29	PLB_ABus30:31
PCI Address Bus	AD31:2	AD1:0

Table 17-2 shows the correspondence of data bus bit naming conventions and data lane connections for PowerPC, CoreConnect PLB, and PCI interface. Note that within a data lane (column), the data signal naming indicates that, for example, AD31is connected to PLB Write Data24.

Table 17-2. PowerPC, CoreConnect PLB, and PCI Data Bus Bit-Naming Conventions

Functional Unit/ Interface	Most Significant Byte (MSB)	↔	↔	Least Significant Byte (LSB)
Data Byte Value (0x*nn*)	11	22	33	44
Little Endian Byte Address (0b*nn*)	11	10	01	00
PPC405GP Processor Core (Write) Data Bus	Data24:31	Data16:23	Data8:15	Data0:7
CoreConnect — PLB Write Data Bus — Byte Group	PLB Write Data24:31	PLB Write Data16:23	PLB Write Data8:15	PLB Write Data0:7
PLB Byte Enable	PLB_BE3	PLB_BE2	PLB_BE1	PLB_BE0
PCI Byte Enable	C/BE3	C/BE2	C/BE1	C/BE0
PCI Data Bus — Byte Group	AD31:24	AD23:16	AD15:8	AD7:0
Note 1: Logical data work (32-bit word) == 0x11223344 **Note 2:** 405 CPU performing either: • Store word to little endian memory space • Store word—byte reversed—to big endian address space				

17.1.4 Reference Information

Subject	Pointer
PLB Overview	Chapter 2, "On-Chip Bus"
Register Summary	Chapter 25, "Register Summary"
Clocking	Chapter 7, "Clocking"
PCI	*PCI Specification*, Version 2.2

17.2 PCI Bridge Functional Blocks

The following sections describe the PCI bridges and the associated arbiter.

17.2.1 PLB-to-PCI Half-Bridge

As shown in Figure 17-2, the 64-bit PLB slave interface and PCI master interface function together as a PLB-to-PCI half-bridge to enablePLB master devices to access PCI target devices. The half-bridge configuration contains a 32-byte write post buffer and a 64-byte read prefetch buffer.

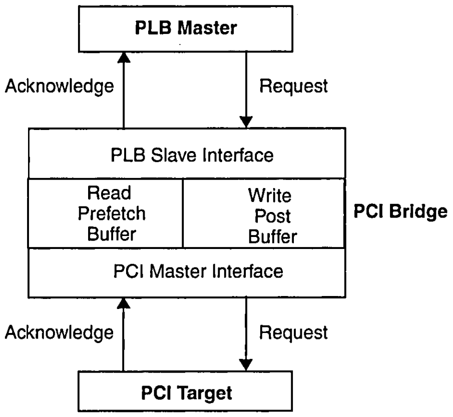


Figure 17-2. PLB-to-PCI Half-Bridge Block Diagram

17.2.2 PCI-to-PLB Half-Bridge

As shown in Figure 17-3, the PCI target interface and 64-bit PLB master interface function together as a PCI-to-PLB half-bridge to enable PCI master devices to access PLB slave devices. The half-bridge configuration contains a 63-byte write post buffer and a 96-byte read prefetch buffer.

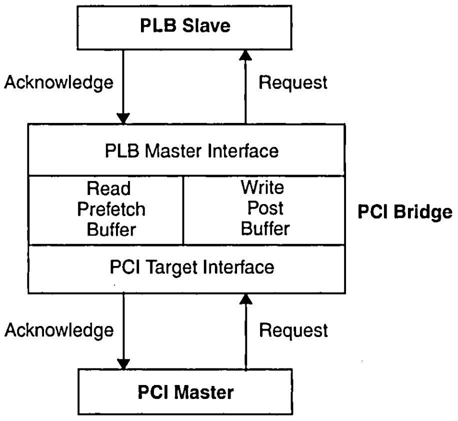


Figure 17-3. PCI-to-PLB Half-Bridge Block Diagram

17.2.3 PCI Arbiter

The internal arbiter can be used with up to six external PCI masters (six $\overline{\text{Req}}/\overline{\text{Gnt}}$ pairs) or can be disabled. When the internal arbiter is disabled, there is one $\overline{\text{Req}}/\overline{\text{Gnt}}$ pair that must be attached to an external arbiter. A strapping pin determines whether the internal arbiter is enabled or not. Priority is round-robin (rotating). Priority switches when a master begins a transfer by asserting $\overline{\text{Frame}}$. Each block keeps a priority bit that only switches if its highest priority requestor receives a grant. Assuming that all priority bits are initially cleared and all requests are active, an example rotation would be PPC405GP 3, 1, 5, 0, 4, 2. *PCI Specification,* Version 2.2 requires that all PCI devices three-state their pins during reset. The PPC405GP PCI arbiter supports bus parking during normal operation.

Figure 17-4 shows the logical arbitration structure.

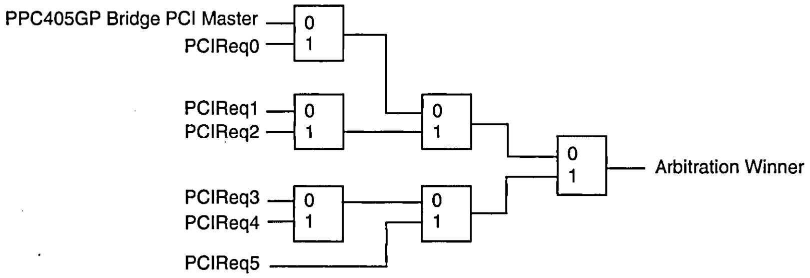


Figure 17-4. Arbitration Structure

17.3 PCI Bridge Address Mapping

The following sections describe the address maps supported by the PCI bridge.

17.3.1 PLB-to-PCI Address Mapping

The PCI bridge responds as a slave on the PLB bus in several address ranges. These ranges enable a PLB master to configure the PCI bridge, and to cause the PCI bridge to generate memory, I/O, configuration, interrupt acknowledge, and special cycles to the PCI bus. Table 17-3 shows the address map from the view of the PLB, that is, as decoded by the PCI bridge as a PLB slave.

Table 17-3. PLB Address Map

PLB Address Range	Description	PCI Address Range
0xE8000000– 0xE800FFFF	PCI I/O Accesses to this range are translated to an I/O access on PCI in the range 0 to 64KB – 1.	0x00000000– 0x0000FFFF
0E8010000– 0E87FFFFF	Reserved PCI bridge does not respond. (Other bridges use this space for non-contiguous I/O.)	
0xE8800000– 0xEBFFFFFF	PCI I/O Accesses to this range are translated to an I/O access on PCI in the range 8MB to 64MB – 1.	0x00800000– 0x03FFFFFF
0xEC000000– 0xEEBFFFFF	Reserved PCI bridge does not respond	
0xEEC00000– 0xEECFFFFF	PCIC0_CFGADDR and PCIC0_CFGDATA 0xEEC00000: PCIC0_CFGADDR 0xEEC00004: PCIC0_CFGDATA 0xEEC00008–0xEECFFFFF: Reserved (can mirror PCIC0_CFGADDR and PCIC0_CFGDATA).	
0xEED00000– 0xEEDFFFFF	PCI Interrupt Acknowledge and Special Cycle 0xEED00000 read: Interrupt Acknowledge 0xEED00000 write: Special Cycle 0xEED00004–0xEEDFFFFF: Reserved (can mirror Interrupt Acknowledge and Special Cycle).	
0xEEE00000– 0xEF3FFFFF	Reserved PCI bridge does not respond.	

Table 17-3. PLB Address Map (continued)

PLB Address Range	Description	PCI Address Range
0xEF400000–0xEF4FFFFF	PCI Bridge Local Configuration Registers 0xEF400000: PCIL0_PMM0LA 0xEF400004: PCIL0_PMM0MA 0xEF400008: PCIL0_PMM0PCILA 0xEF40000C: PCIL0_PMM0PCIHA 0xEF400010: PCIL0_PMM1LA 0xEF400014: PCIL0_PMM1MA 0xEF400018: PCIL0_PMM1PCILA 0xEF40001C: PCIL0_PMM1PCIHA 0xEF400020: PCIL0_PMM2LA 0xEF400024: PCIL0_PMM2MA 0xEF400028: PCIL0_PMM2PCILA 0xEF40002C: PCIL0_PMM2PCIHA 0xEF400030: PCIL0_PTM1MS 0xEF400034: PCIL0_PTM1LA 0xEF400038: PCIL0_PTM2MS 0xEF40003C: PCIL0_PTM2LA 0xF400040–0xEF4FFFFF: Reserved (can mirror PCI local registers)	
0x00000000–0xFFFFFFFF/t*	PCI Memory—Range 0 PMM 0 registers map a region in PLB space to a region in PCI memory space. The address ranges are fully programmable. The PCI address is 64 bits.	0x0000000000000000–0xFFFFFFFFFFFFFFFF
0x00000000–0xFFFFFFFF*	PCI Memory—Range 1 PMM 1 registers map a region in PLB space to a region in PCI memory space. The address ranges are fully programmable. The PCI address is 64 bits.	0x0000000000000000–0xFFFFFFFFFFFFFFFF
0x00000000–0xFFFFFFFF*	PCI Memory—Range 2 PMM 2 registers map a region in PLB space to a region in PCI memory space. The address ranges are fully programmable. The PCI address is 64 bits.	0x0000000000000000–0xFFFFFFFFFFFFFFFF

* Memory map ranges are fully programmable. The ranges must not overlap with each other or conflict with any other memory mappings.

Three PCI bridge address ranges, associated with PLB masters in PLB space, are mapped to PCI memory space: PCI master map (PMM) 0, PMM1, and PMM2.

Each PMM is configured using the following registers (*n* is 0, 1, or 2, corresponding with PMM0, PMM1, and PMM2, respectively):

- PMMnLocal Address (PCIL0_PMMnLA)
- PMMnMask/Attribute (PCIL0_PMMnMA)
- PMMnPCI Low Address (PCIL0_PMMnPCILA)
- PMMnPCI High Address (PCIL0_PMMnPCIHA)

The location of each PMM in PLB space is programmable, using the PCIL0_PMMnLA registers. The PLB address range assigned to each PMM should not overlap any other PLB address space range that is used or reserved. Overlapping results in undefined behavior.

The range of PCI memory address space associated with each PMM is also programmable, and is a 64-bit address space to enable address translation between the PCI bus and the PLB. The PCIL0_PMMnPCILA registers contain the low-order word of a PCI address; the PCIL0_PMMnPCIHA registers contain the high-order word of a PCI address. If the high-order word of a PCI address is greater than 0, the PCI bridge generates dual address cycles to the PCI.

The size of each PMM is programmable, using the mask portion of the PCIL0_PMMnMA registers. The size is a power of 2, ranging from 4KB–4GB. The PLB and PCI address spaces for each PMM are aligned to the size contained in the associated PCIL0_PMMnMA registers.

The attribute portion of the PCIL0_PMMnMA registers specify whether the associated PMM is enabled or disabled, and marked as prefetchable or not prefetchable.

Address ranges and attributes should be initialized before a PMM is enabled.

Figure 17-5 shows the detail of the PMM register sets used to map PLB memory regions to PCI address space.

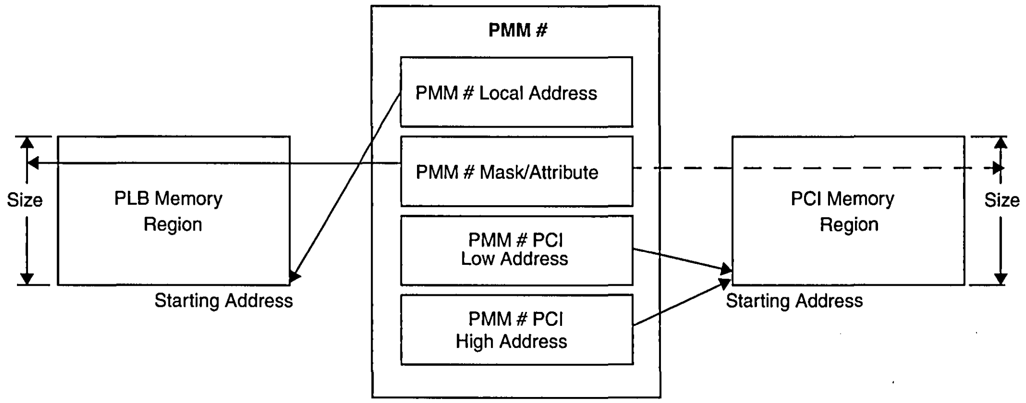


Figure 17-5. PMM Register Sets Map PLB Address Space to PCI Address Space

17.3.2 PCI-to-PLB Address Mapping

The PCI bridge responds as a PCI target for memory accesses and configuration Type 0 accesses. Table 17-4 shows the PCI memory address map from the view of PCI, that is, as decoded by the PCI bridge as a PCI target.

Table 17-4. PCI Memory Address Map

PCI Memory Address	Description	PLB Address
0x00000000– 0xFFFFFFFF	System Memory or ROM—Range 0 PTM 1 maps a region of PCI memory space to PLB space, which can map to system memory or ROM. Size and location is programmable. The space supports address translation between the PCI and the PLB.	0x00000000– 0xFFFFFFFF
0x00000000– 0xFFFFFFFF	System Memory or ROM—Range 1 PTM 2 maps a region of PCI memory space to PLB space, which can map to system memory or ROM. Size and location is programmable. The space supports address translation between the PCI and the PLB.	0x00000000– 0xFFFFFFFF

17.3.3 PCI Target Map Configuration

Two PCI bridge address ranges in PCI memory space are mapped to PLB space: PCI target map (PTM1) and PTM2 (PTM0 is reserved).

Each PTM is configured using the following registers (n is 1 or 2, corresponding with PTM1 and PTM2, respectively).

- PTMnMemory Size (PCILO_PTMnMS)
- PTMnLocal Address (PCILO_PTMnLA)
- PTMnBAR (PCILO_PTMnBAR)

The size of each PTM is programmable, using the PCILO_PTMnMS registers. The size is a power of 2, and ranges from 4KB–4GB. The PLB and PCI address spaces for each PTM are aligned to this size.

The address range of PLB space accessed through each PTM is also programmable, enabling address translation between the PCI bus and the PLB. The PLB address range is defined in the PCILO_PTMnLA registers.

The location of each PTM in PCI memory space is programmable, using the PCIPCILO_PTMnBAR registers.

The PTMs are enabled and disabled using PCICO_CMD[MA]. PTM address ranges and sizes should be initialized before being enabled. If the PCI bridge is not the host bridge, the local processor must initialize the PTM size before enabling host configuration setting the Host Configuration Enable (HCE) field of the Bridge Options 2 register (PCICO_BRDGOPT2). This ensures that the host experiences proper behavior from the PCILO_PTMnBAR registers. Note that PTM1 is always enabled. The PTM1 registers must always be initialized.

Figure 17-6 shows the detail of the PMM/BAR register sets used to map PCI memory regions to PLB address space.

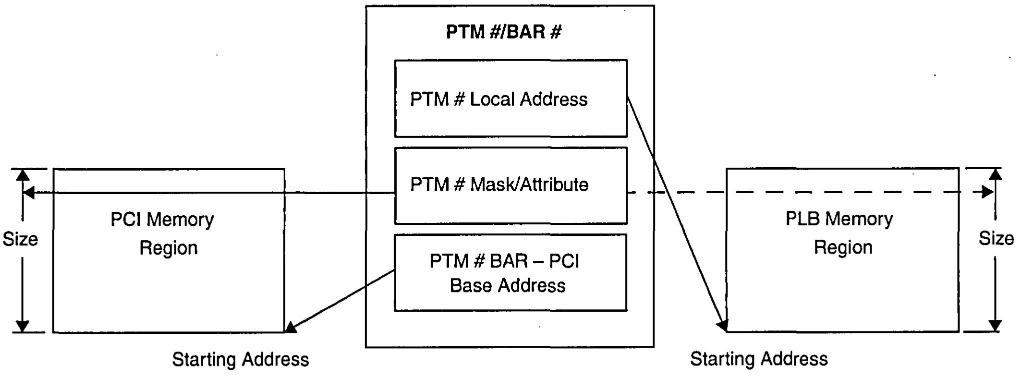


Figure 17-6. PTM Register Sets Map PCI Address Space to PLB Address Space

17.4 PCI Bridge Transaction Handling

The following sections discuss PCI bridge transactions and completion ordering.

17.4.1 PLB-to-PCI Transaction Handling

This section describes how the PCI bridge responds to read and write requests from a PLB master. The PCI bridge decodes and accepts PLB transactions to different address ranges resulting in the generation of memory, I/O, configuration, interrupt acknowledge and special cycles on the PCI bus.

Table 17-5. Transaction Mapping: PLB —> PCI

PLB Transaction PLB Master → Bridge (PLB Slave Interface)	Bridge Mapping and Qualifications	PCI Transaction Bridge (PCI Master Interface) → PCI Target
Single-beat 1 → 8-byte Read	64KB or 56MB PCI I/O address range	I/O Read
Single-beat 1 → 8-byte Write	64KB or 56MB PCI I/O address range	I/O Write
Single-Beat 1 → 8-byte Read	Access to PCIC0_CFGDATA register	Configuration Read (Type 0, 1)
Single-Beat 1 → 8-byte Write	Access to PCIC0_CFGDATA register	Configuration Write (Type 0, 1)
Single-Beat 1 → 8-byte Read	PLB address decodes to PMM0, PMM1, or PMM2, nonprefetchable	Memory Read
Burst Read	PLB address decodes to PMM0, PMM1, or PMM2, nonprefetchable	Memory Read
PLB 4-word and 8-word Line Reads	PLB address decodes to PMM0, PMM1, or PMM2	Memory Read Line
Single-Beat 1 → 4-byte Read	PLB address decodes to PMM0, PMM1, or PMM2, nonprefetchable	Memory Read Multiple
Burst Read	PLB address decodes to PMM0, PMM1, or PMM2, nonprefetchable	Memory Read Multiple
Single-Beat 1 → 4-byte Write	PLB address decodes to PMM0, PMM1, or PMM2	Memory Write
Burst Write	PLB address decodes to PMM0, PMM1, or PMM2	Memory Write
Single-Beat 1 → 4-byte Read	Address 0xFED00000	Interrupt Acknowledge
Single-Beat 1 → 4-byte Write	Address 0xFED00000	Special Cycle
—	Not supported	Memory Write and Invalidate
—	Not supported	Memory Write Line

17.4.1.1 PCI Master Commands

The type of cycle generated to the PCI bus depends on the PLB address, the type of PLB transfer, and the data size. The following sections describe the transaction types supported and outlines the translation of commands from one bus to the other.

The terms "single beat" or "1–8-byte," in reference to PLB transfers, refer to the M_size=0000 transaction type.

PCI bridge initiates the following commands as a PCI master:

- I/O Read and I/O Write

  This command is generated in response to PLB 1–8-byte read or write requests that decode to one of the two PCI I/O spaces.

- Configuration Read and Configuration Write (type 0 and type 1)

  This command is generated in response to PLB 1–8-byte read or write requests that decode to the PCIC0_CFGDATA register.

- Memory Read

  This command is generated in response to PLB 1–8-byte reads or byte and halfword burst reads that decode to one of the three PMMs when the PMM is marked as nonprefetchable.

- Memory Read Line

  This command is generated in response to PLB 4- and 8-word line reads or word and doubleword reads of 32 bytes or less that decode to one of the three PMMs.

- Memory Read Multiple

  This command is generated in response to PLB 1–8-byte reads or byte and halfword burst reads that decode to one of the three PMMs when the PMM is marked as prefetchable. This command is also generated in response to word and doubleword burst reads of greater than 32 bytes that decode to one of the three PMMs. For prefetches, the PCI bridge bursts up to a 64 bytes from the PCI.

- Memory Write

  This command is generated in response to PLB 1–8-byte writes or burst writes to one of the three PMMs.

- Interrupt Acknowledge

  This command is generated in response to a PLB 1–8-byte read from address 0xEED00000.

- Special Cycle

  This command is generated in response to a PLB 1–8-byte write to address 0xEED00000.

The Memory Write and Invalidate command is not generated. All PCI memory writes are performed with Memory Write.

The PLB slave responds as a 64-bit device to word and doubleword bursts. All other commands receive a 32-bit response.

The PCI bridge supports PLB size 1–8-byte encodings. Burst reads of all sizes are also supported. Read line sizes greater than eight words are not supported, and no line writes are supported. The PCI bridge posts all writes which are decoded to PCI memory and PCI I/O space. Posted data is kept in internal write buffers until it can be transferred to the PCI bus. All other writes and all reads are connected, that is, they complete on the PCI bus before completing on the PLB.

17.4.1.2 PLB Slave Read Handling

PLB master read requests are decoded into four types: PCI Memory, I/O, Configuration, and Interrupt Acknowledge. If the request falls within any of these ranges, and is a supported command type, the bridge claims the cycle initially by asserting a PLB wait signal (as opposed to a PLB address acknowledge signal). The bridge must first gain access to the PCI bus before acknowledging a PLB read request. The specific timing of the address acknowledge is dependent upon the type of transfer. All posted writes must be flushed before a read is allowed to complete.

For PLB line reads, the PCI bridge must wait for all read data to be received before acknowledging the PLB request. This is because PCI targets are allowed to disconnect in the middle of a transfer, and the PLB requires line transfers to be atomic. If the system can guarantee that PCI targets do not disconnect these reads, PCIC0_BRDGOPT1[APLRM] can be set to 1. In this mode, line read performance is improved by having the bridge PLB slave assert an address acknowledge signal and begin its data tenure as soon as the first word is received on the PCI bus. If the above guarantee cannot be made, the setting of this bit could hang the bridge.

If the PCI cycle Master Aborts, all beats of read data are returned as 0xFFFFFFFF.

PLB master reads to the PCI bridge configuration registers are allowed to execute regardless of whether any write data is posted in the bridge. The configuration registers are described in "PCI Bridge Configuration Registers" on page 17-19.

17.4.1.3 Prefetching

When the PCI bridge receives a PLB 1–8-byte or byte or halfword burst read request that decodes to a PMM marked as nonprefetchable. The PCI bridge runs a single beat read to the PCI. If the PCI cycle is retried, the PLB cycle is rearbitrated.

When the PCI bridge receives a PLB 1–8-byte or byte or halfword burst read request that decodes to a PMM marked as prefetchable, the PCI bridge burst reads up to a 64 bytes from the PCI and saves the data in the PLB slave read prefetch buffer. Less than 64 bytes can be read if the PCI target disconnects, or if the PCI bridge PCI master disconnects due to a master latency time out. Note that PCI bridge prefetching is not affected by memory management page boundaries (PLB_Guarded is ignored). If a subsequent PLB 1–8-byte or byte or halfword burst read is contained in the prefetch buffer, the data is returned to the PLB directly from the prefetch buffer, and no cycle is generated on the PCI.

If a PLB read to the PCI bridge occurs while the PCI bridge is prefetching and does not hit in the prefetch buffer, then the PLB read is rearbitrated. After prefetching completes, any PLB read (of any type or address range) to the PCI bridge that does not hit in the prefetch buffer causes the prefetch buffer to be emptied, and a new PCI read to begin. PLB writes, including configuration writes, will invalidate the prefetch buffer.

17.4.1.4 PLB Slave Write Handling

PLB master write requests are decoded into four types: PCI Memory (one of three PMM ranges), PCI I/O, PCI Configuration, or Special Cycles. If the request falls within any of these ranges, and is a supported command type, the bridge claims the cycle by asserting a PLB wait signal. If the write is connected, or translates to a PCI Configuration or Special Cycle, the bridge must gain access to the PCI bus and successfully transfer the data before it may assert a PLB address acknowledge signal. If the address is to PCI I/O or memory, the bridge immediately asserts a PLB address acknowledge signal and posts the write if there is sufficient buffer space.

Internal configuration writes are not allowed to execute if posted write data exists in either the PCI slave write buffer or the PLB slave write buffer. The internal configuration mechanism is described in "PCI Bridge Configuration Registers" on page 17-19.

PLB Slave Write Post Buffer

The PCI bridge has a 32-byte write post buffer that may contain four separate single-beat PLB write transactions or one burst. New PLB write requests are rearbitrated if there is not enough room in the write post buffer.

The buffers are not snooped, and are always completed on the PCI bus in the same order as they are received on the PLB bus.

Each write buffer entry preserves the master ID and drives the appropriate PLB bus busy signal until the write is deallocated (it completes on the PCI bus). It is recommended that PLB masters do not use PLB bus busy signal. Instead, PLB masters generating cycles to the PCI should use the standard PCI mechanisms for data coherency.

17.4.1.5 Aborted PLB Requests

The PCI bridge aborts PLB reads only.

A PLB master accessing the PCI bridge can abort PLB write cycles only under the following conditions:

- The PCI bridge rearbitrates the cycle.
- The PCI bridge does not see the cycle because the PLB bus is granted to some other master. A CPU/System Memory interface is expected to do this when a CPU cycle is pending to PCI bridge, but a PLB Master requests system memory access requiring snooping.

  Note: If a PLB master aborts the write cycle at any other time, the results are undefined and the bus may hang.

17.4.1.6 Retried PCI Reads

The PCI specification requires that a PCI master must repeat any read that is retried. The PCI bridge adheres to this requirement. It is only mentioned here because, under certain conditions with respect to aborted PLB reads, the PCI bridge must execute a PCI read and discard the data.

17.4.2 PCI-to-PLB Transaction Handling

This section describes how PCI bridge handles read and write requests from a PCI master device. PCI bridge responds as a PCI target to PCI memory transactions when the PCI address is in one of the two PTM ranges and PCIC0_CMD[MA] = 1. PCI bridge responds by claiming the PCI cycle and mastering a cycle on the PLB.

PCI bridge is also a PCI target for configuration cycles when its PCIIDSel pin is active. PCI bridge will master abort if a configuration cycle is run to itself.

Table 17-6. Transaction Mapping: PCI → PLB

PCI Transaction PCI Master → Bridge (PCI Target Interface)	Bridge Mapping and Qualifications	PLB Transaction Bridge (PLB Master Interface) → PLB Slave
Single-Beat Memory Read	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	8-byte/doubleword read
Burst Memory Read	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	8-byte/doubleword read
Memory Read Line	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	Doubleword burst read
Memory Read Multiple	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	Doubleword burst read
Memory Read	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	Doubleword burst read
Single-Beat Memory Write	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	1 → 8-byte write
Single-Beat Memory Write and Invalidate	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	1 → 8-byte write
Burst Memory Write	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	Word burst write
Burst Memory Write and Invalidate	PCI address decodes to PTM1/BAR1 or PTM2/BAR2, memory access flag	Word burst write
—	Not supported	Memory line reads
—	Not supported	Memory line writes

17.4.2.1 PLB Master Commands

PCI bridge generates PLB transactions based on the type and length of received PCI transactions. The following sections describe the transaction types supported and outline the translation of commands from one bus to the other.

The term "single-beat" refers to the M_size = 0000 transaction type. PCI slave devices are referred to as "targets."

PCI bridge initiates the following PLB master commands:

- 8-Byte Read:

  Generated in response to single beat or burst Memory Read commands from the PCI bus.

- Doubleword Burst Read:

  Generated in response to Memory Read Line and Memory Read Multiple commands on the PCI bus. PCI bridge can also be programmed to perform doubleword bursts on behalf of Memory Read.

- 1–8-Byte Write:

  Generated in response to single-beat (1–4 byte) Memory Write commands on the PCI bus; also generated when the PCI master uses non-contiguous byte enables (see "Byte Enable Handling" on page 17-17).

- Doubleword Burst Write:

  Generated in response to burst Memory Write and Memory Write and Invalidate commands on the PCI bus.

The PLB treats Memory Write and Memory Write and Invalidate identically (nothing on the PLB distinguishes a Memory Write from a Memory Write and Invalidate.)

PCI bridge does not generate line reads or line writes on the PLB.

17.4.2.2 Handling of Reads from PCI Masters

PCI bridge responds to PCI Memory Read, Memory Read Line, and Memory Read Multiple commands. The PCI bridge initiates all PLB reads as single-beat or doubleword burst transfers.

Memory Read generates a PLB single-beat doubleword read. Memory Read Line and Memory Read Multiple commands generate PLB doubleword bursts. For Memory Read Line, PCI bridge encodes a burst length on the byte enable pins of the PLB that corresponds to the number of doublewords from the start address to the end of a word boundary and terminates when the encoded number of words has been transferred. This is called a PLB fixed length burst. If the starting address is the last doubleword on a word boundary (typically, this should not occur), PCI bridge executes a single-beat read. For Memory Read Multiple, PCI bridge sets the byte enables to 0s, indicating a variable length burst.

The PCI target can be programmed to treat Memory Reads as Memory Read Line commands or Memory Read Multiple commands in terms of PLB read behavior.

The PCI bridge guarantees the PCI initial target latency requirement by retrying the PCI cycle if read data is not immediately available in the read buffer. Subsequent latency is programmable using PCIC0_BRDGOPT2[STLD].

The PCI bridge master latency timer can limit the length of read bursts using PCIC0_BRDGOPT1[MLTC]. The timer limits the duration of a burst to the programmed value in units of PLB clock cycles.

Read Buffer

The PCI bridge read buffer stores all read data (including delayed read and prefetched data) when the data is received from the PLB, before it is passed to the PCI. The 96-byte read buffer can store one transaction.

Delayed Reads

A delayed read is queued if a PCI master requests a read while PCI master writes are posted. Posted writes are completed on the PLB before the read is run. PCI bridge continues to post PCI master writes (if buffer space is available) while a delayed read is in progress. Such writes complete on the PLB after the read, even though they complete on the PCI before the read.

A delayed read is also queued if data is not immediately available in the read buffer and a delayed read does not already exist.

When a PCI master returns for a previously requested delayed read, the data is passed out of the read buffer. While the PCI master accepts delayed read data, the PCI bridge can begin to prefetch more read data, if the PCI master posted write buffer is empty. See "Read Prefetching" on page 17-16 for more details.

Any data remaining in the read buffer after delayed read data has been passed to a PCI master is marked as prefetch data and discarded upon a write in either direction.

PCI bridge can hold one delayed read transaction. PCI bridge retries all other PCI master reads until the delayed read completes on the PCI. The read buffer discards data from a delayed read under only one condition. The PCI discard timer is used to track the amount of time it takes for a PCI master to re-request the read. If the PCI master does not re-request the read in 2^{15} PCI clocks (about one millisecond for a 33 MHz PCI clock), PCI bridge discards the delayed read data. This timer begins counting at the beginning of the initial PCI cycle (delayed read request). If PCI bridge is used in a system on which the PLB target (memory) maximum latency (including PLB arbitration) is a significant portion of the timer duration, the timer can expire despite normal bus operation. One solution to this problem is to disable the PCI Discard Timer.

If a delayed read is burst terminated on the PLB (a rare occurrence), PCI bridge will not repeat the request until the PCI master re-requests and only then if the PCI master requests more data than is already buffered.

Read Prefetching

PCI bridge attempts to prefetch data to maximize burst throughput on PCI read requests. Read prefetching occurs only in response to Memory Read Multiple commands or Memory Read, if the PCI target is programmed to treat them as Memory Read Multiple (Memory Read Line causes prefetching to the next word boundary only). This prefetch buffer is 96 bytes.

If a PCI master reads from the read buffer while a PLB read is in progress, data is passed to PCI as it is being filled from the PLB. If the read buffer goes empty long enough for the PCI subsequent latency timer to expire, the PCI is target disconnected. If the read buffer fills up, the PLB cycle is master terminated. The bridge PLB master will not attempt to reacquire the PLB bus if its posted write buffer is not empty.

Prefetched data is discarded if a write is accepted from either the PLB or the PCI. A PCI master read that misses the prefetch buffer also causes current read data to be discarded and the new request to be serviced.

Byte Enable Handling

PCI byte enables are treated as don't cares for PCI reads. The PCI bridge performs doubleword burst or single-beat doubleword reads on the PLB, regardless of the byte enables presented by the requesting PCI master.

Note: This rule assumes that all PLB memory is prefetchable and that all PLB memory accesses are nondestructive.

17.4.2.3 Handling Writes from PCI Masters

PCI bridge responds to Memory Write and Memory Write and Invalidate commands. All PCI master writes are posted. A 64-byte write buffer is used for this purpose. The write buffer accepts up to two separate PCI write transactions. Two single-beat writes, one burst write, or a combination of a single-beat and a burst writes can be posted. If the write buffer is full, new writes are retried until buffer space becomes available.

Note: The maximum of two posted transactions is as viewed from the PCI master. The number of writes performed on the PLB can be more than two, depending on the setting of byte enables of write burst data. See "Byte Enable Handling" on page 17-17.

The PCI bridge begins a PLB write request as soon as a PCI master write has completed on the PCI bus, or a bursting PCI master has written at least six words of data. The PCI bridge continues to receive data from a bursting PCI master as it transfers data to the PLB. If the write post buffer fills, the PCI master is target disconnected. If the write post buffer empties, the PLB cycle is master terminated.

Writes are executed in the same order they are received.

Byte Enable Handling

The PLB does not support non-contiguous byte enables, whereas the PCI bus does. The PLB supports the use of byte enables only for non-line, non-burst transactions, whereas the PCI bus supports any combination of byte enables for any data phase. Therefore, when a PCI master presents a data phase without all byte enables asserted, the bridge disconnects and treats that data phase as one or two single-beat writes on PLB, depending on whether or not byte enables are non-contiguous.

Masters presenting writes without all byte enables asserted experience degraded performance.

17.4.2.4 Miscellaneous

The PCI target forces single-beat transfers when reserved burst or cache line wrap order is used.

The PCI target causes master abort of reserved command encodings, and does not respond to I/O, interrupt acknowledge, or special cycle commands.

The PLB master does not abort requests.

17.4.3 Completion Ordering

PCI bridge implements the following completion ordering rules:

1. PCI master writes are accepted if there is room in the PCI write post buffer.
2. New PCI master reads are accepted if there is no delayed read (DRR or DRC) in progress:
   a. If PCI write post buffer is empty and read data is not buffered, then begin a delayed read (enter DRR state).
   b. If PCI write post buffer is not empty, then begin delayed read (enter DRR state).

   Delayed reads are handled as follows:

   a. While in DRR state, retry all PCI master reads. Wait for all PCI master writes if any that were posted before entering DRR state to complete on PLB.
   b. Execute PLB read, enter DRC state.
   c. While in DRC state, retry all PCI master reads if the address does not match. If it does match, pass the read data to the PCI master. If data is passed, exit the DRC state.
3. PLB master writes are accepted if there is room in the PLB write post buffer.
4. PLB master reads are accepted if the PLB write post buffer and the PCI write post buffer are both empty.

17.4.3.1 PCI Producer-Consumer Model

The PCI Producer-Consumer model is followed with one exception: PCI master reads do not flush PLB writes and PCI master writes do not cause PLB prefetched read data to be discarded. If the “flag” is stored in system memory (PLB side), but the “data” is stored in a PCI target, the control software must manually force coherency. This can be done by following two rules:

1. To ensure data written by a PLB master has reached the intended PCI target, the PLB master should execute a read from PCI, to any nondestructive address. This is only necessary if the write is postable.
2. To ensure data read by a PLB master is current (rather than old prefetched data), the PLB master should execute a read from PCI to any other nondestructive address. This is only necessary if the read is prefetchable.

17.4.4 Collision Resolution

The PCI bridge must resolve collisions when a PLB master and a PCI master attempt accesses through the PCI bridge at the same time. Table 17-7 summarizes collision resolution.

In general, PLB postable writes are always accepted (if buffer space is available), and passed to the PCI when given the chance. PCI master writes are always accepted (if buffer space is available), but cause PLB reads and non-postable writes to be rearbitrated to clear the path to the PLB. PLB reads and non-postable writes proceed as long as there is no PCI master activity, which causes the PLB cycle to be rearbitrated. PCI master reads are always allowed to proceed, and cause PLB reads and non-postable writes to be rearbitrated.

Internal configuration accesses have their own rules. Configuration writes are not allowed to complete while any write data is posted in the PCI bridge, or while the PCI master is prefetching. Otherwise, there are no restrictions. Configuration reads have no restrictions; however, only one internal configuration access (PLB or PCI side) may be serviced at a time.

Table 17-7. Collision Resolution

Access Type	PLB Read from PCI	PLB Postable Write to PCI	PLB Non-postable Write to PCI
PCI Write to PLB	Rearbitrate PLB master (reads flush writes)	No conflict	Rearbitrate PLB master
PCI Read from PLB	Rearbitrate PLB master	No conflict	Rearbitrate PLB master

17.5 PCI Bridge Configuration Registers

The PCI bridge has two sets of configuration registers for configuring the bridge, handling errors, and reporting status. Local configuration registers control PLB functions, and are accessed only from the PLB. The PCI configuration registers control PCI functions, and can be accessed from the PLB and the PCI. In addition, the mechanism used to access the bridge configuration registers may also be used to configure other devices on the PCI bus.

17.5.1 PCI Bridge Register Summary

Table 17-8 provides a summary of all of the PCI bridge registers. The registers are discussed in detail in the following sections. See Chapter 8, “Reset and Initialization,” for register reset values.

Table 17-8. Directly Accessed MMIO Registers

Register	Address	Access	Description	Page
PCIL0_PMM0LA	0xEF400000	R/W	PMM 0 Local Address	17-21
PCIL0_PMM0MA	0xEF400004	R/W	PMM 0 Mask/Attribute	17-25
PCIL0_PMM0PCILA	0xEF400008	R/W	PMM 0 PCI Low Address	17-22
PCIL0_PMM0PCIHA	0xEF40000C	R/W	PMM 0 PCI High Address	17-23
PCIL0_PMM1LA	0xEF400010	R/W	PMM 1 Local Address	17-23
PCIL0_PMM1MA	0xEF400014	R/W	PMM 1 Mask/Attribute	17-21
PCIL0_PMM1PCILA	0xEF400018	R/W	PMM 1 PCI Low Address	17-24
PCIL0_PMM1PCIHA	0xEF40001C	R/W	PMM 1 PCI High Address	17-25
PCIL0_PMM2LA	0xEF400020	R/W	PMM 2 Local Address	17-25
PCIL0_PMM2MA	0xEF400024	R/W	PMM 2 Mask/Attribute	17-25
PCIL0_PMM2PCILA	0xEF400028	R/W	PMM 2 PCI Low Address	17-26
PCIL0_PMM2PCIHA	0xEF40002C	R/W	PMM 2 PCI High Address	17-27
PCIL0_PTM1MS	0xEF400030	R/W	PTM 1 Memory Size/Attribute	17-27
PCIL0_PTM1LA	0xEF400034	R/W	PTM 1 Local Address	17-27
PCIL0_PTM2MS	0xEF400038	R/W	PTM 2 Memory Size/Attribute	17-28
PCIL0_PTM2LA	0xEF40003C	R/W	PTM 2 Local Address	17-28

Table 17-9. PCI Configuration Address and Data Registers

Register	Address	Access	Description
PCIC0_CFGADDR	0xEEC00000	R/W	PCI Configuration Address Register
PCIC0_CFGDATA	0xEEC00004	R/W	PCI Configuration Data Register

Table 17-10. PCI Configuration Register Offsets

Register	Offset	Access		Description
		PLB	PCI	
PCIC0_VENDID	0x01–0x00	R/W	R	PCI Vendor ID
PCIC0_DEVID	0x03–0x02	R/W	R	PCI Device ID
PCIC0_CMD	0x05–0x04	R/W	R/W	PCI Command Register
PCIC0_STATUS	0x07–0x06	R/W	R/W	PCI Status Register
PCIC0_REVID	0x08	R/W	R/W	PCI Revision ID
PCIC0_PCICLS	0x0B–0x09	R/W	R	PCI Class Register
PCIC0_CACHELS	0x0C	R	R	PCI Cache Line Size
PCIC0_LATTIM	0x0D	R/W	R/W	PCI Latency Timer
PCIC0_HDTYPE	0x0E	R	R	PCI Header Type
PCIC0_BIST	0x0F	R	R	PCI Built In Self Test Control
Reserved PCI BAR	0x13–0x10	R	R	Reserved, PCI BAR
PCIC0_PTM1BAR	0x17–0x14	R/W	R/W	PCI PTM 1 BAR
PCIC0_PTM2BAR	0x1B–0x18	R/W	R/W	PCI PTM 2 BAR
Reserved PCI BAR	0x1F–0x1C	R	R	Reserved PCI BAR. Refer to *PCI Specification*, Version 2.2 for more information on values.
Reserved PCI BAR	0x23–0x20	R	R	Reserved PCI BAR. Refer to *PCI Specification*, Version 2.2 for more information on values.
Reserved PCI BAR	0x27–0x24	R	R	Reserved PCI BAR. Refer to *PCI Specification*, Version 2.2 for more information on values.
Reserved Cardbus CIS Pointer	0x2B–0x28	R	R	Reserved Cardbus CIS Pointer. Refer to *PCI Specification*, Version 2.2 for more information on values.
PCIC0_SBSYSVID	0x2D–0x2C	R/W	R	PCI Subsystem Vendor ID
PCIC0_SBSYSID	0x2F–0x2E	R/W	R	PCI Subsystem ID
Reserved Exp ROM Base Addr	0x33–0x30	R	R	Reserved Expansion ROM Base Address. Refer to *PCI Specification*, Version 2.2 for more information on values.
PCIC0_CAP	0x34	R	R	PCI Capabilities Pointer
Reserved	0x3B–0x35	R	R	Reserved
PCIC0_INTLN	0x3C	R/W	R/W	PCI Interrupt Line
PCIC0_INTPN	0x3D	R	R	PCI Interrupt Pin
PCIC0_MINGNT	0x3E	R	R	PCI Minimum Grant
PCIC0_MAXLTNCY	0x3F	R	R	PCI Maximum Latency
PCIC0_PCIICS	0x44	R/W	R/W	PCI Interrupt Control/Status

Table 17-10. PCI Configuration Register Offsets (continued)

Register	Offset	Access		Description
		PLB	PCI	
PCIC0_ERREN	0x48	R/W	R/W	Error Enable
PCIC0_ERRSTS	0x49	R/W	R/W	Error Status
PCIC0_BRDGOPT1	0x4B–0x4A	R/W	R/W	PCI Bridge Options 1
PCIC0_PLBBESR0	0x4F–0x4C	R/W	R/W	PLB Slave Error Syndrome 0
PCIC0_PLBBESR1	0x53–0x50	R/W	R/W	PLB Slave Error Syndrome 1
PCIC0_PLBBEAR	0x57–0x54	R/W	R/W	PLB Slave Error Address Register
PCIC0_CAPID	0x58	R	R	Capability Identifier
PCIC0_NEXTIPTR	0x59	R	R	Next Item Pointer
PCIC0_PMC	0x5B–0x5A	R	R	Power Management Capabilities
PCIC0_PMCSR	0x5D–0x5C	R/W	R/W	Power Management Control Status
PCIC0_PMCSRBSE	0x5E	R	R	PMCSR PCI-to-PCI Bridge Support Extensions
PCIC0_DATA	0x5F	R	R	Data
PCIC0_BRDGOPT2	0x63–0x60	R/W	R/W	PCI Bridge Options 2
PCIC0_PMSCRR	0x64	R/W	R/W	Power Management State Change Request Register

17.5.2 PCI Bridge Local Configuration Registers

The PCI bridge local configuration registers have fixed addresses in PLB space and must be accessed using single beat PLB read or write cycles of the same size as shown in the register descriptions.

Failure to access all bytes of a particular register could produce unexpected results. Reading of reserved bit locations produces unpredictable values. Software must use appropriate masks to extract the desired bits. Writes must preserve the values of reserved bit positions by first reading the register, merging the new value, and writing the result.

17.5.2.1 PMM 0 Local Address Register (PCIL0_PMM0LA)

PCIL0_PMM0LA defines the PLB starting address of range 0 in PLB space that is mapped to PCI memory. Only bits that are 1 in the PCIL0_PMM0MA are used to determine the starting address; all other bits are don't cares. Only bits 31:12 are writable. Bits 11:0 are always 0.

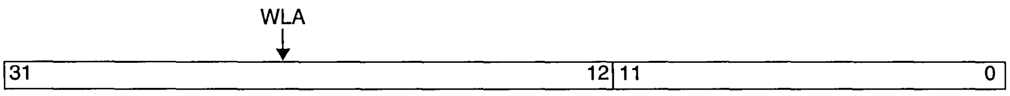


Figure 17-7. PMM 0 Local Address Register (PCIL0_PMM0LA)

31:12	WLA	Writable PLB Local Address	
11:0		PLB Local Address	Always 0

17.5.2.2 PMM 0 Mask/Attribute Register (PCILO_PMM0MA)

PCILO_PMM0MA controls the size and attributes of the PLB space mapped to PCI memory for range 0.

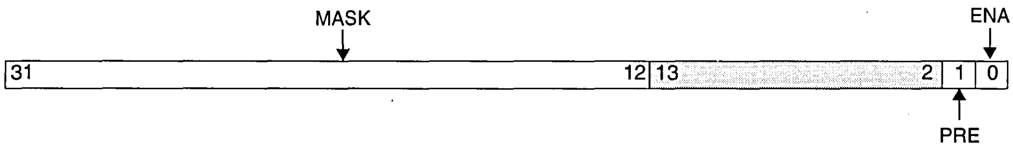


Figure 17-8. PMM 0 Mask/Attribute Register (PCILO_PMM0MA)

31:12	MASK	The mask bits determine the size of the address map range.	The mask must be of the form 111....0000. Bits set to 1 cause the corresponding PCILO_PMM0LA bits to be compared with incoming PLB addresses. Note that the minimum range size is 4KB, and valid ranges are powers of 2. For example, a 128MB range would be encoded as 0xF8000 and a 4KB range would be encoded as all ones.
11:2		Reserved	Returns 0 when read.
1	PRE	Read Prefetching Enable 1 Read prefetching is enabled.	If read prefetch is enabled, the PCI bridge prefetches 64 bytes from PCI memory in response to a PLB single-beat, byte-burst, or half word burst read from PMM 0.
0	ENA	PLB to PCI Memory Mapping Enable 1 Memory mapping is enabled.	Note that PCILO_PMM0LA, PCILO_PMM0PCIHA, and PCILO_PMM0PCILA must be initialized before enabling.

17.5.2.3 PMM 0 PCI Low Address Register (PCILO_PMM0PCILA)

PCILO_PMM0PCILA defines the low-order 32 bits of the PCI address generated in response to a PLB access to range 0. Only bits that are 1 in PCILO_PMM0MA are passed to the PCI address. The other (least significant) bits of the PCI address are passed through from the PLB address. Only bits 31:12 are writable. Bits 11:0 are always 0.

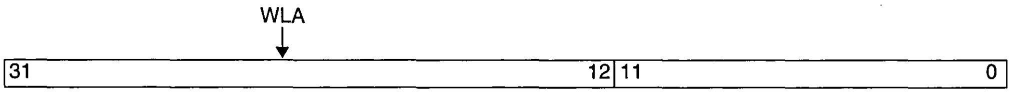


Figure 17-9. PMM 0 PCI Low Address Register (PCIL0_PMM0PCILA)

31:12	WLA	Writable PCI Low Address	
11:0		PCI Low Address	Always 0

17.5.2.4 PMM 0 PCI High Address Register (PCIL0_PMM0PCIHA)

PCIL0_PMM0PCIHA defines the high-order 32 bits of the PCI address generated in response to a PLB access to range 0. If PCIL0_PMM0PCIHA is greater than 0, the PCI bridge generates a PCI dual address cycle using the value in PCIL0_PMM0PCIHA as the high-order 32 bits of the PCI address.

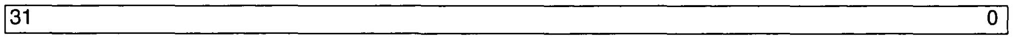


Figure 17-10. PMM 0 High Address Register (PCIL0_PMM0PCIHA)

31:0		PCI High Address

17.5.2.5 PMM 1 Local Address Register (PCIL0_PMM1LA)

PCIL0_PMM1LA defines the PLB starting address of range 1 in PLB space that is mapped to PCI memory. See “PMM 0 Local Address Register (PCIL0_PMM0LA)” on page 17-21.

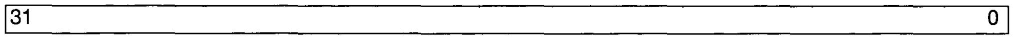


Figure 17-11. PMM 1 Local Address Register (PCIL0_PMM1LA)

31:0		PLB Local Address

17.5.2.6 PMM 1 Mask/Attribute Register (PCIL0_PMM1MA)

PCIL0_PMM1MA defines the size and attributes of range 1 in PLB space that is mapped to PCI memory. See "PMM 0 Mask/Attribute Register (PCIL0_PMM0MA)" on page 17-22.

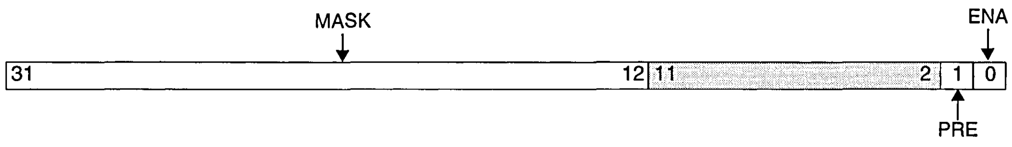


Figure 17-12. PMM 1 Mask/Attribute Register (PCIL0_PMM1MA)

31:12	MASK	The mask bits determine the size of the address map range.	The mask must be of the form 111....0000. Bits set to 1 cause the corresponding PCIL0_PMM1LA bits to be compared with incoming PLB addresses. Note that the minimum range size is 4KB, and valid ranges are powers of 2. For example, a 128MB range would be encoded as 0xF8000 and a 4KB range would be encoded as 0x11111.
11:2	▨	Reserved	Returns 0 when read.
1	PRE	Read Prefetching Enable 1 Read prefetching is enabled.	If read prefetch is enabled, the PCI bridge prefetches 64 bytes from PCI memory in response to a PLB single-beat, byte-burst, or half word burst read from PMM 0.
0	ENA	PLB to PCI Memory Mapping Enable 1 Memory mapping is enabled.	Note that PCIL0_PMM1LA, PCIL0_PMM1PCIHA, and PCIL0_PMM1PCILA must be initialized before enabling.

17.5.2.7 PMM 1 PCI Low Address Register (PCIL0_PMM1PCILA)

PCIL0_PMM1PCILA defines the low-order 32 bits of the PCI address generated in response to a PLB access to range 1. See "PMM 0 PCI Low Address Register (PCIL0_PMM0PCILA)" on page 17-22.

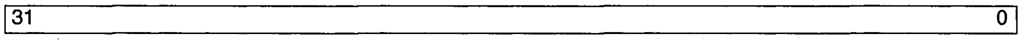


Figure 17-13. PMM 1 PCI Low Address Register (PCIL0_PMM1PCILA)

31:0		PCI Low Address

17.5.2.8 PMM 1 PCI High Address Register (PCILO_PMM1PCIHA)

PCILO_PMM1PCIHA defines the high-order 32 bits of the PCI address generated in response to a PLB access to range 1. See "PMM 0 PCI High Address Register (PCILO_PMM0PCIHA)" on page 17-23.

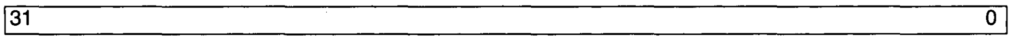


Figure 17-14. PMM 0 High Address Register (PCILO_PMM0PCIHA)

31:0		PCI High Address

17.5.2.9 PMM 2 Local Address Register (PCILO_PMM2LA)

PCILO_PMM2LA defines the PLB starting address of range 2 in PLB space that is mapped to PCI memory. See "PMM 0 Local Address Register (PCILO_PMM0LA)" on page 17-21.

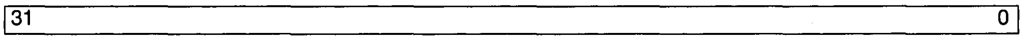


Figure 17-15. PMM 2 Local Address Register (PCILO_PMM2LA)

31:0		PLB Local Address

17.5.2.10 PMM 2 Mask/Attribute Register (PCILO_PMM2MA)

PCILO_PMM2MA defines the size and attributes of range 2 in PLB space that is mapped to PCI memory. See "PMM 0 Mask/Attribute Register (PCILO_PMM0MA)" on page 17-22.

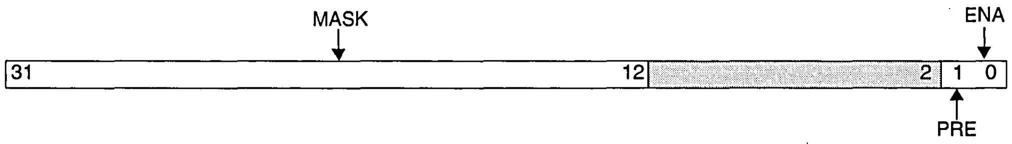


Figure 17-16. PMM 2 Mask/Attribute Register (PCILO_PMM2MA)

31:12	MASK	The mask bits determine the size of the address map range.	The mask must be of the form 111....0000. Bits set to 1 cause the corresponding PCILO_PMM2LA bits to be compared with incoming PLB addresses. Note that the minimum range size is 4KB, and valid ranges are powers of 2. For example, a 128MB range would be encoded as 0xF8000 and a 4KB range would be encoded as 0x11111.
11:2	▨	Reserved	Returns 0 when read.
1	PRE	Read Prefetching Enable 1 Read prefetching is enabled.	If read prefetch is enabled, the PCI bridge prefetches 64 bytes from PCI memory in response to a PLB single-beat, byte-burst, or half word burst read from PMM 0.
0	ENA	PLB to PCI Memory Mapping Enable 1 Memory mapping is enabled.	Note that PCILO_PMM2LA, PCILO_PMM2PCIHA, and PCILO_PMM2PCILA must be initialized before enabling.

17.5.2.11 PMM 2 PCI Low Address Register (PCILO_PMM2PCILA)

PCILO_PMM2PCILA defines the low-order 32 bits of the PCI address generated in response to a PLB access to range 2. See “PMM 0 PCI Low Address Register (PCILO_PMMOPCILA)” on page 17-22.

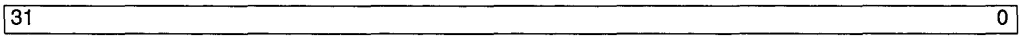


Figure 17-17. PMM 2 Low Address Register (PCILO_PMM2PCILA)

31:0		PCI Low Address

17.5.2.12 PMM 2 PCI High Address Register (PCIL0_PMM2PCIHA)

PCIL0_PMM2PCIHA defines the high-order 32 bits of the PCI address generated in response to PLB access to range 2. See "PMM 0 PCI High Address Register (PCIL0_PMM0PCIHA)" on page 17-23.

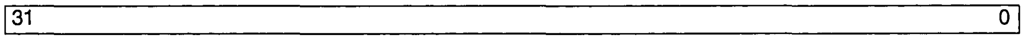


Figure 17-18. PMM 2 PCI High Address Register (PCIL0_PMM2PCIHA)

31:0		PCI High Address

17.5.2.13 PTM 1 Memory Size/Attribute Register (PCIL0_PTM1MS)

PCIL0_PTM1MS defines the size and attributes of the of PCI memory region mapped to local (PLB) space through PTM 1. PCIL0_PTM1MS affects the operation of PCI PTM 1 BAR.

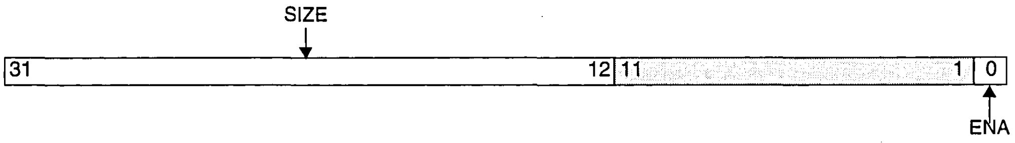


Figure 17-19. PTM 1 Memory Size/Attribute Register (PCIL0_PTM1MS)

31:12	MASK	Defines the size of the region of PCI memory space that is mapped to local (PLB) space using PTM 1.	The minimum range size is 4KB. Valid ranges are always a power of 2. For example, a value of 0xFF000000 indicates that the region contains 16MB.
11:1	▨	Reserved	Returns 0 when read.
0	EMM	Determines if range 1 is enabled to map PCI memory space to PLB space. Always 1 (enabled).	

17.5.2.14 PTM 1 Local Address Register (PCIL0_PTM1LA)

PCIL0_PTM1LA defines the local (PLB) address that is generated in response to a PCI access to local (PLB) space through PTM 1. Only bits that are 1 in PCIL0_PTM1MS are passed to the PLB address. The other (least significant) bits of the PLB address are passed through from the PCI address. Only bits 31:12 are writable. Bits 11:0 are always 0.

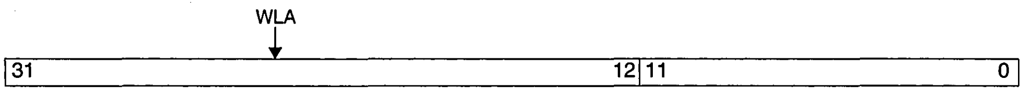


Figure 17-20. PTM 2 Local Address Register (PCIL0_PTM1LA)

31:12	WLA	Writable PTM 1 Local Address	Writable
11:0		PTM 1 Local Address	Always 0

17.5.2.15 PTM 2 Memory Size/Attribute Register (PCIL0_PTM2MS)

PCIL0_PTM2MS defines the size of the region of PCI memory space mapped to local (PLB) space through PTM 2.

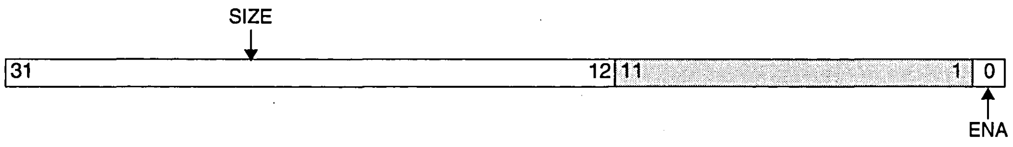


Figure 17-21. PTM 2 Memory Size/Attribute Register (PCIL0_PTM2MS)

31:12	MASK	Defines the size of the region of PCI memory space mapped to local (PLB) space using PTM 2.	The minimum range size is 4KB. Valid ranges are always a power of 2. For example, a value of 0xFF000000 indicates that the region contains 16MB.
11:1	▨	Defines the size of the region of PCI memory space mapped to local (PLB) space using PTM 2.	The minimum range size is 4KB. Valid ranges are always a power of 2. For example, a value of 0xFF000000 indicates that the region contains 16MB.
0	EMM	Determines if range 2 is enabled to map PCI memory space to PLB space.	When EMM is disabled, PCIC0_PTM2BAR cannot be written. Set PCIC0_PTM2BAR to 0 before disabling EMM.

17.5.2.16 PTM 2 Local Address Register (PCIL0_PTM2LA)

This register defines the local (PLB) address generated in response to a PCI access to local (PLB) space through PTM 2. See “PTM 1 Local Address Register (PCIL0_PTM1LA)” on page 17-27 for more information.

31	0

Figure 17-22. PTM 2 Local Address Register (PCIL0_PTM2LA)

31:0		PTM 2 Local Address

17.5.3 PCI Configuration Registers

The PCI configuration registers can be accessed from both the PLB and PCI buses.

PLB side configuration is supported using the mechanism defined in the *PCI Local Bus Specification* Version 2.2. This mechanism uses PCIC0_CFGADDR and PCIC0_CFGDATA to access the configuration registers indirectly. These registers reside at addresses 0xEEC00000 and 0xEEC00004, respectively.

To access (from the PLB side) the configuration space of other devices on the PCI bus, write a value to PCIC0_CFGADDR that specifies the following:

- Bus number
- Device number on that bus
- Register number to be accessed

The value must also set PCIC0_CFGADDR[EN] = 1. An access to PCIC0_CFGDATA then results in a configuration cycle on the PCI bus.

To access the bridge configuration registers from the PLB side, use the same mechanism as described above, but set PCIC0_CFGADDR[BN, DN] = 0. The bridge is assumed to reside on PCI bus 0 and to have a device number of 0.

The bridge configuration registers can be accessed from the PCI side by Type 0 configuration reads or writes, with the PCIIDSel pin asserted to the bridge. There are some restrictions on PCI side accesses that are noted in the register descriptions that follow.

PCIC0_CFGADDR and CONFIG_DATA should be accessed with single-beat PLB commands. All registers are byte addressable. Reading of reserved bit locations produces unpredictable values. Software must use appropriate masks to extract the desired bits. Writes must preserve the values of reserved bit positions by first reading the register, merging the new value, and writing the result.

17.5.3.1 PCI Configuration Address Register (PCIC0_CFGADDR)

PCIC0_CFGADDR controls the type of cycle generated when PCIC0_CFGDATA is accessed. Its fields are shown in Figure 17-23.

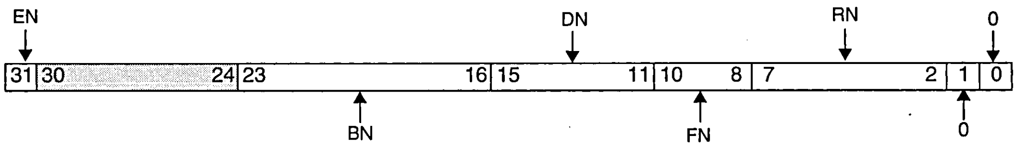


Figure 17-23. PCI Configuration Address Register (PCIC0_CFGADDR)

31	EN	Enable 0 Disabled 1 Enabled
30:24		Reserved
23:16	BN	Bus Number
15:11	DN	Device Number
10:8	FN	Function Number
7:2	RN	Register Number
1	0	
0	0	

See the *PCI Local Bus Specification* Version 2.2 for details about how the fields are used.

17.5.3.2 PCI Configuration Data Register (PCIC0_CFGDATA)

Accessing PCIC0_CFGDATA causes one of three things to happen, depending on the value of PCIC0_CFGADDR.

1. Generation of a Type 0 configuration cycle on the PCI bus (PCIC0_CFGADDR[BN] = 0, PCIC0_CFGADDR[DN] > 0).
2. Generation of a Type 1 configuration cycle on the PCI bus (PCIC0_CFGADDR[BN] > 0).
3. Access of a PCI bridge PCI configuration register (PCIC0_CFGADDR[BN, DN] = 0).

Figure 17-24 illustrates the PCIC0_CFGDATA register.

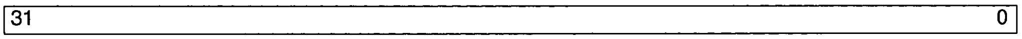


Figure 17-24. PCI Configuration Data Register (PCIC0_CFGDATA)

31:0		Configuration Data

17.5.3.3 PCI Vendor ID Register (PCIC0_VENDID)

PCIC0_VENDID identifies the manufacturer of a PCI device. This register contains 0x1014 (index 0x00 = 0x14, index 0x01 = 0x10) at reset. This vendor ID is assigned for all IBM PCI devices. The local CPU (PLB master) can write a different value to this register.

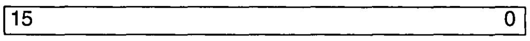


Figure 17-25. PCI Vendor ID Register (PCIC0_VENDID)

15:0		Vendor ID

17.5.3.4 PCI Device ID Register (PCIC0_DEVID)

PCIC0_DEVID identifies the PCI device. This value is 0x0156 (index 0x03 = 0x01, index 0x02 = 0x56) at reset. The local CPU (PLB master) can write a different value to this register.

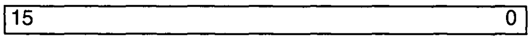


Figure 17-26. PCI Device ID Register (PCIC0_DEVID)

15:0		PCI Device ID

17.5.3.5 PCI Command Register (PCIC0_CMD)

PCIC0_CMD controls the operation of the PCI bridge on the PCI bus. Figure 17-27 describes the bits.

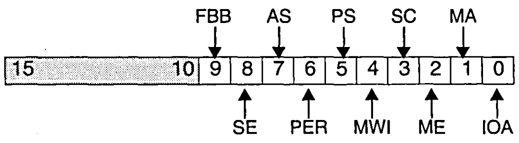


Figure 17-27. PCI Command Register (PCIC0_CMD)

15:10		Reserved	.
9	FBB	Fast Back-to-Back Write Enable	Enables PCI masters to perform fast back-to-back transactions. Because he PCI bridge does not perform fast back-to-back transactions; FBB is read-only and returns 0 when read.
8	SE	$\overline{\text{PCISErr}}$ Enable 0 Disabled 1 Enabled	Enables driving $\overline{\text{PCISErr}}$ when a PCI bus parity error is detected when the PCI bridge is the PCI target. PCIC0_CMD[PER] must be enabled for address parity errors. PCIC0_CMD[PER] and PCIC0_ERREN[WDPE] must be enabled for write data parity errors.
7	AS	Address stepping wait states.	The PCI bridge does not address step (except for address stepping when generating a Config Type 0 cycle); AS is read-only and returns 0 when read.
6	PER	Parity error response 0 Disabled 1 Enabled	This bit is enabled for all types of PCI bus parity errors, including the following: • PCI data bus parity errors while PCI is master. • PCI data bus parity errors while PCI is target. • PCI address bus parity errors. When parity error response is disabled, detection of these errors is masked and $\overline{\text{PCIPErr}}$ (PERR#) is not asserted, although parity is still generated.
5	PS	Palette Snooping	Enable special palette snooping. The PCI bridge is not a VGA device; PS is read-only and returns 0 when read
4	MWI	Memory Write and Invalidate Enable	The PCI bridge does not generate this command; MWI is read-only and returns 0 when read.
3	SC	Special Cycle Operations Enable	The PCI bridge never monitors special cycles; SC is read-only and returns 0 when read.

2	ME	Master Enable 0 Disabled 1 Enabled	Enables PCI bridge-to-master cycles on the PCI bus. When ME is 0, the PCI bridge only responds as a PLB slave to PCIC0_CFGADDR, PCIC0_CFGDATA, and PCI bridge local configuration register access. Except for configuration cycles, the PCI bridge cannot master cycles to the PCI bus. If the pin strapping setting reflected in CPC0_PSR[RL] = 1, ME resets to 1.
1	MA	Memory Access 0 Disabled 1 Enabled	Controls PCI bridge response as a PCI memory target. MA is disabled at reset.
0	IOA	I/O Access	Controls the PCI bridge response as a PCI I/O target. The PCI bridge does not respond to I/O space accesses; IOA is read-only and returns 0 when read.

17.5.3.6 PCI Status Register (PCIC0_STATUS)

PCIC0_STATUS is a read/bit-reset register used to record status information for PCI bus events. Bits in PCIC0_STATUS are set only as a result of specific events occurring on the PCI bus. They are reset by writing a 1 to the desired bit. Writing a 0 to a bit location leaves that bit unchanged.

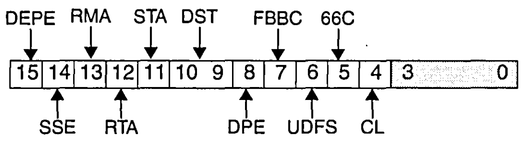


Figure 17-28. PCI Status Register (PCIC0_STATUS)

15	DEPE	Detected Parity Error Write 1 to clear.	The PCI bridge sets DEPE when the PCI bridge detects a PCI bus parity error, regardless of the setting of any enable bits (DEPE is non-maskable). The following events set DEPE: • PCI address bus parity error detected when PCI bridge is a target. • PCI data bus parity error detected when a PCI master writes to PLB memory (PCI bridge is the target). • PCI data bus parity error detected when PCI bridge masters a PCI read cycle.
14	SSE	Signaled System Error Write 1 to clear.	The PCI bridge sets SSE if the PCI bridge asserts $\overline{\text{PCISErr}}$ (see “Error Handling” on page 17-55 for causes of $\overline{\text{PCISErr}}$ assertion).

13	RMA	Received Master Abort Write 1 to clear.	The PCI bridge sets RTA when a PCI cycle for which the PCI bridge is the master is terminated with master abort.
12	RTA	Received Target Abort Write 1 to clear.	The PCI bridge sets RTA when a PCI cycle for which it is the master is terminated with target abort.
11	STA	Signaled Target Abort Write 1 to clear.	The PCI bridge sets STA when a PCI cycle for which it is the target is terminated with target abort.
10:9	DST	$\overline{\text{PCIDevSel}}$ Response Timing Read-only.	The PCI bridge asserts $\overline{\text{PCIDevSel}}$ on the second clock after $\overline{\text{PCIFframe}}$ is asserted (called medium response time). Read-only; always returns 0b01 when read.
8	DPE	Data Parity Error Detected Write 1 to clear.	DPE is set when the following conditions are met: • The PCI bridge detects a data parity error ($\overline{\text{PCIPErr}}$ is asserted) when the PCI bridge is the master on a PCI read cycle, or is the master when it samples $\overline{\text{PCIPErr}}$ asserted on a PCI write cycle. • PCIC0_CMD[PER] = 1.
7	FBBC	Fast Back-to-Back Capable Read-only; returns 0 when read.	Indicates that the PCI target can accept fast back-to-back transactions when the transactions are not to the same agent. The PCI bridge target does not accept this type of fast back-to-back transaction.
6	UDFS	UDF Supported Read-only; returns 0 when read.	Indicates device support of user-definable features. The PCI bridge does not support user-definable features.
5	66C	66 MHz Capable 0 At reset 1 PCI bridge is configured for 66MHz operation.	Indicates that the device can run at 66 MHz. The PCI bridge can be configured to run at 33 MHz max or 66 MHz. The local CPU (PLB master) sets 66C to 1 if PCI bridge is configured for 66 MHz operation.
4	CL	Capabilities List This bit is read only and returns 1 when read.	Indicates that the value at offset 0x34 is a pointer in configuration space to a linked list of new capabilities.
3:0	▨	Reserved	These bits return 0s when read.

17.5.3.7 PCI Revision ID Register (PCIC0_REVID)

PCIC0_REVID holds the current incremental revision number. The reset value is the version number of the PCI bridge macro (first version is 00). However, the local CPU (PLB master) can write a value to this register.

Figure 17-29. PCI Revision ID Register (PCIC0_REVID)

7:0		Revision ID	Revision level of device.

17.5.3.8 PCI Class Register (PCIC0_CLS)

This register holds the class code. This register is 0x060000 at reset, which indicates that the PCI bridge is a bridge device located between the PLB and the PCI bus; however, the local CPU (PLB master) can write a value to this register for the case where PCI bridge is not the host bridge.

Class information is defined in the *PCI Local Bus Specification*, Version 2.2.

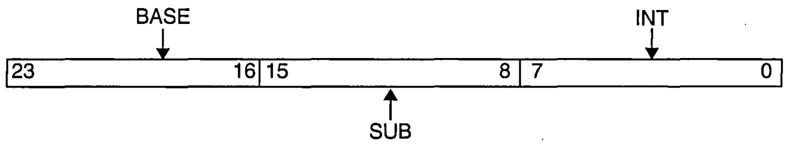


Figure 17-30. PCI Class Register (PCIC0_PCICLS)

23:16	BASE	Base Class	Reset to 0x06, which indicates bridge device. Users of the RISCWatch debugger must use the PCIC0_BASECC register to access this field.
15:8	SUB	Subclass	Reset to 00, which indicates host bridge. Users of the RISCWatch debugger must use the PCIC0_SUBCLS register to access this field.
7:0	INT	Interface Class	Reset to 00. Users of the RISCWatch debugger must use the PCIC0_INTCLS register to access this field.

17.5.3.9 PCI Cache Line Size Register (PCIC0_CACHELS)

PCIC0_CACHELS determines the size of a PCI cache line. PCI bridge does not support a PCI cache. Therefore, this register is read-only and returns 0x00 when read.

Figure 17-31. PCI Cache Line Size Register (PCIC0_CACHELS)

7:0		PCI Cache Line Size

17.5.3.10 PCI Latency Timer Register (PCIC0_LATTIM)

PCIC0_LATTIM holds the value of the PCI latency timer. The granularity of the latency timer is 8 PCI cycles. Therefore, the 3 low-order bits of this register are read-only and return 0x111.

Figure 17-32. PCI Latency Timer Register (PCIC0_LATTIM)

7:0		PCI Latency Timer

PCI Local Bus Specification Version 2.2 specifies that PCI masters capable of multi-beat bursts must, after losing their PCI grant, get off the bus after a number of clock cycles equivalent to the value in PCIC0_LATTIM.

The actual number of clock cycles to disconnect varies somewhat when in asynchronous mode. If PCIC0_LATTIM is programmed in asynchronous mode to a value that is less than 64, the PCI bridge PCI master interface could timeout, regardless of the state of its grant line.

In asynchronous mode, the PCI master starts its timer and can timeout regardless of the state of the PCI grant. This is strictly a performance issue and does not limit functionality or affect data integrity.

Several factors affect the frequency of timeouts.

- The amount of PCI bus traffic. In moderate to heavily loaded systems, this is less of an issue because a PCI master tends to lose its grant more often after gaining the bus.
- Fast targets that introduce few or no wait states reduce the chances of timeouts occurring.

17.5.3.11 PCI Header Type Register (PCIC0_HDTYPE)

PCIC0_HDTYPE (bits 0:6) identifies the second part of the PCI header, which begins at offset 0x10. It also determines whether a device contains multiple functions (bit 7). The PCI bridge implements the standard header and is not a multi-function device; therefore, PCIC0_HDTYPE is read-only and returns 0x00 when read.

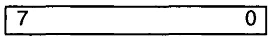


Figure 17-33. PCI Header Type Register (PCIC0_HDTYPE)

7:0		PCI Header Type

17.5.3.12 PCI Built-In Self Test (BIST) Control Register (PCIC0_BIST)

PCIC0_BIST is used for control and status of BIST. PCI bridge does not implement BIST. PCIBIST is read-only and returns 0x00 when read.

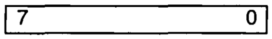


Figure 17-34. PCI Built-in Self Test Control Register (PCIC0_BIST)

7:0		PCI BIST Control

17.5.3.13 Unused PCI Base Address Register Space

PCI base address register space is defined to begin at offset 0x10; however, the first 32 bits of this space are unused by PCI bridge, and the defined base address registers begin at offset 0x14.

17.5.3.14 PCI PTM 1 BAR (PCIC0_PTM1BAR)

PCIC0_PTM1BAR defines a space in PCI memory space mapped to PLB space (system memory or ROM).

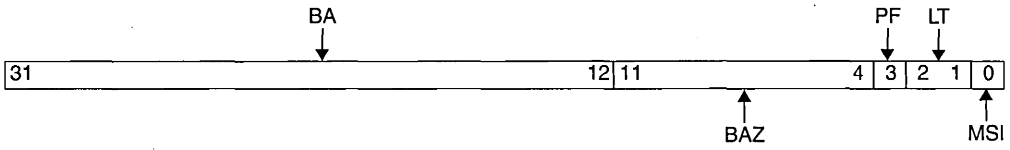


Figure 17-35. PCI PTM 1 BAR Register (PCIC0_PTM1BAR)

31:12	BA	Base Address These bits determine where in PCI memory address space this region is located.	Only corresponding bits in PCIL0_PTM1MS that are set to 1 are writable. Bits in PCIL0_PTM1MS that are set to 0 cause the corresponding Base Address register bits to be always 0. PCIL0_PTM1MS must be initialized by a PLB master before any PCI device is allowed to configure this register.
11:4	BAZ	Base Address Always Zero	BAZ = 0x00 because the minimum size of this range is 4KB.
3	PF	Prefetchable	PR = 1 to indicate that prefetching is allowed.
2:1	LT	Location Type	LT = 0b00 to indicate that the memory space can be located anywhere in the 32-bit address space.
0	MSI	Memory Space Indicator	MSI = 0 to indicate memory space, rather than I/O space.

17.5.3.15 PCI PTM 2 BAR (PCIC0_PTM2BAR)

PCIC0_PTM2BAR defines a second space in PCI memory space that is mapped to PLB space (system memory or ROM). Note that if PCIC0_PTM2BAR is disabled using PCIL0_PTM2MSA, PCIC0_PTM2BAR cannot be written. Set PCIC0_PTM2BAR to 0 before disabling this bit. If disabled in this way, reads to PCIC0_PTM2BAR always return 0.

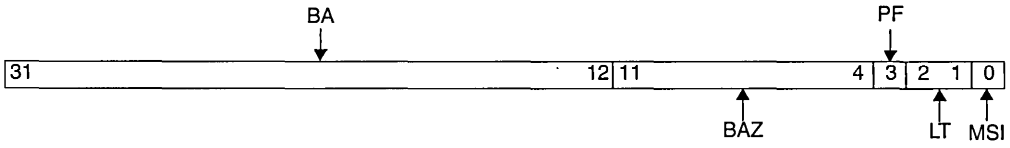


Figure 17-36. PCI PTM 2 BAR Register (PCIC0_PTM2BAR)

31:12	BA	Base Address These bits determine where in PCI Memory address space this region is located.	Only corresponding bits in PCIL0_PTM2MS that are set to 1 are writable. Bits in PCIL0_PTM2MS that are set to 0 cause the corresponding Base Address register bits to be always 0. PCIL0_PTM2MS must be initialized by a PLB master before any PCI device can configure this register.
11:4	BAZ	Base Address Always Zero	BAZ = 0x00 because the minimum size of this range is 4KB.
3	PF	Prefetchable	PF = 1 to indicate that prefetching is allowed.
2:1	LT	Location Type	LT = 0b00 to indicate that the memory space can be located anywhere in the 32-bit address space.
0	MSI	Memory Space Indicator	MSI = 0 to indicate memory space, rather than I/O space.

17.5.3.16 PCI Subsystem Vendor ID Register (PCIC0_SBSYSVID)

PCIC0_SBSYSVID holds the vendor ID for a subsystem or add-in board.

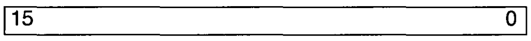


Figure 17-37. PCI Subsystem Vendor ID Register (PCIC0_SBSYSVID)

15:0		PCI Subsystem Vendor ID

17.5.3.17 PCI Subsystem ID Register (PCIC0_SBSYSID)

PCIC0_SBSYSID holds the device ID of a subsystem or add-in board.

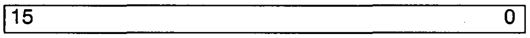


Figure 17-38. PCI Subsystem ID Register (PCIC0_SBSYSID)

15:0		PCI Subsystem ID

17.5.3.18 PCI Capabilities Pointer (PCIC0_CAP)

PCIC0_CAP contains an 8-bit pointer in configuration space to the next capability. This data structure is indicated by PCIC)_STATUS[CL]. PCIC0_CAP points to the first item in the list of capabilities at address offset 0x58, which is the PCI power management capability structure.

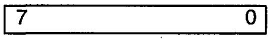


Figure 17-39. PCI Capabilities Pointer (PCIC0_CAP)

7:0		PCI Capabilities Pointer

17.5.3.19 PCI Interrupt Line Register (PCIC0_INTLN)

PCIC0_INTLN contains interrupt line routing information.

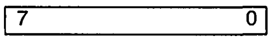


Figure 17-40. PCI Interrupt Line Register (PCIC0_INTLN)

7:0		PCI Interrupt Line

17.5.3.20 PCI Interrupt Pin Register (PCIC0_INTPN)

PCIC0_INTPN specifies the PCI interrupt line that the device uses. The value 0x01 indicates INTA#.

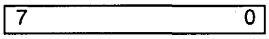


Figure 17-41. PCI Interrupt Pin Register (PCIC0_INTPN)

7:0		PCI Interrupt Pin

17.5.3.21 PCI Minimum Grant Register (PCIC0_MINGNT)

PCIC0_MINGNT specifies the burst period length of a PCI device. PCIC0_MINGNT is read-only and returns 0x00 when read.

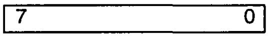


Figure 17-42. PCI Minimum Grant Register (PCIC0_MINGNT)

7:0		PCI Minimum Grant

17.5.3.22 PCI Maximum Latency Register (PCIC0_MAXLTNCY)

PCIC0_MAXLTNCY specifies how often a PCI device needs to access to the PCI bus. PCIC0_MAXLTNCY is read-only and returns 0x00 when read.

Figure 17-43. PCI Maximum Latency Register (PCIC0_MAXLTNCY)

7:0		PCI Maximum Latency

17.5.3.23 PCI Interrupt Control/Status Register (PCIC0_ICS)

A PLB master or a PCI master device may generate an interrupt to the PCI bus by writing a 1 to bit 0. Clearing this bit clears the interrupt. Bit 0 also reports the status of the interrupt. A value of 1 means that the interrupt is asserted, a value of 0 means that the interrupt is deasserted.

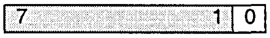


Figure 17-44. PCI Interrupt Control/Status Register

7:1	▨	Reserved	These bits return 0 when read.
0	API	Asset PCI interrupt	When software sets this bit, the PCI bridge asserts its Interrupt pin.

17.5.3.24 Error Enable Register (PCIC0_ERREN)

ERREN enables detection and reporting of various errors for the PCI bridge (see “Error Handling” on page 17-55).

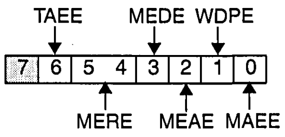


Figure 17-45. Error Enable Register (PCIC0_ERREN)

7	▨	Reserved	
6	TAEE	Target Abort Error Enable 0 Disabled 1 Enabled	While the PCI bridge is the PCI master, this bit enables the detection of a target abort as an error condition. If TAEE is enabled, the PCI bridge reports PLB bus errors.

5:4	MERE	PLB Bus Error Response Enable 00 No action is taken. 01 The PCI target should drive $\overline{\text{PCISErr}}$ on the PCI bus. 10 Target should target abort the offending read. 11 Indicates the PCI target should drive $\overline{\text{PCISErr}}$ and target abort.	MERE controls the response taken by the PCI bridge on the PCI bus (as the PCI target) when PLB bus errors are asserted to the PCI bridge PLB master. **Note:** Only reads can be target aborted. **Note:** Modes 10 and 11 cannot be used in asynchronous mode.
3	MEDE	PLB Master Error Detection Enable 0 Disables detection of PLB master errors. 1 Enables detection of PLB master errors.	MEDE enables the detection of PLB bus errors when the PCI bridge is a PLB master.
2	MEAE	PLB Bus Error Assertion Enable 0 Disabled 1 Enabled	MEAE enables the reporting of a PLB bus error when the PCI bridge is a PLB slave.
1	WDPE	Write Data Parity $\overline{\text{PCISErr}}$ Enable 0 Disabled 1 Enabled.	The PCI bridge drives $\overline{\text{PCISErr}}$ when a data parity error is detected on a write cycle when the PCI bridge is the PCI target. PCIC0_CMD[SE] must also be 1.
0	MAEE	Master Abort Error Enable 0 Disabled 1 Enabled	MAEE enables the detection of a master abort as an error condition when the PCI bridge is the master. The PCI bridge drives SI_MErr on the PLB bus in response to a master abort. If this bit is disabled, driving of SI_Merr in response to master abort is masked.

17.5.3.25 Error Status Register (PCIC0_ERRSTS)

PCIC0_ERRSTS contains status for detected error conditions (see "Error Handling" on page 17-55). Bits in PCIC0_ERRSTS can be set to 1 only as the result of a system error. Writing a 1 to a PCIC0_ERRSTS bit clears the bit. Writing a 0 to a bit leaves that bit unchanged.

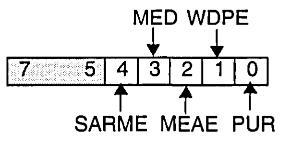


Figure 17-46. Error Status Register (PCIC0_ERRSTS)

7:5	▨	Reserved	
4	SARME	$\overline{\text{PCISErr}}$ Asserted on Received PLB Bus Error	Set when PCI bridge asserts $\overline{\text{PCISErr}}$ on the PCI bus in response to PCI bridge receiving a PLB bus error while PLB master.

3	MED	PLB Bus Error Detected 1 Error detected	Set when a PLB bus error signal is asserted when PCI bridge is the PLB master. MED is set regardless of whether the PCI bridge is enabled to treat this as an error condition (the setting of MED is not maskable).
2	MEAE	PLB Bus Error Assertion Event 1 An PCI bridge error, which can cause a PLB bus error, occurred.	Set when an error occurs that would cause PCI bridge (as PLB slave) to assert a PLB bus error signal. MEAE is set regardless of whether the the PLB bus error assertion is enabled (the setting of MEAE is not maskable).
1	WDPE	$\overline{\text{PCISerr}}$ on Write Data Parity Error	Set when the PCI bridge drives $\overline{\text{PCISErr}}$ in response to a data parity error detected on a PCI write to PLB memory. $\overline{\text{PCIPErr}}$ is also driven.
0	PUR	PLB Unsupported Request	Set when the PCI bridge is a PLB slave and detects an unsupported request from a PLB master to an address range that PCI bridge decodes. The PCI bridge allows such requests to time out.

17.5.3.26 Bridge Options 1 Register (PCIC0_BRDGOPT1)

PCIC0_BRDGOPT1 controls various operating parameters of the PCI bridge. The parameters must be initialized before PCI masters access the PCI bridge.

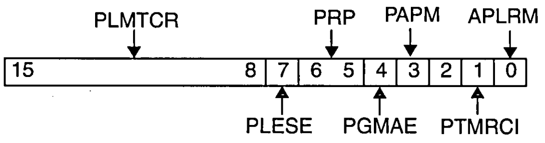


Figure 17-47. Bridge Options 1 Register (PCIC0_BRDGOPT1)

15:8	PMLTCR	PLB Master Latency Timer Count Register	PMLTCR contains the value used by the PLB master to load its latency timer. The granularity of this timer is 16 PLB cycles; therefore, the low-order bits of this register are read-only and are hardwired to 1.
7	PLESE	PLB Lock Error Status Enable 0 Slave error locking is disabled. 1 Slave error locking is enabled.	PLESE controls the handling of slave error locking.

6:5	PRP	PLB Request Priority 11 Highest 10 Next highest 01 Next highest 00 Lowest	PRP controls the request priority for PLB accesses.
4	PGMAE	PLB Guarded Memory Access Enable 0 Bridge PLB master memory accesses are unguarded. 1 Bridge PLB master memory accesses are guarded.	PGMAE controls whether PLB accesses are guarded or unguarded.
3	PAPM	PCI Arbiter Park Mode 0 The arbiter parks on requester 0 (the bridge PCI master). 1 The arbiter parks on the last master granted the bus.	PAPB defines how the internal PCI arbiter handles bus parking.
2:1	PTMRCI	PCI Target Memory Read Command Interpretation 00 Memory Read 01 Memory Read Line 10 Memory Read Multiples 11 Reserved	PTMRCI enables the PCI bridge to be forced to treat a PCI memory read as a memory read multiple, or as a memory read line, with respect to the burst size implied by the read commands. This is for masters that use memory read for multiple beat bursts.
0	APLRM	Atomic PLB Line Read Mode 0 1 PLB slave asserts Addrack and begins its data tenure immediately after the PCI master receives the first read data word.	APLRM controls the behavior of the bridge PLB slave with respect to PLB line reads. APLRM must not be se t to 1 unless all PCI target devices can guarantee no disconnects for PLB line reads.

17.5.3.27 PLB Slave Error Syndrome Register 0 (PCIC0_PLBBESR0)

PCIC0_PLBBESR0 stores information about errors reported by the PCI bridge PLB slave. There are four groups of errors, one each for PLB masters 0–3. PCIC0_PLBBESR0[MxET] fields (x represents a particular PLB master ID) contain information about the type of error. PCI parity errors set PCIC0_PLBBESR0[MxET] to 0b001. Master and target aborts set PCIC0_PLBBESR0[MxET] to 0b101 (non-configured bank error). The PCIC0_PLBBESR0[MxRWS] fields show whether the transaction causing the error was a read or write.

Each error field can be locked by PCIC0_PLBBESR0[MxFL], which is set by a PLB lock error signal to the bridge PLB slave. If the PCIC0_PLBBESR0[MxFL] field associated with a master is 0, the PLB lock error signal is driven high to the bridge PLB slave, and an error associated with that master occurs. The error is then reported, and PCIC0_PLBBESR0[MxFL] is set. Subsequent errors do not set PCIC0_PLBBESR0 fields for that master until PCIC0_PLBBESR0[MxFL] is cleared. If PCIC0_PLBBESR0[MxFL] = 0, and the PLB lock error signal is low, the error is reported, and PCIC0_PLBBESR0[MxFL] is not set. Additional errors are also reported. Only software can clear PCIC0_PLBBESR0[MxFL].

Writing 1 to a PCIC0_PLBBESR0 field clears the field.

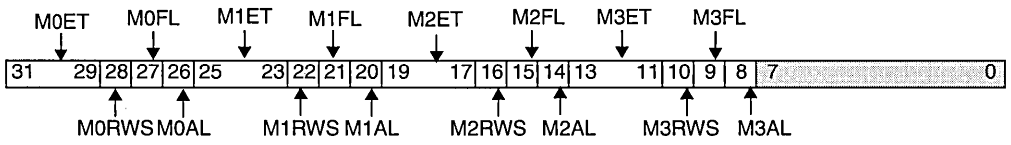


Figure 17-48. PLB Slave Error Syndrome Register 0 (PCIC0_PLBBESR0)

31:29	M0ET	Master 0 Error Type 000 No Error 001 Parity Error 010 Reserved 011 Reserved 100 Reserved 101 Non-configured Bank Error 110 Reserved 111 Reserved
28	M0RWS	Master 0 Read/Write Status 0 Error operation was a write 1 Error operation was a read
27	M0FL	Master 0 PCIC0_PLBBESR0 Field Lock 0 PCIC0_PLBB ESR0 unlocked 1 PCIC0_PLBB ESR0 locked
26	M0AL	Master 0 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 0 1 PCIC0_PLBBEAR locked by Master 0
25:23	M1ET	Master 1 Error Type See PCIC0_PLBBESR0[M0ET]
22	M1RWS	Master 1 Read/Write Status 0 Error operation was a write 1 Error operation was a read
21	M1FL	Master 1 PCIC0_PLBBESR0 Field Lock 0 PCIC0_PLBB ESR0 unlocked 1 PCIC0_PLBB ESR0 locked
20	M1AL	Master 1 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 1 1 PCIC0_PLBBEAR locked by Master 1
19:17	M2ET	Master 2 Error Type See PCIC0_PLBBESR0[M0ET]
16	M2RWS	Master 2 Read/Write Status 0 Error operation was a write 1 Error operation was a read
15	M2FL	Master 2 PCIC0_PLBBESR0 Field Lock 0 PCIC0_PLBB ESR0 unlocked 1 PCIC0_PLBB ESR0 locked
14	M2AL	Master 2 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 2 1 PCIC0_PLBBEAR locked by Master 2

13:11	M3ET	Master 3 Error Type See PCIC0_PLBBESR0[M0ET]
10	M3RWS	Master 3 Read/Write Status 0 Error operation was a write 1 Error operation was a read
9	M3FL	Master 3 PCIC0_PLBBESR0 Field Lock 0 PCIC0_PLBBESR0 unlocked 1 PCIC0_PLBBESR0 locked
8	M3AL	Master 3 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR Unlocked by Master 2 1 PCIC0_PLBBEAR Locked by Master 2
7:0		Reserved

17.5.3.28 PLB Slave Error Syndrome Register 1 (PCIC0_PLBBESR1)

PCIC0_PLBBESR1 stores information about errors reported by the bridge PLB slave. There are four groups of errors, one each for PLB masters 4–7. See "PLB Slave Error Syndrome Register 0 (PCIC0_PLBBESR0)" on page 17-45 for additional information about the fields of this register.

Only software can clear PCIC0_PLBBESR1[MxFL]. The PCIC0_PLBBESR1[MxAL] fields control the and PCIC0_PLBBESR0 and PCIC0_PLBBESR1 in the same way. Writing a 1 to a field of the PCIC0_PLBBESRx clears the bit.

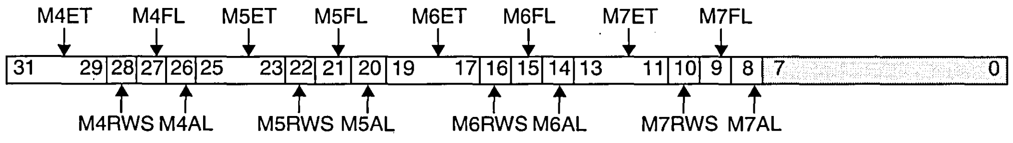


Figure 17-49. PLB Slave Error Syndrome 1 (PCIC0_PLBBESR1)

31:29	M4ET	Master 4 Error Type 000 No Error 001 Parity Error 010 Reserved 011 Reserved 100 Reserved 101 Non-configured Bank Error 110 Reserved 111 Reserved
28	M4RWS	Master 4 Read/Write Status 0 Write error operation 1 Read error operation
27	M4FL	Master 4 PCIC0_PLBBESR1 Field Lock 0 PCIC0_PLBBESR1 unlocked 1 PCIC0_PLBBESR1 locked

26	M4AL	Master 4 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 4 1 PCIC0_PLBBEAR locked by Master 4	
25:23	M5ET	Master 5 Error Type	See PCIC0_PLBBESR1[M4ET]
22	M5RWS	Master 5 Read/Write Status 0 Write error operation 1 Read error operation	
21	M5FL	Master 5 PCIC0_PLBBESR1 Field Lock 0 PCIC0_PLBBESR1 Unlocked 1 PCIC0_PLBBESR1 Locked	
20	M5AL	Master 5 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 5 1 PCIC0_PLBBEAR locked by Master 5	
19:17	M6ET	Master 6 Error Type	See PCIC0_PLBBESR1[M4ET]
16	M6RWS	Master 6 Read/Write Status 0 Write error operation 1 Read error operation	
15	M6FL	Master 6 PCIC0_PLBBESR1 Field Lock 0 PCIC0_PLBBESR1 unlocked 1 PCIC0_PLBBESR1 locked	
14	M6AL	Master 6 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 6 1 PCIC0_PLBBEAR locked by Master 6	
13:11	M7ET	Master 7 Error Type	See PCIC0_PLBBESR1[M4ET]
10	M7RWS	Master 7 Read/Write Status 0 Write error operation 1 Read error operation	
9	M7FL	Master 7 PCIC0_PLBB ESR1 Field Lock 0 PCIC0_PLBBESR1 unlocked 1 PCIC0_PLBBESR1 locked	
8	M7AL	Master 7 PCIC0_PLBBEAR Address Lock 0 PCIC0_PLBBEAR unlocked by Master 7 1 PCIC0_PLBBEAR locked by Master 7	
7:0	▨	Reserved	

17.5.3.29 PLB Slave Error Address Register (PCIC0_PLBBEAR)

PCIC0_PLBBEAR contains addresses associated with errors, as indicated by the PLB slave asserting Sl_MErr for transactions initiated by the PCI bridge on the PCI bus. PCIC0_PLBBEAR is read-only.

Figure 17-50. PLB Slave Error Address Register (PCIC0_PLBBEAR)

31:0		PLB Slave Error Address

17.5.3.30 Capability Identifier (PCIC0_CAPID)

When PCIC0_CAPID contains 0x01, the PCI bridge supports power management and the data structure currently being pointed to is the PCI power management capability structure.

Figure 17-51. Capability Identifier (PCIC0_CAPID)

7:0		PCI Capability Identifier

17.5.3.31 Next Item Pointer (PCIC0_NEXTIPTR)

PCIC0_NEXTIPTR describes the location of the next item in the capability list of the function. PCIC0_NEXTIPTR is set to 0x00 to indicate that this is the last item on the capability list.

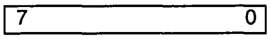


Figure 17-52. Next Item Pointer (PCIC0_NEXTIPTR)

7:0		PCI Next Item Pointer

17.5.3.32 Power Management Capabilities (PCIC0_PMC)

PCIC0_PMC provides information about the capabilities of the function related to power management. A value of 0x0202 indicates no specific capabilities.

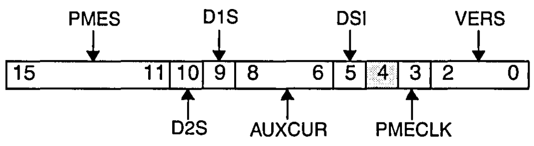


Figure 17-53. Power Management Capabilities Register (PCIC0_PMC)

15:11	PMES	PME Support	The PCI bridge does not support PME#; therefore, PMES is hardwired to 0b00000.
10	D2S	D2 Support Determines if the D2 power management state is supported.	The PCI bridge does not support the D2 power management state; therefore, D2S is hardwired to 0.
9	D1S	D1 Support Determines if the D1 power management state is supported.	The PCI bridge supports the D1 power management state; therefore, D1S is hardwired to 1.
8:6	AUXCUR	Auxiliary Current Support	The PCI bridge does not support Aux_Current; therefore, AUXCUR is hardwired to 0b000.
5	DSI	Device Specific Initialization 0 after reset	This bit indicates whether special initialization of this function is required (beyond the standard PCI configuration header) before the generic class device driver is able to use it.
4	▨	Reserved	Always read as 0.
3	PMECLK		This bit is hardwired to 0 indicating that the function does not support PME# generation in any state.
2:0	VERS		Returns 0b010 on reads, indicating that PMC complies with Revision 1.1 of *PCI Power Management Interface Specification.*

17.5.3.33 Power Management Control/Status Register (PCIC0_PMCSR)

PCIC0_PMCSR is used to manage the PCI power management state and to enable and monitor PMEs.

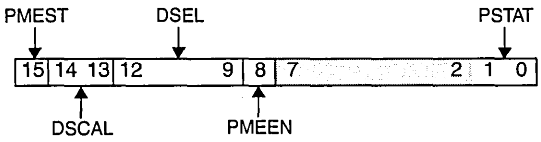


Figure 17-54. Power Management Control/Status Register (PCIC0_PMCSR)

15	PMEST		The PCI bridge does not support PME#; therefore, PMEST is hardwired to 0.
14:13	DSCAL		The PCI bridge does not support data register; therefore, DSCAL is hardwired to 0b00.
12:9	DSEL		The PCI bridge does not support a data register; therefore, DSEL is hardwired to 0b0000.
8	PMEEN		The PCI bridge does not support PME generation; therefore, PMEEN is hardwired to 0.
7:2		Reserved	Returns 0 when read.
1:0	PSTAT	Determine the current power state of a function and sets the function into a new power state. 00 D0 01 D1 10 D2 11 D3 Hot	If software attempts to write a value for an unsupported power state to PSTAT, its value does not change. Writing this fiield may change PCI0_PMSCRR.

17.5.3.34 PMCSR PCI-to-PCI Bridge Support Extensions (PCIC0_PMCSRBSE)

PCIC0_PMCSRBSE is required for all PCI-to-PCI bridges. The PCI bridge in not a PCI-to-PCI bridge; therefore, it returns 0 when this register is read.

Figure 17-55. PMCSR PCI to PCI Bridge Support Extensions (PCIC0_PMCSRBSE)

7:0		PCI to PCI Bridge Support Extensions

17.5.3.35 PCI Data Register (PCIC0_DATA)

PCIC0_DATA is an optional register that provides a mechanism for the function to report state dependent operating data such as power consumed or heat dissipation. The PCI bridge does not implement this register; therefore, it returns 0 when this register is read.

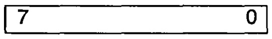


Figure 17-56. PCI Data (PCIC0_DATA)

7:0		PCI Data

17.5.3.36 Bridge Options 2 Register (PCIC0_BRDGOPT2)

PCIC0_BRDGOPT2 controls various operating parameters of the PCI bridge.

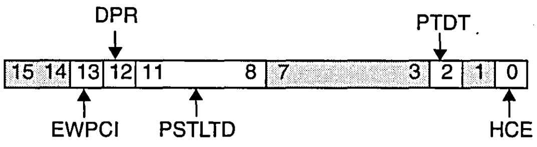


Figure 17-57. Bridge Options 2 Register (PCIC0_BRDGOPT2)

15:14	▨	Reserved	
13	EWPCI	External Write to PCI Command Interrupt 0 No write to PCIC0_CMD has occurred. 1 External PCI master has written to PCIC0_CMD.	Software can also set or clear this bit.
12	DPR	Drive PCI Reset 0 Normal operation 1 Causes $\overline{\text{PCIReset}}$ pin to be asserted.	Software that asserts this bit must leave is asserted long enough to guarantee the PCI pulse width requirements. DPR does not reset PLB bus interface registers or PCI bridge registers. $\overline{\text{PCIReset}}$ is also asserted when the PCI bridge is reset.
11:8	PSSTLD	Subsequent Target Latency Timer Duration Specifies the number of PCI clocks that a PCI master burst can be held in a wait state before a target disconnect is initiated.	Only set on reads. In synchronous mode, PSSLTD equals the maximum number of PCI clocks to disconnect. In asynchronous mode, PSSLTD plus 3 equals the maximum number of PCI clocks to disconnect. The asynchronous value must be 2 or less.

7:3		Reserved	
2	PTDT	PCI Discard Timer Disable 0 Disabled 1 Enabled	When enabled, the PCI bridge never discards delayed read data.
1		Reserved	
0	HCE	Host Configuration Enable 0 Disabled 1 Enabled	HCE controls host PCI access to the PCI bridge configuration registers. All host attempts to access the PCI bridge PCI configuration registers are retried. This give the local CPU (PLB master) time to initialize them before the host sees them.

In synchronous mode, the PCI subsequent target latency timer duration equals the maximum number of PCI clocks to disconnect. In asynchronous mode, PCI subsequent target latency timer duration plus 3 equals the maximum number of PCI clocks to disconnect. The asynchronous value must be 2 or less.

17.5.3.37 Power Management State Change Request Register (PCIC0_PMSCRR)

PCIC0_PMSCRR provides a method of informing the local processor of a power management state change request and prevents the completion of the write to PCIC0_PMCSR until the local processor indicates it is ready for the state change. All writes to PCIC0_PMCSR are retried until the local processor sets PCIC0_PMSCRR[APW] = 1. PCIC0_PMSCRR is used with the registers in the capability structure for power management. Descriptions of each bit are shown in Figure 17-58.

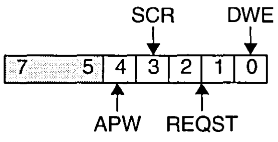


Figure 17-58. Power Management State Change Request Register (PCIC0_PMSCRR)

7:5	▨	Reserved	Always read as 0.
4	APW	Accept PMCI0_PCMSR Writes Always 1 if DWE is 0.	The local processor sets APW when the local processor is ready to change the power management state. APW is cleared when the host configuration writes to the PCIC0_PMCSR register is accepted. The local processor can write 0 to APW.
3	SCR	State Change Request	The PCI bridge sets SCR when a host writes PCIC0_PMCSR to request a power management state change. This drives an interrupt to the local processor informing it of a state change request. The local processor must simultaneously clear SCR and set APW = 1 when the local processor is ready to change the state. After SCR is cleared, new requests are not detected until the outstanding delayed write is accepted. The local processor can set SCR = 1. Note that any host side write to any byte (0x5C–0x5F) is considered a power state change request.
2:1	REQST	Request State	Indicates the new power management state requested by a delayed host write to PCIC0_PMCSR. This field is read-only from the PLB side.
0	DWE	Delayed Write Enable 0 Immediate write 1 Delayed write	When DWE is set to 1, any configuration write to the PCIC0_PMCSR is completed as a delayed write. All writes to PCIC0_PMCSR are retried until the local processor sets the "Accept PCIC0_PMCSR Write bit" (bit 4). When 0, any configuration write to the PCIC0_PMCSR is completed immediately. DWE is a don't care if a host write to PCIC0_PMCSR requests a state change from D3hot to D0.

17.6 Error Handling

The PCI bridge detects and reports several types of errors, which are reported to the PLB or the PCI. Status information is saved in the configuration registers to enable error type determination.

All errors are associated with either a cycle on the PLB or a cycle on the PCI bus.

Each error that can be detected is associated with a mask. If the mask is set, detection of that error condition is disabled. There are also masks for the $\overline{\text{PCISErr}}$, $\overline{\text{PCIPErr}}$, and PLB bus error signals that prevent reporting of any error by these signals. The masks do not prevent error detection.

The error types are as follows:

- PLB unsupported transfer type
- PCI master abort generated (while PCI master)
- PCI target abort received (while PCI master)
- PCI target data bus parity error detection
- PCI master data bus parity error detection
- PCI target address parity error detection
- PLB bus error detection

The following sections describe in detail how these errors are generated, what actions are taken for each, and how to reset a given error.

17.6.1 PLB Unsupported Transfer Type

This error occurs when the bridge PLB slave encounters an unsupported PLB transfer type. Table 17-11 outlines transfers not supported by the bridge PLB slave.

Table 17-11. PLB Unsupported Transfer Types

PLB Transaction	PCI Address Space
4- and 8-word line read/write	Nonmemory
16-word line read/write	Any
Burst	Nonmemory

Upon detection of this error, the bridge sets PCIC0_ERRSTS[PUR] = 1.

17.6.2 PCI Master Abort

This error is generated by the bridge PCI master when no target responds with $\overline{\text{PCIDevSel}}$ within the required time-out window and error detection is enabled. The bridge PLB slave may assert a PLB bus error signal on the PLB in response to this error, as explained below.

Two masks are associated with a PCI master abort. PCIC0_ERREN[MAEE] masks error reporting. If the error is detected, a PLB bus error signal is asserted if PCIC0_ERREN[MAEE] = 1. For reads, the bridge PLB slave still completes the transfer on the PLB, but drives 1s on the read data bus and the appropriate PLB bus error signal for each data beat. For posted writes, a PLB bus error is asserted for

1 cycle, asynchronously to the corresponding write data beat on the PLB. For connected writes, a PLB bus error signal is asserted with the data transfer, and the data is discarded. If PCIC0_ERREN[MAEE] = 0, error reporting is masked. No PLB bus error signal is asserted, regardless of the setting of PCIC0_ERREN[MEAE].

The following status bits are set:

1. If a master abort is signalled, PCIC0_STATUS[RMA] = 1. Setting of this field is non-maskable. Writing a 1 to PCIC0_STATUS[RMA] resets the field.
2. If master abort is detected as an error, PCIC0_ERREN[MEAE] is set to 1 to indicate an event that would cause a PLB bus error to be asserted by the bridge PLB slave, regardless of the setting of PCIC0_ERREN[MEAE]. This field can be reset by writing a 1 to PCIC0_ERREN[MEAE].
3. PCIC0_PLBBEAR and PCIC0_PLBBESRx are updated as follows:

   The address of the aborted request is saved in PCIC0_PLBBEAR if all PCIC0_PLBBESRx[MxAL] = 0 (PCIC0_PLBBEAR is unlocked). If all PCIC0_PLBBESRx[MxFL] = 0, PCIC0_PLBBESRx[MxET] = 0b101 to indicate a non-configured bank error; and PCIC0_PLBBESRx[MxRWS] is set to 0 on a write, 1 on a read. If PCIC0_ERREN[MAEE] = 0 or PCIC0_ERREN[MEAE] = 0, no PCIC0_PLBBEAR or PCIC0_PLBBESRx update is performed.

17.6.3 Bridge PCI Master Receives Target Abort While PCI Bus Master

This error is generated when the bridge PCI master receives a target abort while mastering a cycle on the PCI bus. Upon detection of this error, the bridge PLB slave may assert a PLB bus error signal on the PLB in response to this error, as explained below.

Two masks are associated with a target abort. PCIC0_ERREN[TAEE] masks error reporting. If the error is detected, a PLB bus error signal is asserted if PCIC0_ERREN[TAEE] = 1. For reads, the bridge PLB slave still completes the transfer on the PLB and drives the appropriate PLB bus error line for each data beat (note that for line reads, a PLB bus error signal is asserted for all data beats). For posted writes, if PCIC0_ERREN[TAEE] = 1, the bridge PLB slave asserts a PLB bus error for 1 cycle, asynchronously to the corresponding write data beat on the PLB. For connected writes, a PLB bus error signal is asserted with the data transfer, and the data is discarded. If PCIC0_ERREN[TAEE] = 0, error reporting is masked No PLB bus error signal is asserted, regardless of the setting of PCIC0_ERREN[MEAE]. If prefetching is occurring when a target abort is received, data preceding the target abort is kept in a prefetch buffer.

The following status bits are set:

1. If a target abort is received, PCIC0_STATUS[RTA] = 1. Setting this field is non-maskable. Writing a 1 to PCIC0_STATUS[RTA] clears the field.
2. If a target abort is detected as an error, PCIC0_ERRSTS[MEAE] = 1 to indicate an event that would cause the bridge PLB slave to assert a PLB bus error signal, regardless of the setting of PCIC0_ERREN[MEAE]. Writing a 1 to PCIC0_ERRSTS[MEAE] resets the field.
3. PCIC0_PLBBEAR and PCIC0_PLBBESRx are updated as follows:

   The address of the aborted request is saved in PCIC0_PLBBEAR if all PCIC0_PLBBESRx[MxAL] = 1 (PCIC0_PLBBEAR is unlocked). If all PCIC0_PLBBESRx[MxFL] = 1, PCIC0_PLBBESRx[MxET] = 0b101 to indicate a nonconfigured bank error, and PCIC0_PLBBESRx[MxRWS] is set to 0 on a write, or to 1 on a read. If PCIC0_ERREN[MAEE] = 0 or PCIC0_ERREN[MEAE] = 0, no PCIC0_PLBBEAR or PCIC0_PLBBESRx update is performed.

17.6.4 PCI Target Data Bus Parity Error Detection

This error is generated when the bridge PCI target detects a data bus parity error on write data from a PCI master doing a write cycle to PLB memory. PCI uses even parity.

Setting PCIC0_CMD[PER] = 0 masks this error.

The following status bits are set:

1. PCIC0_STATUS[DEPE] = 1 to indicate a PCI bus parity error. Setting this field is non-maskable. Writing a 1 to PCIC0_STATUS[DEPE] clears the field.
2. PCIC0_STATUS[SSE] = 1 if PCIC0_ERREN[WDPE] = 1. Writing a 1 to PCIC0_STATUS[SSE] clears the field.
3. PCIC0_ERRSTS[WDPE] = 1 if PCIC0_ERREN[WDPE] = 1. Writing a 1 to PCIC0_STATUS[SSE] clears the field.

17.6.5 PCI Master Data Bus Parity Error Detection

This error is generated when a data bus parity error is detected on the PCI bus during a cycle mastered by the bridge PCI master. The bridge PCI master checks parity on read cycles and samples $\overline{\text{PCIPErr}}$ on write cycles. The bridge PCI master may assert $\overline{\text{PCIPErr}}$ if the master detects a parity error on a read. PCI uses even parity.

Setting PCIC0_CMD[PER] = 0 masks this error. PCIC0_STATUS[DEPE] = 1. If a parity error is detected, writing a 1 to PCIC0_STATUS[DEPE] = 1 clears the field.

If PCIC0_ERREN[MEAE] = 1 and the error is detected as described, the PLB slave asserts a PLB error signal on the PLB in response to the error. For reads, a PLB bus error is asserted for each data beat in which bad parity was detected. For writes, a PLB bus error is asserted for each data beat in which bad parity was detected, but asynchronously to the actual transfer of write data on the PLB.

The following status bits are set:

1. PCIC0_STATUS[DEPE] = 1 if the bridge PCI master detects bad parity on read data, regardless of the state of PCI0_CMD[PER]. Writing a 1 to PCIC0_STATUS[DEPE] clears the field.
2. If a data bus parity error is detected as an error, PCIC0_ERRSTS[MEAE] = 1 to indicate an event that would cause a PLB bus error signal to be asserted by the bridge PLB slave, regardless of the state of PCIC0_ERREN[MEAE]. Writing a 1 to PCIC0_ERRSTS[MEAE] = 1 clears the field.
3. PCIC0_PLBBEAR and the PCIC0_PLBBESRx are updated as follows:

   The address of the PCI transaction where parity errors occurred is saved in the PCIC0_PLBBEAR. PCIC0_PLBBEAR is set if all PCIC0_PLBBESRx[MxAL] = 1 (PCIC0_PLBBEAR is unlocked). If PCIC0_PLBBESRx[MxFL] = 1, PCIC0_PLBBESRx[MxET] = 0b001 to indicate a parity error, and PCIC0_PLBBESRx[MxRWS] is set to 0 on a write, 1 on a read. If PCI0_CMD[PER] = 0 or PCIC0_ERRSTS[MEAE] = 0, no PCIC0_PLBBEAR or PCIC0_PLBBESRx update is performed.

Note: For clock ratios greater than 2:1 (independent of asynchronous/synchronous mode), the PCI bridge detects errors but does not assert a PLB bus error signal or log the error in PCIC0_PLBBEAR, PCIC0_PLBBESRx, and PCIC0_ERRSTS[MEAE].

17.6.6 PCI Address Bus Parity Error While PCI Target

This error occurs when a PCI address bus parity error is detected during the address phase of a cycle in which the bridge is the PCI target. PCI uses even parity.

Setting PCIC0_CMD[PER] = 0 masks this error. This error does not have an explicit status bit, however the following actions are taken:

1. PCIC0_STATUS[SSE] = 1 to indicate assertion of $\overline{\text{PCISErr}}$, if the mask at PCIC0_CMD[SE] = 1. Writing a 1 to PCIC0_STATUS[SSE] clears the field.
2. PCIC0_STATUS[DEPE] = 1 to indicate a PCI bus parity error, regardless of the state of PCI0_CMD[PER]. PCIC0_STATUS[DEPE] = 1 when any type of PCI parity error is detected. Writing a 1 to PCIC0_STATUS[DEPE] clears the field.

17.6.7 PLB Master Bus Error Detection

This error occurs when the bridge PLB master detects a PLB bus error. If the bridge PLB master receives a PLB bus error while mastering a read, the master associates the error with the currently executing read. If the master receives a PLB bus error while mastering a write or while idle, the master associates the error with a write.

PLB bus error detection is masked by PCIC0_ERREN[MEDE] = 0. If PCIC0_ERREN[MEDE] = 1, PLB bus error detection is enabled.

PCIC0_ERREN[MERE] controls the response of the bridge PCI target to PLB bus error detection. If PCIC0_ERREN[MERE] = 10 or 11, the bridge PCI target will execute a target abort. If PCIC0_ERREN[MERE] = 01 or 11, the bridge PCI target asserts $\overline{\text{PCISErr}}$ and allows the transaction to continue. If PCIC0_ERREN[MERE] = 11, the bridge PCI target both target aborts and asserts $\overline{\text{PCISErr}}$.

The following status bits are set:

1. If the bridge PCI target executes a target abort, PCIC0_STATUS[STA] = 1. The setting of PCIC0_STATUS[STA] in such an event is non-maskable. Writing a 1 to PCIC0_STATUS[STA] clears the field.
2. f the bridge PCI target asserts $\overline{\text{PCISErr}}$, PCIC0_STATUS[SSE] = 1. The setting of PCIC0_STATUS[SSE] is non-maskable. Writing a 1 to PCIC0_STATUS[SSE] clears the field.
3. If the bridge PCI target asserts $\overline{\text{PCISErr}}$, PCIC0_ERRSTS[SARME] = 1 to indicate that the bridge PCI target asserted $\overline{\text{PCISErr}}$ in response to a received PLB bus error signal. The setting of PCIC0_ERRSTS[SARME] is non-maskable. Writing a 1 to PCIC0_ERRSTS[SARME] clears the field.
4. PCIC0_ERRSTS[MED] = 1 to indicate that the bridge PLB master received a PLB bus error signal. Setting of PCIC0_ERRSTS[MED] is non-maskable. Writing a 1 to PCIC0_ERRSTS[MED] clears the field.

17.7 PCI Bridge Clocking Configuration

See “PCI Clocking” on page 7-7 for detailed information regarding the choice and setup involved with both synchronous and asynchronous PCI clocking modes.

17.8 PCI Power Management Interface

The PCI bridge supports *PCI Power Management Interface Specification* Revision 1.1 (PCI-PM).

17.8.1 Capabilities and Power Management Status and Control Registers

The PCI bridge has a capabilities structure in the PCI configuration space that indicates that the PCI bridge core is PCI Power Management capable. The capabilities structure includes the following registers:

- PCIC0_CAPID, value 0x01, indicates Power Management
- PCIC0_NEXTIPTR, points to next capabilities structure
- PCIC0_PMC, value 0x0202, indicates no specific capabilities
- PCIC0_PMCSR, indicates hold the current PowerState
- PCIC0_PMCSR_BSE, value 0x00, unused in PCI-to-PCI bridge
- PCIC0_Data, value 0x00, not used
- See “PCI Configuration Registers” on page 17-29 for details.

17.8.2 Power State Control

The current power management state is reported by reading PCIC0_PMCSR. The PCI bridge supports states D0, D1, D3hot, and D3cold. State D2 is not supported. When the state is not D0, the PCI bridge is masked from being a master or a memory or I/O target on the PCI bus. The PCI bridge can still be a config target. Thus, accesses claimed by the PCI bridge when in state D0 are no longer claimed, resulting in master aborts on the PLB or PCI if such an access is attempted. Note that this mask is independent of the state of the PCI Command register.

17.8.3 Changing Power States

The PCI bridge has two registers that control changing the power state. The host requests a change in the power state by writing to the PCIC0_PMCSR. The other register is PCIC0_PMSCRR), which provides a method of informing the local processor of a state change request and of preventing completion of the write to the PCIC0_PMCSR until the local processor indicates that it is ready for the state change.

Power state changes are handled as follows:

- If a host write to PCIC0_PMCSR requests an unsupported state change (such as a change to D2), the write is accepted but is ignored (no state change occurs).
- If a host write to PCIC0_PMCSR requests a change from D3hot to D0, the write is accepted. Then, the PCI bridge asserts the power management reset signal, which causes the entire SOC to be reset.

Note: The PCI bridge assumes that any requested state change from D3hot is always to D0.

- All other change requests are handled with the following sequence:

  1. The host requests a new power state by a PCI write to the PCIC0_PMCSR.
  2. The host PCI write is retried (unless PCIC0_PMSCRR[DWE] = 0).
  3. The host PCI write (retried or not) sets PCIC0_PMSCRR[SCR] = 1, which drives an interrupt to the local processor.
  4. The local processor recognizes the interrupt. The local processor checks PCIC0_PMSCRR[SCR, REQST] to determine the nature of the request.
  5. The local processor proceeds to power down the subsystem if the requested state is valid.
  6. When the subsystem has been powered down and is ready to change state, the local processor clears the PCIC0_PMSCRR[SCR] and sets PCIC0_PMSCRR[APW] = 1.
  7. When the host PCI write reoccurs:
     - It is accepted.
     - PCIC0_PMSCR is updated (only if the transition is valid).
     - PCIC0_PMSCRR[APW] = 0, unless PCIC0_PMSCRR[DWE] = 0, in which case PCIC0_PMSCRR[APW] = 1 always.
     - The PCI bridge enters a new power state.

The PCI bridge operates with the clock power management (CPM) logic to enable the bridge to be put into sleep mode under control of software. See Chapter 13, "Clock and Power Management" for discussion of the CPM function.

17.9 PCI Bridge Reset and Initialization

The following sections discuss resetting and initializing the PCI bridge.

17.9.1 Address Map Initialization

When the PCI bridge is the PCI master, it can generate memory, I/O, configuration, interrupt acknowledge, and special cycles. The method of cycle generation, and the associated address ranges, are fixed, except for memory cycles. PCI memory cycles are generated when the PCI bridge detects a cycle in one of three specified PLB address ranges. The sizes and address spaces of these ranges are specified using the PMM registers. Also, the address of the resulting PCI memory cycle can be an offset from the PLB address (address translation occurs). This translation is also specified in the PMM registers. The PMM registers *do not* default to usable values following reset; they must be initialized before attempting to generate PCI memory cycles.

When the PCI bridge is a target on the PCI bus, the PCI bridge can respond to memory cycles. The memory cycle address ranges that the PCI bridge responds to are specified in PCIC0_PTM1BAR and PCIC0_PTM2BAR. These registers are typically initialized as part of the standard PCI initialization process.

Figure 17-59 shows the desired address map. System memory resides from 0x00000000–0x0FFF FFFF in the CPU/PLB address space, which is accessible from the PCI in the same address space (PCI bridge as a memory target) as defined by PTM1/BAR1. PTM2/BAR2 is disabled in this example. The CPU/PLB master has two spaces in which to access PCI Memory space. Range 0 is 0x20000000 to 0x27FFFFFF and is mapped to the same address on the PCI bus, and is nonprefetchable. Range 1 is 0x28000000 to 0x2BFFFFFF, and is translated to address range 0x30000000 to 0x33FFFFFF of PCI memory space. Range 2 is disabled.

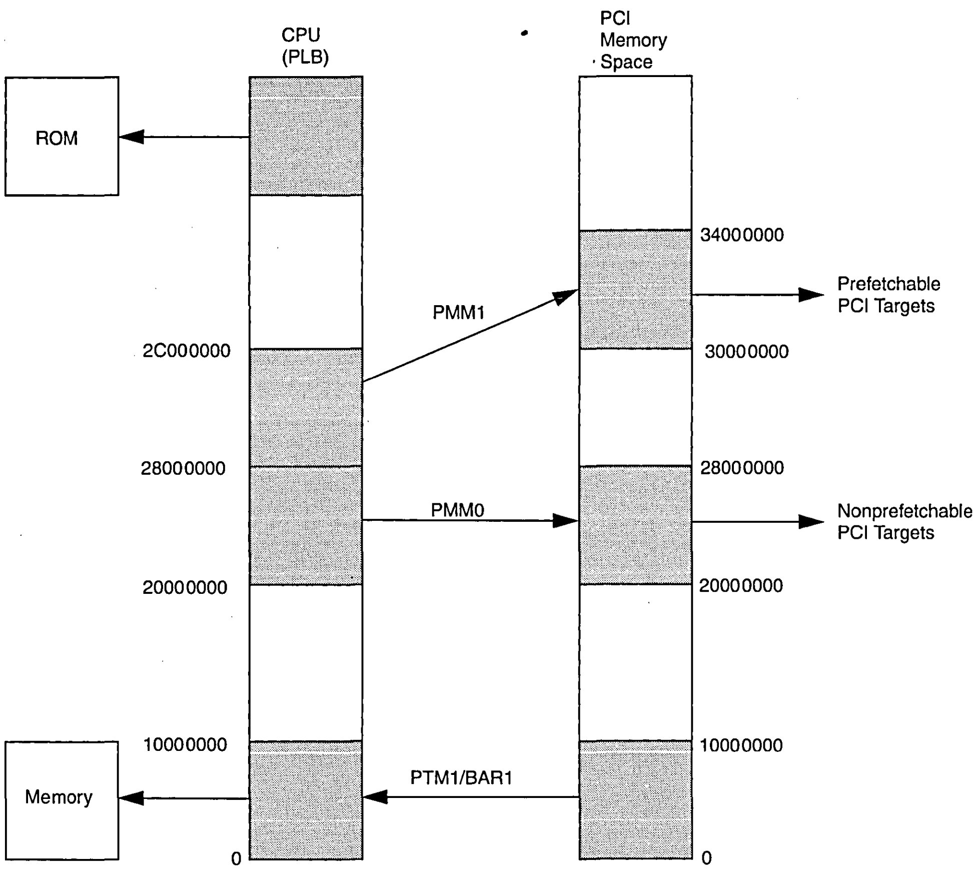


Figure 17-59. Example Address Map

The following register values provide the address map shown in Figure 17-59:

Table 17-12. Address Map Register Values

Register Name	Value	Comments
PCIL0_PMM0LA	0x20000000	
PCIL0_PMM0MA	0xF8000001	128MB; enabled; read prefetching not allowed.
PCIL0_PMM0PCILA	0x20000000	
PCIL0_PMM0PCIHA	0x00000000	
PCIL0_PMM1LA	0x28000000	
PCIL0_PMM1MA	0xFC000003	64MB; enabled; prefetching allowed.
PCIL0_PMM1PCILA	0x30000000	
PCIL0_PMM1PCIHA	0x00000000	
PCIL0_PMM2LA	0x00000000	
PCIL0_PMM2MA	0x00000000	Not enabled.
PCIL0_PMM2PCILA	0x00000000	
PCIL0_PMM2PCIHA	0x00000000	
PCIL0_PTM1MSr	0xF0000001	256MB; enabled.
PCIL0_PTM1LA	0x00000000	
PCIL0_PTM2MS	0x00000000	Not enabled.
PCIL0_PTM2LA	0x00000000	
PCIC0_PTM1BAR	0x00000008	PCI memory space; address decode starts at PCI address 0x0000 0000.
PCIC0_PTM2BAR	0x00000000	

17.9.2 Other Configuration Register Initialization

Additional register initialization is required, as follows:

- Error handling is initially disabled (error detection is masked). If error handling is to be enabled, PCIC0_ERREN must be initialized appropriately.
- PCIC0_BRDGOPT1 contains options for controlling the PLB. Its default values can be used.
- PCIC0_BRDGOPT2 contains options for controlling the PCI bus. Its default values assume that the PCI is run synchronously to the PLB, and that the PCI bridge is the primary host bridge. If the PCI bridge is used otherwise, the values must be changed accordingly.

17.9.3 Target Bridge Initialization

The PCI bridge can also respond as a configuration target; however, the PCI bridge only responds as a configuration target when the PCIIDSel pin is attached, rather than pulled inactive. Note that if the size and local address of these ranges are not strapped to desired values at reset, the local CPU must specify them by setting the PTM Memory Size and Local Address registers before initializing PCIC0_PTM1BAR and PCIC0_PTM2BAR.

The local CPU must update the following registers (if the default value is not suitable or they were not strapped to appropriate values at reset) before setting the Host Config Enable bit:

- The address map registers (see "Address Map Initialization" on page 17-61)
- PCIC0_VENDID
- PCIC0_DEVID
- PCIC0_REVID
- PCIC0_CLS
- PCIC0_SBSYSID
- PCIC0_SBSYSVID

17.9.4 Local Processor Boot from PCI Memory

The PCI bridge has a mode that enables a PLB master to access a PCI memory range without initial configuration cycles. This mode is enabled when CPC0_PSR[RL] = 1. System designers can use this mode to enablea processor to access a boot ROM in PCI memory space.

The PCI bridge comes out of reset with PMM0 enabled and programmed for the address range, 0xFFFE0000–0xFFFFFFFF. Also, PCIC0_CMD[ME] = 1 after reset. Note that enabling PCI boot mode does not prevent subsequent updates to the PMM0 registers.

Note: The PPC405GP allows booting from PCI memory. See Chapter 9, "Pin Strapping and Sharing" for more information.

17.9.5 Type 0 Configuration Cycles for Other Devices

Twenty-one devices can be accessed using the PCIIDSel mechanism. The PCI master asserts 1 bit of AD(31:11) for type 0 configuration cycles based on the value in the Device Number field. The mapping is as follows:

- If device number is 1, AD(11) is asserted
- If device number is 2, AD(12) is asserted

  .
  .
  .

- If device number is 21, AD(31) is asserted

If device number contains a value of 22–31, no bit of AD(31:11) is asserted.

17.10 Timing Diagrams

This section contains timing diagrams of several different PCI bridge operations. The following paragraphs describe each diagram in detail. Each description assumes basic knowledge of PCI and PLB protocols.

Each operation is shown in both synchronous and asynchronous modes. The PCI is clocked at 33 MHz in synchronous mode and 66 MHz in asynchronous mode. The PLB is clocked at 100 MHz in all cases.

The SDRAM uses a 32-bit, PC100 memory interface configured for $\overline{\text{CAS}}$ latency of 2, command leadoff of 2, and RAS to CAS delay (T_{rcd}) of 2. All memory accesses are page idle, unless indicated otherwise.

Note: The PLB signals shown in the following timing diagrams are not externally accessible. They are included for information purposes and as an aid to understanding the PCI operations. For more information on these signals, refer to *Processor Local Bus Architecture Specifications*.

17.10.1 PCI Timing Diagram Descriptions

The following sections briefly describe each of the timing diagrams that follow the descriptions. Each description covers both the asynchronous and synchronous clocking modes for that operation. The timing diagrams then follow with all of the asynchronous diagrams grouped together followed by all of the synchronous diagrams grouped together.

17.10.1.1 PCI Master Burst Read From SDRAM

Figure 17-60 (asynchronous) and Figure 17-67 (synchronous) show a PCI Master executing a 128-byte Read Multiple from SDRAM. PCI bridge retries the initial request and performs a delayed read. PCI bridge executes a variable-length, doubleword read burst on PLB to SDRAM, filling its 96-byte read buffer. The read is a page hit. When the PCI master re-requests its read, PCI bridge begins bursting out of its read buffer, while continuing to prefetch from SDRAM.

17.10.1.2 PCI Master Burst Write To SDRAM

Figure 17-61 and Figure 17-68 show a PCI Master executing a 128-byte Write to SDRAM. PCI bridge accepts several beats of data into its 64-byte write buffer before executing variable-length, doubleword write bursts on PLB to SDRAM. The final write burst is fixed-length, since the PCI write has completed, and PCI bridge knows the exact burst length.

17.10.1.3 CPU Read From PCI Memory Slave, Nonprefetching

In Figure 17-62 and Figure 17-69, a PLB Master (CPU) executes a single-beat 64-bit read from a region of PCI memory marked as nonprefetchable. PCI bridge responds as a 32-bit PLB slave, so the CPU executes conversion cycles for each read. PCI bridge executes a single-beat PCI read for both PLB read requests.

17.10.1.4 CPU Read From PCI Memory Slave, Prefetching

In Figure 17-63 and Figure 17-70, a PLB Master (CPU) executes 8, 64-bit, single-beat reads from a region of PCI memory marked as prefetchable. The first PLB read causes PCI bridge to execute a 64-byte Read Multiple to fill its 64-byte read prefetch buffer. The data for subsequent PLB reads is provided from the read buffer and no PCI cycles are generated. PCI bridge responds as a 32-bit PLB slave, so the CPU executes conversion cycles for each read.

17.10.1.5 CPU Write To PCI Memory Slave

Figure 17-64 and Figure 17-71 show a PLB Master (CPU) executing 4, 64-bit, single-beat writes to PCI memory. PCI bridge responds as a 32-bit slave, so the CPU executes conversion cycles for each write. PCI bridge posts the writes in its 4-entry write buffer, and executes a PCI single-beat Memory Write for each request.

17.10.1.6 PCI Memory To SDRAM DMA Transfer

Figure 17-65 and Figure 17-72 show a DMA transfer of data from PCI memory to SDRAM. The DMA PLB Master executes a 4-doubleword read burst to PCI bridge followed by a 4-doubleword write burst to SDRAM. For the read, PCI bridge executes a 32-byte PCI Read Line.

17.10.1.7 SDRAM To PCI Memory DMA Transfer

Figure 17-66 and Figure 17-73 show a DMA transfer of data from SDRAM to PCI memory. The DMA PLB Master executes a 4-doubleword read burst from SDRAM followed by a 4-doubleword write burst to PCI memory. PCI bridge then executes a 32-byte write on the PCI bus.

17.10.2 Asynchronous

The following diagrams are for asynchronous clocking mode. Note that all of the diagrams flow across multiple pages. Each diagram begins with cycle 1 on the left facing page.

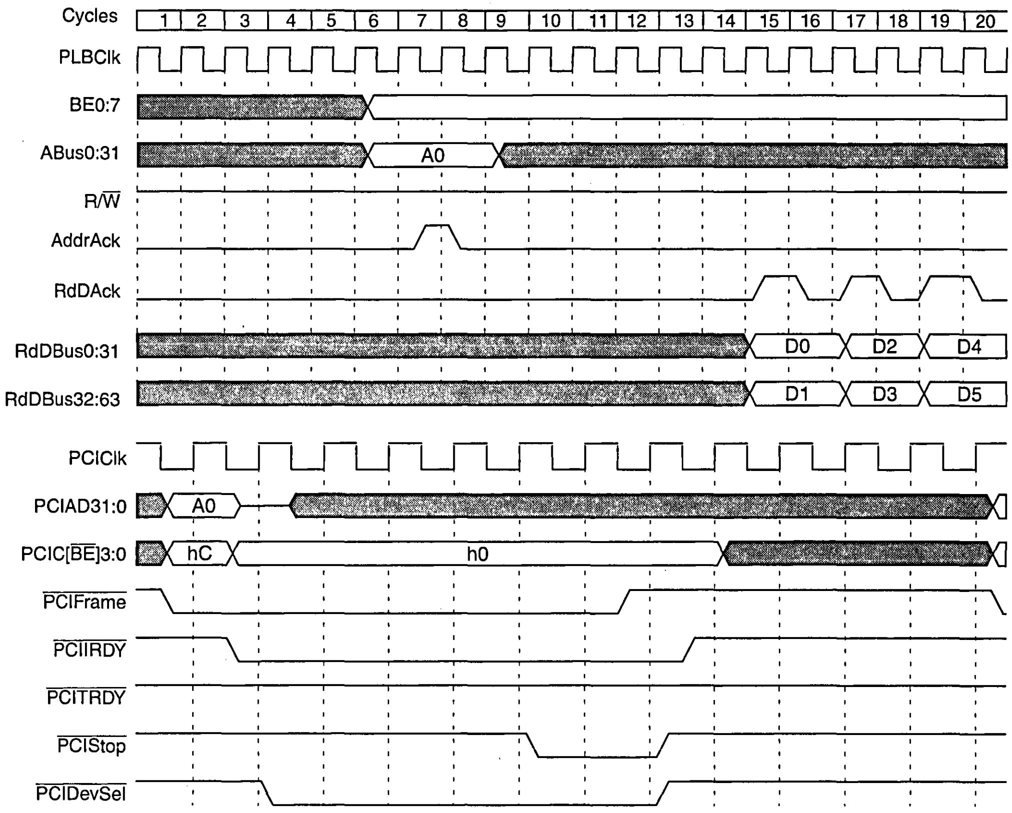


Figure 17-60. PCI Master Burst Read From SDRAM

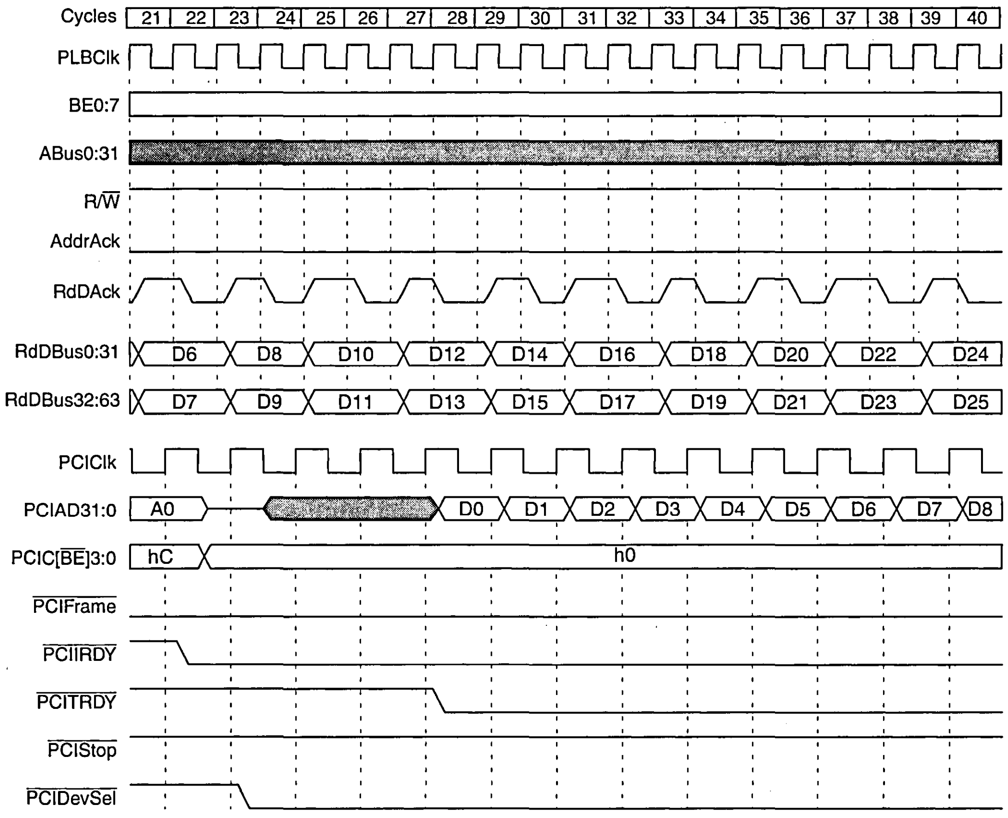


Figure 17-60. PCI Master Burst Read From SDRAM (Continued)

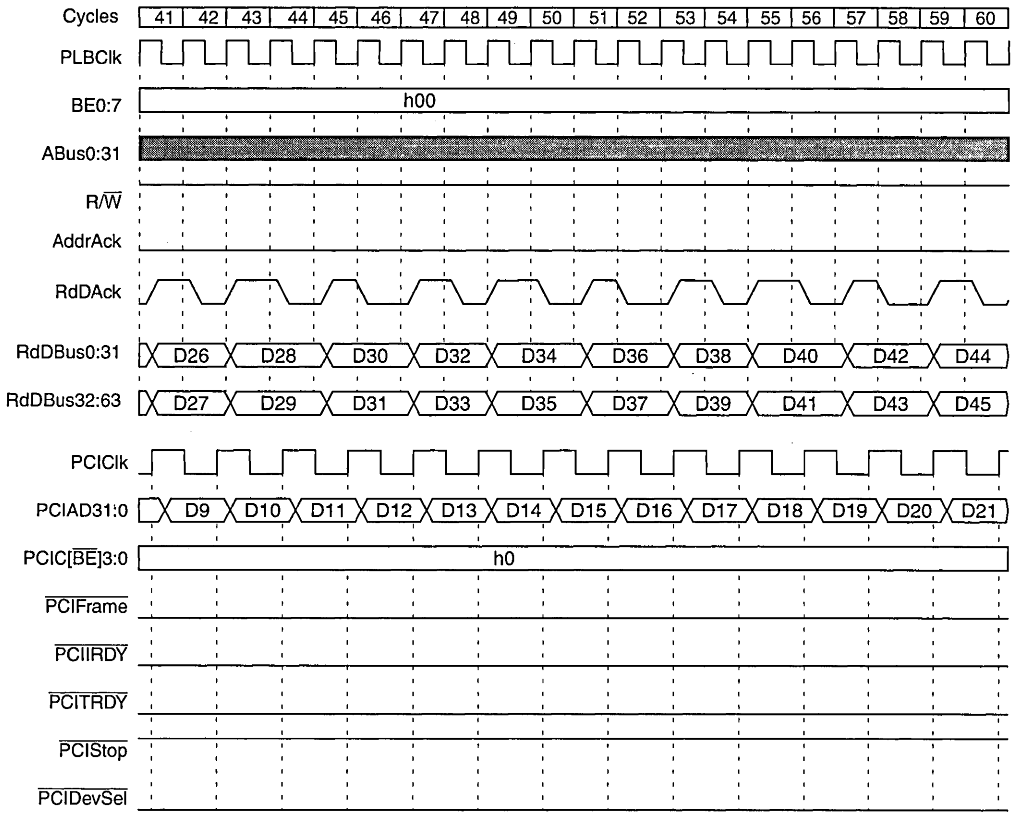


Figure 17-60. PCI Master Burst Read From SDRAM (Continued)

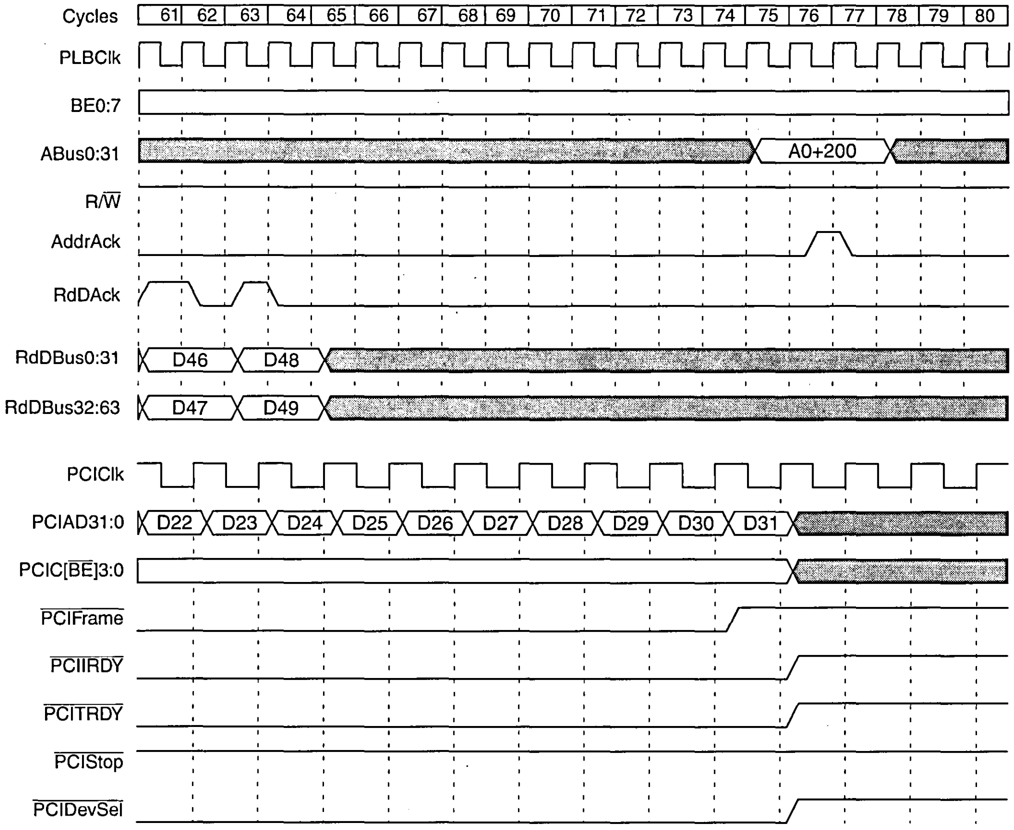


Figure 17-60. PCI Master Burst Read From SDRAM (Continued)

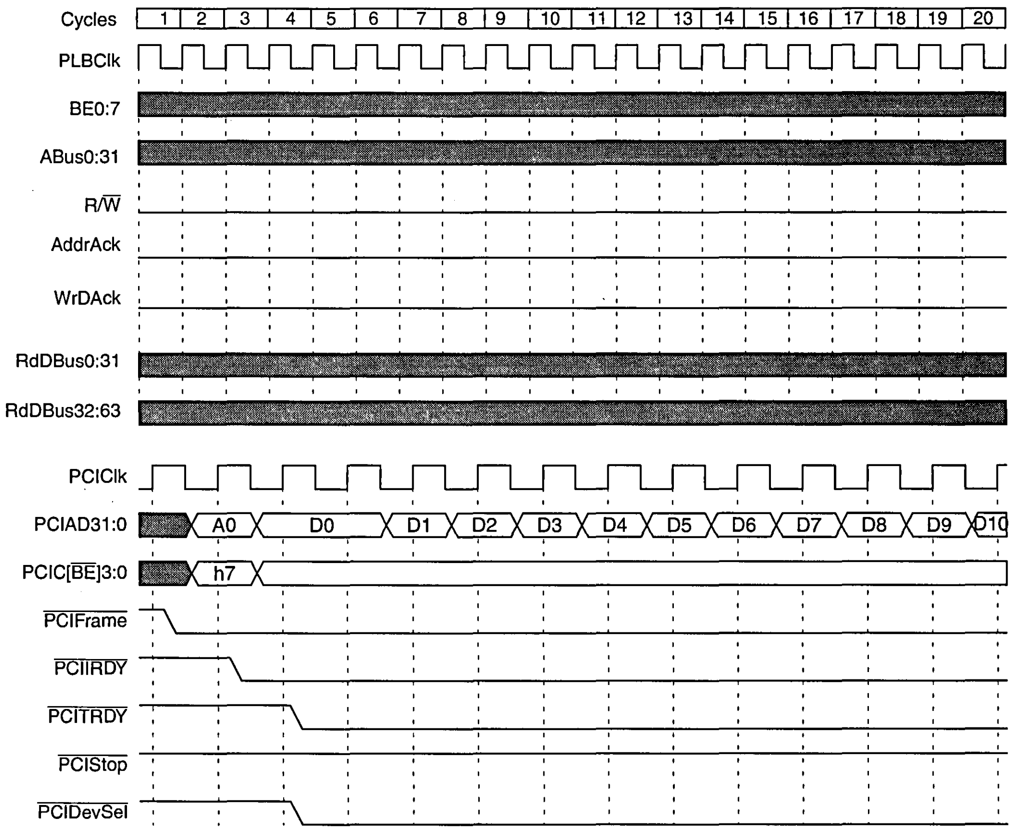


Figure 17-61. PCI Master Burst Write To SDRAM

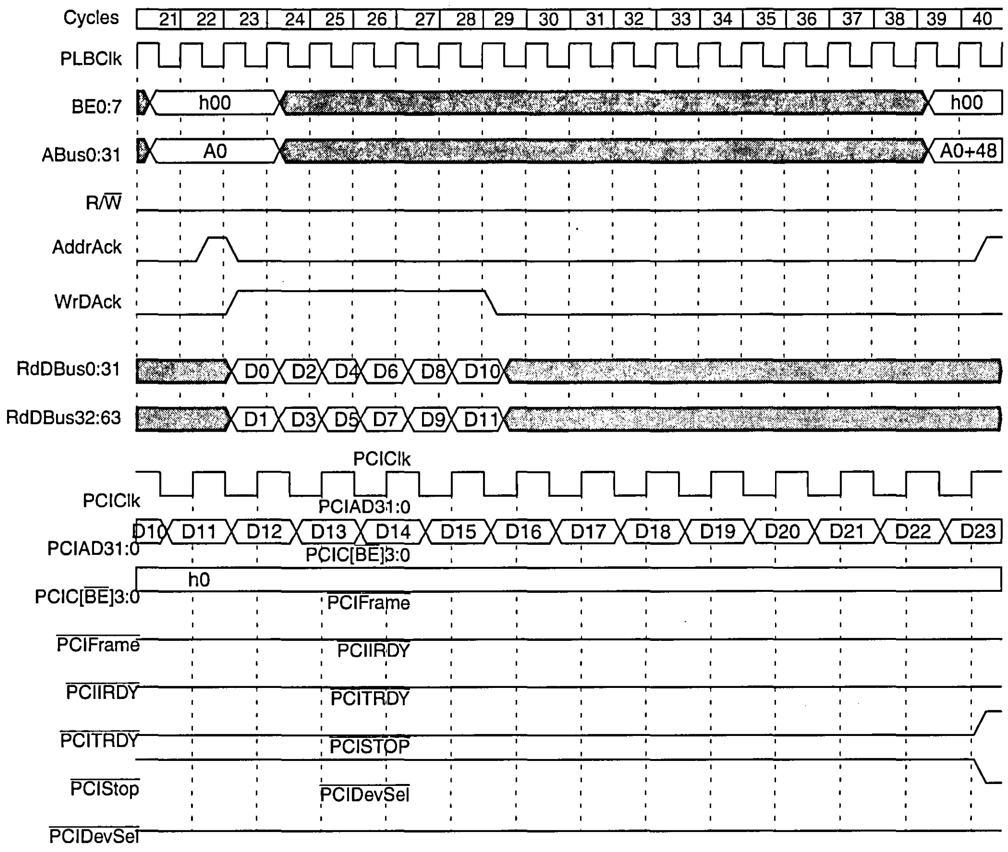


Figure 17-61. PCI Master Burst Write To SDRAM (Continued)

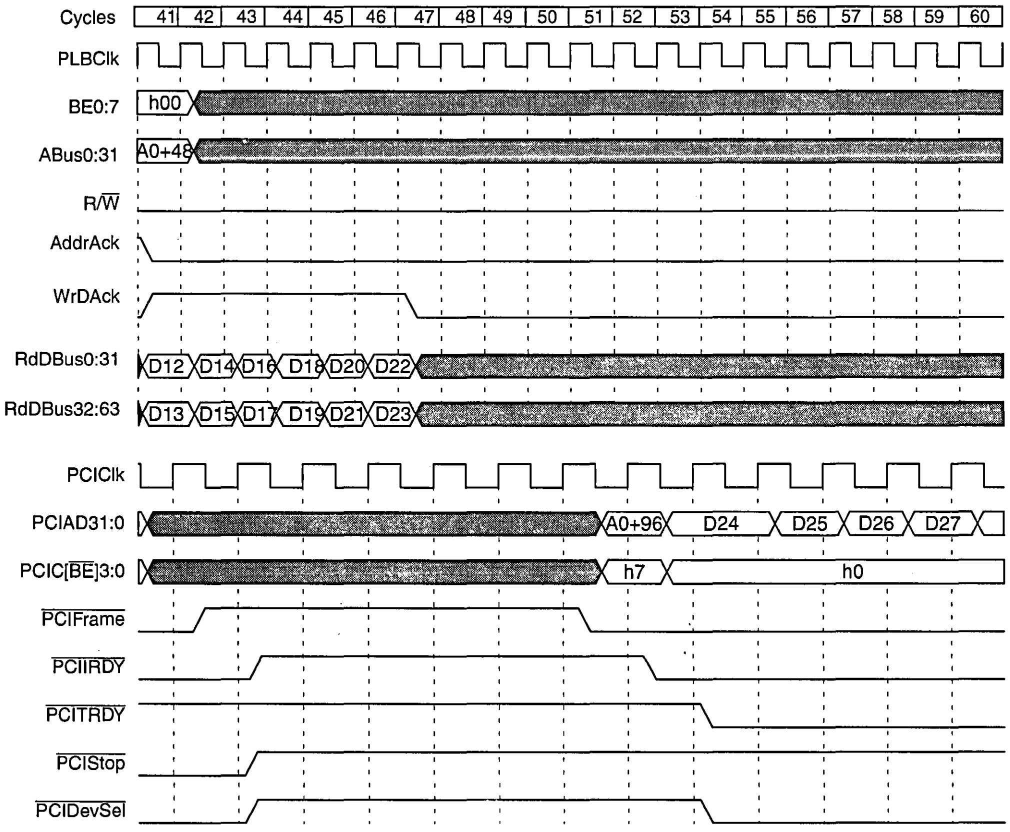


Figure 17-61. PCI Master Burst Write To SDRAM (Continued)

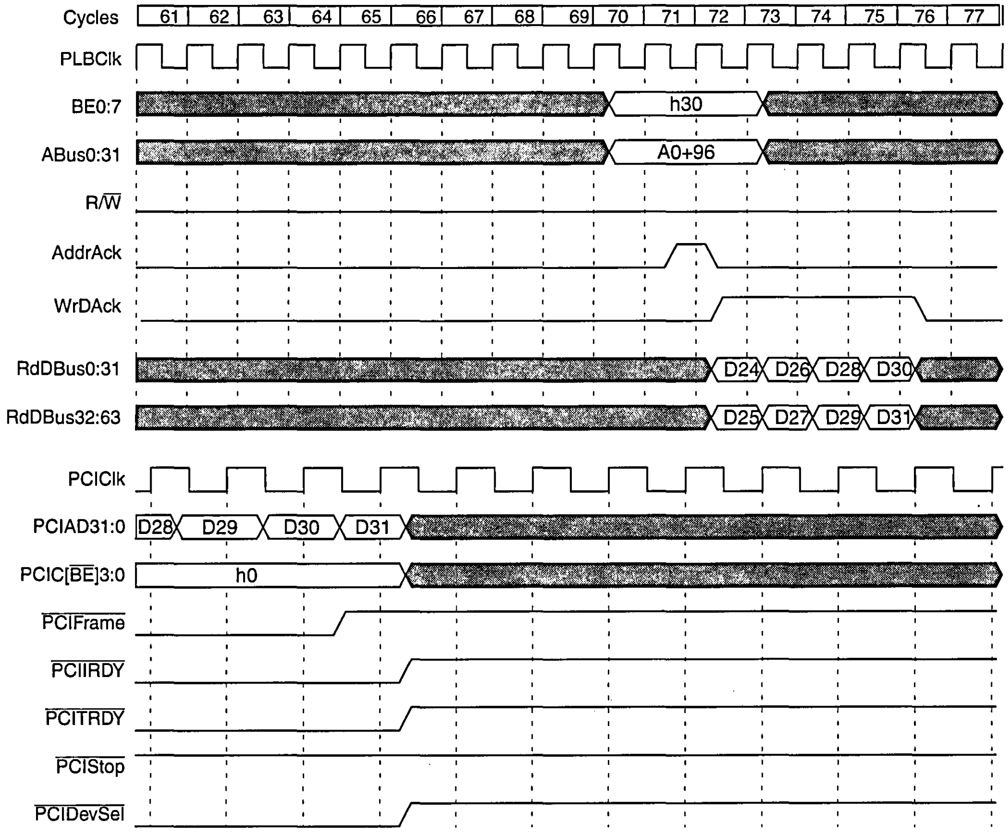


Figure 17-61. PCI Master Burst Write To SDRAM (Continued)

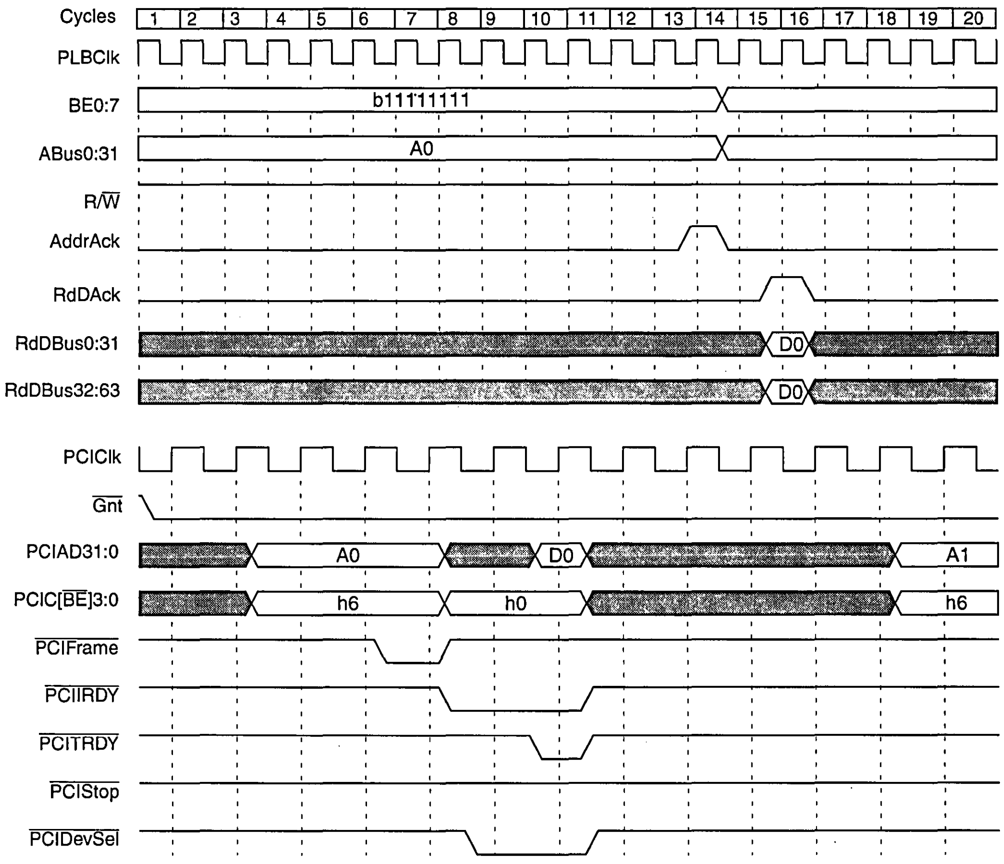


Figure 17-62. CPU Read From PCI Memory Slave, Nonprefetching

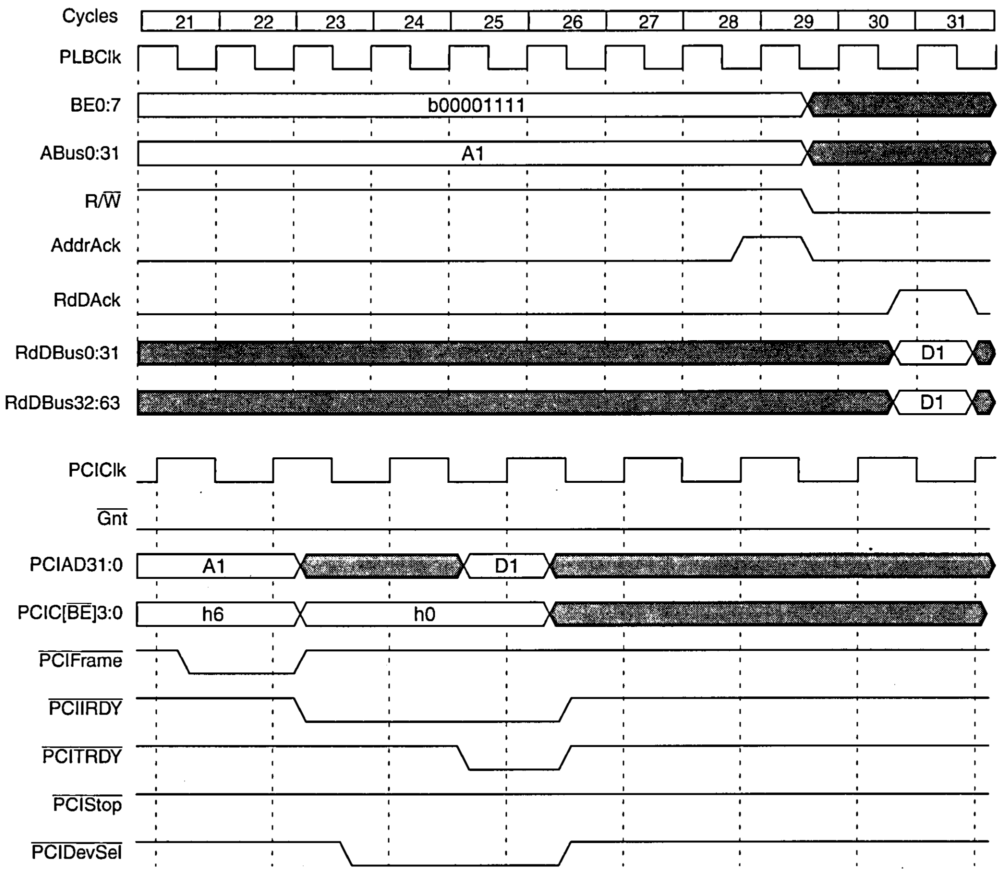


Figure 17-62. CPU Read From PCI Memory Slave, Nonprefetching (Continued)

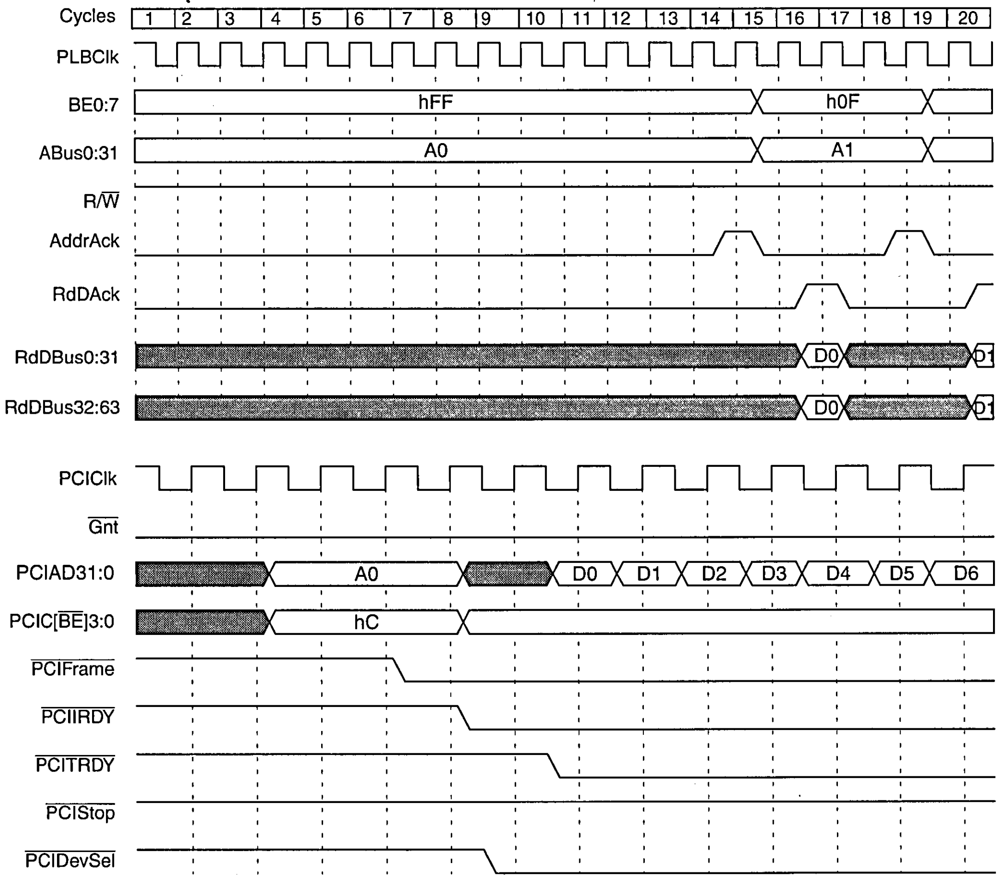


Figure 17-63. CPU Read From PCI Memory Slave, Prefetching

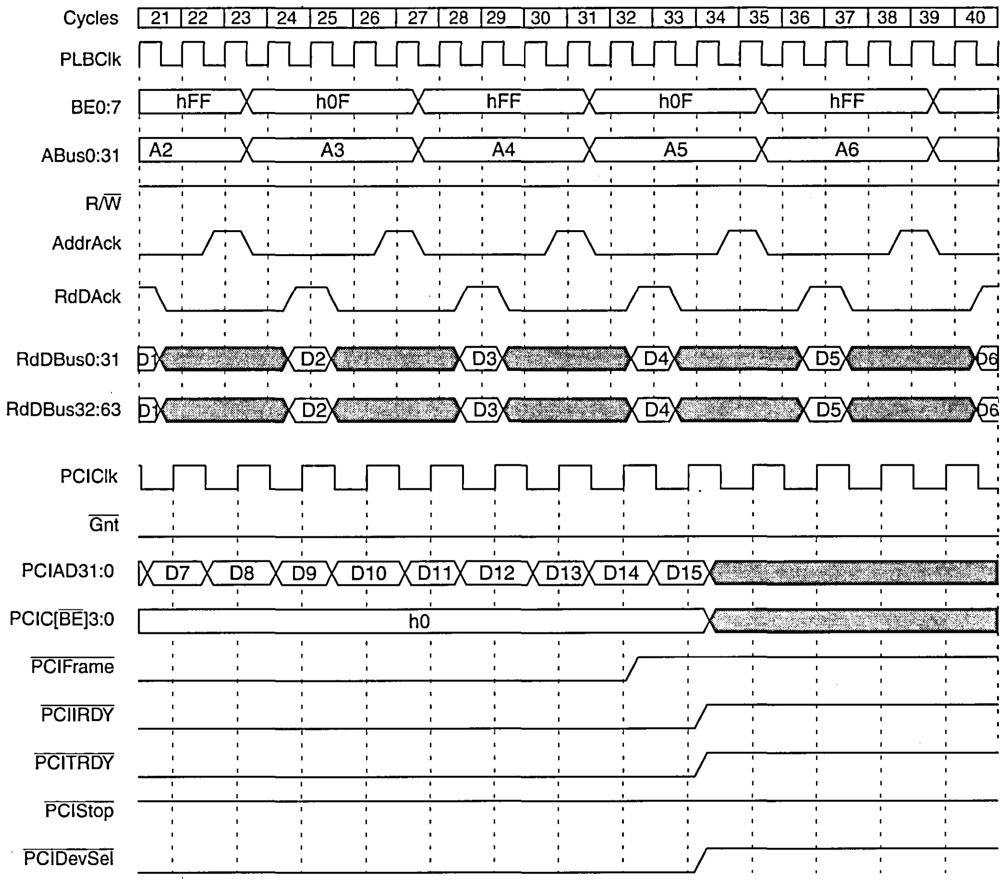


Figure 17-63. CPU Read From PCI Memory Slave, Prefetching (Continued)

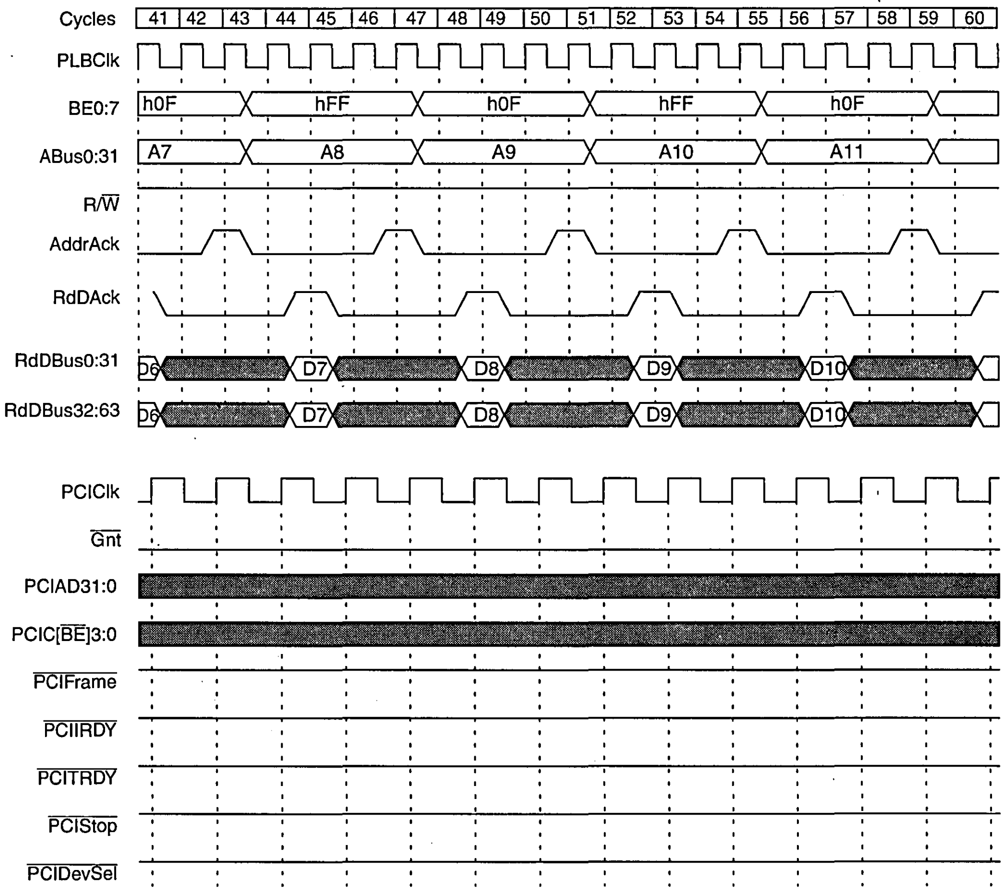


Figure 17-63. CPU Read From PCI Memory Slave, Prefetching (Continued)

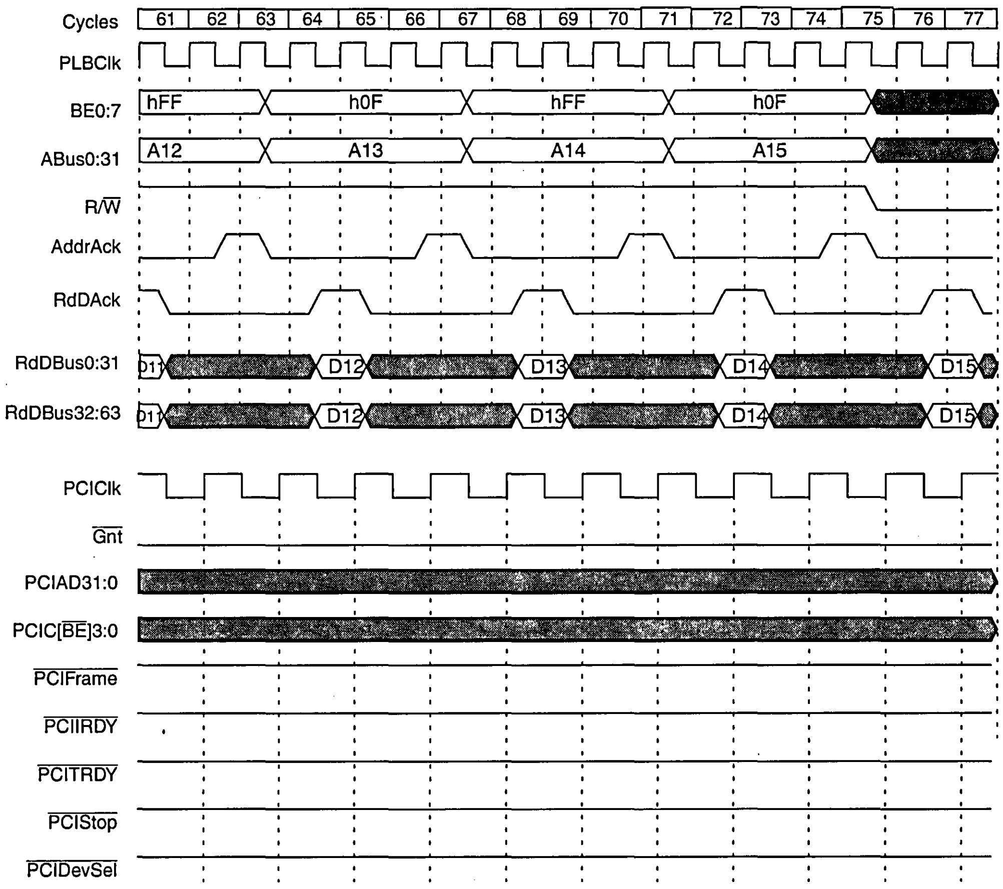


Figure 17-63. CPU Read From PCI Memory Slave, Prefetching (Continued)

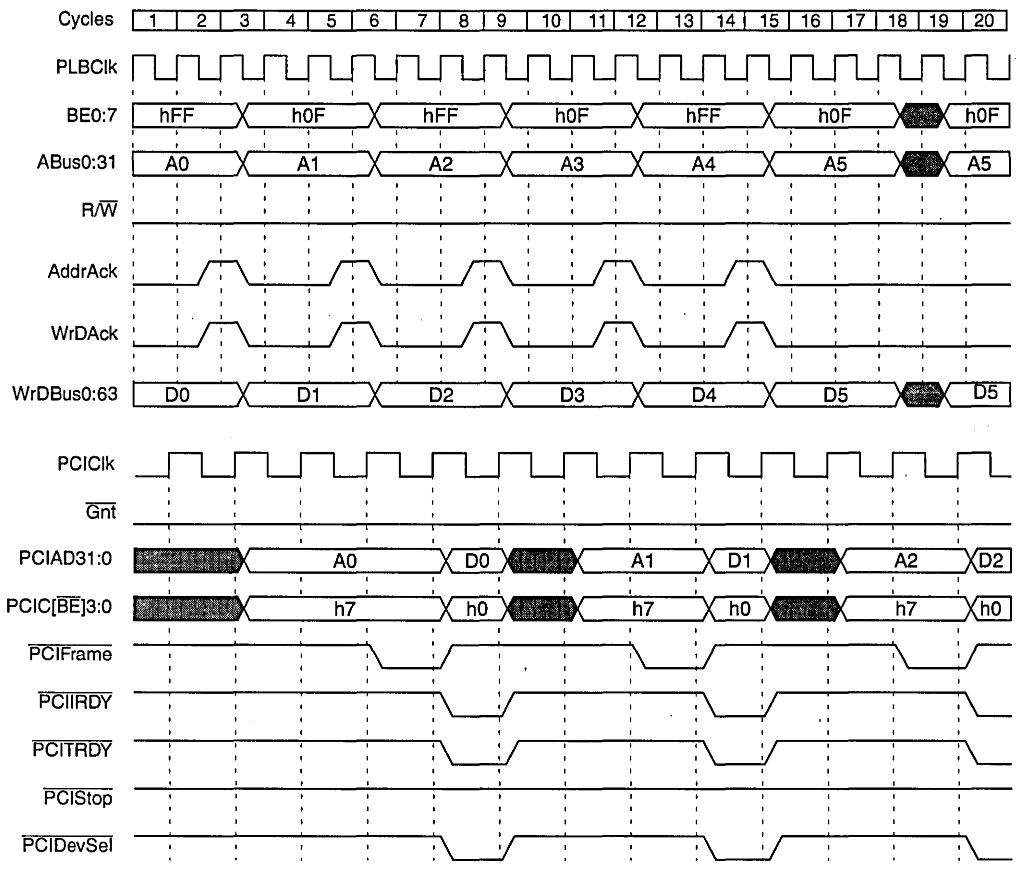


Figure 17-64. CPU Write To PCI Memory Slave

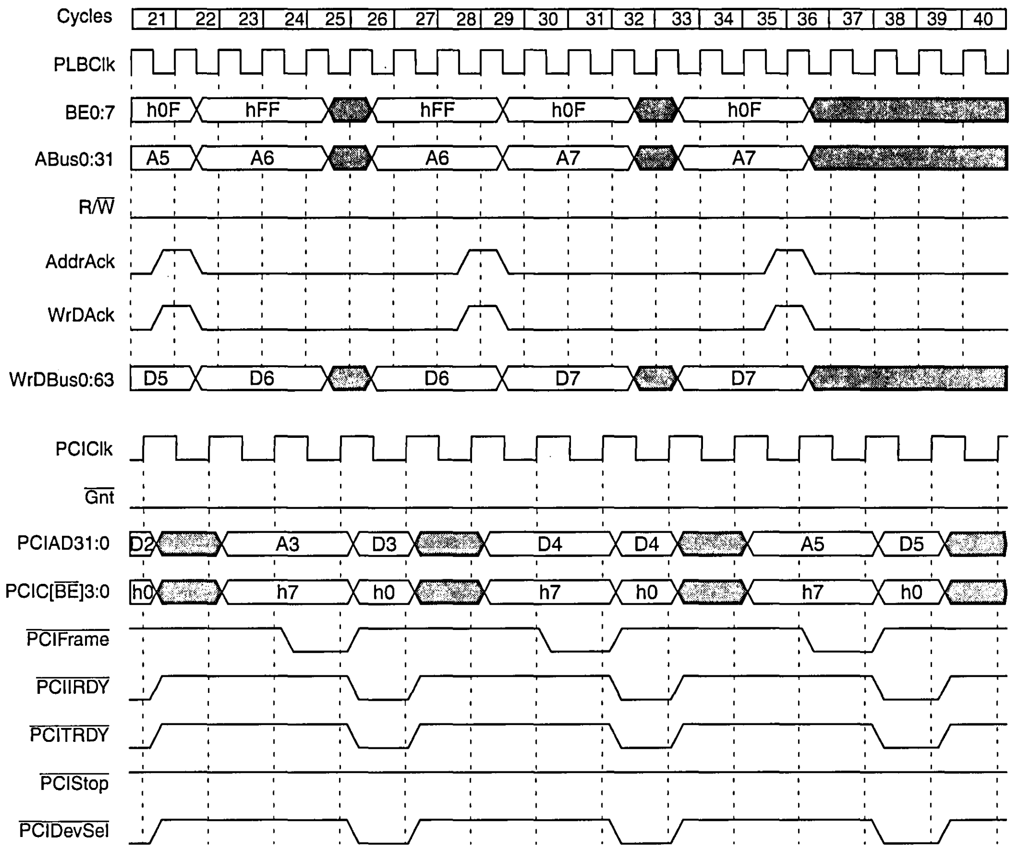


Figure 17-64. CPU Write To PCI Memory Slave (Continued)

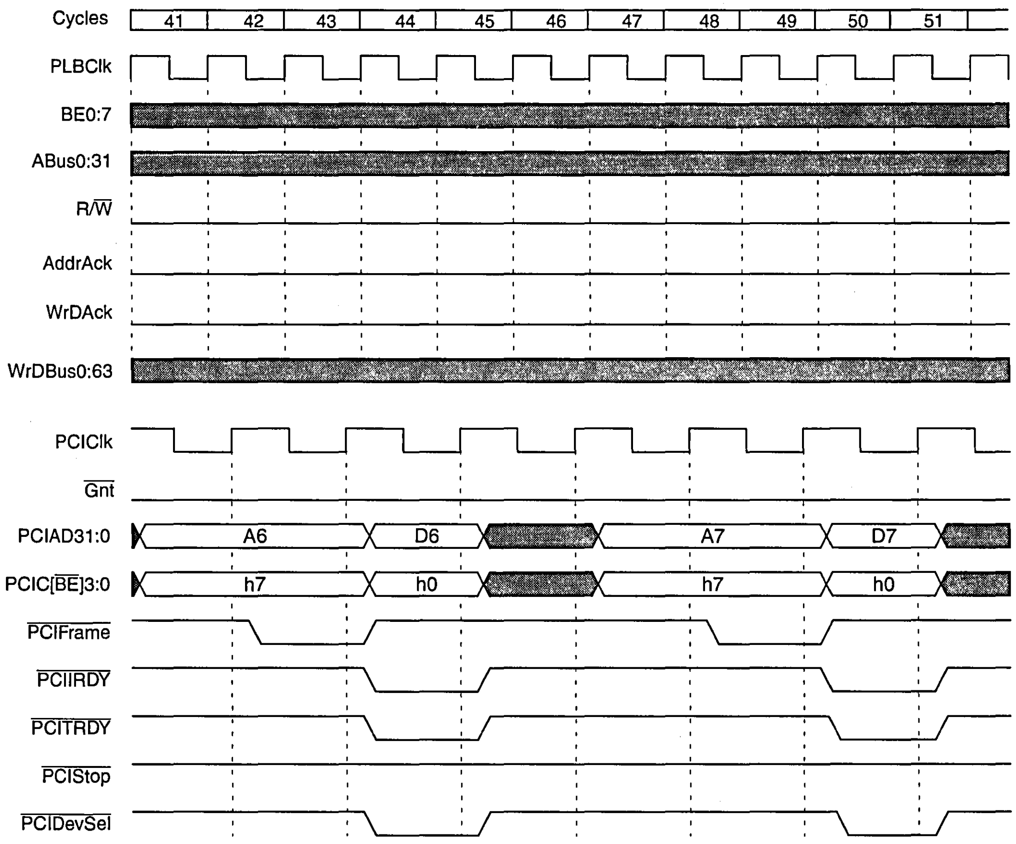


Figure 17-64. CPU Write To PCI Memory Slave (Continued)

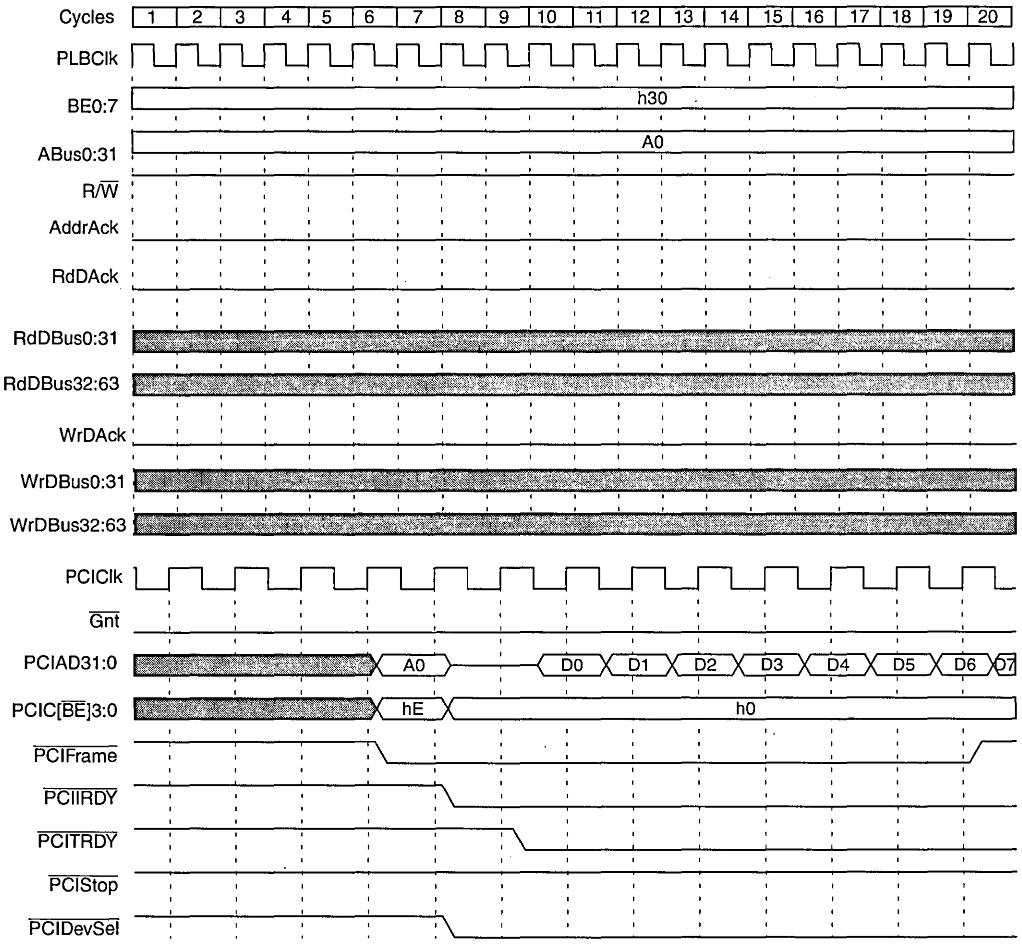


Figure 17-65. PCI Memory To SDRAM DMA Transfer

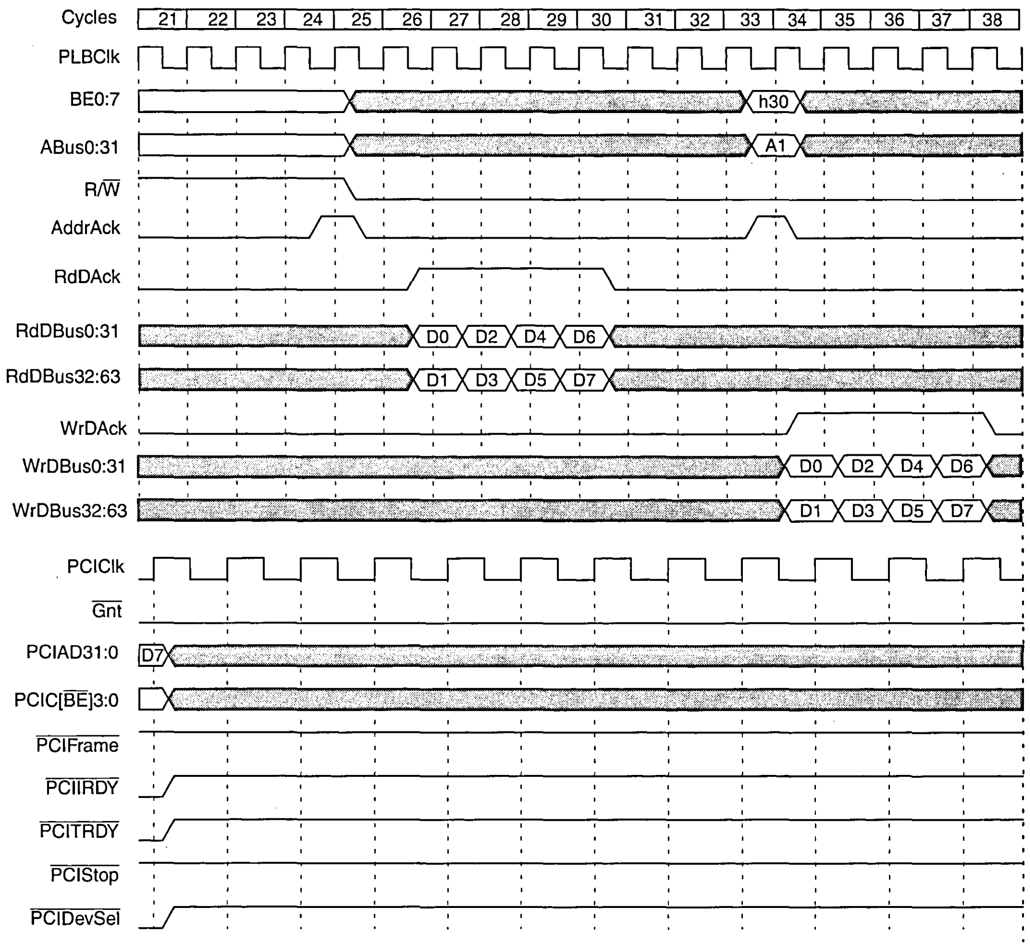


Figure 17-65. PCI Memory To SDRAM DMA Transfer (Continued)

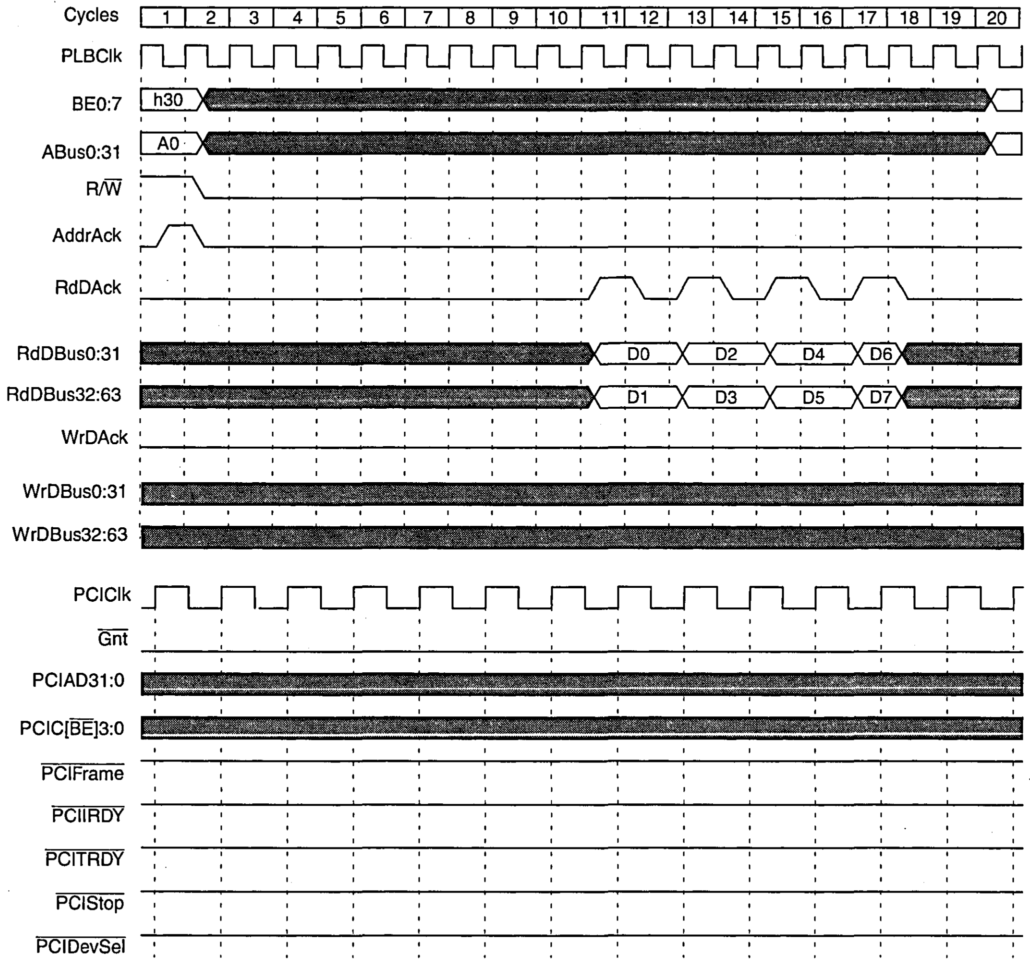


Figure 17-66. SDRAM To PCI Memory DMA Transfer

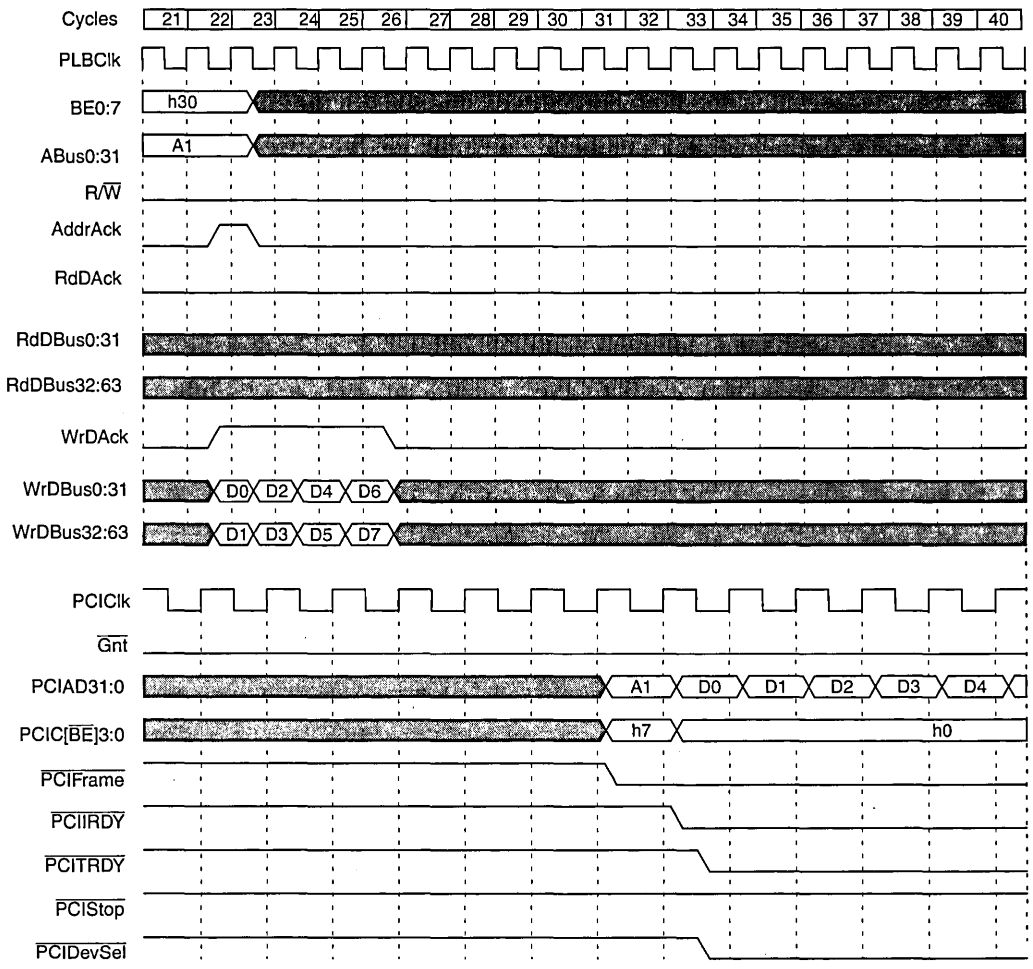


Figure 17-66. SDRAM To PCI Memory DMA Transfer (Continued)

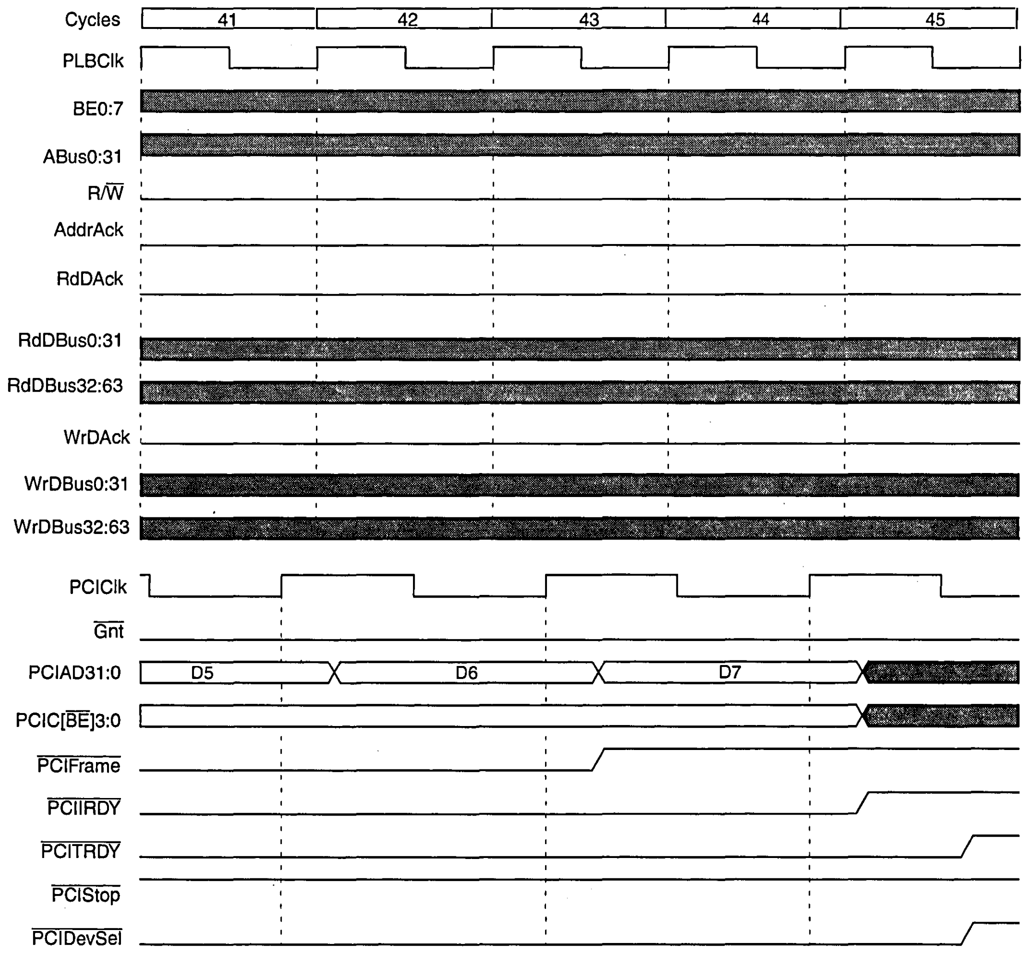


Figure 17-66. SDRAM To PCI Memory DMA Transfer (Continued)

17.10.3 Synchronous

The following diagrams are for synchronous clocking mode. Note that all of the diagrams flow across multiple pages. Each diagram begins with cycle 1 on the left facing page.

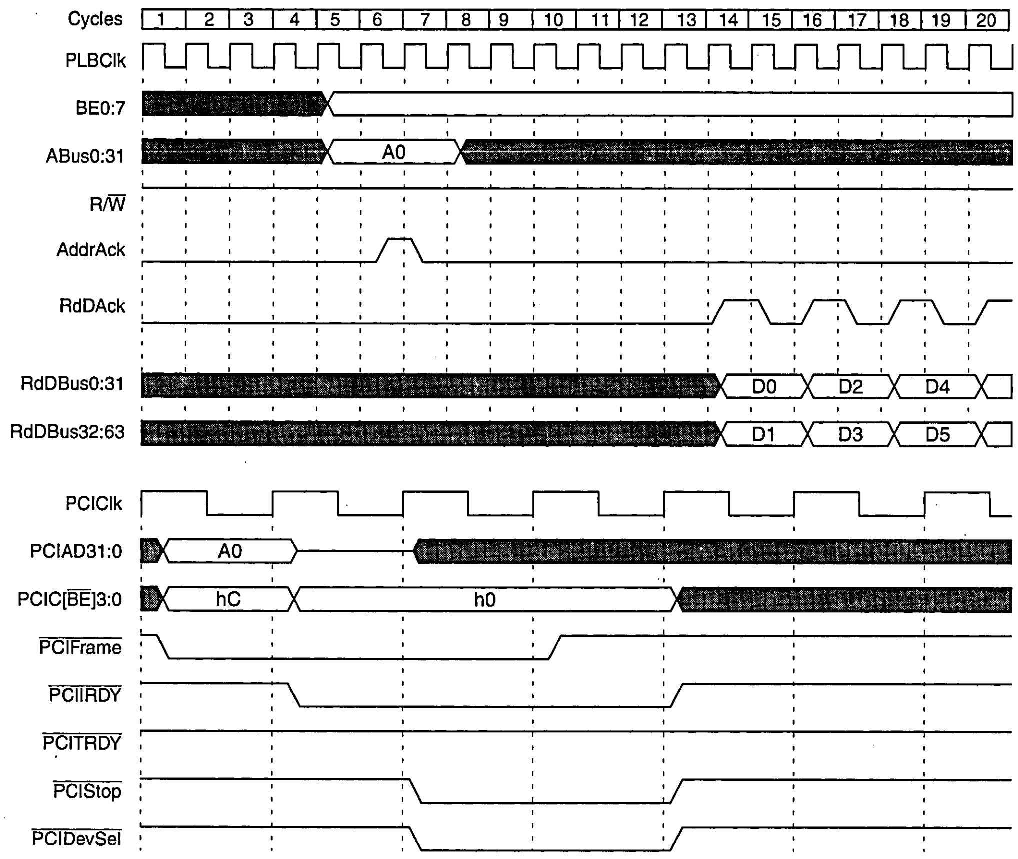


Figure 17-67. PCI Master Burst Read From SDRAM

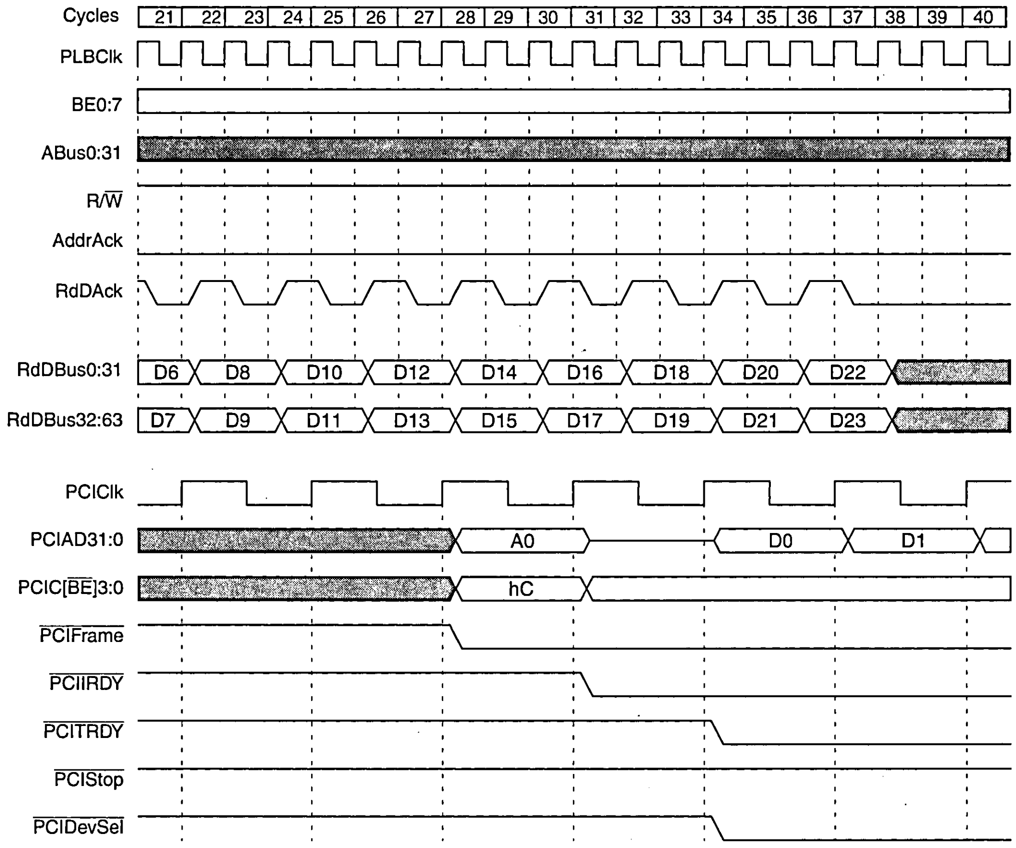


Figure 17-67. PCI Master Burst Read From SDRAM (Continued)

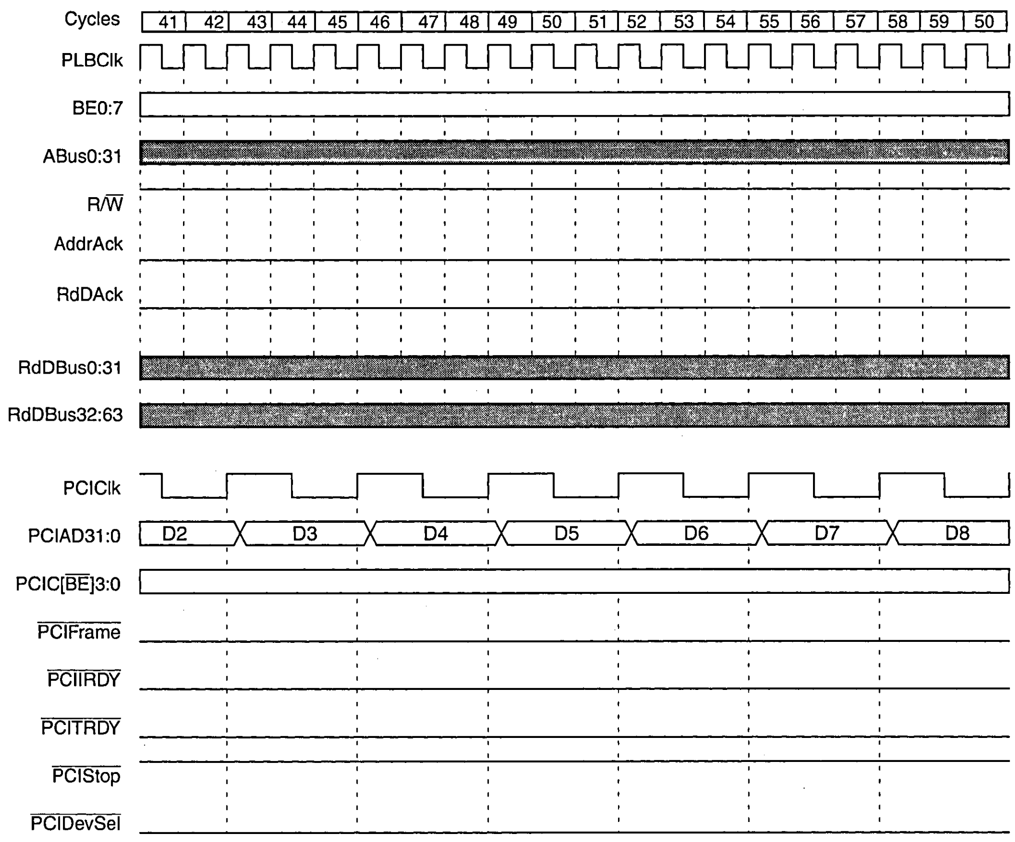


Figure 17-67. PCI Master Burst Read From SDRAM (Continued)

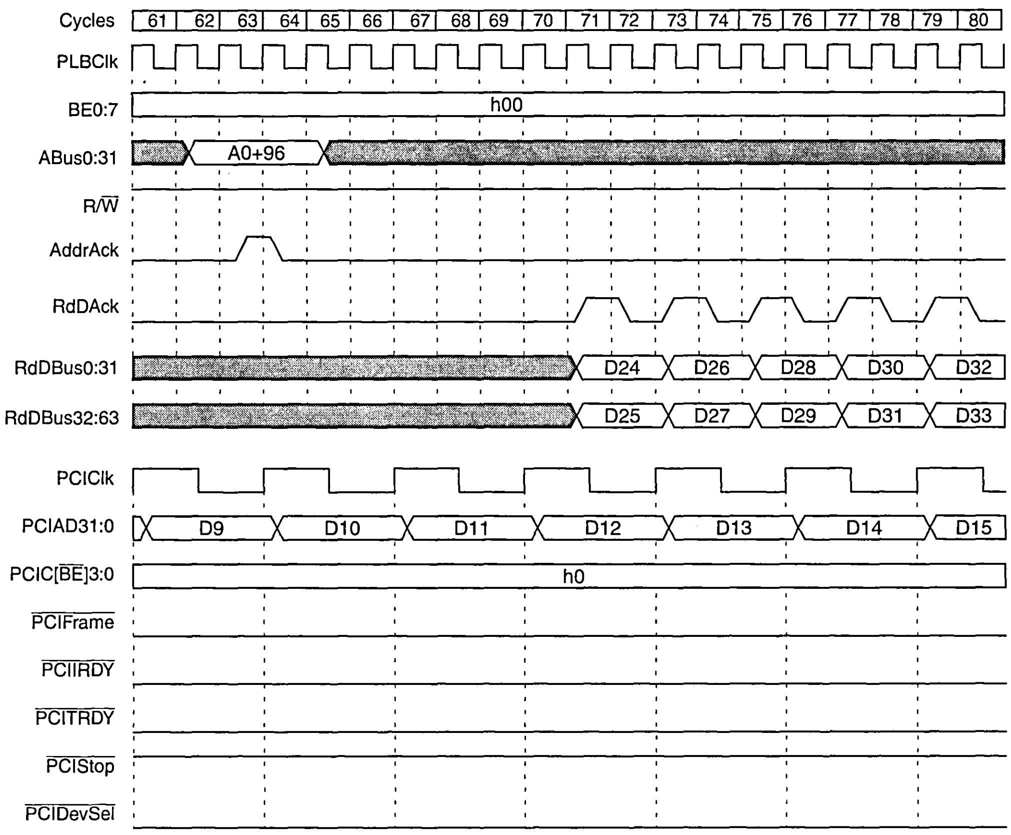


Figure 17-67. PCI Master Burst Read From SDRAM (Continued)

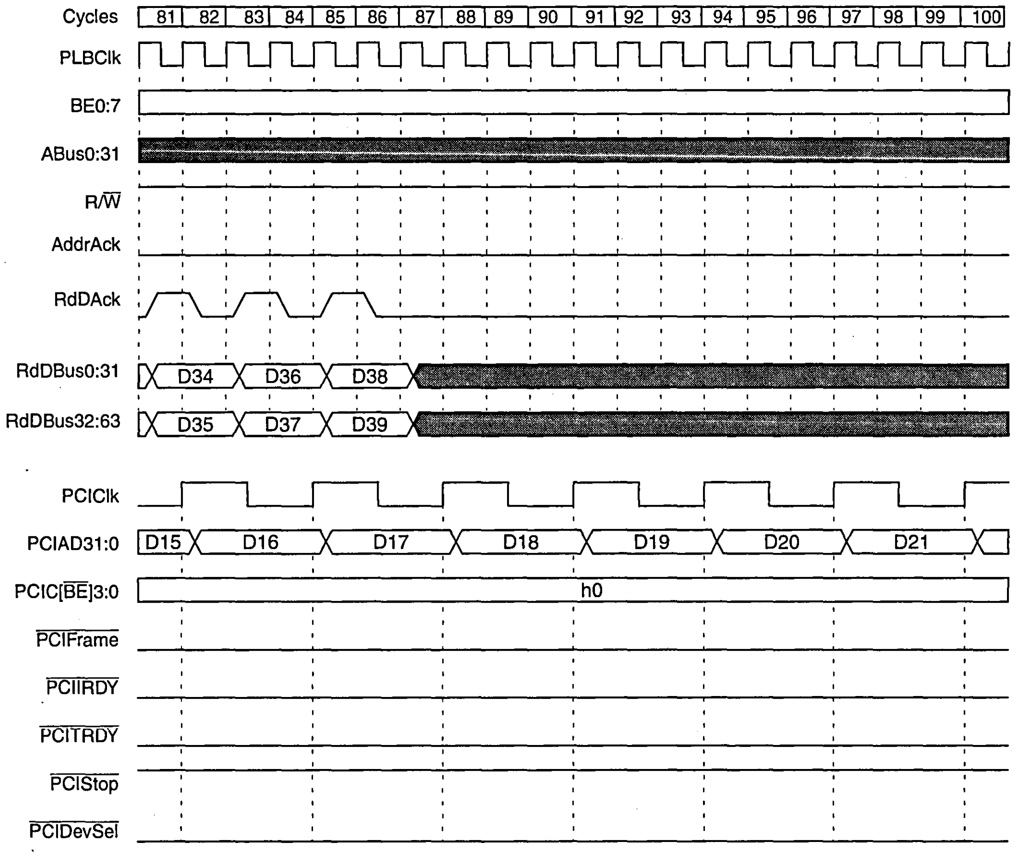


Figure 17-67. PCI Master Burst Read From SDRAM (Continued)

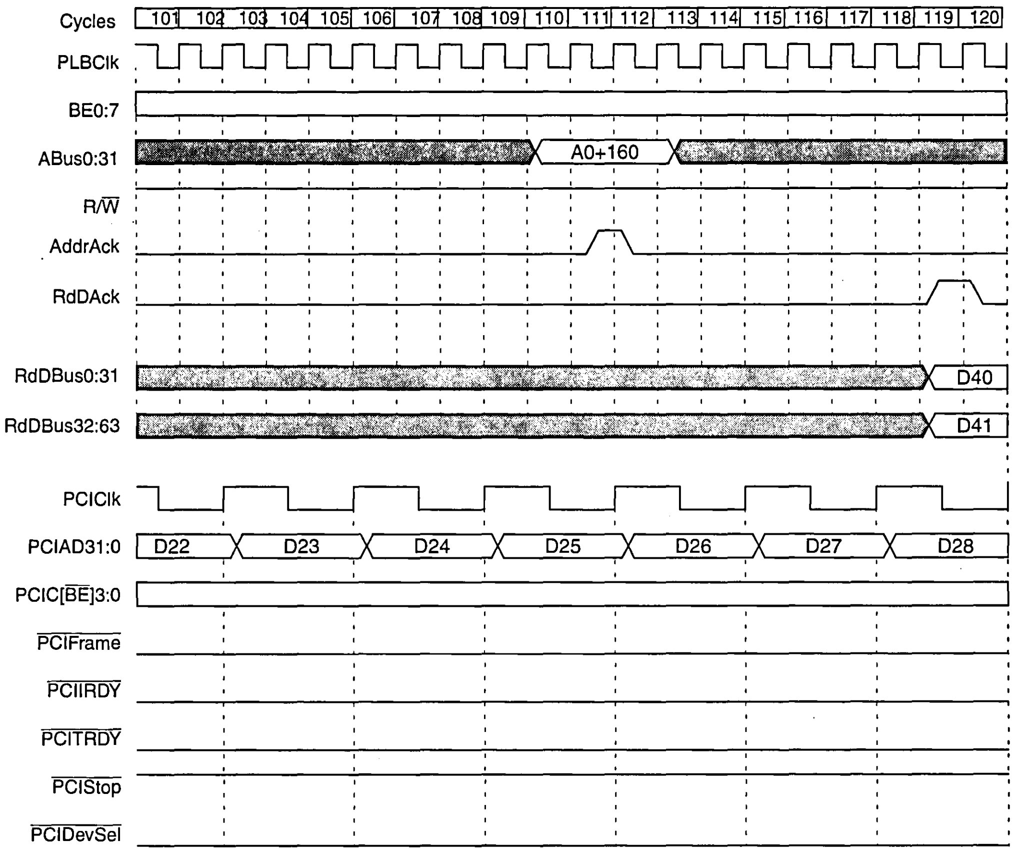


Figure 17-67. PCI Master Burst Read From SDRAM (Continued)

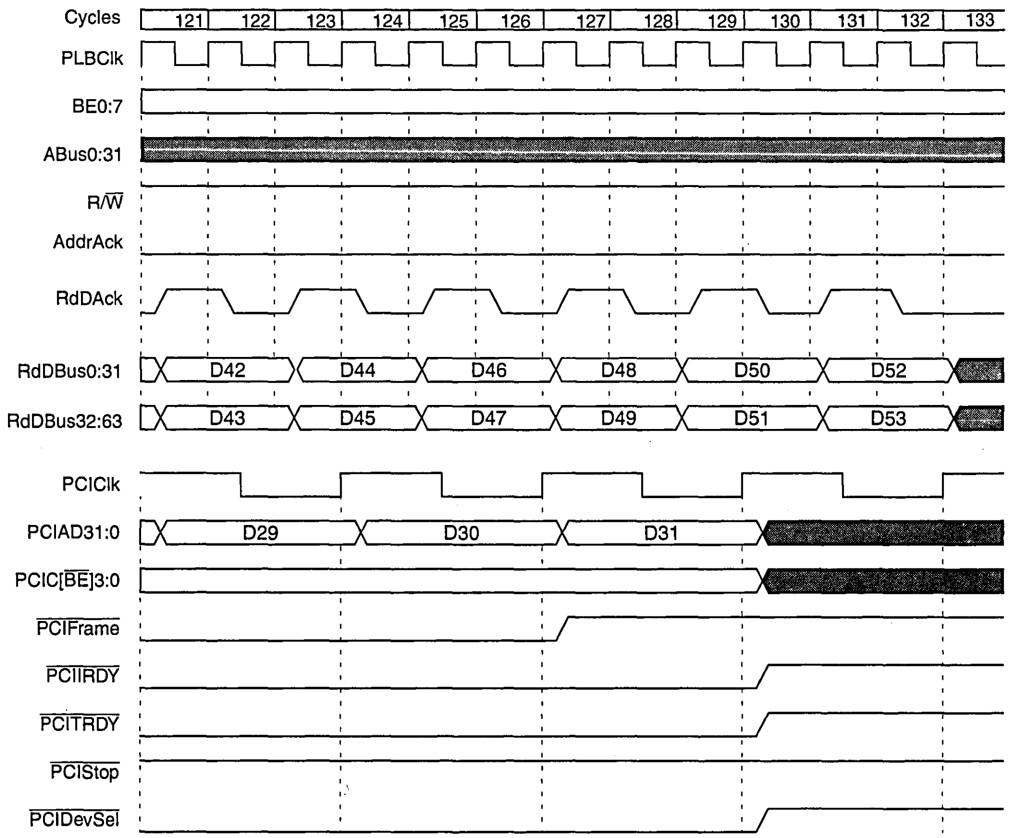


Figure 17-67. PCI Master Burst Read From SDRAM (Continued)

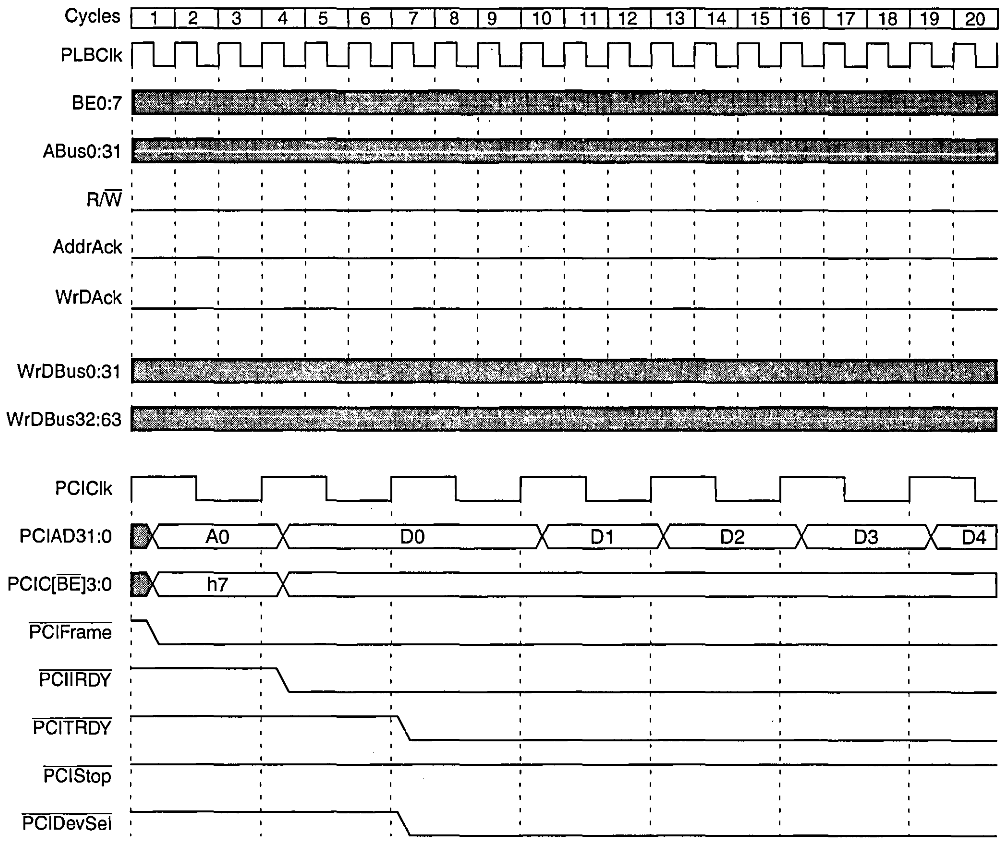


Figure 17-68. PCI Master Burst Write To SDRAM

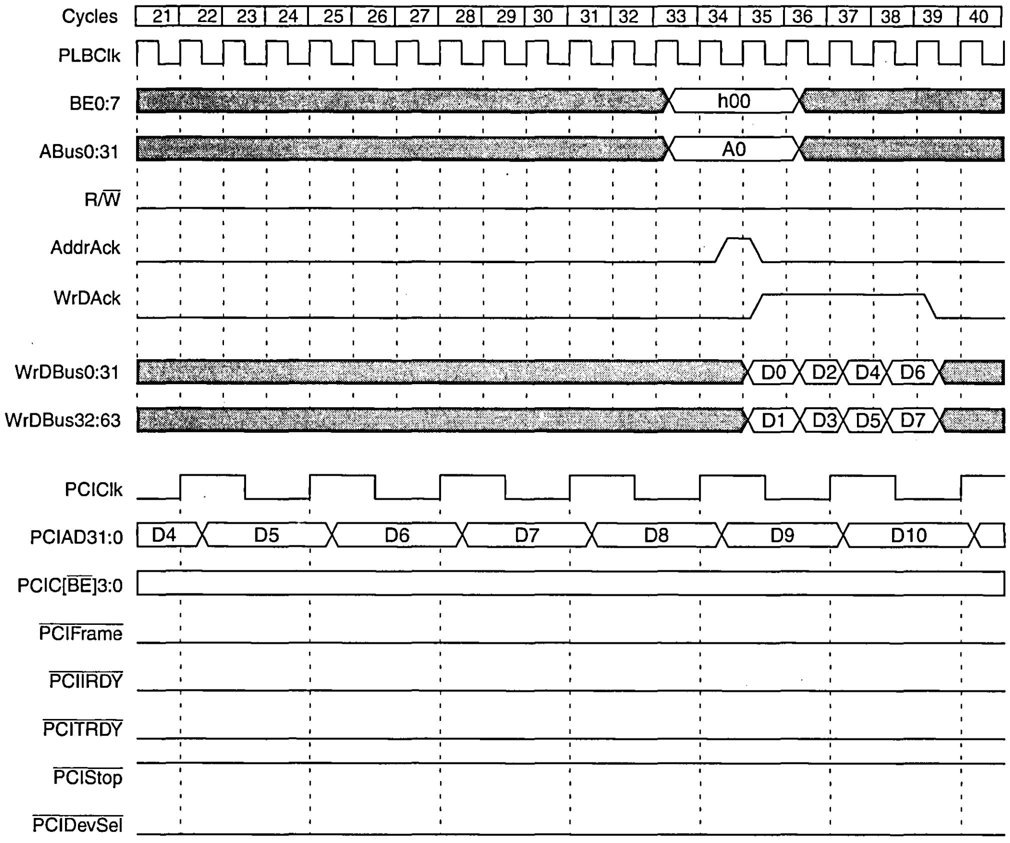


Figure 17-68. PCI Master Burst Write To SDRAM (Continued)

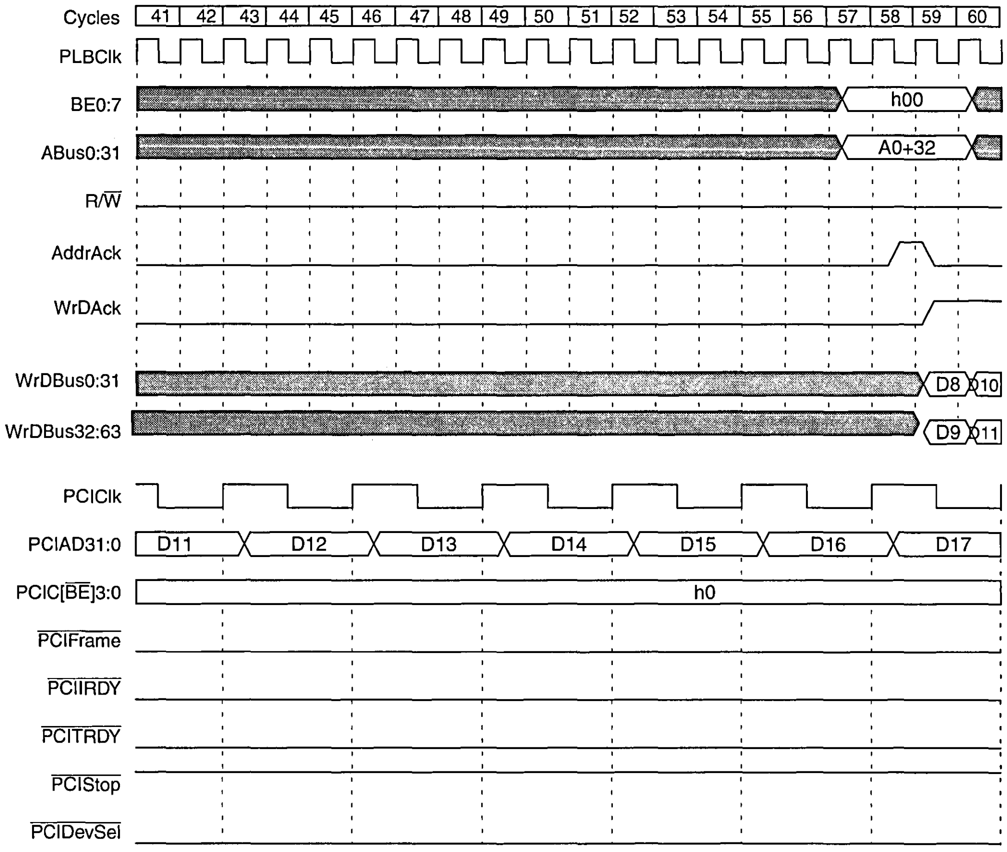


Figure 17-68. PCI Master Burst Write To SDRAM (Continued)

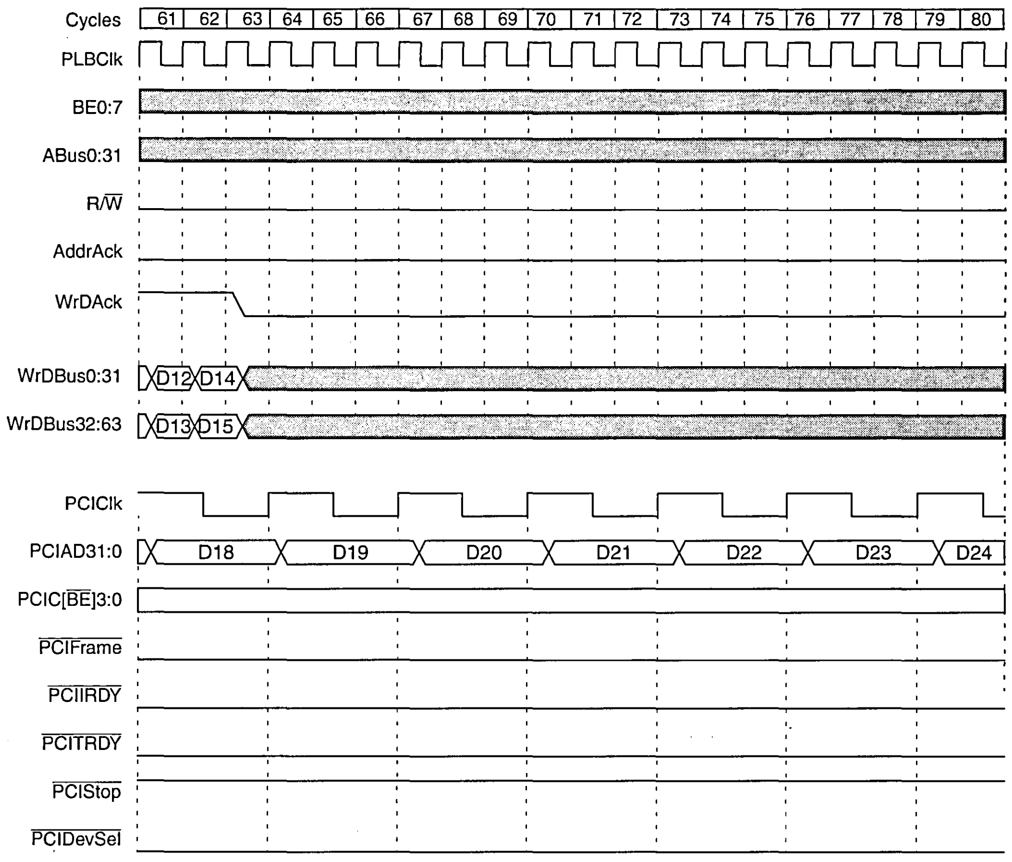


Figure 17-68. PCI Master Burst Write To SDRAM (Continued)

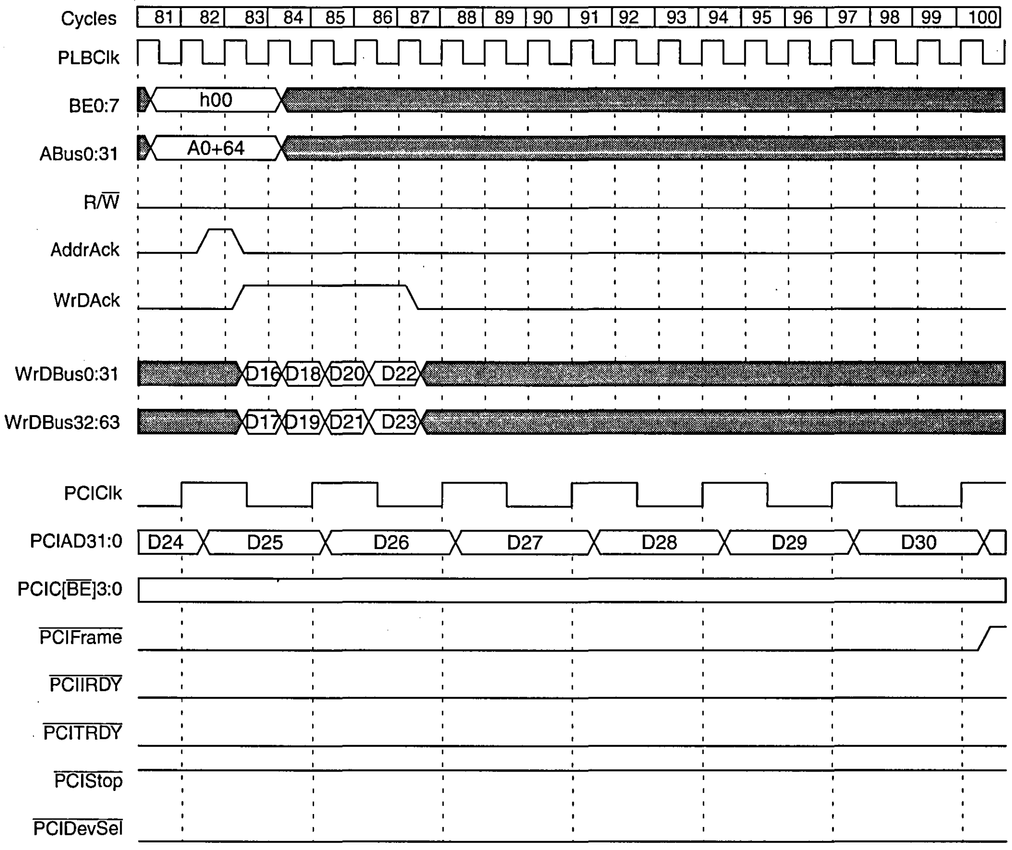


Figure 17-68. PCI Master Burst Write To SDRAM (Continued)

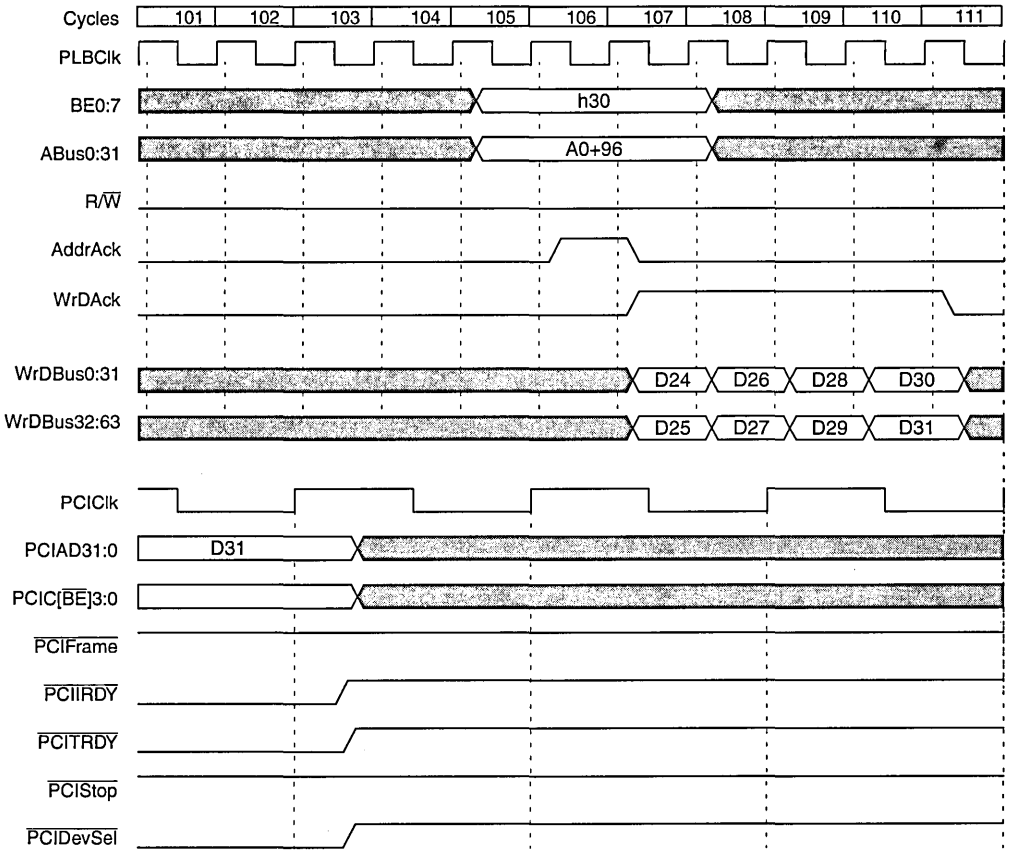


Figure 17-68. PCI Master Burst Write To SDRAM (Continued)

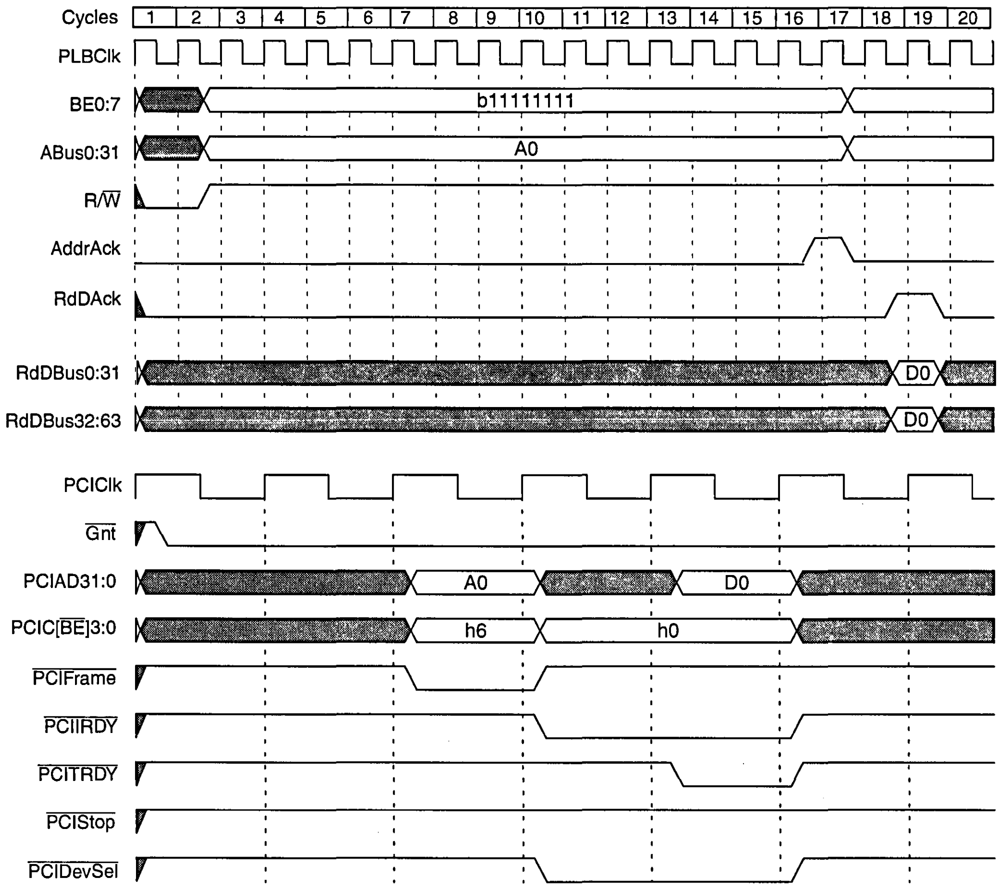


Figure 17-69. CPU Read From PCI Memory Slave, Nonprefetching

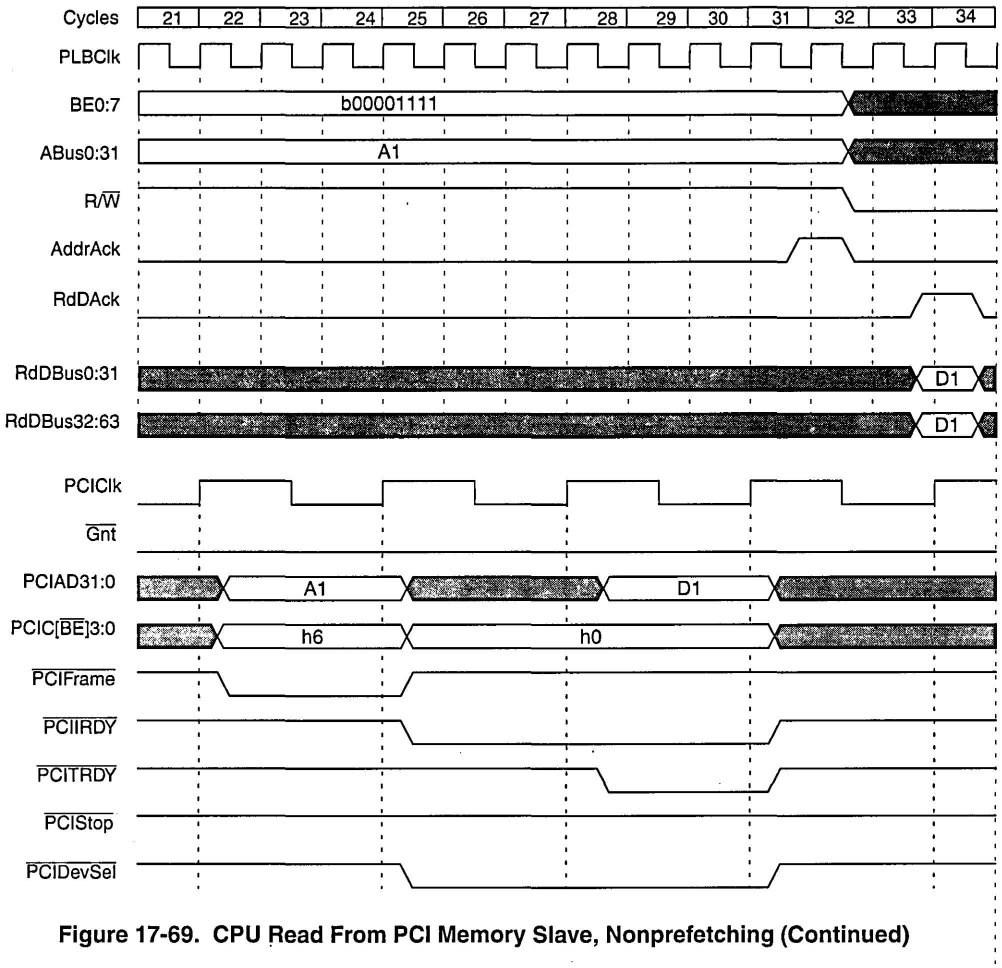


Figure 17-69. CPU Read From PCI Memory Slave, Nonprefetching (Continued)

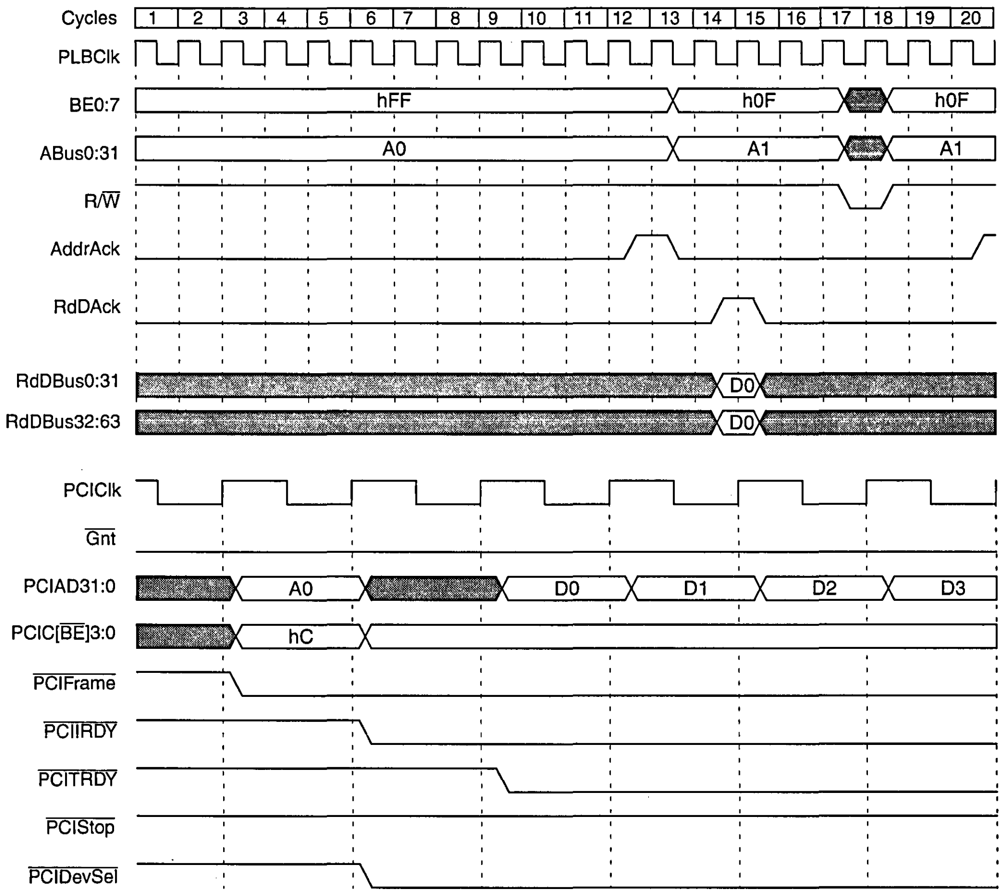


Figure 17-70. CPU Read From PCI Memory Slave, Prefetching

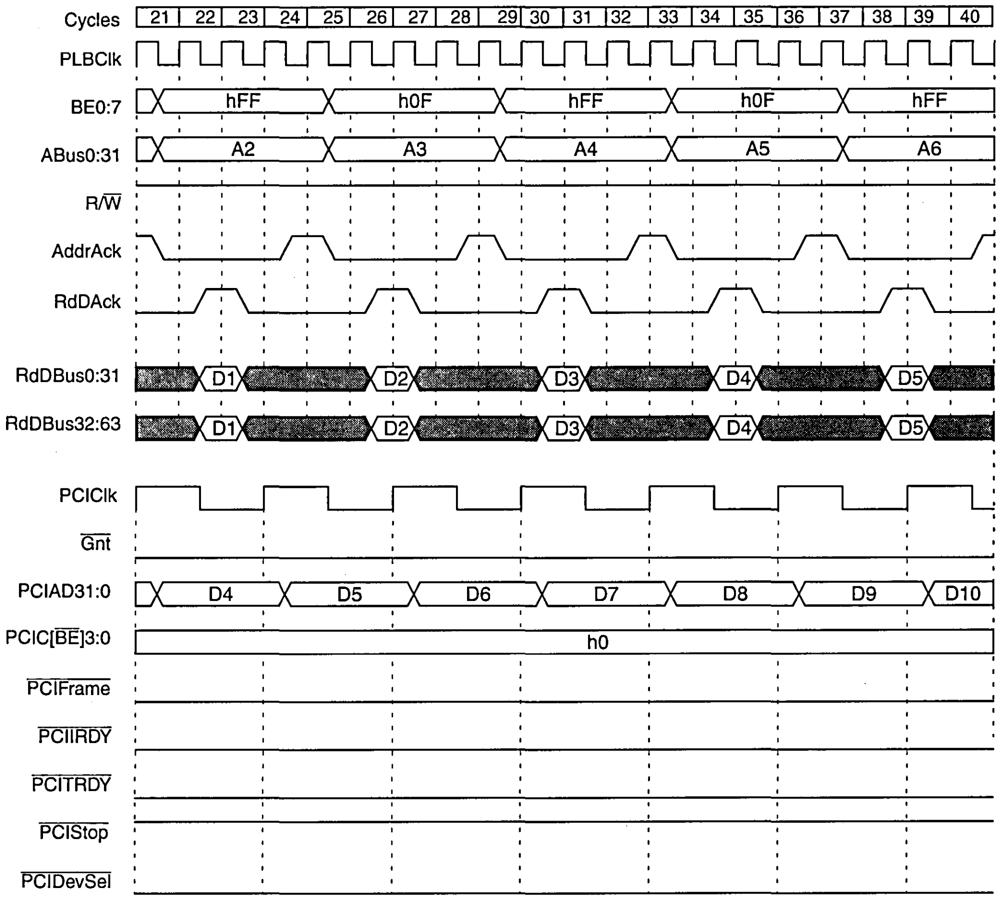


Figure 17-70. CPU Read From PCI Memory Slave, Prefetching (Continued)

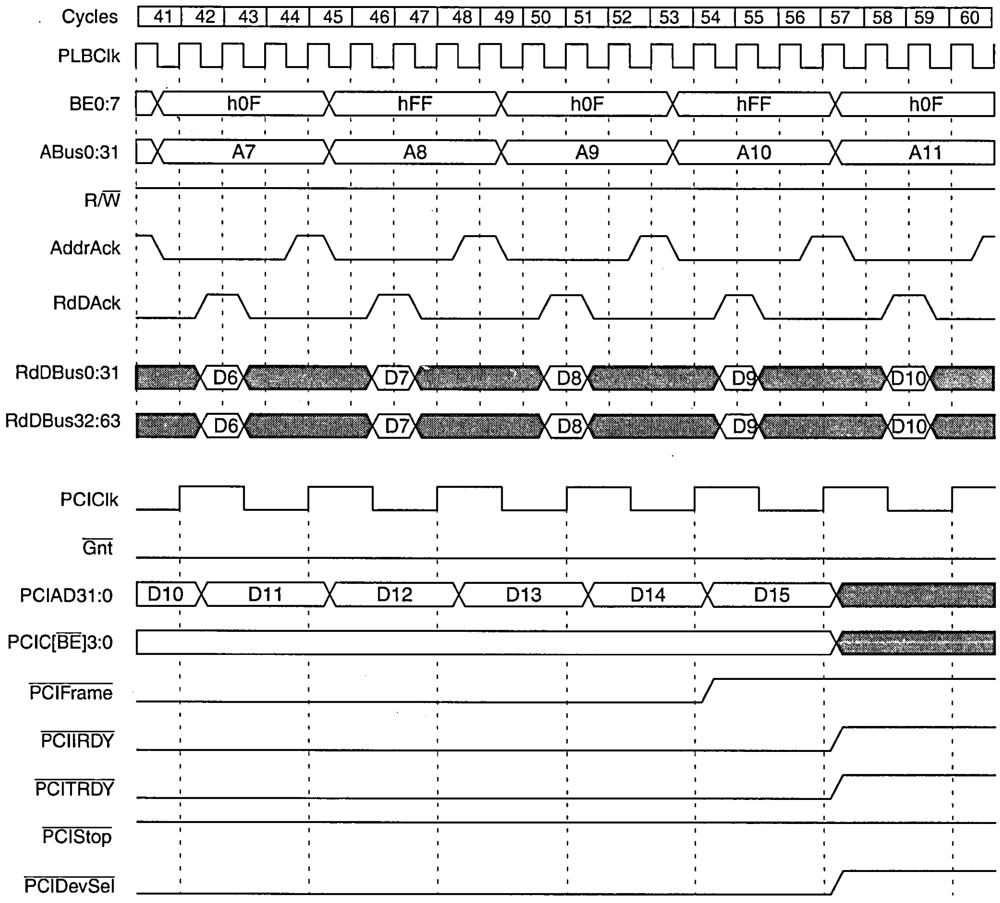


Figure 17-70. CPU Read From PCI Memory Slave, Prefetching (Continued)

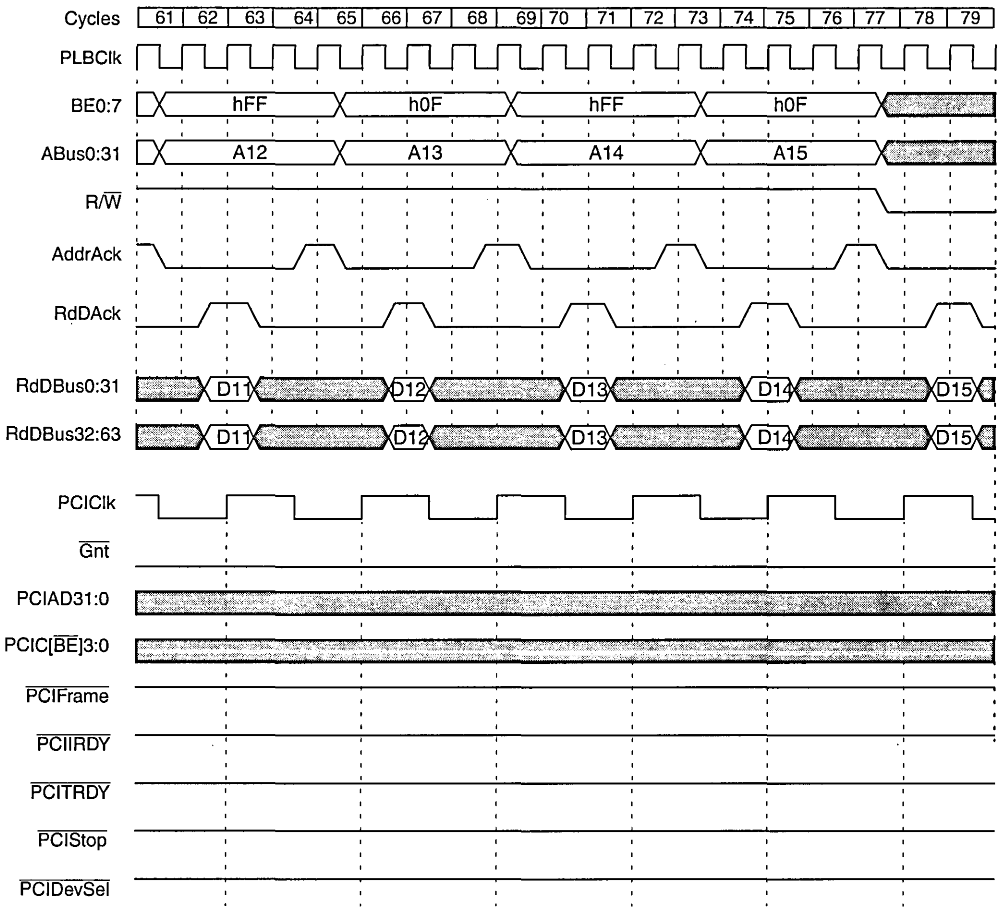


Figure 17-70. CPU Read From PCI Memory Slave, Prefetching (Continued)

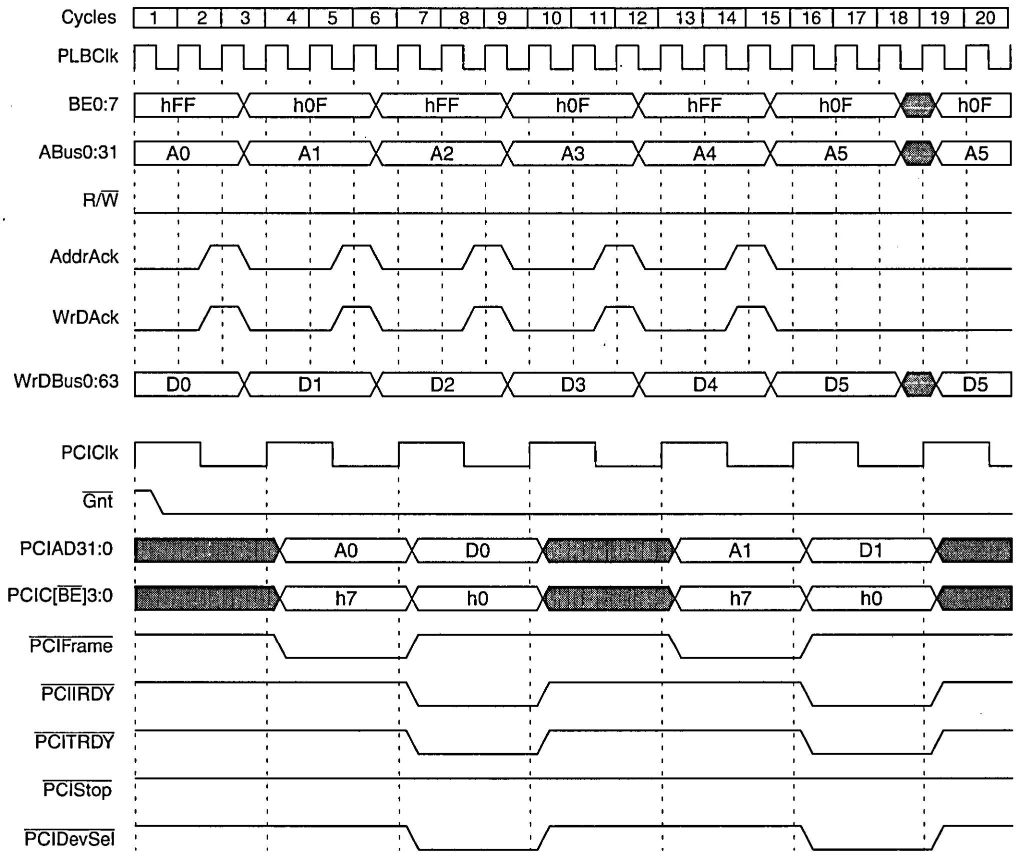


Figure 17-71. CPU Write To PCI Memory Slave

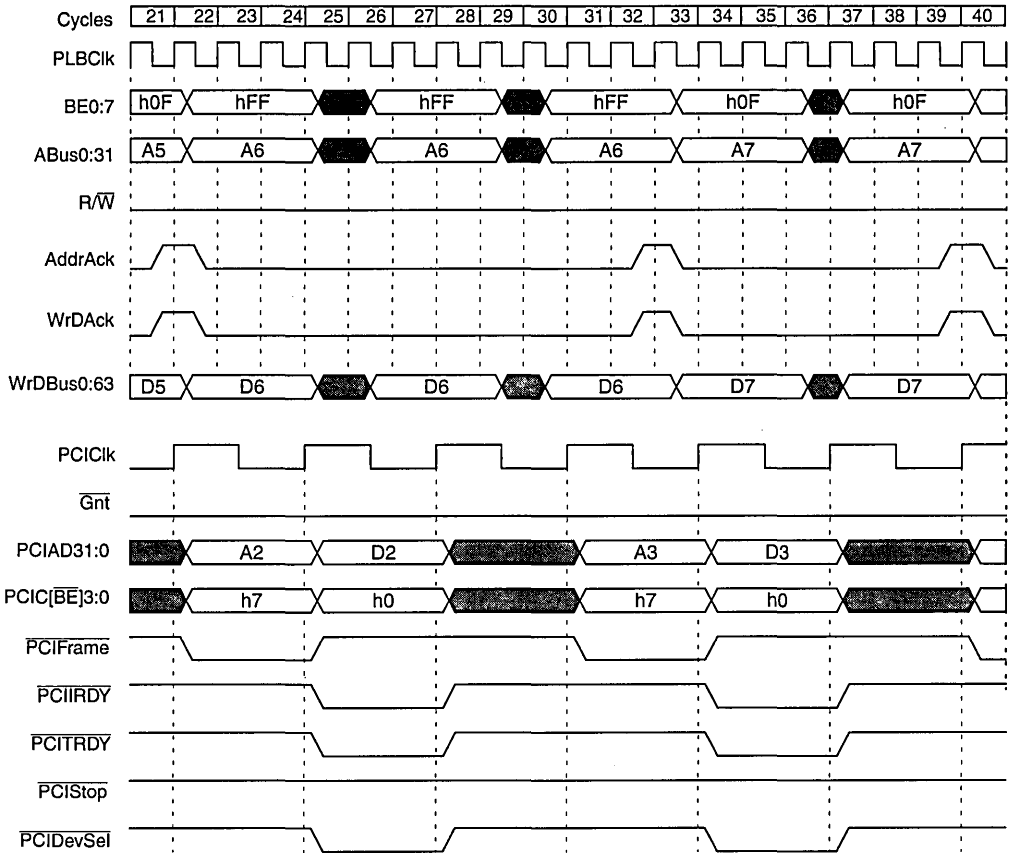


Figure 17-71. CPU Write To PCI Memory Slave (Continued)

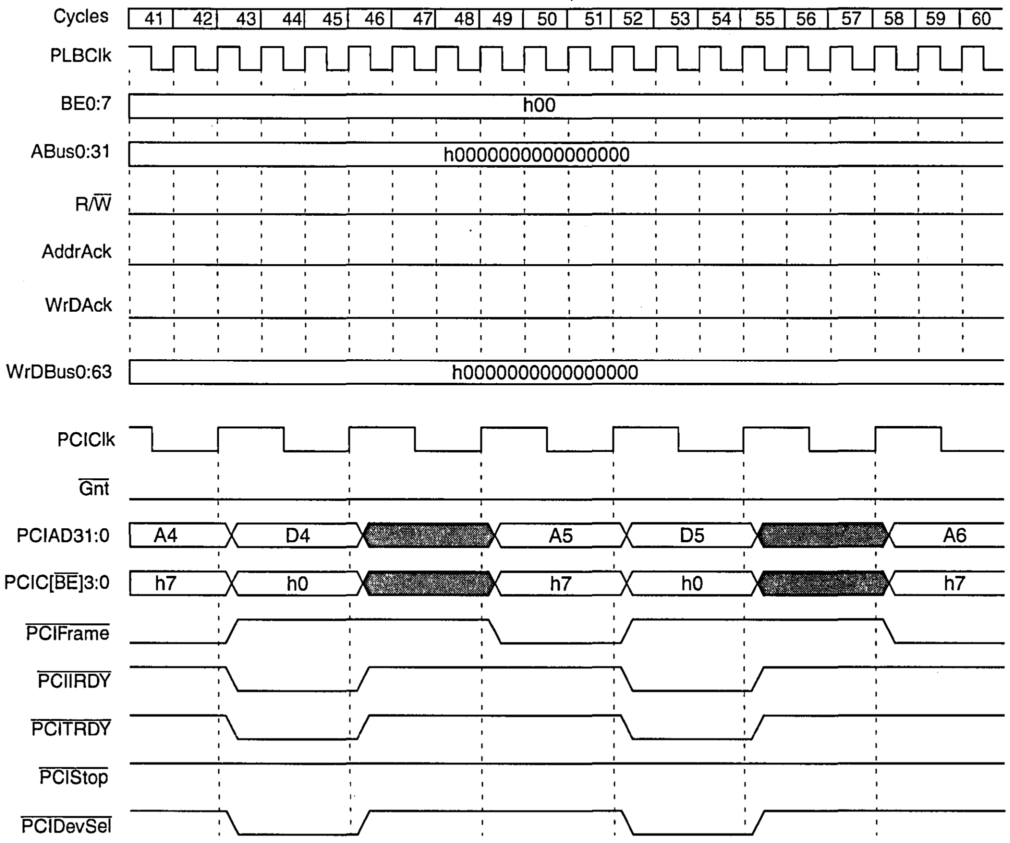


Figure 17-71. CPU Write To PCI Memory Slave (Continued)

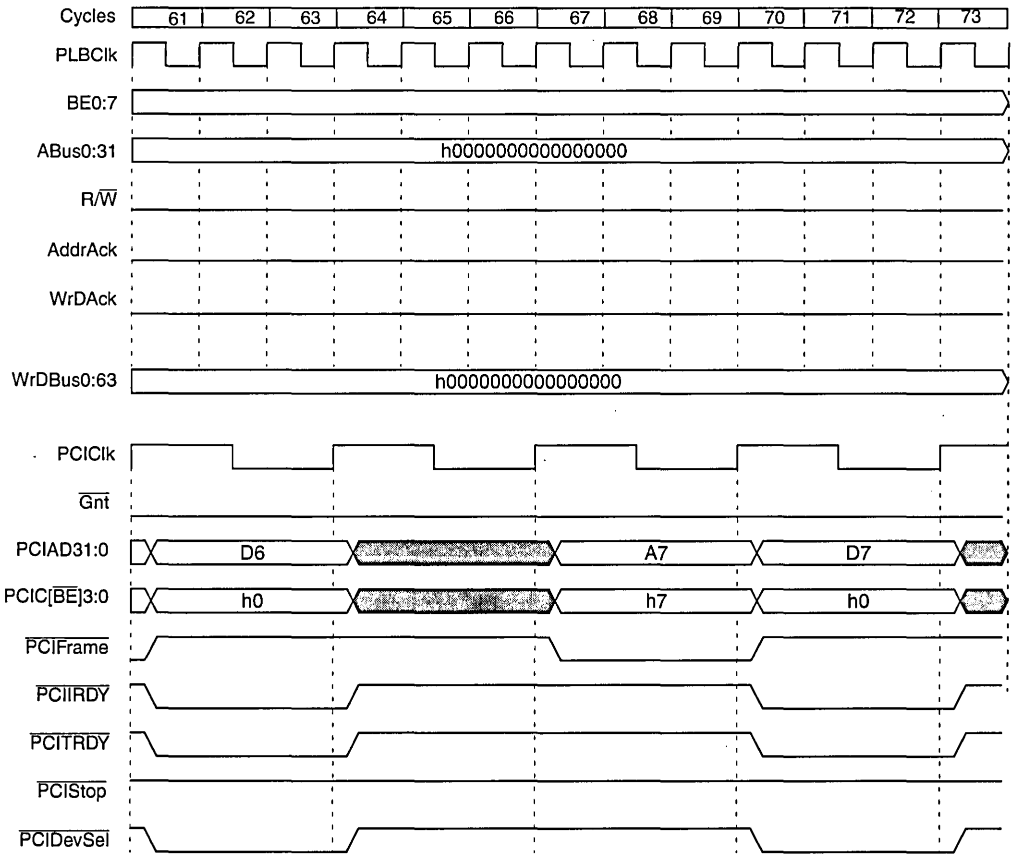


Figure 17-71. CPU Write To PCI Memory Slave (Continued)

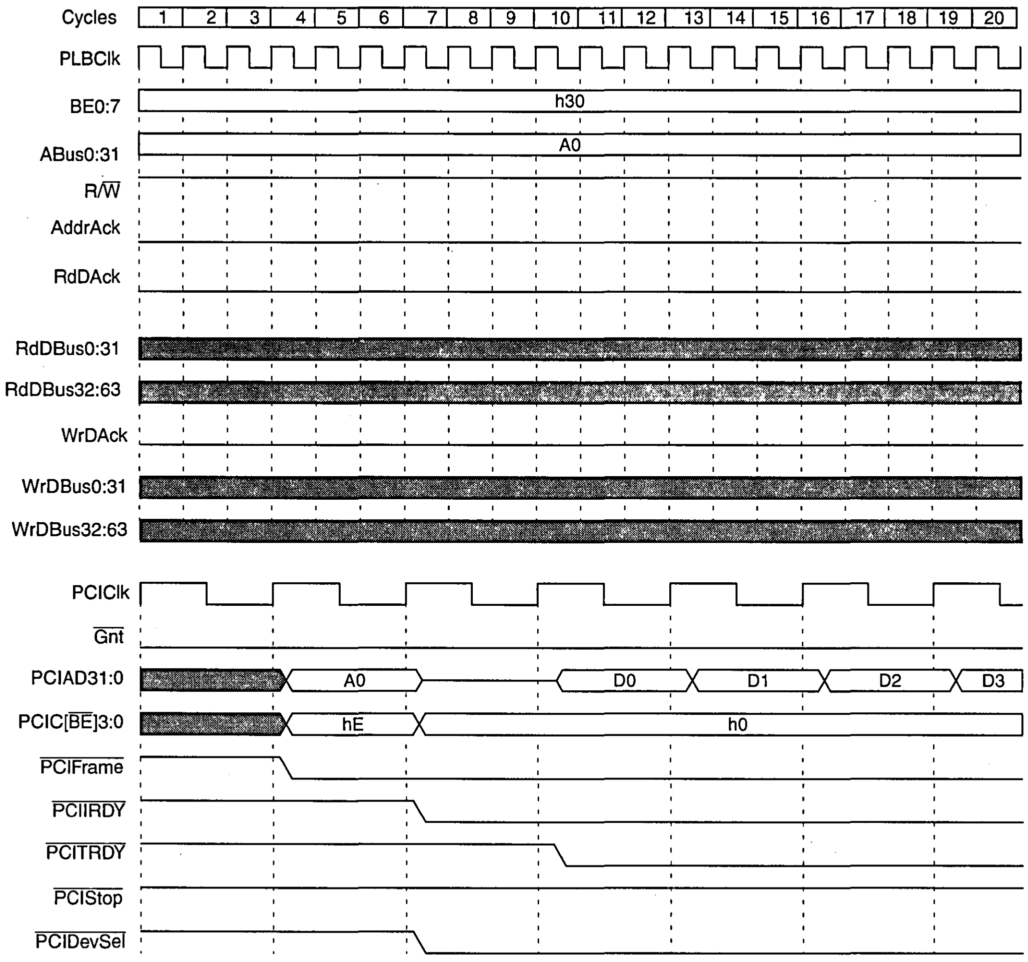


Figure 17-72. PCI Memory To SDRAM DMA Transfer

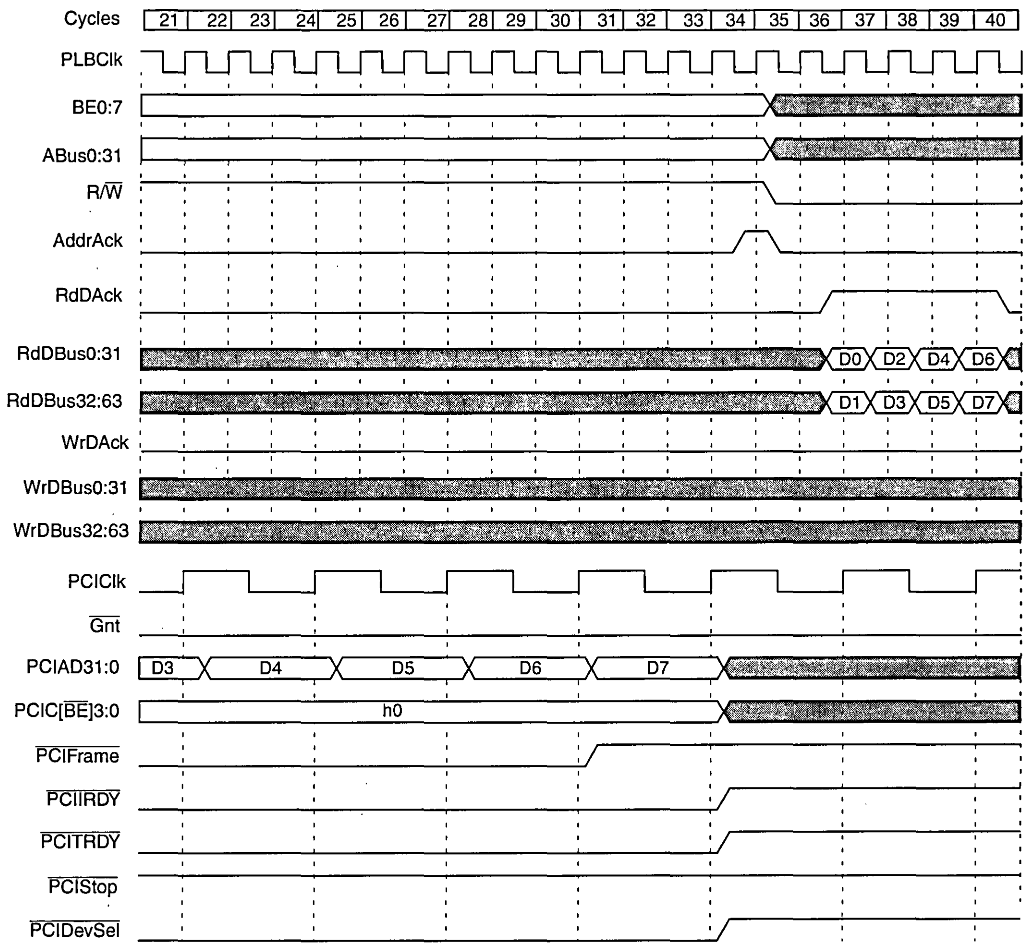


Figure 17-72. PCI Memory To SDRAM DMA Transfer (Continued)

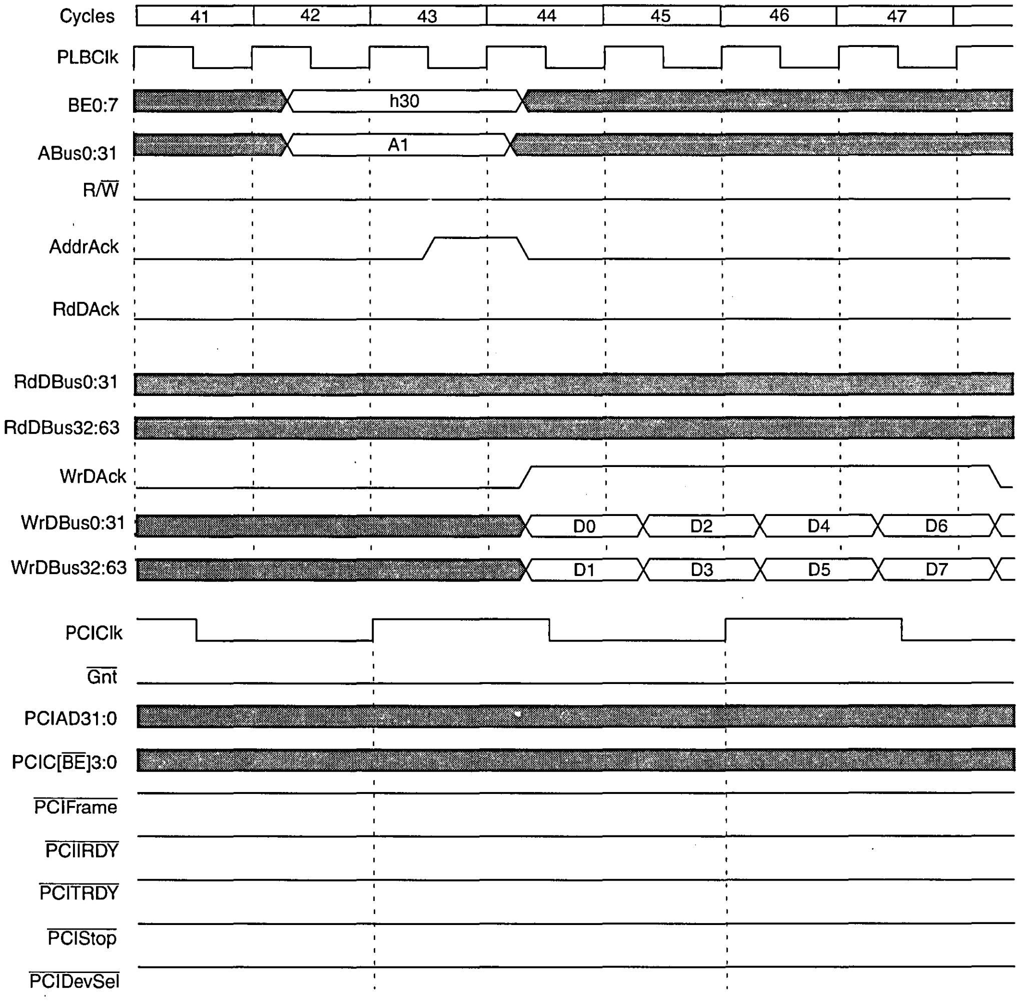


Figure 17-72. PCI Memory To SDRAM DMA Transfer (Continued)

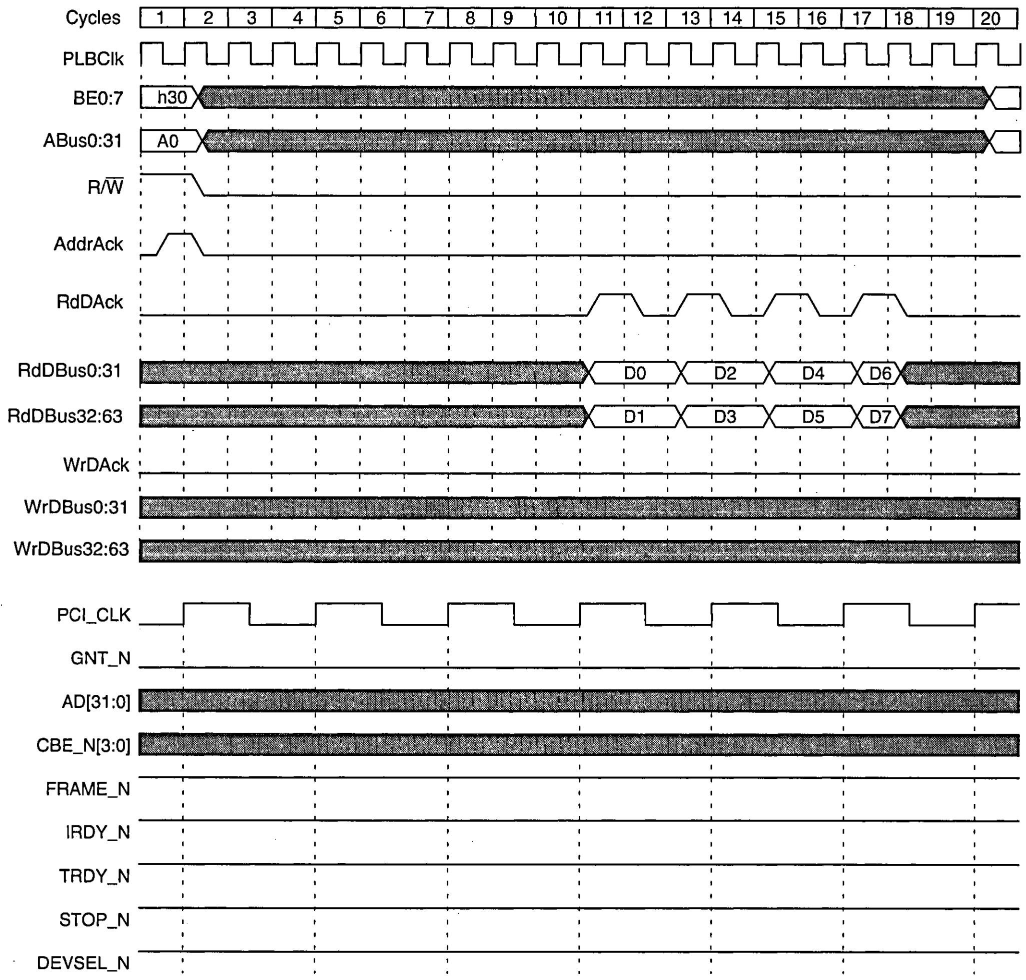


Figure 17-73. SDRAM To PCI Memory DMA Transfer

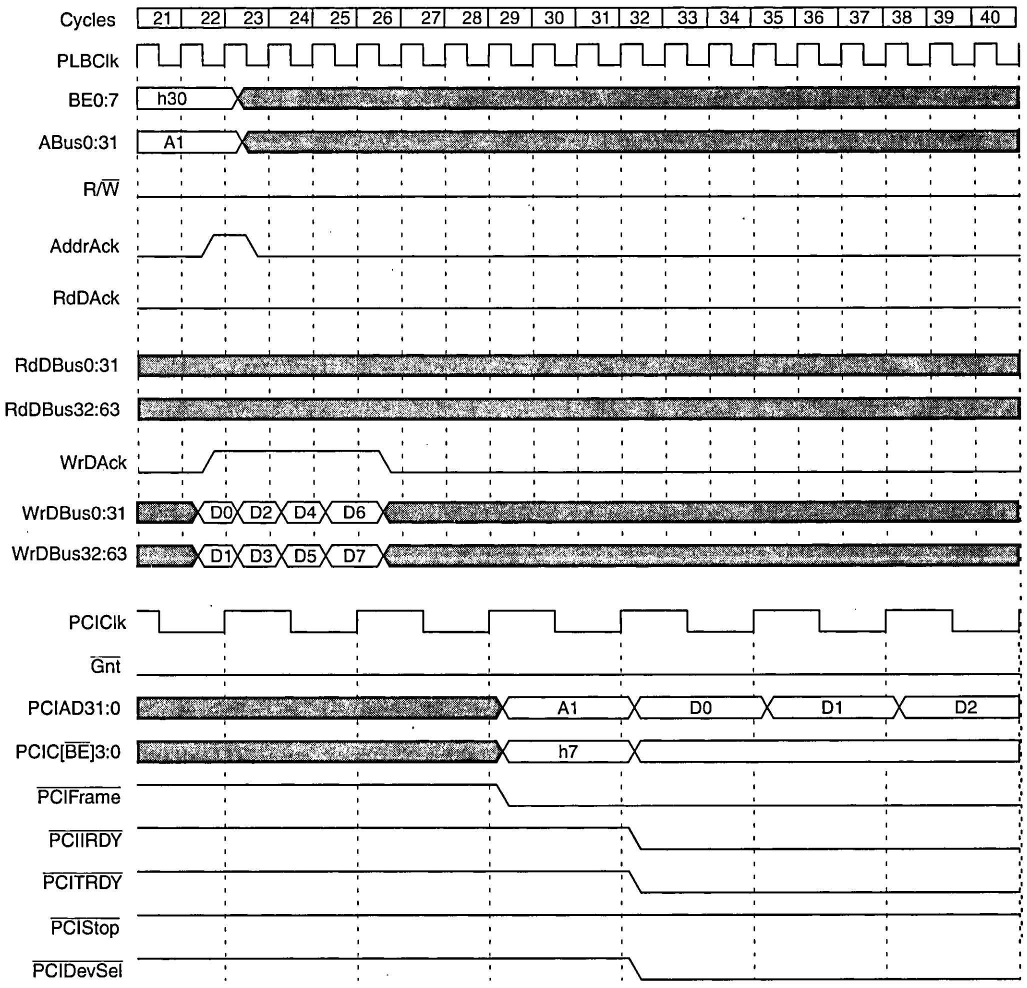


Figure 17-73. SDRAM To PCI Memory DMA Transfer (Continued)

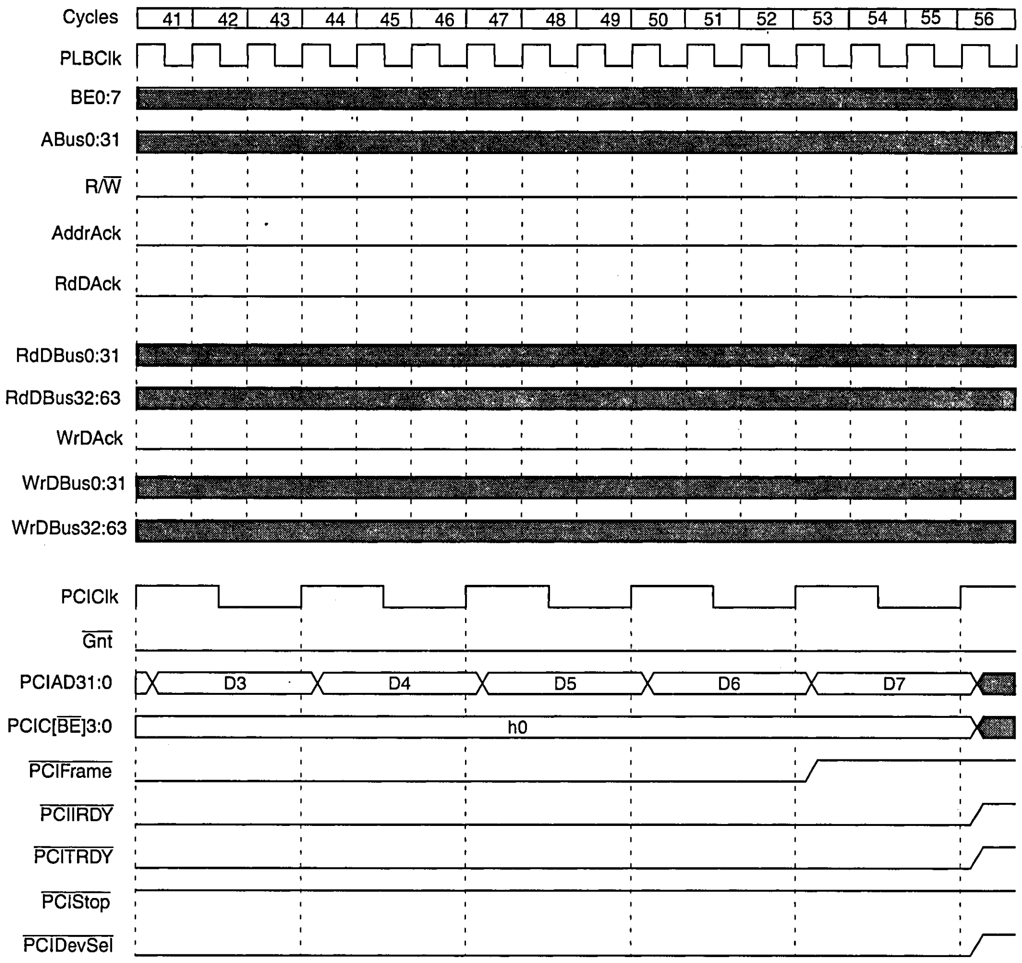


Figure 17-73. SDRAM To PCI Memory DMA Transfer (Continued)

Chapter 18. Direct Memory Access Controller

The Direct Memory Access (DMA) controller is a Processor Local Bus (PLB) and On-chip Peripheral Bus (OPB) master which supports the autonomous transfer of data between memory and peripherals and from memory-to-memory. The controller provides four DMA channels, each of which has an independent set of configuration registers. Each channel has its own control, source address, destination address, count, and scatter/gather address registers. Once these registers are programmed by the PPC405 processor, the DMA controller performs the requested data transfer without the need for processor intervention.

The four DMA channels also support scatter/gather transfers. During a scatter/gather transfer the configuration registers for a particular DMA channel are automatically loaded from a data structure in memory instead of being individually programmed. Since the scatter/gather address register is updated in this process, the channel can optionally reconfigure itself for another transfer when the current one completes.

As master on both the PLB and OPB the DMA controller can read and write any address accessible by the PPC405 processor. This includes memory and memory-mapped peripherals on the EBC interface, SDRAM memory and PCI addresses that have been mapped into PLB address space. The DMA controller can also service DMA peripherals attached to the EBC via the DMAReqn, DMAAckn and EOTn[TCn] I/Os, along with the OPB-attached UART0.

18.1 External Interface Signals

Figure 18-1 illustrates the external I/Os associated with the DMA controller and the EBC I/Os used during external peripheral transfers. External peripheral and EBC device-paced memory transfers request service from the DMA controller by driving a DMA request line (DMAReq0-3) active. For peripheral mode transfers the DMA controller acknowledges the request and transfers data by asserting a DMA acknowledge signal (DMAAck0-3). In contrast, an EBC device-paced memory-to-memory transfer occurs when the chip select ($\overline{\text{PerCSn}}$) associated with the memory location is driven active. The timing of $\overline{\text{PerCSn}}$ and the other EBC I/Os is determined by the Bank Access Parameter Register (EBC0_BnAP) for the particular memory location. See Chapter 16, "External Bus Controller" for details on EBC configuration and timings.

Table 18-1. DMA Controller External I/Os

Signal	Usage
DMAAckn	DMA Request, used to request either a peripheral mode transfer or an EBC device-paced memory-to-memory transfer.
DMAAckn	DMA Acknowledge, instructs an EBC-attached DMA peripheral to transfer data.
EOTn[TCn]	End of Transfer or Terminal Count. Used to stop the channel when programmed as EOT. When configured as TC, goes active when the channel transfer count register (DMA0_CTn) reaches zero.

Note 1: The active level (polarity) of DMAReqn, DMAAckn, and EOTn[TCn] are individually programmable. See "DMA Polarity Configuration Register (DMA0_POL)" on page 18-5.

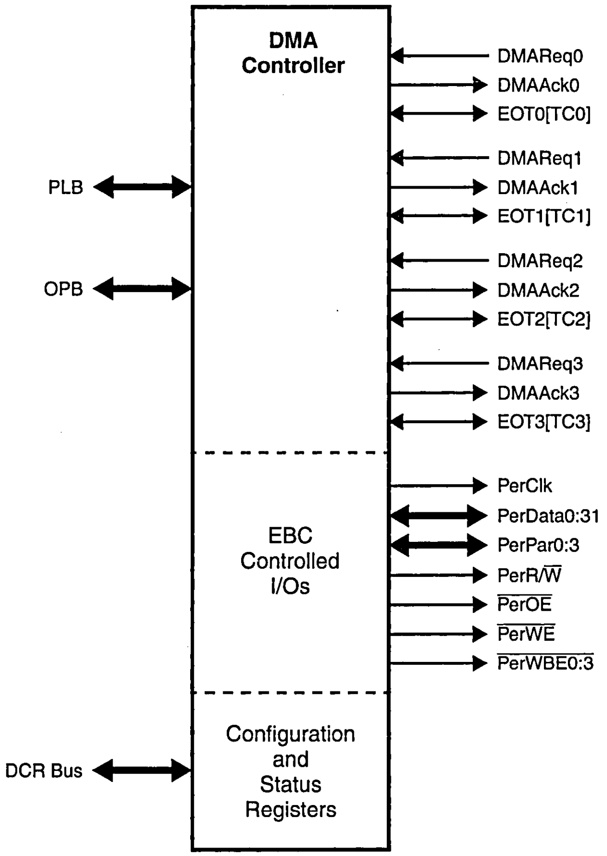


Figure 18-1. DMA Controller External Bus Control Signals

18.2 Functional Overview

As a specialized controller, the DMA unit provides system designers and programmers with a highly efficient method of moving data. During any DMA transfer the controller always buffers data read from the source prior to writing the data to the destination. Since many buses, including the internal PLB and the SDRAM interface, provide substantially better performance when bursting data, the DMA controller includes a 32-byte (4 doubleword) buffer. This buffer is enabled on a per-channel basis by setting DMA0_CRn[BEN] and serves to minimize the number of discrete memory transactions. Each of the four DMA channels is configurable for either peripheral or memory-to-memory transfers.

18.2.1 Peripheral Mode Transfers

Peripherals are either devices attached to the EBC interface via the DMAReqn and DMAAckn lines, or the internal serial port (UART0). Memory is any address accessible from the PLB, including PLB-mapped PCI address space, SDRAM memory, and memory and memory-mapped devices on the peripheral bus. During a peripheral mode transfer the peripheral requests a DMA transfer by asserting a DMA request line. For UART0, this signal is internal to the PPC405GP, while external peripherals use one of the DMAReqn lines. When the requesting channel has the highest priority of any active channel, the requesting device receives a DMA acknowledge. In the case of external peripherals, the appropriate DMAAckn line is driven to the active state, with the timing specified in the DMA Control Register for the channel (DMA0_CRn).

There are two types of peripheral mode transfers: peripheral-to-memory and memory-to-peripheral. A peripheral-to-memory transfer reads data from a DMA device, while a memory-to-peripheral transfer writes data. In both cases, the peripheral interface never bursts and data is transferred at the width of the peripheral. If the DMA buffer is disabled for the active channel (DMA0_CRn[BEN]=0), each peripheral transfer causes a corresponding memory operation.

When buffering is enabled during a peripheral-to-memory transfer, data is collected until the 32-byte buffer is full, the peripheral deasserts DMAReqn, a higher priority DMA request becomes pending, or the channel completes. The buffer contents are then written to the target memory as efficiently as possible. If the initial programming of the channel's destination address register (DMA0_DAn) is 32-byte aligned, the buffer is emptied in one burst operation to the target memory.

Memory-to-peripheral transfers differ since the amount of data that will be requested by the peripheral is unknown. If the DMA buffer is disabled (DMA0_CRn[BEN]=0) a discrete source memory read occurs for each element in the DMA transfer. Since this is inefficient, the buffer should only be disabled for low data rate transfers or when the source memory is FIFO-like, and reads are therefore destructive. When the 32-byte buffer is enabled the controller uses the setting in DMA0_CRn[PF] to prefetch 1, 2, or 4 64-bit doublewords from the source memory. The DMA controller provides data from the buffer until the peripheral deasserts its request, the channel is interrupted by one of higher priority, or the transfer completes. Whenever any of these conditions occurs any unused data in the DMA buffer is discarded.

18.2.2 Memory-to-Memory Transfers

The DMA controller can perform either device-paced (hardware-initiated) or software-initiated memory-to-memory transfers. Device-paced memory transfers function identically to peripheral mode transfers, except that a chip select ($\overline{\text{PerCSn}}$) serves as the DMA acknowledge instead of a DMAReqn output. As with peripheral mode transfers, bursts never occur on the device-paced side of the transaction.

Software-initiated memory-to-memory transfers between memories with fixed timings provide the best overall performance. During a software-initiated transfer the DMA controller knows the exact amount of data to be transferred. As a result, when the 32-byte DMA buffer is enabled (DMA0_CRn[BEN]=1) the controller uses bursts as much as possible. To ensure the highest bandwidth, source and destination addresses should be aligned on 32-byte boundaries.

There are three cases that limit the ability to burst during software-initiated memory-to-memory transfers. Bursting is not possible from the source memory when the source address increment is zero (DMA0_CRn[SAI]=0). Similarly, the DMA controller does not burst to the destination when the destination address increment is zero (DMA0_CRn[DAI]=0). Finally, the DMA controller cannot burst to or from any address that maps to a device-paced (EBC0_BnAP[RE]=1) EBC chip select ($\overline{\text{PerCSn}}$).

18.2.3 Scatter/Gather Transfers

Each of the four DMA channels supports scatter/gather transfers. This scatter/gather capability allows the chaining of multiple DMA controller operations within a channel. During a normal DMA operation software must program the control, source address, destination address, and count registers for each transfer. Scatter/gather transfers differ in that these registers are automatically loaded from a linked list data structure in system memory. When a channel completes one transfer the DMA controller loads the next set of configuration values into the channel's registers and the channel continues with the new programmings.

18.3 Configuration and Status Registers

Table 18-2 on page 18-4 lists the DMA configuration and status registers, each of which is accessed using the PowerPC **mtdcr** and **mfdcr** instructions. As example, the following PowerPC assembly code writes the control register for DMA channel 0 and then reads the DMA status register:

```
#define DMA0_CR0  0x100
#define DMA0_SR   0x120

    mtdcr     DMA0_CR0,r3                ! write r3 to channel 0 control register
    mfdcr     r4,DMA0_SR                 ! read contents of status register into r4
```

The DMA configuration and status registers are readable at any time. However, since each register read requires a separate operation, it is not possible to guarantee that the values read from multiple registers correspond to a state that ever existed in the DMA controller. To illustrate, consider software that reads the destination address for channel 0 (DMA0_DA0) and then the count for channel 0 (DMA0_CT0). If the DMA controller updates the count between these two operations, the values read differ from what is expected.

While reads can occur at any time, software must not write the configuration registers for any channel that is currently enabled (DMA0_CRn[CE]=1). The only exception is that a channel may be disabled by reading the channel control register, clearing the channel enable bit, and then writing the new value to the control register. Once a channel is disabled, all of its configuration registers may be reprogrammed as desired.

Table 18-2. DMA Controller Configuration and Status Registers

Mnemonic	DCR Address	Access	Description	Page
DMA0_CR0	0x100	R/W	DMA Channel Control Register 0	18-8
DMA0_CT0	0x101	R/W	DMA Count Register 0	18-11
DMA0_DA0	0x102	R/W	DMA Destination Address Register 0	18-11
DMA0_SA0	0x103	R/W	DMA Source Address Register 0	18-10
DMA0_SG0	0x104	R/W	DMA Scatter/Gather Descriptor Address Register 0	18-12
DMA0_CR1	0x108	R/W	DMA Channel Control Register 1	18-8
DMA0_CT1	0x109	R/W	DMA Count Register 1	18-8
DMA0_DA1	0x10A	R/W	DMA Destination Address Register 1	18-11
DMA0_SA1	0x10B	R/W	DMA Source Address Register 1	18-10
DMA0_SG1	0x10C	R/W	DMA Scatter/Gather Descriptor Address Register 1	18-12
DMA0_CR2	0x110	R/W	DMA Channel Control Register 2	18-8
DMA0_CT2	0x111	R/W	DMA Count Register 2	18-11

Table 18-2. DMA Controller Configuration and Status Registers (continued)

Mnemonic	DCR Address	Access	Description	Page
DMA0_DA2	0x112	R/W	DMA Destination Address Register 2	18-11
DMA0_SA2	0x113	R/W	DMA Source Address Register 2	18-10
DMA0_SG2	0x114	R/W	DMA Scatter/Gather Descriptor Address Register 2	18-12
DMA0_CR3	0x118	R/W	DMA Channel Control Register 3	18-8
DMA0_CT3	0x119	R/W	DMA Count Register 3	18-11
DMA0_DA3	0x11A	R/W	DMA Destination Address Register 3	18-11
DMA0_SA3	0x11B	R/W	DMA Source Address Register 3	18-10
DMA0_SG3	0x11C	R/W	DMA Scatter/Gather Descriptor Address Register 3	18-12
DMA0_SR	0x120	R/Clear	DMA Status Register	18-7
DMA0_SGC	0x123	R/W	DMA Scatter/Gather Command Register	18-13
DMA0_SLP	0x125	R/W	DMA Sleep Mode Register	18-6
DMA0_POL	0x126	R/W	DMA Polarity Configuration Register	18-5

18.3.1 DMA Polarity Configuration Register (DMA0_POL)

The Polarity Configuration Register (DMA0_POL) is used to set the polarity (active state) of the external DMA I/O signals: DMAReqn, DMAAckn, and EOTn[TCn]. As shown in Figure 18-2, if a bit in DMA0_POL is zero, the corresponding signal is active high, otherwise the signal is active low.

Whenever any of the EOT polarities are changed (DMA0_POL[EnP), software must subsequently clear the corresponding EOT status bits in the DMA Status Register (DMA0_SR[TS0:3]) prior to enabling the associated DMA channel. This is necessary to prevent a channel from being disabled because of an incorrect EOT status stored in the status bits.

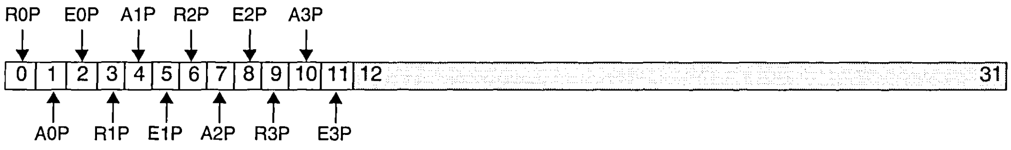


Figure 18-2. DMA Polarity Configuration Register (DMA0_POL)

0	R0P	DMAReq0 Polarity 0 DMAReq0 is active high 1 DMAReq0 is active low
1	A0P	DMAAck0 Polarity 0 DMAAck0 is active high 1 DMAAck0 is active low
2	E0P	EOT0[TC0] Polarity 0 EOT0[TC0] is active high 1 EOT0[TC0] is active low
3	R1P	DMAReq1 Polarity 0 DMAReq1 is active high 1 DMAReq1 is active low

4	A1P	DMAAck1 Polarity 0 DMAAck1 is active high 1 DMAAck1 is active low
5	E1P	EOT1[TC1] Polarity 0 EOT1[TC1] is active high 1 EOT1[TC1] is active low
6	R2P	DMAReq2 Polarity 0 DMAReq2 is active high 1 DMAReq2 is active low
7	A2P	DMAAck2 Polarity 0 DMAAck2 is active high 1 DMAAck2 is active low
8	E2P	EOT2[TC2] Polarity 0 EOT2[TC2] is active high 1 EOT2[TC2] is active low
9	R3P	DMAReq3 Polarity 0 DMAReq3 is active high 1 DMAReq3 is active low
10	A3P	DMAAck3 Polarity 0 DMAAck3 is active high 1 DMAAck3 is active low
11	E3P	EOT3[TC3] Polarity 0 EOT3[TC3] is active high 1 EOT3[TC3] is active low
12:31	▨	Reserved

18.3.2 DMA Sleep Mode Register (DMA0_SLP)

The Sleep Mode Register (DMA0_SLP) enables the DMA controller to enter sleep (low-power) mode and programs the number of PLB clock cycles to wait when the controller is idle before going to sleep. The DMA controller only goes to sleep when no DMA channels are enabled and no configuration or status (DCR) register operations are in progress. Reading or writing any of the DMA control or status register awakens the DMA controller.

To enable sleep mode set DMA0_SLP[SME] and CPM0_ER[DMA]. When sleep mode is enabled and the DMA controller becomes idle the 10-bit idle timer begins counting down from the programmed value. Only the upper 5 bits of the idle counter are programmable; the lower 5 bits are hardcoded to 0b11111. Therefore, the minimum granularity of the idle timer is 32 PLB clock cycles. When the counter reaches zero, the controller is placed in sleep mode.

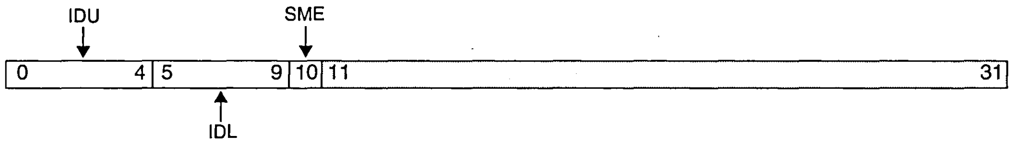


Figure 18-3. DMA Sleep Mode Register (DMA0_SLP)

0:4	IDU	Idle Timer Upper 0-31	Upper 5-bits of the idle timer.
5:9	IDL	Idle Timer Lower Hardcoded to 0b11111	Lower 5-bit portion of the idle timer. Writing this field has no effect.
10	SME	Sleep Mode Enable 0 Sleep disabled 1 Sleep enabled	If SME=1, also set CPM0_ER[DMA] to enable the Clock and Power Management macro to put the DMA controller to sleep.
11:31		Reserved	

18.3.3 DMA Status Register (DMA0_SR)

As shown in Figure 18-4, the DMA Status Register (DMA0_SR) provides status information for each of the DMA channels. Bits in DMA0_SR are set in hardware, and can be either read or cleared by software. Clearing is performed by writing a word to DMA0_SR containing a 1 in any bit position to be cleared and 0 in all other bit positions.

The terminal count status (DMA0_SR[CSn]), end of transfer status (DMA0_SR[TSn]) and error status (DMA0_SR[RIn]) must be cleared for a DMA channel to operate. If a scatter/gather operation generates an interrupt for any of the above conditions, the channel pauses until software clears the associated status field(s) in DMA0_SR.

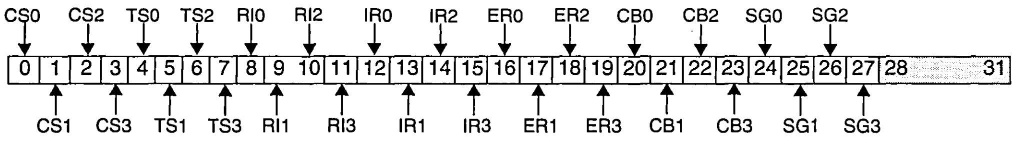


Figure 18-4. DMA Status Register (DMA0_SR)

0:3	CS[0:3]	Channel 0-3 Terminal Count Status 0 Terminal count has not occurred 1 Terminal count has been reached	Set when the transfer count reaches 0.
4:7	TS[0:3]	Channel 0-3 End of Transfer Status 0 End of transfer has not been requested 1 End of transfer has been requested	Only valid for channels with DMA0_CRn[ETD]=0.

8:11	RI[0:3]	Channel 0-3 Error Status 0 No error 1 Error occurred	See "Errors" on page 18-14 for more information.
12:15	IR[0:3]	Internal DMA Request 0 No internal DMA request pending 1 Internal DMA request is pending	
16:19	ER[0:3]	External DMA Request 0 No external DMA request pending 1 External DMA request is pending	
20:23	CB[0:3]	Channel Busy 0 Channel is idle 1 Channel currently active	
24:27	SG[0:3]	Scatter/Gather Status 0 No scatter/gather operation in progress 1 Scatter/gather operation in progress	
28:31		Reserved	

18.3.4 DMA Channel Control Registers (DMA0_CR0–DMA0_CR3)

The DMA Channel Control Registers (DMA0_CR0-DMA0_CR3) are used to configure and enable their respective DMA channels. Before a DMA channel can transfer data, the channel control, source address, destination address, and transfer count registers must be programmed. If a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1) the DMA channel control register is automatically loaded from memory. For additional details see "Scatter/Gather Transfers" on page 18-16.

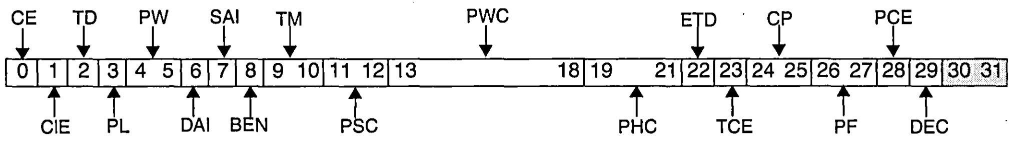


Figure 18-5. DMA Channel Control Registers (DMA0_CR0-DMA0_CR3)

0	CE	Channel Enable 0 Channel is disabled 1 Channel is enabled	This field is automatically cleared when the transfer completes or an error occurs.
1	CIE	Channel Interrupt Enable 0 Disable interrupts from this channel 1 Enable interrupts from this channel	When enabled, interrupts are generated for terminal count, end of transfer, and errors conditions. See "DMA Interrupts" on page 18-15.

2	TD	In peripheral mode: 0 Transfers are from memory-to-peripheral 1 Transfers are from peripheral-to-memory In device-paced memory-to-memory mode: 0 Peripheral is at the destination address 1 Peripheral is at the source address	TD is not used (don't care) for software-initiated memory-to-memory transfers.
3	PL	Peripheral Location 0 External peripheral (EBC) bus 1 OPB (UART0)	
4:5	PW	Peripheral Width/Memory alignment 00 Byte (8 bits) 01 Halfword (16 bits) 10 Word (32 bits) 11 Doubleword (64 bits) memory-to-memory transfers only	Transfer width equals peripheral width for peripherals.
6	DAI	Destination Address Increment 0 Do not increment destination address 1 After each data transfer increment the destination address by: 1, if the transfer width is a byte (8 bits) 2, if the transfer width is a halfword (16 bits) 4, if the transfer width is a word (32 bits) 8, if the transfer width is a doubleword (64 bit)	
7	SAI	Source Address Increment 0 Do not increment source address 1 After each data transfer increment the source address by: 1, if the transfer width is a byte (8 bits) 2, if the transfer width is a halfword (16 bits) 4, if the transfer width is a word (32 bits) 8, if the transfer width is a doubleword (64 bit)	
8	BEN	Buffer Enable 0 Disable DMA 32-byte buffer 1 Enable DMA 32-byte buffer	If BEN=0 discrete read and write operations occur for each data transfer.
9:10	TM	Transfer mode 00 Peripheral 01 Reserved 10 Software-initiated memory-to-memory 11 Device-paced memory-to-memory	
11:12	PSC	Peripheral Setup Cycles 0-3	Number of PerClk cycles that the EBC peripheral bus is idle from the last peripheral bus transaction to DMAAckn becoming active. Used only for the peripheral side of peripheral mode transfers.
13:18	PWC	Peripheral Wait Cycles 0-63	DMAAckn remains active for PWC+1 PerClk cycles. Used only for the peripheral side of peripheral mode transfers.

19:21	PHC	Peripheral Hold Cycles 0-7	The number of PerClk cycles between the time that DMAAckn becomes inactive until the peripheral bus is available for the next bus access. Used only during the peripheral side of peripheral mode transfers.
22	ETD	End-of-Transfer/Terminal Count (EOTn[TCn]) Pin Direction 0 EOTn[TCn] is an EOT input 1 EOTn[TCn] is a TC output	ETD must be set to 1 if the channel is configured for software-initiated memory-to-memory transfers.
23	TCE	Terminal Count (TC) Enable 0 Channel does not stop when TC is reached 1 Channel stops when TC is reached	If TCE=1, it is required that ETD=1.
24:25	CP	Channel Priority 00 Low priority 01 Medium low priority 10 Medium high priority 11 High priority	Actively requesting channels of the same priority are ranked in order by channel number, channel 0 having the highest priority. See "Channel Priorities" on page 18-13 for more information.
26:27	PF	Memory Read Prefetch Transfer 00 Prefetch 1 doubleword 01 Prefetch 2 doublewords 10 Prefetch 4 doublewords 11 Reserved	Used only during memory-to-peripheral and deviced-paced memory-to-memory transfers. To enable prefetching it is required that BEN=1.
28	PCE	Parity Check Enable 0 Disable parity checking 1 Enable parity checking	Enables parity checking for peripheral mode transfers. See "Data Parity During DMA Peripheral Transfers" on page 18-14.
29	DEC	Address Decrement 0 SAI and DAI fields control memory address incrementing. 1 After each data transfer the memory address is decremented by the transfer width.	If DEC=1, it is required that BEN=0. This field is valid only for peripheral mode transfers (TM=00).
30:31	▨	Reserved	

18.3.5 DMA Source Address Registers (DMA0_SA0–DMA0_SA3)

The DMA Source Address Registers (DMA0_SA0–DMA0_SA3) contain the source address for memory-to-memory and memory-to-peripheral transfers. If a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1) the source address register is automatically loaded from memory. For additional details see "Scatter/Gather Transfers" on page 18-16.

The source address must be aligned at the transfer width programmed in DMA0_CRn[TW], otherwise the error bit (DMA0_SR[RIn]) is set for the channel and no transfer occurs. If the source address increment bit in the channel's control register is set (DMA0_CRn[SAI]) the address is incremented by the transfer width after each data transfer. In contrast, if the channel is performing a memory-to-peripheral transfer and the address decrement bit is set (DMA0_CRn[DEC]=1), the address is decremented by the transfer width after each transfer.

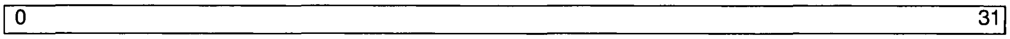


Figure 18-6. DMA Source Address Registers (DMA0_SA0–DMA0_SA3)

0:31		Source address for memory-to-memory and memory-to-peripheral transfers.

18.3.6 DMA Destination Address Registers (DMA0_DA0–DMA0_DA3)

The DMA Destination Address Registers (DMA0_DA0-DMA0_DA3) contain the destination address for memory-to-memory and peripheral-to-memory transfers. When a DMA channel is configured for scatter/gather transfers (DMA_SGC[SSGn]=1) the destination address register is automatically loaded from memory. For additional details see “Scatter/Gather Transfers” on page 18-16.

The destination address must be aligned at the transfer width programmed in DMA0_CRn[TW], otherwise the error bit (DMA0_SR[RIn]) is set for the channel and no transfer occurs. If the destination address increment bit in the channel's control register is set (DMA0_CRn[DAI]) the address is incremented by the transfer width after each data transfer. However, if the channel is performing a peripheral-to-memory transfer and the address decrement bit is set (DMA0_CRn[DEC]=1), the destination address is decremented by the transfer width after each transfer.

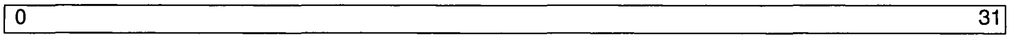


Figure 18-7. DMA Destination Address Registers (DMA0_DA0-DMA0_DA3)

0:31		Destination address for memory-to-memory and peripheral-to-memory transfers.

18.3.7 DMA Count Registers (DMA0_CT0–DMA0_CT3)

The DMA Count Registers (DMA0_CT0-DMA0_CT3) contain the number of transfers left in the DMA transaction for their respective channels when EOTn[TCn] is programmed as a terminal count output. When EOTn[TCn] is programmed as an end-of-transfer input (DMA0_CRn[ETD]=1), DMA0_CTn continues to count down past zero until EOTn is asserted.

If a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1) the count register is automatically loaded from memory. For additional details see "Scatter/Gather Transfers" on page 18-16.

The value in the DMA count register is interpreted as the number of **transfers** of the width specified in DMA0_CRn[PW], **not** the total number of bytes. The maximum number of transfers is 64K, and each transfer can be either 1, 2, 4, or 8 bytes as programmed in DMA0_CR[PW]. The maximum count of 64K transfers is programmed by writing zero to DMA0_CTn.

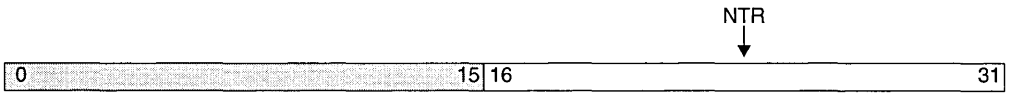


Figure 18-8. DMA Count Registers (DMA0_CT0-DMA0_CT3)

0:15	▨	Reserved
16:31	NTR	Number of transfers remaining

18.3.8 DMA Scatter/Gather Descriptor Address Registers (DMA0_SG0–DMA0_SG3)

When a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1), the Scatter/Gather Descriptor Address Register (DMA0_SGn) contains the memory address of the next scatter/gather descriptor table. Prior to starting a scatter/gather transfer, software must write the address of the channel's descriptor table to DMA0_SGn. Once the scatter/gather transfer starts, DMA0_SGn is automatically updated from the descriptor table. For additional details see "Scatter/Gather Transfers" on page 18-16.

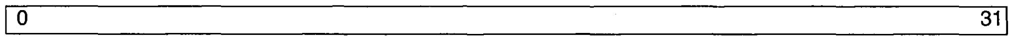


Figure 18-9. DMA Scatter/Gather Descriptor Address Registers (DMA0_SG0–DMA0_SG3)

0:31		Address of next scatter/gather descriptor table.

18.3.9 DMA Scatter/Gather Command Register (DMA0_SGC)

The DMA Scatter/Gather Command Register (DMA0_SGC) is a 32-bit register, of which 8-bits are implemented. Bits 0:3 are the Start Scatter/Gather Enable bits for channels 0 to 3, and bits 16:19 are the corresponding Enable Mask bits for the Start Scatter/Gather Enable bits. Setting a Start Scatter/Gather Enable bit causes the selected channel to begin a scatter/gather operation, while writing a 0 stops the Scatter/Gather operation. To start or stop a specific Scatter/Gather channel, the corresponding Enable Mask bit must be set to 1; otherwise, the register holds the previous value. Note that halting a scatter/gather transfer does not stop the transfer currently in progress.

Upon completion of a scatter/gather sequence of transfers the DMA controller clears DMA0_SGC[SSGn].

If an error occurs when the DMA controller is reading the Scatter/Gather descriptor table, DMA0_SGC[SSGn] is cleared for the affected channel, and the channel's error status bit (DMA0_SR[RIn]) is set.

For additional details see "Scatter/Gather Transfers" on page 18-16.

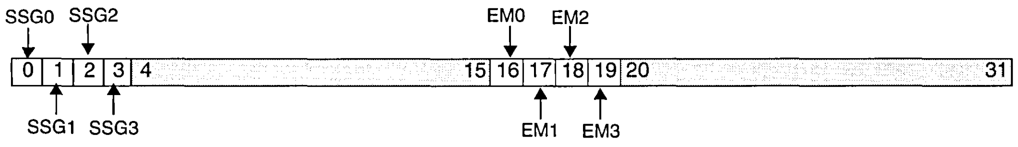


Figure 18-10. DMA Scatter/Gather Command Register (DMA0_SGC)

0:3	SSG[0:3]	Start Scatter/Gather for channels 0-3. 0 Scatter/gather support is disabled 1 Scatter/gather support is enabled	To start a scatter/gather operation for channel n, EM[n] must also be set.
4:15		Reserved	
16:19	EM[0:3]	Enable Mask for channels 0-3. 0 Writes to SSG[n] are ignored 1 Allow writing to SSG[n]	To write SSG[n], EM[n] must be set. Otherwise, writing SSG[n] has no effect.
20:31		Reserved	

18.4 Channel Priorities

The priority of DMA transfers is controlled on a per-channel basis by the channel priority field in the channel control register. Table 18-3 shows the different priority settings for DMA0_CRn[PW].

Table 18-3. DMA Transfer Priorities

DMA0_CRn[CP]	Priority Level
0b00	Low
0b01	Medium Low
0b10	Medium High

Table 18-3. DMA Transfer Priorities

DMA0_CRn[CP]	Priority Level
0b11	High

These priorities serve two purposes. First, the DMA controller arbitrates among all actively requesting channels and selects the highest priority one for service. If multiple channels request at the same priority, the arbiter selects the lowest numbered channel for service. Secondly, DMA0_CRn[CP] determines the priority of the internal PLB transactions that the DMA controller uses to read and write data.

18.5 Data Parity During DMA Peripheral Transfers

The DMA controller works in conjunction with the peripheral bus controller (EBC) to generate and check parity during peripheral mode DMA transfers. When DMA0_CRn[PCE]=1 parity checking is enabled for peripheral mode transfers on channel n.

During memory-to-memory transfers any data error checking and/or correction is dependent on the configuration of the relevant memory controller. For example, if ECC is enabled on the SDRAM controller, only uncorrectable errors are reported to the DMA controller. Similarly, an EBC memory bank with parity enabled will report an error if a parity error is encountered during a memory read operation. See "Peripheral Bank Access Parameters (EBC0_BnAP)" on page 16-26 for more information on peripheral bus parity checking.

18.6 Errors

The DMA controller detects and reports three types of errors: address alignment, PLB timeout and slave errors. The DMA controller reports errors through the channel error status bit in the DMA status register (DMA0_SR[RIn]). If the error status bit for a channel is set, the channel enable bit (DMA0_CRn[CE]) is cleared, disabling the channel. An interrupt signal is also presented to the interrupt controller if DMA0_CRn[CIE] is set. See "DMA Interrupts" on page 18-15 for more information on interrupt processing.

When the DMA controller has multiple channels active, an error may be reported on the current channel which was in actuality caused by a previously active channel. This causes the current channel to have its error status bit set. Therefore, for deterministic error analysis with multiple DMA channels active the PLB slave bus controller's error status registers (the bus error address register in particular), must be queried to isolate the actual channel which encountered the error. In any case, the channel causing the errors will eventually cause all active channels, including itself, to be disabled.

18.6.1 Address Alignment Error

The source address (DMA0_SAn) and destination address (DMA0_DAn) registers must be aligned to the programmed transfer width (DMA0_CRn[TW]).The address alignment rules are outlined in Table 18-4. In addition, when a channel is configured for scatter/gather transfers, the scatter/gather table must be word-aligned. If the source, destination and scatter/gather address registers are not appropriately aligned an error occurs immediately after the channel is enabled.

Table 18-4. Address Alignment Requirements

DMA0_CRn[PW] Setting	Required Alignment for: Source Address Register (DMA0_SAn) and Destination Address (DMA0_DAn)
0b00	Byte (8-bit)
0b01	Halfword (16-bit)
0b10	Word (32-bit)
0b11	Doubleword (64-bit)

18.6.2 PLB Timeout

The DMA controller uses PLB operations to read and write memory. A PLB timeout results if the DMA controller attempts to access a non-existent memory location. This will occur if the source, destination or scatter/gather address registers do not map to valid memory locations.

18.6.3 Slave Errors

If the DMA controller detects an error from a PLB slave, it finishes any active read/write pair transfer on the channel and then reports an error. An SDRAM uncorrectable ECC error and an EBC bank protection error are examples of PLB slave errors.

18.7 DMA Interrupts

Each DMA channel can generate interrupts for end of transfer, terminal count and error conditions. Interrupts from a particular DMA channel are enabled by setting the channel enable bit in the channel's control register (DMA0_CRn[CIE]=1). When an interrupt occurs for a given channel, the DMA controller sends a signal to the Universal Interrupt Controller. For the PPC405GP's CPU to take an exception, interrupts from the particular DMA channel must be enabled in the interrupt controller's interrupt enable register (UIC0_ER). Also, the CPU's machine state register's interrupt enable bit must be enabled for the appropriate interrupt type (critical or non-critical), MSR[EE,CE]. See Chapter 10, "Interrupt Controller Operations" for more information on interrupt controller processing.

For DMA channels with interrupts enabled (DMA0_CRn[CIE]=1) and not performing a scatter/gather transfer an interrupt is generated while any of the following are true:

DMA0_CRn[TCE]=1 and DMA0_SR[CSn]=1
DMA0_SR[TSn]=1
DMA0_SR[RIn]=1

When a channel is performing a scatter/gather transfer, interrupt generation is further qualified by the TCI, ETI and ERI bits loaded from the descriptor table (see Table 18-5, "Scatter/Gather Descriptor Table," on page 18-16). Any of the following conditions cause an interrupt during a scatter/gather transfer when interrupts are enabled for the channel (DMA0_CRn[CIE=1]):

TCI=1 and DMA0_CRn[TCE]=1 and DMA0_SR[CSn]=1
ETI=1 and DMA0_SR[TSn]=1
ERI=1 and DMA0_SR[RIn]=1

For both normal DMA and scatter/gather transfers the interrupt remains active until the appropriate bits are cleared in the DMA Status Register (DMA0_SR). In addition, interrupts from a channel performing a scatter/gather transfer cause the channel to pause until the interrupt is cleared.

18.8 Scatter/Gather Transfers

With a normal DMA transfer it is necessary to program a channel's control, source, destination, and count registers for each transfer. The scatter/gather capability of the DMA controller provides a more efficient solution for applications that require multiple transactions on a single DMA channel. Instead of individually programming a channel's registers, software creates a set (linked list) of descriptor tables in system memory. Table 18-5 illustrates the required table format.

Table 18-5. Scatter/Gather Descriptor Table

Memory Address	Byte 0 (MSB)						Byte 1	Byte 2	Byte 3 (LSB)
x (word aligned)	DMA Channel Control Word								
x + 4	Source Address								
x + 8	Destination Address								
x + 12	LK	▨	TCI	ETI	ERI	▨	▨	Count	
x + 16	Next Scatter/Gather Descriptor Table Address								

Table 18-6 details the usage of the bit fields in the scatter/gather table.

Table 18-6. Bit Fields in the Scatter/Gather Descriptor Table

Bit	Mnemonic	Description
0	LK	Link 0 This is the last descriptor. 1 Fetch next descriptor from address DMA0_SGn when the channel completes
2	TCI	Enable Terminal Count Interrupt 0 Do not interrupt when terminal count occurs 1 Allow an interrupt when terminal count occurs
3	ETI	Enable End of Transfer Interrupt 0 Do not interrupt when end of transfer occurs 1 Allow an interrupt when end of transfer occurs.
4	ERI	Enable Error Interrupt 0 Do not interrupt if an error occurs 1 Allow an interrupt if an error occurs

To configure a channel for a scatter/gather transfer the DMA Scatter/Gather Descriptor Address Register (DMA0_SGn) for the channel is set to the address of the first descriptor table, which must be word-aligned. To begin the scatter/gather transfer, software then writes a start scatter/gather and enable mask to the Scatter/Gather Command Register (DMA0_SGC). The DMA controller then reads the descriptor table at address DMA0_SGn and updates the DMA controller registers as shown in

Table 18-7. Upon receiving the data from the scatter/gather descriptor table, the channel's terminal count status bit (DMA0_SR[TCn]) is automatically cleared.

Table 18-7. DMA Registers Loaded from Scatter/Gather Descriptor Table

Descriptor Table Entry	Register Loaded
Channel Control Word	DMA0_CRn
Source Address	DMA0_SAn
Destination Address	DMA0_DAn
Count	DMA0_CTn
Next Descriptor Address	DMA0_SGn

After loading the channel's registers from the descriptor table, the transfer functions as a normal non-scatter/gather operation.

If the channel control word loaded from the descriptor table enables interrupts for the channel (DMA0_CRn[CIE]=1), the TCI, ETI, and ERI bits further qualify the generation of interrupts. See "DMA Interrupts" on page 18-15 for more information on scatter/gather interrupts.

If the LK (link) bit was not set the scatter/gather process stops when the current transfer completes. Otherwise, the DMA controller reads the descriptor table at address DMA0_SGn and the process repeats.

18.9 Programming the DMA Controller

Before the DMA controller can transfer data it must be configured, both globally and on a per-channel basis. Global settings include the DMA Polarity Register (DMA0_POL) and DMA Sleep Mode Register (DMA0_SLP). For most applications, these registers should be configured when the DMA controller is first initialized. To prevent spurious activity resulting from changing the active level for DMAReqn, DMAAckn, or EOTn[TCn], a channel's configuration in the Polarity Register should not be altered when the channel is enabled (DMA0_CRn[CE]=1).

Each channel has a Control (DMA0_CRn), Source Address (DMA0_SAn), Destination Address (DMA0_DAn), Count (DMA0_CTn), and Scatter/Gather Descriptor Address (DMA0_SGn) register. The type of DMA transfer determines which of these registers must be programmed and what causes the channel to start. In all cases, the terminal count (CSn), end of transfer (TSn) and error status (RIn) bits in the DMA Status Register (DMA0_SR) must be cleared or the channel will not start.

The programming information that follows assumes that the DMA controller is operating in non-scatter/gather mode. To use scatter/gather transfers the channel configuration data must be written into a set of descriptor tables in system memory. See "Scatter/Gather Transfers" on page 18-16 for additional details.

18.9.1 Peripheral Mode Transfers

DMA peripherals are either devices attached to the EBC interface via the DMAReqn and DMAAckn lines, or the internal serial port (UART0). During a peripheral mode transfer an external peripheral asserts DMAReqn to request a DMA transfer. For metastability protection DMAReqn is double latched in the DMA upon assertion, and sampled with a single latch on the deassertion.

Timings on the memory-access portion of peripheral mode DMA transfers are governed by the configuration of the associated memory controller. In contrast, timing during the peripheral portion of the transfer is controlled by the PSC, PWC and PHC fields in the DMA Channel Control Registers. The effect of these parameters on peripheral timings is illustrated in Figure 18-11 on page 18-18 and Figure 18-12 on page 18-19. Although shown as active high, the polarity (active state) of DMAReqn, DMAAckn, and EOTn[TCn] are programmable via the DMA Polarity Register (DMA0_POL).

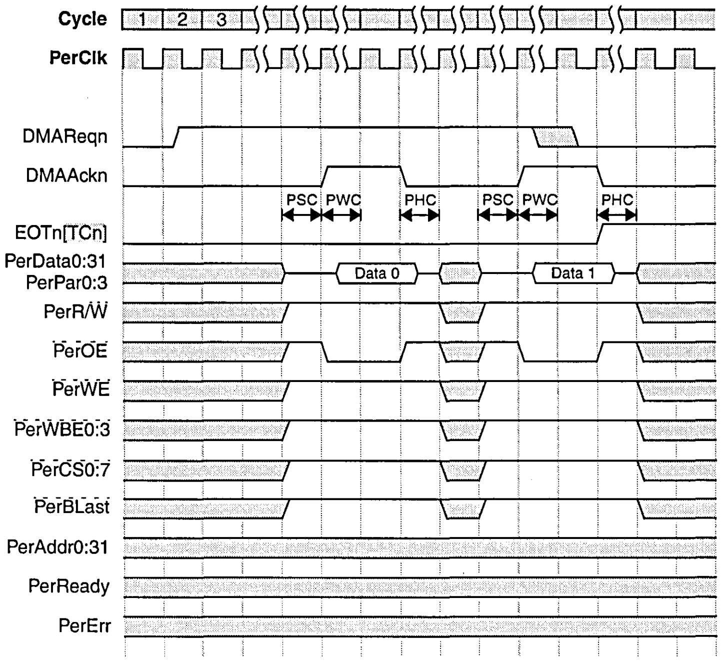


Figure 18-11. Peripheral-to-Memory DMA Transfer

Peripheral setup cycles (DMA0_CRn[PSC]) provide a delay between any previous operation on the peripheral bus and DMAAckn becoming active. Following the setup time DMAAckn is driven active for (DMA0_CRn[PWC] + 1) PerClk cycles. During peripheral-to-memory transfers read data from the peripheral is sampled on the PerClk edge where DMAAckn rises. After DMAAckn becomes inactive the peripheral bus is held idle for DMA0_CRn[PHC] PerClk cycles.

The second transfer in Figure 18-11 on page 18-18 illustrates the required DMAReqn timing to prevent a subsequent DMA transfer. For all peripheral mode transfers DMAReqn must be sampled inactive at the end of the last PerClk cycle where DMAAckn is active.

The EOTn[TCn] I/O can be configured either as an end of transfer input (DMA0_CRn[ETD]=0) or a terminal count output (DMA0_CRn[ETD]=1). When programmed as a terminal count output, EOTn[TCn] is asserted in the cycle after DMAAckn became inactive and the channel's count register (DMA0_CTn) reached zero. EOTn[TCn] remains active until the terminal count status bit is cleared in the DMA status register (DMA0_SR[CSn]).

If EOTn[TCn] is configured as an end of transfer input (DMA0_CRn[ETD]=0), EOTn[TCn] must be sampled active during the last DMAAckn cycle. If the channel is configured for scatter/gather transfers

EOTn[TCn] should be immediately deasserted to prevent the subsequent transfer from ending prematurely. Figure 18-12 on page 18-19 shows the required timing.

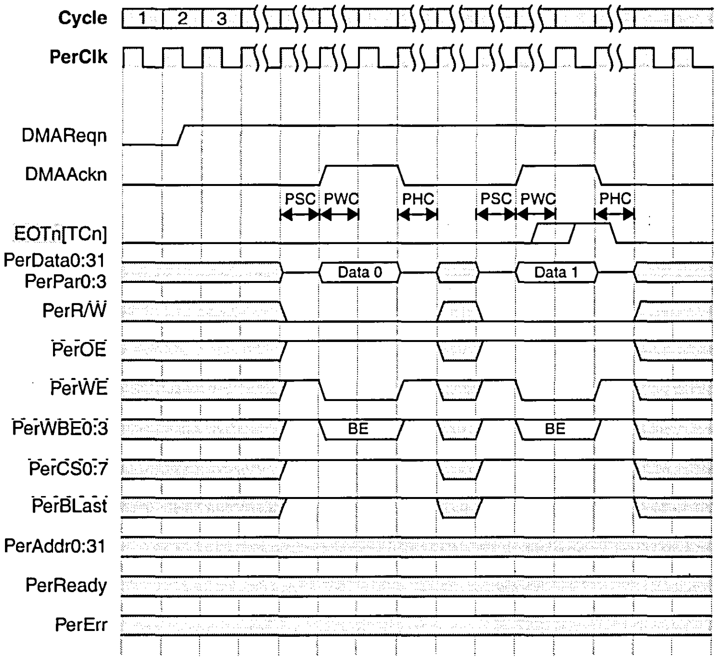


Figure 18-12. Memory to Peripheral DMA Transfer

For both peripheral-to-memory and memory-to-peripheral transfers the transfer width (DMA0_CR[PW]) must be set to the data bus width of the peripheral. This is because the DMA controller does not pack or unpack data on the peripheral side of transaction.

Peripheral-to-Memory Transfer

To perform a peripheral-to-memory DMA transfer from an EBC-attached DMA peripheral:

1. Set destination address register (DMA0_DAn) to the desired memory location. The address must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
2. Program the count register (DMA0_CTn) for the number of transfers.
3. Clear the channel's status bits in the DMA status register (DMA0_SR).
4. In the channel control register (DMA0_CRn):
   a. Optionally enable the DMA buffer, BEN=1.
   b. Optionally enable parity checking, PCE=1.
   c. Set the destination address increment, DAI=1.
   d. Set the transfer mode to peripheral, TM=0b00.
   e. Set the peripheral location to external, PL=0.
   f. Set the transfer direction to peripheral-to-memory, TD=1.

g. Enable the channel CE=1.

Once the DMA channel is active, the peripheral initiates a transfer by activating the DMAReqn pin for the channel. The PPC405GP then activates the DMAAckn pin to read data from the peripheral. This continues until either a terminal count or end of transfer condition occurs.

Memory-to-Peripheral Transfer

To perform a memory-to-peripheral DMA transfer to an EBC-attached DMA peripheral:

1. Set source address register (DMA0_SAn) to the desired memory location. The address must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
2. Program the count register (DMA0_CTn) for the number of transfers.
3. Clear the channel's status bits in the DMA status register (DMA0_SR).
4. In the channel control register (DMA0_CRn):

   a. Optionally enable the DMA buffer, BEN=1, and set the desired prefetch count, PF.

   b. Optionally enable parity generation, PCE=1.

   c. Set the source address increment, SAI=1.

   d. Set the transfer mode to peripheral, TM=0b00.

   e. Set the peripheral location to external, PL=0.

   f. Set the transfer direction to memory-to-peripheral TD=0.

   g. Enable the channel, CE=1.

Once the DMA channel is active, the peripheral initiates a transfer by activating the DMAReqn pin for the channel. The PPC405GP then reads the source memory and subsequently activates the DMAAckn pin to write data to the peripheral. This continues until either a terminal count or end of transfer condition occurs.

18.9.2 Memory-to-Memory Transfers

Memory-to-memory transfers can be initiated either by software or by an external device. If initiated via software, the transfer begins as soon as the channel is configured and enabled. When initiated by hardware (also known as a device-paced memory-to-memory transfer), software configures the channel for a memory-to-memory move and transfers begin when an external device places an active request on the channel request line, DMAReqn.

Hardware-Initiated (Device-Paced) Memory-to-Memory Transfers

To perform a device-paced memory-to-memory DMA transfer:

1. Set the transfer width (DMA0_CRn[PW]) to the width of the device-paced memory.
2. Set the source (DMA0_SAn) and destination (DMA0_DAn) address registers to the desired memory locations. These addresses must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
3. Program the count register (DMA0_CTn) for the number of transfers.
4. Clear the channel's status bits in the DMA status register (DMA0_SR).
5. In the channel control register (DMA0_CRn):

   a. Optionally enable the DMA buffer, BEN=1, and set the desired prefetch count, PF.

b. If the device-paced memory is at the source memory location set PL=1.

c. Set the source address increment, SAI, and destination address increment, DAI, as desired.

d. Set the transfer mode to device-paced memory-to-memory, TM=0b11.

e. Enable the channel, CE=1.

Once the DMA channel is configured for this mode, the external device initiates a transfer by activating the DMAReqn input. The PPC405GP then reads the source memory, buffers the data in the DMA controller and then outputs the data to the destination memory address. Transfers continue as long as the controlling device maintains an active signal on DMAReqn and the channel count register (DMA0_CTn) is non-zero. To pause a device paced memory-to-memory transfer, the controlling device must deassert DMAReqn one PerClk cycle before the last cycle in the device-paced memory access.

18.9.3 Software-Initiated Memory-to-Memory Transfers (Non-Deviced Paced)

To perform a software-initiated memory-to-memory DMA transfer:

1. Set the transfer width (DMA0_CRn[PW]) as desired.
2. Set the source (DMA0_SAn) and destination (DMA0_DAn) address registers to the desired memory locations. These addresses must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
3. Program the count register (DMA0_CTn) for the number of transfers.
4. Clear the channel's status bits in the DMA status register (DMA0_SR).
5. In the channel control register (DMA0_CRn):
   a. Optionally enable the DMA buffer, BEN=1.
   b. Set the source address increment, SAI, and destination address increment, DAI, as desired.
   c. Set the transfer mode to software-initiated memory-to-memory, TM=0b10.
   d. Enable the channel, CE=1.

Once the channel is enable the DMA controller transfers data from source to destination until the channel count reaches zero. Note that memory-to-memory transfers initiated by software do not use DMAReqn or DMAAckn.

Chapter 19. Ethernet Media Access Controller

The PPC405GP Ethernet media access controller (EMAC) is a generic implementation of the Ethernet Media Access Control (MAC) protocol complying with ANSI/IEEE Std 802.3 and IEEE 802.3u supplement. EMAC supports half-duplex (CSMA/CD) and full-duplex operation for both 10-Mbps and 100-Mbps operations.

EMAC provides two on-chip peripheral bus (OPB) slave interfaces. The first OPB interface provides access to EMAC configuration and status registers. The PLB/OPB bridge allows the 405 processor core to access these registers.

The second OPB interface is used to exchange packet information with the memory access layer (MAL). The MAL is a multi-channel, intermediate hardware layer that resides between packet-based communication macros (such as EMAC) and external memory (such as SDRAM or SRAM). The MAL transfers packet information and packet status between the EMAC and external memory separately for each of the three EMAC channels (one receive and two transmit). Software (a device driver) is responsible for maintaining a buffer descriptor ring and a set of data buffers in external memory for each of the channels, and manages the exchange of packet data between the data buffers and the software protocol.

The MAL performs functions such as arbitration between service requests, handling the buffer descriptor memory structure, updating the descriptor status/control fields at the end of packet transfer, and so on. EMAC supports unlimited burst length transactions on the MAL interface.

EMAC utilizes its media independent interface (MII) to communicate with standard physical interface devices (PHYs).

EMAC utilizes independent Receive (RX) and Transmit (TX) FIFOs. Programmable FIFO thresholds minimize overflows and underruns and can launch integrated IEEE 802.3x Pause Packets for flow control.

As part of the Remote Monitoring (RMON) and Management Information Base (MIB) defined in IEEE 802.3z the EMAC contains registers that count the number of octets transmitted and received.

Figure 19-1 illustrates EMAC in a typical Ethernet application.

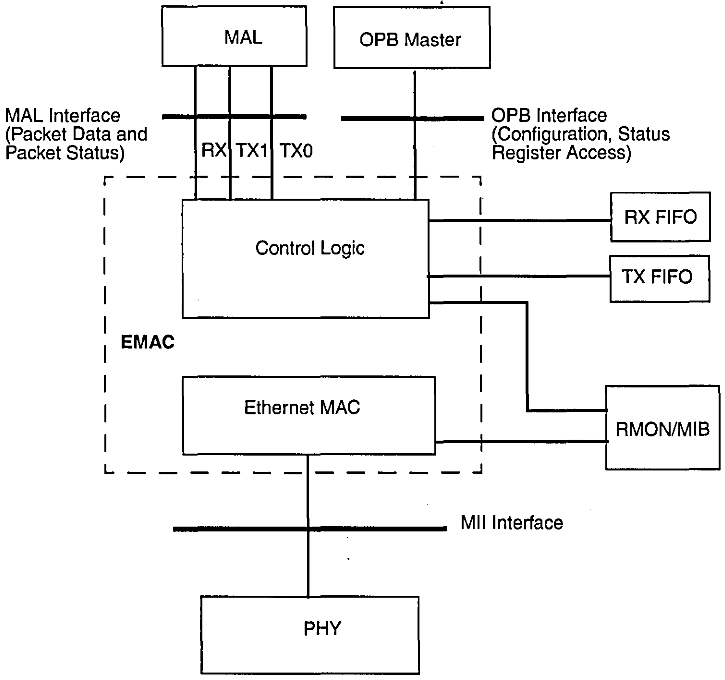

MAL
OPB Master
MAL Interface (Packet Data and Packet Status)
RX
TX1
TX0
OPB Interface (Configuration, Status Register Access)
Control Logic
RX FIFO
TX FIFO
EMAC
Ethernet MAC
RMON/MIB
MII Interface
PHY

Figure 19-1. EMAC in a Typical Ethernet Application

Note: EMAC is connected to a single OPB, to which both the OPB master and MAL are also connected. The EMAC transmit channels operate independently from the receive channel.

19.1 EMAC Features

The PPC405GP EMAC features:

- Dual speed (10/100 Mbps) CSMA/CD (half-duplex) and full-duplex Ethernet MAC-compliant with ANSI/IEEE Std. 802.3 and IEEE 802.3u supplement.
- Automatic source address insertion or replacement for transmitted packets is a programmable option.
- Automatic stripping of frame padding bytes and frame check sequence (FCS) is a programmable option.

  When padding bytes are stripped, the padding and FCS field are removed. FCS stripping removes only the FCS field.
- FCS control for transmit/receive packets.
- Access to registers with support for burst processing.
- MAL for packet moving having one-cycle MAL slave latency.

- Independent, large (2 KB) transmit and (4 KB) receive FIFOs with programmable thresholds to minimize overruns and underruns.
- Multiple packet handling in transmit and receive FIFOs.
- Unicast, multicast, broadcast, and promiscuous address filtering capabilities.
- Two 64-bit hash filters for unicast and multicast packets.
- Automatic retransmission of collided packets.
- Rejection of runt packets before providing them to MAL.
- MII interface for connection to a variety of PHY layer devices.
- Programmable inter-packet gap to enable tuning for better system performance.
- Compliance with IEEE 802.3x standard packet-based flow control, including self-assembled control pause packet transmitting.
- Support for VLAN tag ID in compliance with IEEE Draft 802.3ac/D1.0 standard.
- VLAN tag insertion or replacement for transmit packets is a programmable option.
- Wake On LAN (WOL) handling.
- Programmable internal and external loop-back capabilities.
- Extensive error/status vector generation for each processed packet.
- Power management using a clock and power management (CPM) unit.

19.2 EMAC Operation

The EMAC hardware components and its internal structure are illustrated in the block diagram in Figure 19-2.

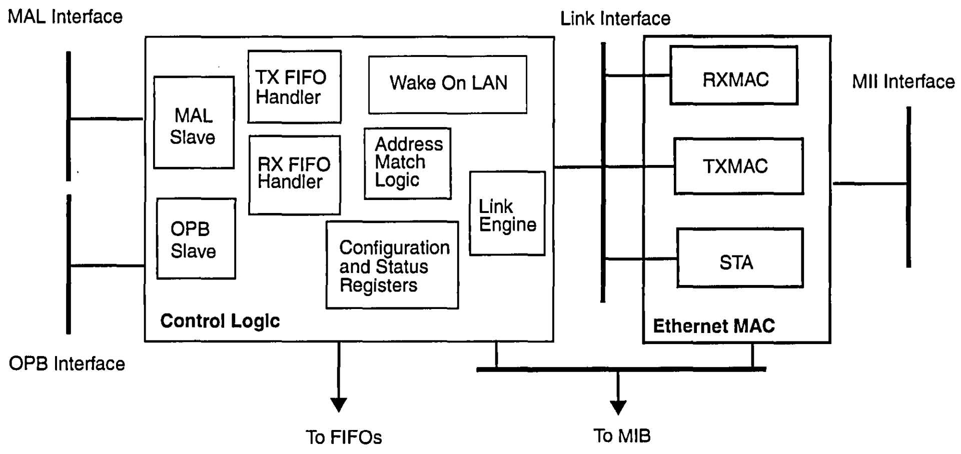


Figure 19-2. Internal EMAC Structure

The control logic sub-block implements the following functions:

- OPB slave device
- MAL slave device
- FIFO management logic
- Ethernet address and pause packet match logic
- Register file for FIFOs and Ethernet MAC handler management
- Logic for support of WOL technology

The Ethernet MAC sub-block implements the following functions:

- Transmit MAC Handler (TXMAC)
- Receive MAC Handler (RXMAC)
- MII management function unit (STA)

These functions are described in the following sections.

19.2.1 MAL Slave Logic

The MAL slave (MALS) logic controls MAL transactions. MALS transfers TX and RX data between MAL and the OPB on one side, and the EMAC FIFO handlers on the other side. MALS is a dedicated MAL slave.

19.2.2 OPB Slave Logic

The OPB slave (OPBS) logic controls all OPB transactions between the processor core and the EMAC configuration and status registers.

19.2.3 Ethernet Address Match Logic

Address match logic checks the destination address of received packets against a set of predefined addresses specified by the current address filtering mode. EMAC contains one unicast (individual address) register, two hash tables for filtering individual and group address, and logic for detecting broadcast address (all ones). EMAC supports promiscuous mode and multicast promiscuous mode.

This logic also checks the destination address of the incoming packet against a special multicast address used for control (pause) packet recognition.

All checks for address matching are performed only after the entire destination address field is received (except for promiscuous and multicast promiscuous modes).

19.2.4 Configuration and Status Registers

Configuration and status registers define the EMAC configuration and reflect error/status of recent transmitted or received packets.

19.2.5 Wake On LAN Logic

EMAC supports Wake On LAN (WOL) technology, an industry standard in the *Wired for Management (WFM)* specification. This technology allows a sleeping or powered-off network node to be awakened with a special packet called a Magic Packet. In the 405GP, with WOL mode enabled, the EMAC discards all incoming packets and does not request data from the MAL for transmission. When a magic packet is detected, the EMAC generates a UIC interrupt on Interrupt 9, called Ethernet WOL.

19.2.6 Ethernet MAC

The Ethernet MAC logic supports the Media Independent Interface (MII).

19.2.7 EMAC Loop-Back Modes

EMAC supports the external and internal loop-back modes illustrated in Figure 19-3.

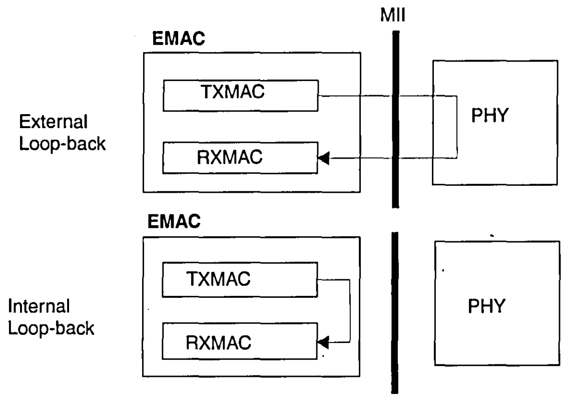


Figure 19-3. EMAC Loop-Back Modes

External loop-back mode uses the PHY device. To EMAC, external loop-back is identical to full-duplex operation. Configuring EMAC to operate in a full-duplex mode enables external loop-back. EMAC does not need an external loop-back configuration signal.

In internal loop-back mode, data from the EMAC transmit channel is routed to the EMAC receive channel. This loop-back mode functions correctly with or without a PHY connected to EMAC. In internal loop-back mode, EMAC does not activate (monitor) any MII signals. The transmit channel signals are buffered internally to the receive channel. However, if internal loop-back is used without a PHY, the TX and RX clocks for EMAC must be provided by another means. In internal loop-back mode, the EMAC transmit clock and receive clock must be sourced from a single clock.

19.3 EMAC Transmit Operation

The transmit part of EMAC is responsible for packet transmission from the MAL device to the MII interface. At the end of the transmission process, EMAC provides a status/error word which allows monitoring the transmission operation.

EMAC implements dual transmit channels with MAL (this means that two transmit channels are allocated within MAL) for efficient utilization of the Transmit FIFO. Both of these channels use the same shared resources inside EMAC. These channels can be configured to independently request packets from MAL and drive them into the TX FIFO or act as a single channel (in dependent mode).

19.3.1 Arbitration Between TX Channels

Because both EMAC transmit channels (referred to as TX Channel 0 and TX Channel 1) drive data into the same FIFO, they cannot request packet data from MAL at the same time. MAL ensures that only one EMAC transmit channel is active at a given time.

19.3.2 Independent Mode

In this mode, each EMAC transmit channel independently requests packets from MAL. Each channel can be configured to work in single packet or multiple packet modes.

In single packet mode, EMAC0_MR1[TR0] = 00 and EMAC0_MR1[TR1] = 00. The channel requests one packet from MAL and resets EMAC0_TMR0[GNP0, GNP1] as appropriate. The channel asks for service again only after EMAC0_TMR0[GNP0] = 1 or EMAC0_TMR0[GNP1] = 1 (set by the device driver).

In multiple packet mode, EMAC0_MR1[TR0] = 01 and EMAC0_MR1[TR1] = 01. After the channel finishes transferring a packet, the channel asks MAL for the next packet as soon as the other channel is in its Idle Phase and there is enough room in the FIFO. The channel continues to request more packets until one of the following events occur:

- The channel receives notification from MAL that the next buffer descriptor is not marked ready for transmission. When this occurs, the channel sets EMAC0_TMR0[GNP0, GNP1] = 0, as appropriate, and waits for software to reactivate it by setting EMAC0_TMR0[GNP0] = 1 or EMAC0_TMR0[GNP1] = 1.
- A transmit error or signal quality error (SQE) occurs and the corresponding interrupt is not masked in the EMAC0_ISER. After such an error, the channel sets EMAC0_TMR0[GNP0, GNP1] = 0, as appropriate, and sets EMAC0_ISR[DB0] = 1 or EMAC0_ISR[DB1] = 1 (the EMAC0_ISR field that is set depends on which channel is active) and the corresponding EMAC0_ISR error. The channel does not request service again until EMAC0_TMR0[GNP0] = 1 or EMAC0_TMR0[GNP1] = 1 and EMAC0_ISR[DB0] = 0 or EMAC0_ISR[DB1] = 0 (again, depending on channel).

In independent mode, if both channels are configured to work in multiple packet mode and both EMAC0_TMR0[GNP0] = 1 and EMAC0_TMR0[GNP1] = 1 at the same time, the channels operate in a sequential repeating manner as long as no errors occur.

19.3.3 Dependent Mode

In dependent mode, EMAC0_MR1[TR0] = 10 and EMAC0_MR1[TR1] = 10. The two TX channels act as if they were one channel, sharing EMAC0_TMR0[GNPD]. When EMAC0_TMR0[GNPD] = 1, the channel specified by EMAC0_TMR0[FC] starts requesting MAL service. Then, both channels continue to request packets from MAL in an alternating, sequential, repeating manner, until one of the following occurs:

- One of the channels receives notification from MAL that the next buffer descriptor is not marked ready for transmission. When this occurs, EMAC clears EMAC0_TMR0[GNPD]. At this point, neither channel requests a packet from MAL until EMAC0_TMR0[GNPD] back to 1. The first channel to request a new packet from MAL when this occurs is the channel that received notification from MAL that no packets were ready to transmit (regardless of the setting of EMAC0_TMR0[FC]). Further requests continue in an alternating, repeating manner.

- Either a transmit error or an SQE occurs on one of the channels and the corresponding interrupt is not masked in the EMAC0_ISER. One of the following scenarios can occur.
  - If the other channel has not yet requested MAL service when the channel for which the error occurred receives notification from MAL that the transmit operation has completed, EMAC0_TMR0[GNPD] is cleared (EMAC0_TMR0[GNPD] = 0) and the EMAC0_ISR[DBDM] is immediately set (EMAC_ISR[DBDM]=1).
  - If the other channel was receiving data from MAL, it initiates early termination. If a second packet was being transmitted on the media, it is stopped. In these cases, EMAC0_TMR0[GNPD] is cleared (EMAC0_TMR0[GNPD] = 0) and the EMAC0_ISR[DBDM] = 1 only after notification from MAL that the transmit operation has completed has been received for the second channel.

At this point, neither channel activates a request to MAL until EMAC0_TMR0[GNPD] = 1 and EMAC0_ISR[DBDM] = 0. The channel specified by EMAC0_TMR0[FC] is the first to request service from MAL. Subsequent requests continue in an alternating, sequential manner.

19.3.3.1 MAL TX Descriptor Control/Status Field

For each packet that is transmitted, MAL uses the descriptor control/status field of the buffer descriptor to both provide EMAC with control information (write) and to obtain packet status from EMAC after transmission is complete (read). Software writes the control bits in the buffer descriptor prior to packet transmission, and then reads the status bits from the buffer descriptor after packet transmission has completed. See “Buffer Descriptor Overview” on page 20-7 for more information on the buffer descriptor structure.

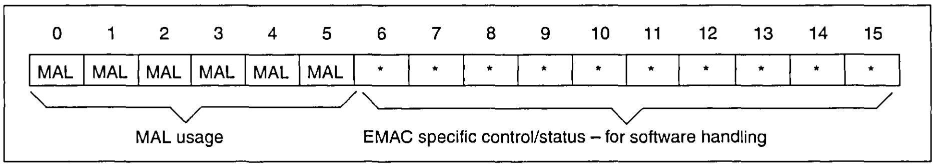

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
MAL MAL MAL MAL MAL MAL * * * * * * * * * *
MAL usage
EMAC specific control/status – for software handling

Figure 19-4. MAL TX Descriptor Control/Status Field

Bits	Bit Name	Bit Description	Mode
0..5	MAL Usage	See “TX Status/Control Field Format” on page 20-15.	R
TX Control Information (Write Access)			
6	Generate FCS	0 FCS is not generated by EMAC. 1 EMAC calculates and adds the FCS field to the packet to be transmitted.	W
7	Generate padding	0 Padding is not generated by EMAC. 1 EMAC adds the padding field to the packet to be transmitted (only when Generate FCS is also set).	W
8	Insert source address	0 EMAC will not insert source address. 1 EMAC inserts the source address field into the packet to be transmitted using the content of the Individual Address High (EMAC0_IAHR) and Individual Address Low (EMAC0_IALR) Registers.	W

Bits	Bit Name	Bit Description	Mode
9	Replace source address	0 EMAC will not replace source address. 1 EMAC replaces the source address field in the packet to be transmitted using the content of the Individual Address High (EMAC0_IAHR) and Individual Address Low (EMAC0_IALR) Registers.	W
10	Insert VLAN Tag	0 EMAC will not insert a VLAN tag. 1 EMAC inserts the VLAN Tag field into the packet to be transmitted using the content of the VLAN TPID register (EMAC0_VTPID).	W
11	Replace VLAN Tag	0 EMAC will not replace the VLAN tag. 1 EMAC replaces the VLAN Tag field in the packet to be transmitted using the content of the VLAN TPID register (EMAC0_VTPID).	W
TX Status Information (Read Access)			
6	Bad FCS on transmitted frame	0 FCS was correct in the transmitted packet. 1 Indicates that a bad FCS was found in the transmitted packet.	R
7	Bad previous packet in dependent mode	0 Packet transmission OK. 1 Indicates that a Descriptor Error, Transmit Error, or SQE Error occurred in the previously transmitted frame. This bit will only be activated in dependent mode.	R
8	Loss of carrier sense	0 No loss of carrier. 1 During the transmission of a frame, the PHY_CRS input was de-asserted after it previously was asserted, or it was not asserted at all.	R
9	Excessive deferral	0 No excessive deferral. 1 Indicates that the current frame has been deferred for an excessive period of time. Applicable only in half duplex mode. The value of this period in bit times is calculated in the following ways: For 10/100 Mbps operation it is: 2 x (maxFrameSize x 8) bit times.	R
10	Excessive collisions	0 Less than 16 collisions. 1 Indicates that the current frame transmission had ended with a collision on the 16th consecutive attempt. Applicable only in half-duplex mode.	R
11	Late collision	0 No late collision. 1 Frame collided outside of the collision window. Applicable only in half-duplex mode.	R
12	Multiple collision	0 More than 1 but less than 16 collisions did not occur. 1 Transmitted frame collided more than once but less than 16 times. Applicable only in half-duplex mode.	R
13	Single collision	0 Single collision did not occur. 1 Activates if transmitted frame collided once. Applicable only in half-duplex mode.	R
14	Underrun	0 Underrun did not occur. 1 Frame transmission was aborted because of underrun; data from the Transmit FIFO was not valid in time to allow continuous data transmission on the MII interface.	R

Bits	Bit Name	Bit Description	Mode
15	SQE	0 Signal Quality Error did not occur. 1 Signal Quality Error test failed during packet transmission. Applicable only in half -duplex mode during 10 Mbps operation.	R

19.3.3.2 Early Packet Termination in Transmit

EMAC can initiate early packet termination during transmit, terminating packet transmission before MAL finishes transferring all packet data from memory to the Transmit FIFO of EMAC. This is typically used when error conditions force the EMAC to abort a transmission.

EMAC performs early termination on the MAL interface if any of the following conditions occur:

1. Underrun in the Transmit FIFO.
2. Excessive Collisions.
3. Excessive Deferral.
4. Late Collision.

19.3.3.3 Empty Packets

EMAC treats empty packets as if a normal packet had been written, but without writing data to the FIFO. A status word of all 0s is returned after an empty packet. EMAC expects that for word-aligned packets, MAL activates the related word transfer indication during the last data transfer, rather than providing an empty packet indication.

19.3.3.4 Automatic Retransmission of Collided Packets

EMAC automatically retransmits packets that collide on the MII interface. The Transmit FIFO always preserves the first 64 bytes of the packet until it receives an indication that the collision window has passed. Otherwise, if a collision was detected within the collision window, the packet is retransmitted automatically without a new request from MAL.

19.3.3.5 Inter-Packet Gap (IPG) Tuning

EMAC allows the user to program the length of the IPG using the EMAC0_IPGVR register, which contains one-third of the IPG value. By changing the content of this register, the user can adjust the fairness or aggressiveness of EMAC on the medium. By programming a lower number (but not less than four), EMAC becomes more aggressive on the media. The aggressive nature can result in EMAC capturing the network by forcing less aggressive nodes to defer. By programming a larger number of bit times, EMAC becomes less aggressive on the network and may defer more often than normal. EMAC performance may decrease as the IPG period is increased from the default value, but the resulting behavior may improve media performance by reducing the occurrence of collisions.

19.3.3.6 Full-Duplex Operation

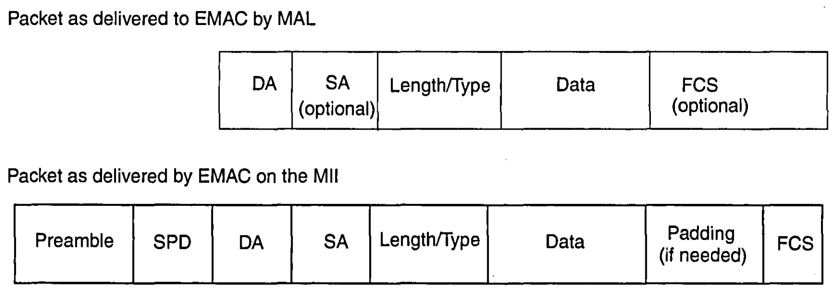
Full-duplex operation allows simultaneous transmit and receive activity on the MII interface. The software can enable full-duplex mode by setting the EMAC0_MR1[FDE] = 1. During operation in full-duplex mode, the following changes are made in EMAC functionality.

- Transmission is not deferred while receive is active.

- The IPG counter, which controls transmit deferral during the IPG between back-to-back transmits, is started when transmit activity for the first packet ends, instead of when transmit and carrier activity end.
- SQE test is not performed.
- Collision indication is ignored.

19.3.3.7 Packet Content Configuration Options

EMAC can modify the content of the packet coming from MAL before issuing it to the MII interface. Figure 19-5 illustrates possible changes in the transmit packet format.



Preamble - combination of alternative zeroes and ones (7 bytes)
SPD - Start Of Packet delimiter (1 byte)
DA - Destination Address (6 bytes)
SA - Source Address (6 bytes)
Length/Type - length of data field / type definition (Ethernet) (2 bytes)
Data - data field including padding if needed in short packets
Padding (optional) - needed to fulfill the packet up to the minimum allowed size
FCS - Cycle Redundancy Check (4 bytes)

Figure 19-5. Transmit Packet Structure (Excluding VLAN Tagged and Control Packets)

The following options can be set for each packet, and can be provided as a part of command write transactions from MAL (see "MAL TX Descriptor Control/Status Field" on page 19-7).

- Perform data padding automatically for short transmit packets. EMAC pads the transmit packet with extra bytes between the data and the FCS field to reach a total length of 64 bytes (including FCS). This feature is supported only when the packet coming from the Transmit FIFO does not contain the FCS. Automatic padding enables software to avoid sending padding as a part of the packet data field, and therefore, reduces the amount of data transferred on the system bus during the transmission of the short packet.
- Source address insertion. When chosen, EMAC adds the source address (SA) field to the transmitted packet. EMAC uses the content of the Individual Address High and Individual Address Low registers for the source address value. This mode is mutually exclusive with source address replacement mode.

- Source address replacement. When chosen, EMAC replaces the source address received from the Transmit FIFO with the content of the Individual Address High and Individual Address Low registers. This mode is mutually exclusive with the source address insertion mode.
- Add four FCS bytes. EMAC calculates the FCS for the transmitted packet. The FCS is appended to data coming from the Transmit FIFO or padding bytes (if added).
- VLAN Tag insertion. When chosen, EMAC adds the content of the VLAN Tag field to the transmitted packet (see "VLAN Support" on page 19-18). EMAC uses the content of the VLAN TPID and VLAN TCI registers for the VLAN Tag value. This feature is supported only when the packet coming from the Transmit FIFO does not contain an FCS. This mode is mutually exclusive with the VLAN Tag replacement mode.
- VLAN Tag replacement. When chosen, EMAC replaces the content of the VLAN Tag field in the transmitted packet. EMAC uses the content of the VLAN TPID and VLAN TCI registers for the VLAN Tag value. This feature is supported only when the packet coming from the Transmit FIFO does not contain an FCS. This mode is mutually exclusive with the VLAN Tag insertion mode.

Table 19-1 summarizes the possible options of adding FCS and source address to the transmitted packet.

Table 19-1. FCS/SA Enable - Possible Configurations

Configuration Options			EMAC Action		
Generate FCS	**Insert SA**	**Replace SA**	**Add FCS**	**Add SA**	**Replace SA**
0	Don't care	Don't care	N	N	N
1	0	0	Y	N	N
1	0	1	Y	N	Y
1	1	0	Y	Y	N

Table 19-2 summarizes the possible options of adding FCS and padding to the transmitted packet.

Table 19-2. FCS/Pad Enable - Possible Configurations

Configuration Options		EMAC Action	
Generate FCS	**Generate Pad**	**Add FCS**	**Add Pad**
0	Don't care	N	N
1	0	Y	N
1	1	Y	Y

Table 19-3 summarizes the possible options of adding FCS and VLAN Tag to the transmitted packet.

Table 19-3. FCS/VLAN Tag Enable - Possible Configurations

Configuration Options			EMAC Action		
Generate FCS	**Insert VLAN Tag**	**Replace VLAN Tag**	**Add FCS**	**Insert VLAN Tag**	**Replace VLAN Tag**
0	Don't care	Don't care	N	N	N
1	0	0	Y	N	N
1	0	1	Y	N	Y

Table 19-3. FCS/VLAN Tag Enable - Possible Configurations

Configuration Options			EMAC Action		
Generate FCS	Insert VLAN Tag	Replace VLAN Tag	Add FCS	Insert VLAN Tag	Replace VLAN Tag
1	1	0	Y	Y	N

19.4 EMAC Receive Operation

The receive part of EMAC is responsible for receiving packets coming from the physical layer (PHY) device and forwarding them to the receive channel of the attached MAL. At the end of the reception process, EMAC provides a status/error word which allows software to monitor the receive operation.

19.4.1 EMAC – MAL RX Packet Transfer Flow

EMAC initiates request for service from MAL whenever the number of occupied entries in the Receive FIFO reaches the low water mark specified in EMAC0_RWMR[RLWM].

19.4.2 MAL RX Descriptor Status

For each packet that is received, MAL obtains status from EMAC after reception is complete, and writes this information into the buffer descriptor status/control field. Software uses this information to monitor the status of received packets. See “Buffer Descriptor Overview” on page 20-7 for more information on the buffer descriptor structure.

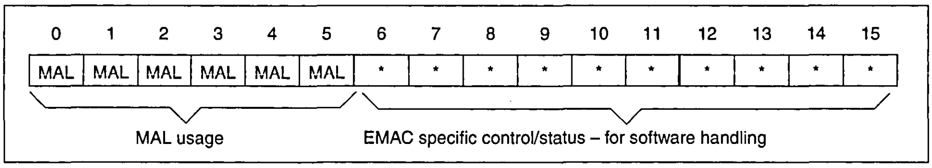


Figure 19-6. MAL RX Descriptor Control/Status Field

Bits	Bit Name	Bit Description	Mode
0..5	MAL Usage	See “RX Status/Control Field Format” on page 20-16.	R
RX Status Information (Read Access)			
6	Overrun Error	0 No overrun error. 1 EMAC detected an overrun error. An overrun error occurs if the flow of received data to the RX FIFO is corrupted because of insufficient empty space.	R
7	Pause Packet	0 Received packet is not a control pause packet. 1 Received packet is a control pause packet.	R
8	Bad Packet	0 No packet errors. 1 Early termination caused by packet error.	R
9	Runt Packet	0 Duration of PHY_RX_DV signal OK. 1 Duration of PHY_RX_DV signal greater than ShortEventMax Time constant and less than collision windows.	R

Bits	Bit Name	Bit Description	Mode
10	Short Event	0 Duration of PHY_RX_DV signal OK. 1 Duration of PHY_RX_DV signal was less than ShortEventMaxTime constant.	R
11	Alignment Error	0 Received packet length OK. 1 Received packet length not an integral number of octets.	R
12	Bad FCS	0 FCS OK. 1 The FCS value does not match the FCS value calculated by EMAC.	R
13	Packet Too Long	0 Received packet length OK. 1 Received packet length exceeded maximum allowed value. The received packet exceeded the maximum packet length: • 1518 octets for standard packet • 1522 octets for VLAN tagged packetData following the maximum packet length is not transferred to MAL	R
14	Out of Range Error	0 Received packet length field value OK. 1 Received packet length field value greater than maximum allowed LLC data size. The maximum allowed logical link control (LLC) data size is greater than 1500 and less than 1536.	R
15	In Range Error	Refer to Table 19-4 for a description of conditions for activating this status bit.	R

Table 19-4. In Range Length Error Behavior for Various Packet Lengths

Programmed Length (Bytes)	Actual Length	EMAC Action
Less than 46 (42 if VLAN Tagged packet)	Differs from 46 (42 in case of VLAN Tagged packet)	In range length error is activated
Less than 46 (42 if VLAN Tagged packet)	46 (42 in case of VLAN Tagged packet)	In range length error is not activated
Greater than or equal to 46 (42 if VLAN Tagged packet) and less than or equal to 1500	Equals the length field value	In range length error is not activated
Greater than or equal to 46 (42 if VLAN Tagged packet) and less than or equal to 1500	Differs from the length field value	In range length error is activated

19.4.3 Early Packet Termination In Receive

Early packet termination occurs when packet reception is aborted by EMAC before the packet data transfer to MAL is completed.

EMAC performs early termination in the following cases.

- An overrun occurs in the receive FIFO
- Packet is too long and the Receive Oversize Packet option is not enabled (EMAC0_RMR[ROP] = 0)

19.4.4 Discarding Packets During Receive

Receive packets can be discarded if certain error conditions are detected. EMAC behavior depends on whether the packet to be discarded is already being output to MAL. If the packet containing the error is not being provided to MAL when the discard condition is detected, the packet is flushed from the Receive FIFO. In this case, EMAC does not provide status information to MAL. If the packet containing the error is already being output to MAL, then EMAC initiates an early packet termination procedure, as described in "Early Packet Termination In Receive" on page 19-13.

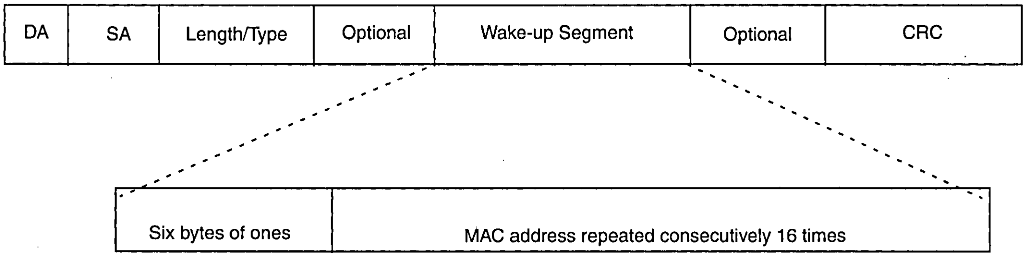
Each receive discard condition can be individually controlled using appropriate settings, as described in "Receive Mode Register (EMAC0_RMR)" on page 19-29.

19.4.5 WOL Support

WOL logic in EMAC supports WOL technology, an industry standard in the Wired for Management (WFM) Specification. WOL remotely awakens sleeping or powered-off nodes on a network. To wake up a node, a specific packet, called a *magic packet*, is sent to network node. When so configured, EMAC monitors the incoming bit stream for a magic packet.

The magic packet, also called a wake-up packet, contains a unique data field not normally expected in typical LAN traffic. When a WOL-enabled adapter on a powered-off client decodes this data field, a wake-up signal is generated. In the 405GP, with WOL mode enabled, the EMAC discards all incoming packets and does not request data from the MAL for transmission. When a magic packet is detected, the EMAC generates an interrupt on UIC Interrupt 9, called Ethernet WOL.

Figure 19-7 shows the wake-up packet format. The key to the wake-up packet is the MAC address of the target client, which is repeated 16 times. This pattern of 16 addresses in the data field is not expected to occur in any packet except the wake-up packet.



DA - destination address (six bytes) - UAA or Broadcast address
SA - source address (six bytes)
Length/Type - length of data field (802.3)/type definition (Ethernet) (two bytes)
optional - for example, IP header
CRC - Cycle Redundancy Check (four bytes)

Figure 19-7. Wake-Up Packet Format

The destination address can be a specific address, called the universally administered address (UAA), or a broadcast address. If the destination address is a UAA, the wake-up packet is sent only to the client at that address. However, since the client is powered off and is no longer transmitting, some protocols remove the client MAC address from routing tables and internal caches at other nodes. In this case, wake-up packets addressed to a target client are discarded because nodes and routers do not know where to send them. The solution to this problem is to use a broadcast address. A directed broadcast has a valid network address and a broadcast host address. Network routers and nodes forward directed broadcasts to the appropriate network, where it is seen as a MAC-level broadcast and detected by the powered-off client.

19.4.5.1 EMAC WOL Support

EMAC enters WOL mode when the wake-up bit in Mode Register 0 is set (EMAC0_MR0[WKE] = 1).

WOL mode should only be changed while the receive MAC idle bit in Mode Register 0 is set (EMAC0_MR0[RXI] = 1) and the receive MAC enable bit is reset (EMAC0_MR0[RXE] = 0). After the wake-up enable bit is set (EMAC0_MR0[WKE] = 1), the receive MAC enable bit can be set (EMAC0_MR0[RXE] = 1).

A reset (soft or hard) should be issued before programming EMAC to WOL mode. When programmed to WOL mode, EMAC does not propagate any received packet to MAL. Also, EMAC transmit channels do not request data from MAL.

19.5 Flow Control

EMAC implements full-duplex flow control for efficient system performance by handling specific MAC control packets contained in the pause opcode. EMAC supports flow control as defined in the IEEE 802.3x-1997 standard.

19.5.1 MAC Control Packet

The flow control mechanism allows the Receive FIFO control logic to automatically notify the node transmitting packets to suspend its transmission for a defined period of time. The pause control packet has a fixed length defined as follows:

MinFrameSize–32 bits (60 bytes)

MAC control packets have a unique value of 0x8808 in the length/type field, and share the same packet format as normal Ethernet packets, except that the data field consists of an opcode field and a parameter field. The opcode field contains an opcode command and the parameter field contains a value associated with the opcode command (0x0001). The only opcode command defined by IEEE 802.3x is the Pause opcode; the parameter field for the Pause opcode defines the pause time. MAC control packets with the Pause opcode, also called pause packets, can have a destination address equal to a reserved multicast address, or can be the address of the receive station itself. The reserved multicast address is 0x0180C2000001.

Figure 19-8 illustrates the control packet format.

Preamble	SPD	DA	SA	Length/ Type	Opcode	Timer Value Field	Reserved	FCS

Preamble - Alternating zeroes and ones (7 bytes)
SPD (Start Of Packet Delimiter), 1 byte
DA (Destination Address) = 0x0180C2000001
SA (Source Address) = Universally Administered Address (UAA)
Length/Type = 0x8808
Opcode Field = 0x0001
Timer Value Field = PauseValue
Reserved = zeroes (42 bytes)
FCS = calculated FCS

Figure 19-8. Control Packet Format

The timer value field contains the value of the delay interval in resolution of *pause_quanta*, defined in IEEE 802.3x as follows: "MAC Control Parameter[s] (pause_time) is a 2-octet, unsigned integer containing the length of time for which the receiving station is requested to inhibit data packet transmission. The field is transmitted most-significant octet first, and least-significant octet second. The pause_time is measured in units of pause_quanta, equal to 512 bit times. The range of possible pause_time is 0 to 65535 pause_quanta."

19.5.2 Control Packet Transmission

There are two options to initiate the transmission of a pause packet from EMAC. The transmitted pause packet will force the node, with the destination address specified, to temporarily suspend the transmission of packets to EMAC.

- Option 1

  Software initiated. The packet transferred to EMAC by MAL for transmission is a pause packet created by software. EMAC transmits this as a normal packet.

- Option 2

  Automatic flow control initiated. The EMAC integrated flow control mechanism detects the need for and then transmits a control (pause) packet automatically. When building the control packet, EMAC obtains the SA (source address) field from the Individual Address Registers (EMAC0_IAHR and EMAC0_IALR), and the timer value from the Pause Timer Register (EMAC0_PTR[TVF]). The contents of the other fields in the packet are represented in Figure 19-8.

19.5.3 Integrated Flow Control

To enable integrated flow control in full-duplex mode, set EMAC0_MR1[EIFC] = 1. When the receive FIFO reaches a predefined threshold level (called a high water mark and specified by EMAC0_RWMR[RHWM]), an internal request for control (pause) packet transmission is activated. EMAC sends a control (pause) packet when a new packet enters the Receive FIFO and the number of vacant entries in the Receive FIFO is less than the high water mark. When the Receive FIFO reaches another predefined threshold level (the low water mark, specified by

EMAC0_RWMR[RLWM]), a new internal request for a pause packet transmission, with a zero pause timer value, is activated. EMAC sends a pause packet, with a zero pause timer value, only once, and only if a pause packet with a non-zero value in the pause timer was transmitted earlier.

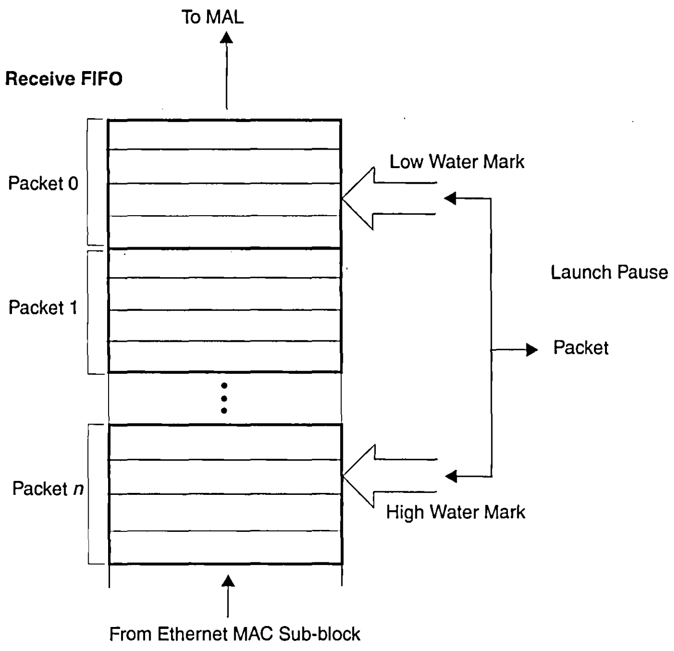


Figure 19-9. Integrated Flow Control Mechanism

19.5.4 Control Packet Reception

In the receive path, EMAC can be configured to respond to pause packets, or ignore them, as specified by the allow pause packet bit in Mode Register 1 (EMAC0_MR1[APP]). When response to pause packets is enabled and EMAC detects a valid MAC control packet with a Pause opcode, EMAC stores the value of the Timer Value field. The received packet is considered a valid control packet only if no error was detected during the packet reception. If, at the end of packet reception, the packet is considered valid, EMAC launches a pause operation state machine, as specified in the IEEE P802.3x standard. Figure 19-10 on page 19-18 illustrates the pause operation state machine.

If a control (pause) packet is received while another packet is being transmitted, the ongoing transmission process is completed and the transmitter is paused. If other packets are in the Transmit FIFO, their transmission is delayed until the pause timer has expired. EMAC normally does not pass the MAC control packets to MAL unless the Propagate Pause Packets bit is set in the Receive mode (EMAC0_RMR[PPP] = 1).

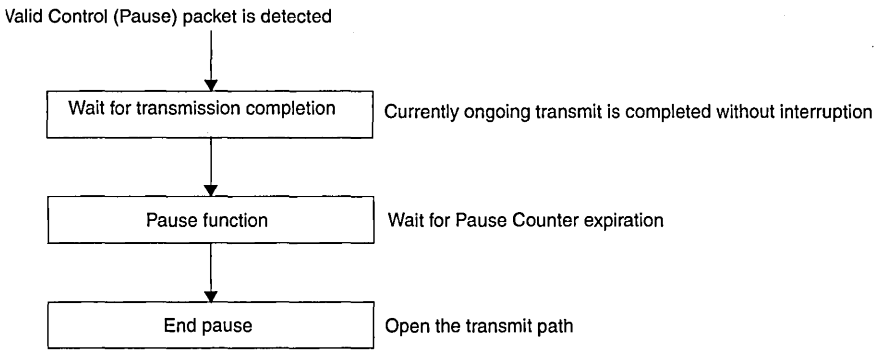

Valid Control (Pause) packet is detected
Wait for transmission completion
Currently ongoing transmit is completed without interruption
Pause function
Wait for Pause Counter expiration
End pause
Open the transmit path

Figure 19-10. Pause Operation State Machine

In the Pause Function state, EMAC decrements its internal pause timer, which was set to the Timer Value field of the received control packet.

Note 1: The transmission of control (pause) packets is not affected by the reception of a receive control (pause) packet. Received control (pause) packets inhibit only the transmission of regular packets from the Transmit FIFO.

Note 2: Receipt of a new valid control (pause) packet causes the pause timer of EMAC to be reloaded with the contents of Timer Value field of the recently received packet, regardless of the current pause timer setting. This indicates new pause operations take precedence over earlier pause operations.

19.6 VLAN Support

EMAC can handle VLAN tagged packets, as specified in IEEE Draft P802.3ac/D1.0a standard when EMAC0_MR1[VLE] = 1.

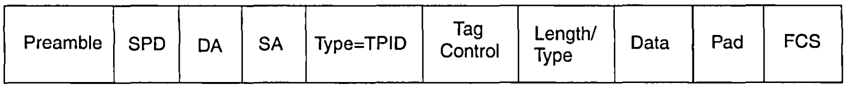
A Tagged MAC Frame is an extension of the standard MAC packet. The extension for VLAN tag support consists of a 4-octet VLAN tag inserted between the end of the Source Address and the beginning of the Length/Type fields of the MAC packet.

The VLAN tag consists of two fields:

- A 2-octet constant Type field value equal to the VLAN Tag Protocol Identifier (0x8100)
- A 2-octet field containing Tag Control Information (TCI)

The MAC Client Data and FCS fields of the basic MAC packet follow the VLAN tag. The length of the packet is extended by four octets by the VLAN tag (up to 1522 bytes). The FCS is calculated over all fields from the Destination Address through the end of the MAC client data or Pad (if present); that is, all fields except the preamble, SPD, and FCS.

Figure 19-11 illustrates the Tagged MAC Frame format.



Preamble – Alternating 0s and 1s (7 bytes)
SPD (Start Of Packet Delimiter)1 byte

DA (Destination Address) 6 bytes
SA (Source Address) 6 bytes
TPID (Tag Protocol Identifier, 2 bytes) = 0x8100

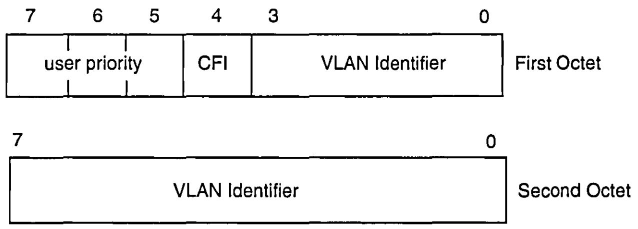
TCI (Tag Control Information) 2 bytes

Length/Type 2 bytes
Data (42–1500 bytes)

Pad (Optional field)
FCS – Calculated FCS

Figure 19-11. Tagged MAC Packet Format

Figure 19-12 illustrates the structure of the TCI field.



CFI is a Canonical Format Indicator (always 1 for Ethernet media)

Figure 19-12. Tag Control Information Field Structure

19.6.1 VLAN Tagged Packet Transmission

When EMAC0_MR1[VLE] = 1, the following configuration options are available, depending on the content of the appropriate bits in the MAL control word (see section 19.3.3.1, "MAL TX Descriptor Control/Status Field," on page 7):

- The Generate FCS bit (bit 6) is not set or both Insert VLAN Tag and Replace VLAN Tag bits (bits 10 and 11, respectively) are not set: EMAC transmits the packet without any changes
- Bit 6 is set and bit 10 is also set: EMAC will insert TPID and Tag control information for the transmitting packet using the content of EMAC0_VTPID and EMAC0_VTCI, respectively
- Bit 6 is set and bit 11 is also set: EMAC will replace TPID and Tag control information for the transmitting packet using the content of EMAC0_VTPID and EMAC0_VTCI, respectively

19.6.2 VLAN Tagged Packet Reception

If EMAC0_MR1[VLE] = 1, EMAC parses the VLAN Tag unique type/length in the incoming packet during the receive process. If the VLAN Tag is equal to the value stored in the EMAC0_VTPID, EMAC continues the receive process and allows the received packet to contain up to 1522 octets. Otherwise, the receive process is continued unless the length is greater than 1518 bytes.

19.6.3 Address Match Mechanism

The address match (or filtering) mechanism is a hardware aid that reduces the average amount of CPU cycles required to determine whether an incoming packet should be accepted.

EMAC uses various address filters for incoming packets by using the following address recognition modes.

- Individual mode (also referred to as physical)
- Multicast mode (also referred to as group)
- Broadcast mode (an all-ones group address)
- Promiscuous mode
- Promiscuous multicast mode
- WOL mode

A flowchart for address recognition of received packets is shown in Figure 19-13 on page 19-22. If the least significant bit (LSb) of the first byte of the destination address (DA) is 0, the packet is considered individual. If the first bit received is 1, the packet is considered multicast. When the DA field contains all 1s, the packet is broadcast, a special type of multicast.

19.6.3.1 Non-WOL Mode

When EMAC operates in single individual mode (EMAC0_RMR[IAE] = 1), the DA of the received packet is compared to the physical address stored in the Individual Address High register (EMAC0_IAHR) and Individual Address Low (EMAC0_IALR) register.

When EMAC operates in multiple individual mode (EMAC0_RMR[MIAE] = 1), EMAC performs a calculation on the contents of the DA field (logical address filter) to determine whether or not to accept the packet.

When EMAC operates in promiscuous mode (EMAC0_RMR[PME] = 1), all properly formed packets are received, regardless of the content of the DA field.

When EMAC operates in multicast promiscuous mode (EMAC0_RMR[PMME] = 1), all multicast packets are received, regardless of the content of the DA field.

When EMAC operates in broadcast address mode (EMAC0_RMR[BAE] = 1), EMAC performs an address compare on received packets with broadcast addresses.

When EMAC operates in multicast address mode (EMAC0_RMR[MAE] = 1), EMAC performs a calculation on the contents of the DA field (logical address filter) as in multiple individual mode, in order to determine whether or not the packet should be accepted.

The logical address filter hardware is an implementation of a hash code searching technique commonly used by software programmers. The hardware maps the DA of the incoming packet into one of the 64 categories that correspond to 64 bits stored in the EMAC0_IAHT1–EMAC0_IAHT4 or EMAC0_GAHT1–EMAC0_GAHT4 registers. The hardware accepts or rejects the packet, depending on the state of the corresponding bit in the EMAC0_IAHT1–EMAC0_IAHT4 or EMAC0_GAHT1–EMAC0_GAHT4 registers that corresponds to the selected category.

Figure 19-14 on page 19-23 shows the details of the hardware mapping algorithm. The example depicts multiple individual address mode, but with changes can be used for the multicast address mode.

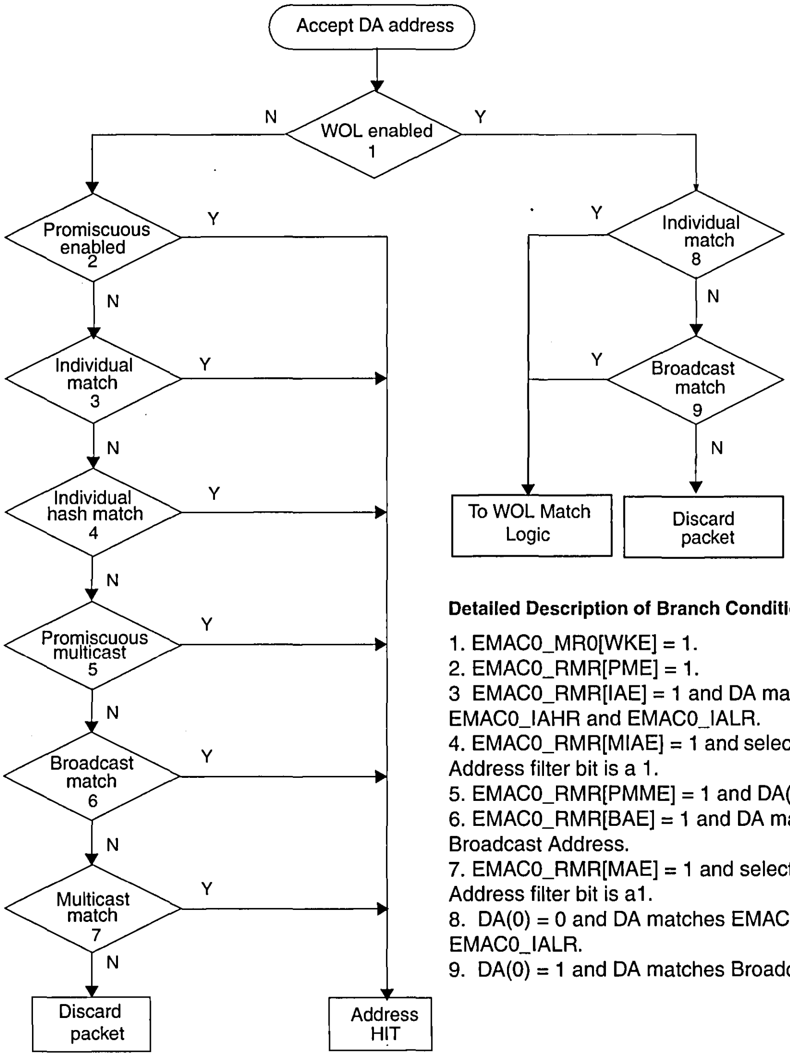
If the most significant bit of an incoming address is a 0, the address is individual and is passed to the individual address filter. If the most significant bit of an incoming address is a 1, the address is multicast and is passed to the multicast address filter. The individual/multicast address filter is a 64-bit mask composed of EMAC0_IAHT1–EMAC0_IAHT4 or the EMAC0_GAHT1–EMAC0_GAHT4 registers (each register contains 16 bits of a 64-bit mask). The incoming address is sent through the FCS circuit. After the 48 address bits have gone through the FCS circuit, the high-order six bits of the resulting FCS (32-bit CRC) are used to select one of the 64-bit positions in the individual/multicast address filter. If the selected filter bit is a 1, the address is accepted.

Note: The individual/multicast address filter ensures only that there is a possibility that the incoming packet belongs to this node. To determine if the packet belongs to the node, the incoming individual/multicast address propagated to the main memory is compared by software to the list of logical addresses to be accepted by this node.

For software, the task of mapping an individual/multicast address to one of 64 bit positions requires a program that uses the same CRC algorithm to calculate the hash.

19.6.3.2 WOL Mode

In WOL mode (EMAC0_MR0[WKE] = 1), EMAC operates only with the broadcast or individual address in the destination address field.



Detailed Description of Branch Conditions

1. EMAC0_MR0[WKE] = 1.
2. EMAC0_RMR[PME] = 1.
3 EMAC0_RMR[IAE] = 1 and DA matches EMAC0_IAHR and EMAC0_IALR.
4. EMAC0_RMR[MIAE] = 1 and selected Individual Address filter bit is a 1.
5. EMAC0_RMR[PMME] = 1 and DA(0) = 1.
6. EMAC0_RMR[BAE] = 1 and DA matches Broadcast Address.
7. EMAC0_RMR[MAE] = 1 and selected Multicast Address filter bit is a1.
8. DA(0) = 0 and DA matches EMAC0_IAHR and EMAC0_IALR.
9. DA(0) = 1 and DA matches Broadcast Address.

Figure 19-13. Receive Address Recognition Flowchart

The example in Figure 19-14 shows that the received individual address maps into category 24, and that bit 8 in EMAC0_IAHT2 is set. The match indication is activated and the packet should be accepted.

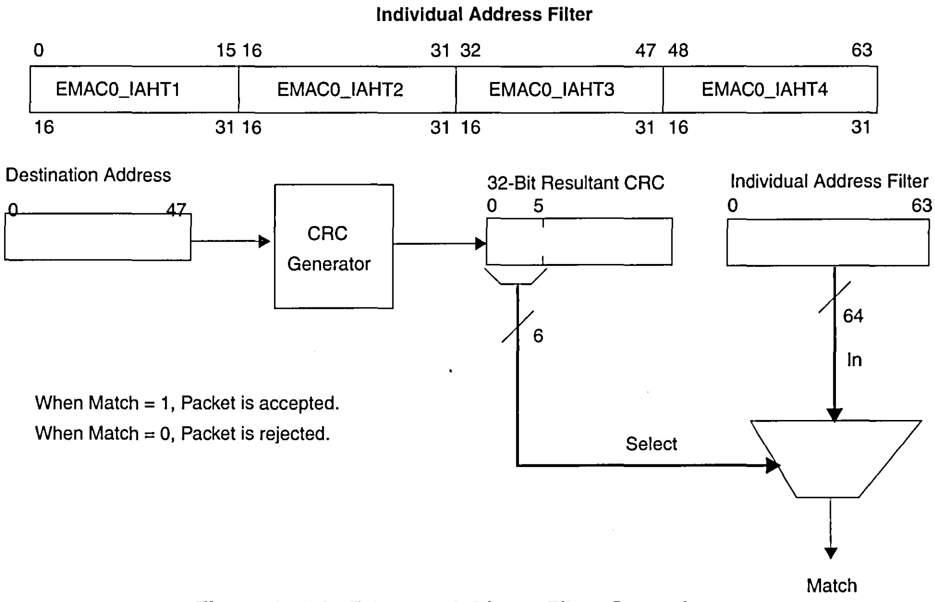


Figure 19-14. Ethernet Address Filter Operation

19.7 EMAC Registers

This section describes the EMAC registers. The EMAC registers are accessed through the OPB. Access to the registers should be word-aligned.

Table 19-5. EMAC Register Summary

Register Name	Address	Write Access	Power-on Reset Value	Access	Page
EMAC0_MR0	0xEF600800	See description in “Scenario 1” on page 19-45	0xC0000000	R/W	19-27
EMAC0_MR1	0xEF600804	Reset	0x00000000	R/W	19-25
EMAC0_TMR0	0xEF600808	See description on page 19-27	0x00000000	R/W	19-27
EMAC0_TMR1	0xEF60080C	See description on page 19-28	0x380F0000	R/W	19-28
EMAC0_RMR	0xEF600810	Reset	0x00000000	R/W	19-29
EMAC0_ISR	0xEF600814	Always	0x00000000	R/W	19-30
Note: Refer to “Power-Up and Initialization” on page 19-44 for definitions of letters in the Write Access Mode column.					

Table 19-5. EMAC Register Summary (continued)

Register Name	Address	Write Access	Power-on Reset Value	Access	Page
EMAC0_ISER	0xEF600818	Reset	0x00000000	R/W	19-33
EMAC0_IAHR	0xEF60081C	Reset, R, T	0x00000000	R/W	19-35
EMAC0_IALR	0xEF600820	Reset, R, T	0x00000000	R/W	19-36
EMAC0_VTPID	0xEF600824	Reset, R, T	0x00008808	R/W	19-36
EMAC0_VTCI	0xEF600828	Reset, R, T	0x00000000	R/W	19-36
EMAC0_PTR	0xEF60082C	Reset, T	0x0000FFFF	R/W	19-37
EMAC0_IAHT1	0xEF600830	Reset, R	0x00000000	R/W	19-37
EMAC0_IAHT2	0xEF600834	Reset, R	0x00000000	R/W	19-37
EMAC0_IAHT3	0xEF600838	Reset, R	0x00000000	R/W	19-37
EMAC0_IAHT4	0xEF60083C	Reset, R	0x00000000	R/W	19-37
EMAC0_GAHT1	0xEF600840	Reset, R	0x00000000	R/W	19-37
EMAC0_GAHT2	0xEF600844	Reset, R	0x00000000	R/W	19-37
EMAC0_GAHT3	0xEF600848	Reset, R	0x00000000	R/W	19-37
EMAC0_GAHT4	0xEF60084C	Reset, R	0x00000000	R/W	19-37
EMAC0_LSAH	0xEF600850	Not applicable	0x00000000	R	19-38
EMAC0_LSAL	0xEF600854	Not applicable	0x00000000	R	19-38
EMAC0_IPGVR	0xEF600858	Reset, T	0x00000004	R/W	19-38
EMAC0_STACR	0xEF60085C	See description on page 19-39	0x00008000	R/W	19-39
EMAC0_TRTR	0xEF600860	See description on page 19-40	0x00000000	R/W	19-40
EMAC0_RWMR	0xEF600864	Reset	0x04001000	R/W	19-41
EMAC0_OCTX	0xEF600868	Not applicable	0x00000000	R	19-42
EMAC0_OCRX	0xEF60086C	Not applicable	0x00000000	R	19-42

Note: Refer to “Power-Up and Initialization” on page 19-44 for definitions of letters in the Write Access Mode column.

19.7.1 Mode Register 0 (EMAC0_MR0)

EMAC0_MR0 defines the operating modes of the EMAC that can be changed at any time during EMAC operation.

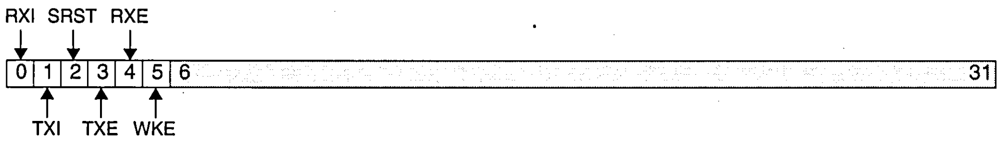


Figure 19-15. Mode Register 0 (EMAC0_MR0)

0	RXI	Receive MAC Idle 0 RX MAC processing packet 1 RX MAC idle; RX packet processing complete	Read-only
1	TXI	Transmit MAC Idle 0 TX MAC processing packet 1 TX MAC idle; TX packet processing complete	Read-only
2	SRST	Soft Reset 0 No effect 1 Soft reset in progress	If EMAC0_MR0[SRST] = 1, writing to any EMAC register, and reading any other bit in this register, is not supported.
3	TXE	Transmit MAC Enable 0 TX MAC is disabled 1 TX MAC is enabled	
4	RXE	Receive MAC Enable 0 RX MAC is disabled 1 RX MAC is enabled	
5	WKE	Wake-Up Enable 0 Incoming packets are not examined for wake-up packet 1 Examine incoming packets for wake-up packet	Software can change EMAC0_MR0[WKE] only while EMAC0_MR0[RXI] = 1 and EMAC0_MR0[RXE] = 0.
6:31		Reserved	

19.7.2 Mode Register 1 (EMAC0_MR1)

EMAC0_MR1 defines the EMAC operating modes that can be changed only after a reset.

Software can use EMAC0_MR1[FDE] and proper programming of the attached PHY to activate external loop-back mode.

When EMAC0_MR1[FDE, ILE] = 1, EMAC wraps transmitted packets back to the receive FIFO without accessing the MII interface.

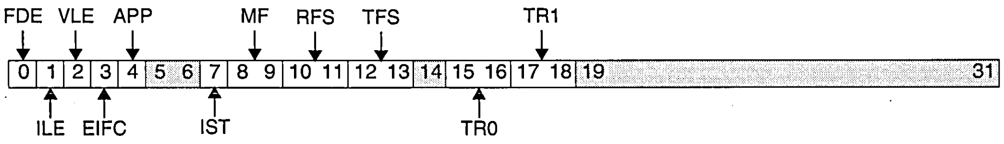


Figure 19-16. Mode Register 1 (EMAC0_MR1)

0	FDE	Full-Duplex Enable 0 Disable simultaneous transmit and receive 1 Enable simultaneous transmit and receive	
1	ILE	Internal Loop-back Enable 0 No wrap back 1 Transmitted packets wrapped back to receive FIFO	Full Duplex must also be set (EMAC0_MRI[FDE]=1).
2	VLE	VLAN Enable 0 Disable processing of VLAN Tags 1 Enable processing of VLAN Tags	
3	EIFC	Enable Integrated Flow Control 0 Disable integrated flow control mechanism 1 Enable integrated flow control mechanism	Refer to “Flow Control” on page 19-15 for more details. Set EMAC0_MR1[EIFC] = 0 in half-duplex mode.
4	APP	Allow Pause Packet 0 Disables processing of incoming control (pause) packets 1 Enables processing of incoming control (pause) packets	
5:6		Reserved	Always zero
7	IST	Ignore SQE test 0 Wait for end of SQE test period before activation of valid signal 1 Do not wait for end of SQE test period before activation of valid signal	EMAC0_MR1[IST] = 0 only during half-duplex operation on 10 Mbps media.
8:9	MF	Medium Frequency 00 10 Mbps (Ethernet mode) 01 100 Mbps (Fast Ethernet mode) 10 Reserved 11 Reserved	Defines the possible operational frequency on the MII interface.
10:11	RFS	Receive (RX) FIFO Size 00 512 bytes 01 1 KB 10 2 KB 11 4 KB	

12:13	TFS	Transmit (TX) FIFO Size 00 Reserved 01 1 KB 10 2 KB 11 Reserved	
14		Reserved	Always zero
15:16	TR0	Transmit Request 0 00 Single packet 01 Multiple packets 10 Dependent mode (bits 17:18 must also be programmed to 10) 11 Reserved	Defines the different modes for using transmit channel 0 of EMAC.
17:18	TR1	Transmit Request 1 00 Single packet 01 Multiple packets 10 Dependent mode (bits 15:16 must also be programmed to 10) 11 Reserved	Defines the different modes for using transmit channel 1 of EMAC.
19:31		Reserved	

19.7.3 Transmit Mode Register 0 (EMAC0_TMR0)

EMAC0_TMR0 defines EMAC operating modes during transmit operations (see "EMAC Transmit Operation" on page 19-5).

EMAC0_TMR0[GNP0, GNP1, GNPD] are self-clearing. Writing 0 to these fields has no effect.

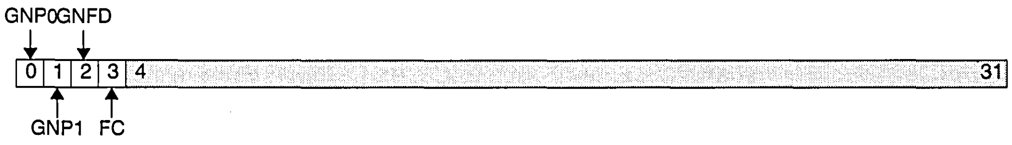


Figure 19-17. Transmit Mode Register 0 (EMAC0_TMR0)

0	GNP0	Get New Packet 0 0 Writing 0 has no effect. 1 Packet ready for transmission on TX Channel 0	EMAC0_TMR0[GNP0] = 0 if EMAC is programmed in dependent mode.
1	GNP1	Get New Packet 1 0 Writing 0 has no effect. 1 Packet ready for transmission on TX Channel 1	EMAC0_TMR0[GNP1] = 0 if EMAC is programmed in dependent mode.
2	GNPD	Get New Packet for Dependent Mode 0 Writing 0 to this bit has no effect 1 Packet ready for transmission in dependent mode	EMAC0_TMR0[GNPD] = 0 if EMAC is not programmed in dependent mode. EMAC0_TMR0[GNPD] = 1 activates the EMAC transmit path in dependent mode.

3	FC	First Channel 0 Activate TX Channel 0 first when GNPD is 1 1 Activate TX Channel 1 first when GNPD is 1	EMAC0_TMR0[FC] is only meaningful in dependent mode, after resetting EMAC0_ISR[DBDM]. EMAC0_TMR0[FC] = 0 if EMAC is not programmed in dependent mode.
4:31		Reserved	

19.7.4 Transmit Mode Register 1 (EMAC0_TMR1)

EMAC0_TMR1 defines conditions for activation of MAL service requests during transmit operations (see “EMAC Transmit Operation” on page 19-5).

19.7.4.1 Low-Priority Requests

EMAC requests low priority service from MAL when the number of vacant entries in the TX FIFO exceeds the decimal TLR value.

EMAC0_TMR1[TRL] must at least equal ((MAL Burst Limit / 2)-1). For example, if MAL supports 16-word bursts, the decimal TLR value should be at least 7.

Note: In the 405GP, all MAL channels are capable of 16 word bursts.

To avoid a deadlock, the sum of EMAC0_TMR1[TRL] and EMAC0_TRTR[TRT] must be at least 4 smaller than the transmit FIFO size specified by EMAC0_MR1[TFS].

19.7.4.2 Urgent-Priority Requests

EMAC requests urgent priority service from MAL if the following conditions occur:

- EMAC begins transmitting the packet to the media before the entire packet is placed in the TX FIFO
- The number of vacant entries for the currently transmitting packet exceeds the decimal TUR value

Software must coordinate the value of EMAC0_TMR1[TUR] with the value of EMAC0_MR1[TFS].The value of EMAC0_TMR1[TUR] must be smaller than that of EMAC0_MR1[TFS] so that the array address encoded in EMAC0_TMR1[TUR] can access the full 66-bit wide array.

The binary value of EMAC0_TMR1[TUR] must be greater than that of EMAC0_TMR1[TLR].

The EMAC0_TMR1 contents can be changed only when EMAC0_TMR0[GNP0, GNP1, GNPD] = 0.

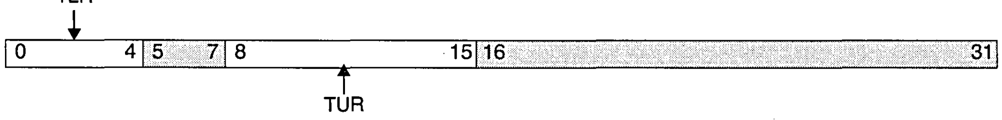


Figure 19-18. Transmit Mode Register 1 (EMAC0_TMR1)

0:4	TLR	Transmit Low Request
5:7		Reserved
8:15	TUR	Transmit Urgent Request

16:31		Reserved

19.7.5 Receive Mode Register (EMAC0_RMR)

The EMAC0_RMR defines EMAC operating modes during receive operations.

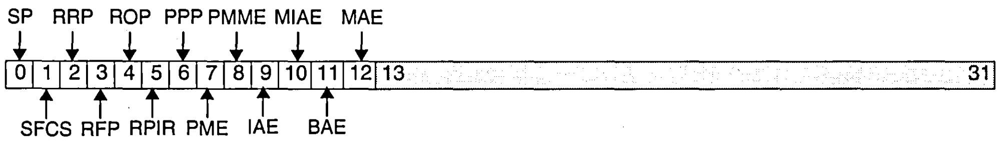


Figure 19-19. Receive Mode Register (EMAC0_RMR)

0	SP	Strip Padding 0 Do not strip pad bytes from the received packet. 1 Strip pad/FCS bytes from the received packet.
1	SFCS	Strip FCS 0 Do not strip FCS bytes from the received packet. 1 Strip FCS bytes from the received packet.
2	RRP	Receive Runt Packets 0 Discard packets less than 64 bytes in length. 1 Receive packets less than 64 bytes in length.
3	RFP	Allow Receive Packets with a FCS Error 0 Discard packets containing a FCS error. 1 Receive packets containing a FCS error.
4	ROP	Receive Oversize Packet 0 Discard packets that activate Packet Is Too Long error. 1 Receive packets that activate Packet Is Too Long error.
5	RPIR	Receive Packets with In Range Error 0 Discard packets that activate In Range Error. 1 Receive packets that activate In Range Error.
6	PPP	Propagate Pause Packet 0 Do not propagate incoming pause packet to MAL; remove packet from FIFO. 1 Propagate incoming pause packet to MAL.

7	PME	Promiscuous Mode Enable 0 Do not enable promiscuous mode. 1 Accept all packets.
8	PMME	Promiscuous Multicast Mode Enable 0 Do not accept all multicast packets. 1 Accept all multicast packets.
9	IAE	Individual Address Enable 0 Do not compare address of received packets with content of individual address register. 1 Compare address of received packets with content of individual address register.
10	MIAE	Multiple Individual Address Enable 0 Do not compare address of received packets with hash table of individual addresses. 1 Compare address of received packets with hash table of individual addresses.
11	BAE	Broadcast Address Enable 0 Do not compare address of received packets with broadcast addresses. 1 Compare address of received packets with broadcast addresses.
12	MAE	Multicast Address Enable 0 Do not compare address of received packets with multicast addresses. 1 Compare address of received packets with multicast addresses.
13:31		Reserved

19.7.6 Interrupt Status Register (EMAC0_ISR)

EMAC generates one distinct interrupt event indication. The event indication is driven out of the EMAC to the UIC Interrupt 15. This interrupt is generated from the content of the EMAC0_ISR. The content of the EMAC0_ISR is first ANDed with the corresponding mask bits in the EMAC0_ISER; the resulting bits are then logically ORed to produce the interrupt signal. Thus, if any of the resulting bits is a 1, an interrupt is generated.

Note: EMAC activates its interrupt signal only after an indication that status for the current packet was accepted by MAL (with the exception of "MMA Operation Succeed/MMA Operation Failed," which causes unconditional activation of interrupt, if it is not masked).

The interrupt indication is cleared by writing 1 to the related bit in the EMAC0_ISR; writing 0 has no effect.

The Event Indication Signal is cleared when all non-masked event indication bits are cleared.

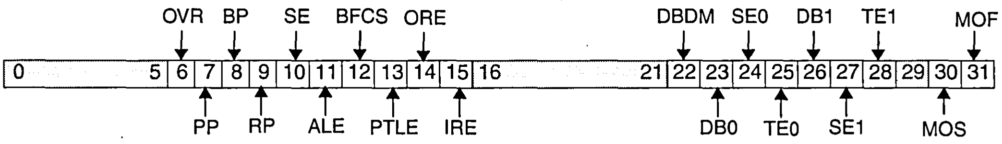


Figure 19-20. Interrupt Status Register (EMAC0_ISR)

0:5		Reserved	
6	OVR	Overrun 0 No overrun error 1 Overrun error during reception of recent packet	
7	PP	Pause Packet 0 Received packet is not a control pause packet 1 Received packet is a control pause packet	
8	BP	Bad Packet 0 Receive operation OK 1 Early termination was initiated because of a packet error	
9	RP	Runt Packet 0 No Runt packets received 1 Runt packet received	Set when EMAC0_RMR[RRP] = 1 and the duration of PHY_RX_DV signal was greater than ShortEventMaxTime constant and less than the collision window.
10	SE	Short Event 0 No short events 1 Duration of PHY_RX_DV signal less than ShortEventMaxTime constant	
11	ALE	Alignment Error 0 No alignment error in received packet 1 Alignment error in received packet	The packet contained an odd number of nibbles (4 bits).
12	BFCS	Bad FCS 0 No FCS error in received packet 1 Packet with an FCS error received	Set if EMAC0_RMR[RFP] = 1.
13	PTLE	Packet Too Long Error 0 No oversized packets received 1 Oversized packet received	Set if EMAC0_RMR[ROP] = 1 and the received packet length exceeded the maximum allowed value: • 1518 octets for standard packet (checked only if the length/type field of the transmitted packet contained length value and jumbo support is disabled) • 1522 octets for VLAN tagged packet (checked only if the length/type field of the transmitted packet contained length value and jumbo support is disabled)

14	ORE	Out Of Range Error 0 Received packet length field value OK 1 Received packet length field value greater than the maximum allowed LLC data size	Indicates that received packet has a length field value greater than the maximum allowed Logical Link Control (LLC) data size (greater than 1500 and less than 1536).
15	IRE	In Range Error 0 Received packet does not contain an In Range Error 1 Received packet contains an In Range Error	
16:21		Reserved	
22	DBDM	Dead Bit Dependent Mode 0 No transmit error or SQE in dependent mode 1 Transmit error or SQE has occurred while in dependent mode	If EMAC0_ISR[DBDM] = 1, EMAC does not request MAL service, even if EMAC0_TMR0[GNPD] = 1. EMAC0_ISR[DBDM] does not affect EMC_INT.
23	DB0	Dead Bit 0 0 No transmit error or SQE for TX Channel 0 while not in dependent mode 1 Transmit error or SQE has occurred for TX Channel 0 while not in dependent mode	If EMAC0_ISR[DB0] = 1, EMAC does not request service for TX Channel 0 from MAL, even if EMAC0_TMR0[GNP0] = 1. EMAC0_ISR[DB0] does not affect EMC_INT.
24	SE0	SQE Error 0 0 No SQEs on TX Channel 0 1 SQE test failure during transmission of a packet from TX Channel 0	Applicable only in half-duplex mode during 10 Mbps operations; 0 in all other modes.
25	TE0	Transmit Error 0 0 TX Channel 0 transmission OK 1 TX Channel 0 transmission aborted	EMAC aborts the transmitted packet if one of the following events takes place: • Late collision detection • Excessive collision detection • Excessive deferral • TX FIFO underrun • Loss of carrier sense
26	DB1	Dead Bit 1 0 No transmit error or SQE for TX Channel 1 while not in dependent mode 1 Transmit error or a SQE has occurred for TX Channel 1 while not in dependent mode	If this bit is set, EMAC does not request MAL service for TX Channel 1 even if EMAC0_TMR1[GNP1] = 1. EMAC0_ISR[DB1] does not affect EMC_INT.
27	SE1	SQE Error 1 0 No Signal Quality Errors on TX Channel 1 1 Signal Quality Error test failure during transmission of a packet from TX Channel 1	Applicable only in half-duplex mode during 10 Mbps operations; 0 in all other modes.

28	TE1	Transmit Error 1 0 TX Channel 1 transmission OK 1 TX Channel 1 transmission aborted	EMAC aborts the transmitted packet if one of the following events takes place: • Late collision detection • Excessive collision detection • Excessive deferral • TX FIFO underrun • Loss of carrier sense
29		Reserved	Always 0
30	MOS	MMA Operation Succeeded 0 MMA_CONTROL addressed on the OPB 1 PHY transfer valid	The device driver should poll assertion of EMAC0_ISR[MOS] or EMAC0_ISR[MOF] before issuing a new command or before using data read from the PHY.
31	MOF	MMA Operation Failed 0 MMA_CONTROL addressed on the OPB 1 PHY transfer *not* valid	The device driver should poll assertion of EMAC0_ISR[MOF] or EMAC0_ISR[MOS] before issuing a new command or before using data read from the PHY.

19.7.7 Interrupt Status Enable Register (EMAC0_ISER)

The EMAC0_ISER indicates which conditions in the EMAC0_ISR can generate an interrupt.

Each masking bit in the EMAC0_ISER corresponds to a related bit in the EMAC0_ISR. If a mask bit is set to 1, the corresponding status bit, when set, causes an interrupt to be generated. Setting a mask bit to 0 suppresses interrupt generation for the associated condition.

Mask bits for reserved bits in the EMAC0_ISR are not implemented, have no effect on write, and return 0 on read.

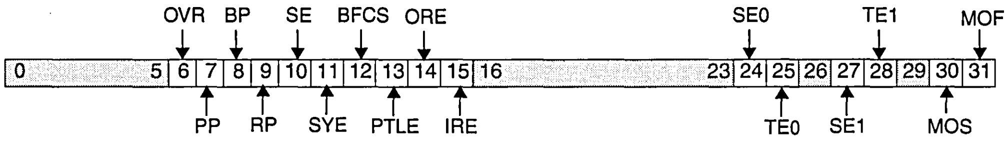


Figure 19-21. Interrupt Status Register (EMAC0_ISER)

0:5		Reserved
6	OVR	Overrun 0 Overrun error will not generate an interrupt. 1 Overrun error will generate an interrupt.
7	PP	Pause Packet 0 Received control pause packet will not generate an interrupt. 1 Received control pause packet will generate an interrupt.

8	BP	Bad Packet 0 Early termination on received packet will not generate an interrupt. 1 Early termination on received packet will generate an interrupt.
9	RP	Runt Packet 0 Received runt packet will not generate an interrupt. 1 Received runt packet will generate an interrupt.
10	SE	Short Event 0 Short event during receive will not generate an interrupt. 1 Short event during receive will generate an interrupt.
11	ALE	Alignment Error 0 Alignment error in received packet will not generate an interrupt. 1 Alignment error in received packet will generate an interrupt.
12	BFCS	Bad FCS 0 FCS error in received packet will not generate an interrupt. 1 FCS error in received packet will generate an interrupt.
13	PTLE	Packet Too Long Error 0 Oversized packets received will not generate an interrupt. 1 Oversized packet received will generate an interrupt.
14	ORE	Out Of Range Error 0 Out of range error on received packet will not generate an interrupt. 1 Out of range error on received packet will generate an interrupt.
15	IRE	In Range Error 0 In range error on received packet will not generate an interrupt. 1 In range error on received packet will generate an interrupt.
16:23	▨	Reserved
24	SE0	SQE Error 0 0 SQE error on TX Channel 0 will not generate an interrupt. 1 SQE error on TX Channel 0 will generate an interrupt.

25	TE0	Transmit Error 0 0 TX error on TX Channel 0 will not generate an interrupt. 1 TX error on TX Channel 0 will generate an interrupt.
26		Reserved
27	SE1	SQE Error 1 0 SQE error on TX Channel 1 will not generate an interrupt. 1 SQE error on TX Channel 1 will generate an interrupt.
28	TE1	Transmit Error 1 0 TX error on TX Channel 1 will not generate an interrupt. 1 TX error on TX Channel 1 will generate an interrupt.
29		Reserved
30	MOS	MMA Operation Succeeded 0 Successful MMA Operation with a PHY will not generate an interrupt. 1 Successful MMA Operation with a PHY will generate an interrupt.
31	MOF	MMA Operation Failed 0 Unsuccessful MMA Operation with a PHY will not generate an interrupt. 1 Unsuccessful MMA Operation with a PHY will generate an interrupt.

19.7.8 Individual Address High (EMAC0_IAHR)

EMAC0_IAHR contains the high-order halfword of the station unique individual address.

During packet reception, if EMAC is programmed in individual address match mode (EMAC0_RMR[IAE] = 1), the contents of EMAC0_IAHR are concatenated with the content of EMAC0_IALR to form a composite address that is compared with the destination address of the received packet. If addresses match, the packet is transferred to MAL.

During packet transmission, EMAC0_IAHR is used in source address inclusion/replacement and as the source address field in the self-assembled control (pause) packet.

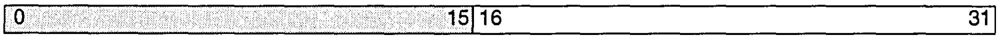


Figure 19-22. Individual Address High Register (EMAC0_IAHR)

0:15		Reserved	
16:31		High-order halfword of the station unique individual address	This field contains bits 0:15 of the destination address (bit 0 is the most significant bit).

19.7.9 Individual Address Low (EMAC0_IALR)

EMAC0_IALR contains the low-order word of the station unique individual address.

During packet reception, EMAC0_IALR is compared with the corresponding address bits of the received packet.

During packet transmission, EMAC0_IALR is used in source address inclusion/replacement and as the source address field in the self-assembled control (pause) packet.

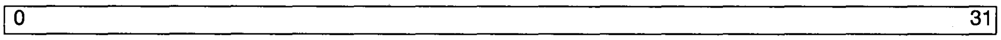


Figure 19-23. Individual Address Low Register (EMAC0_IALR)

0:31		Low-order bits of Receive Individual Address or Transmit Source Address

19.7.10 VLAN TPID Register (EMAC0_VTPID)

EMAC0_VTPID contains the value of the VLAN TPID (Tag Protocol Identifier) field.

During packet reception, packet bytes 13 and 14 are compared to the content of this register to check whether the packet is tagged with a VLAN ID.

During packet transmission, EMAC uses EMAC0_VTPID when VLAN Tag replacement or VLAN Tag inclusion mode is chosen.

The value of this register must be a Type field (8100).

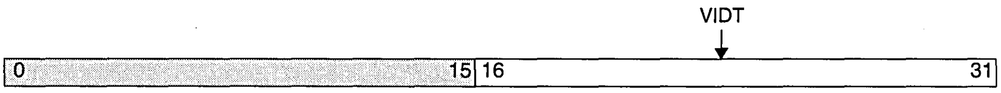


Figure 19-24. VLAN TPID Register (EMAC0_VTPID)

0:15	▨	Reserved
16:31	VIDT	VLAN ID tag

19.7.11 VLAN TCI Register (EMAC0_VTCI)

EMAC0_VTCI contains the value of the VLAN TCI (Tag Control Information) field.

During packet transmission, EMAC uses EMAC0_VTCI when VLAN Tag replacement or VLAN Tag inclusion mode is chosen.

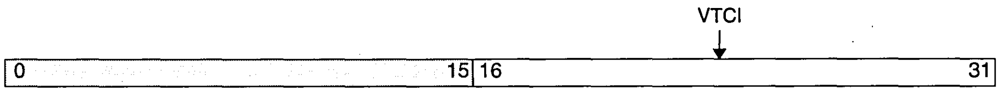


Figure 19-25. VLAN TCI Register (EMAC0_VTCI)

0:15		Reserved
16:31	VTCI	VLAN TCI tag

19.7.12 Pause Timer Register (EMAC0_PTR)

The EMAC0_PTR defines the time period for which the pause function is enabled. EMAC uses EMAC0_PTR[TVR] as the timer value field of control (pause) packets (see "Control Packet Transmission" on page 19-16). Each bit corresponds to 512 bit times.

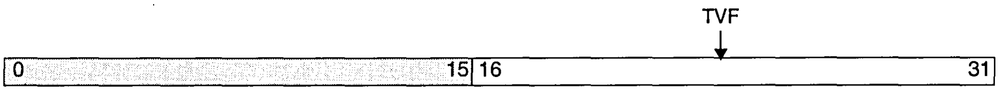


Figure 19-26. Pause Timer Register (EMAC0_PTR)

0:15		Reserved
16:31	TVF	Timer Value Field

19.7.13 Individual Address Hash Tables 1–4 (EMAC0_IAHT1–EMAC0_IAHT4)

These registers are used in the hash table function of the multiple individual addressing mode.

See "Address Match Mechanism" on page 19-20 for more information. See Figure 19-14 on page 19-23 for bit mapping information.

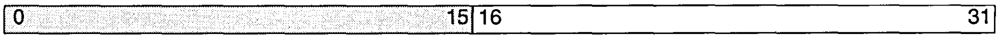


Figure 19-27. Individual Address Hash Tables 1–4 (EMAC0_IAHT1–EMAC0_IAHT4)

0:15		Reserved
16:31		Individual Address Hash Number

19.7.14 Group Address Hash Tables 1–4 (EMAC0_GAHT1–EMAC0_GAHT4)

These registers are used in the hash table function of the group addressing mode.

See "Address Match Mechanism" on page 19-20 for more information. See Figure 19-14 on page 19-23 for bit mapping information.

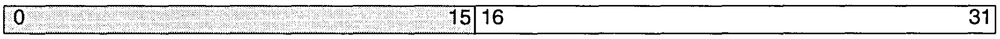


Figure 19-28. Group Address Hash Tables 1–4 (EMAC0_GAHT1–EMAC0_GAHT4)

0:15		Reserved
16:31		Group Address Hash Number

19.7.15 Last Source Address High (EMAC0_LSAH)

EMAC0_LSAH contains the high-order halfword of the source address of the last "good" received packet. The packet is considered to be "good" if EMAC is programmed to provide this packet to MAL.

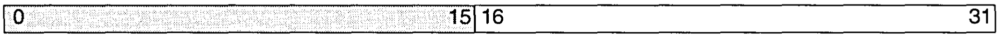


Figure 19-29. Last Source Address High Register (EMAC0_LSAH)

0:15		Reserved
16:31		Last Source Address High-Order Halfword

19.7.16 Last Source Address Low (EMAC0_LSAL)

EMAC0_LSAL contains the low-order word of the source address of the last "good" received packet. The packet is considered "good" if EMAC is programmed to provide this packet to MAL.

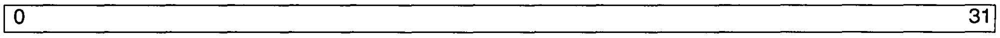


Figure 19-30. Last Source Address Low Register (EMAC0_LSAL)

0:31		Last Source Address Low-Order Word

19.7.17 Inter-Packet Gap Value Register (EMAC0_IPGVR)

EMAC0_IPGVR contains the value of one-third of the inter-packet gap (IPG) for the next packet to be transmitted. ("Frame" is synonymous with "packet.")

The resolution of each bit is 8-bit times. The minimum value in the register is four, causing a minimum IPG period of 96-bit times).

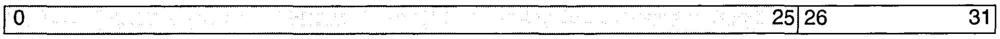


Figure 19-31. inter-Packet Gap Value Register (EMAC0_IPGVR)

0:25		Reserved
26:31		Inter-Packet Gap

19.7.18 STA Control Register (EMAC0_STACR)

The EMAC0_STACR controls the MII Management interface. The software must follow the following steps during access to the EMAC0_STACR:

1. Software polls EMAC0_STACR[OC], waiting for it to be set by EMAC.

   EMAC sets EMAC0_STACR[OC] = 0 when the EMAC0_STACR is written to.

   EMAC then sets EMAC0_STACR[OC] = 1 to indicate that the data has been written to the PHY, or the data read from the PHY is valid. The device driver should poll for EMAC0_STACR[OC] = 1 before issuing a new command, or before using data read from the PHY.
2. The software can perform read/write access to the EMAC0_STACR.
3. EMAC clears EMAC0_STACR[OC] (sets EMAC0_STACR[OC] = 0) and starts activity on the MII management interface.
4. Return to step 1.

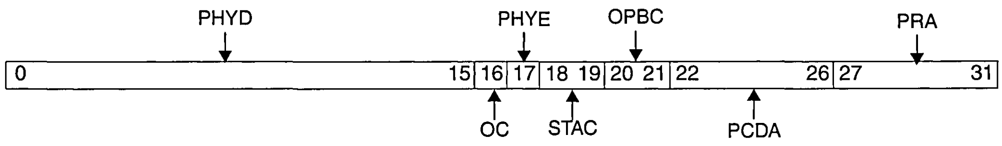


Figure 19-32. STA Control Register (EMAC0_STACR)

0:15	PHYD	PHY data	Data to be sent to the PHY if the command is a write, or data is read from the PHY if the command is a read.
16	OC	Operation Complete 0 EMAC0_STACR is addressed 1 PHY data transfer complete	
17	PHYE	PHY Error 0 Successful read transaction 1 Read transaction was not successful	EMAC0_STACR[PHYE] = 0 when a read is successful.
18:19	STAC	STA Command 00 Reserved 01 Read 10 Write 11 Reserved	EMAC sets EMAC0_STACR[STAC] = 0 when the command is completed.

20:21	OPBC	OPB Bus Clock Frequency 00 50 MHz 01 66 MHz 10 83 MHz 11 100 MHz	EMAC0_STACR[OPBC] is used to generate the Management Data Clock (EMCMDClk. When the operational frequency differs from those in the list, then the next greater frequency should be chosen.
22:26	PCDA	PHY Command Destination Address	
27:31	PRA	PHY Register Address	

19.7.19 Transmit Request Threshold Register (EMAC0_TRTR)

The EMAC0_TRTR defines the conditions that cause EMAC to initiate transmission to the Ethernet MAC sub-block, and for requesting service from MAL.

EMAC0_TRTR[TRT] defines the number of occupied entries in the Transmit FIFO that should be written before the Transmit FIFO control logic initiates a transmit request to the Ethernet MAC sub-block.

If an entire packet is already located in the Transmit FIFO, then EMAC initiates a transmit regardless of the programmed value.

The software must coordinate the value of EMAC0_TRTR[TRT] with the Transmit FIFO size specified in EMAC0_MR1[TFS].

To avoid deadlock, the sum of the Transmit Low Request (Figure 19-18 on page 19-28, bits 0:4) and EMAC0_TRTR[TRT] must be smaller, by at least four, than the chosen size of the Transmit FIFO specified in EMAC0_MR1[TFS].

To avoid an underrun, program this threshold to a high enough value.

In half-duplex mode, in case of collision, to allow packet re-transmission without involving MAL, EMAC preserves the necessary space in the Transmit FIFO unless it gets an indication that the collision window has elapsed.

The EMAC0_TRTR may only be written to while EMAC0_MR0[TXI] = 1.

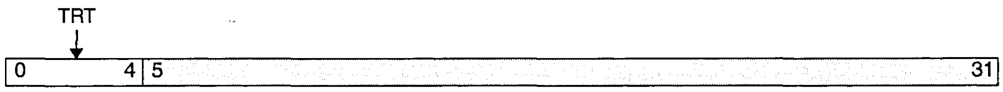


Figure 19-33. Transmit Request Threshold Register (EMAC0_TRTR)

0:4	TRT	Transmit Request Threshold The following number of bytes must be placed in the Transmit FIFO before initiating a transmit request. 00000 64 bytes 00001 128 bytes 00010 192 bytes 00011 256 bytes . . . 11111 2048 bytes
5:31		Reserved

19.7.20 Receive Low/High Water Mark Register (EMAC0_RWMR)

The EMAC0_RWMR defines the conditions that cause EMAC to activate a low or urgent priority MAL request, and that manage flow control.

EMAC activates a low priority request if the number of occupied entries in the Receive FIFO is greater than or equal to the content of EMAC0_RWMR[RLWM] (the receive low water mark is reached). A request for a pause packet with a pause_value of 0 is also issued when the receive low water mark is reached.

Software must coordinate the value of EMAC0_RWMR[RLWM] with the value of EMAC0_MR1[RFS]. EMAC0_RWMR[RLWM] should be smaller than EMAC0_MR1[RFS] and larger than the MAL burst length.

Note: In the 405GP, the MAL burst length is 16 words for all channels.

If the entire packet is already in the Receive FIFO, EMAC initiates a low priority request regardless of the programmed value.

EMAC activates an urgent priority request if the number of occupied entries in the Receive FIFO is greater than or equal to EMAC0_RWMR[RHWM] (the receive high water mark is reached). A request for a pause packet is also issued when the receive high water mark is reached.

Software must coordinate the value of EMAC0_RWMR[RHWM] with the value of EMAC0_MR1[RFS]. EMAC0_RWMR[RHWM] should be greater than the value of EMAC0_RWMR[RLWM] and less then the size of the Receive FIFO.

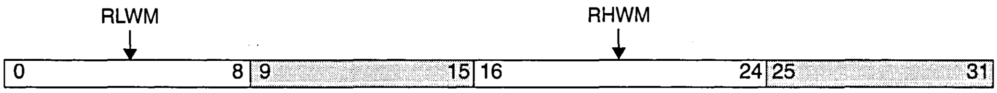


Figure 19-34. Receive Low/High Water Mark Register (EMAC0_RWMR)

0:8	RLWM	Receive Low Water Mark
9:15		Reserved
16:24	RHWM	Receive High Water Mark
25:31		Reserved

19.7.21 Number of Octets Transmitted (EMAC0_OCTX)

Figure 19-35. Number of Octets Transmitted (EMAC0_OCTX)

0:31	OCTX	Number of octets (bytes) transmitted.	This field is Read-Only.

19.7.22 Number of Octets Received (EMAC0_OCRX)

Figure 19-36. Number of Octets Received (EMAC0_OCRX)

0:31	OCRX	Number of octets (bytes) received.	This field is Read-Only.

19.8 MII Interface

EMAC implements all MII interface functionality in accordance with Clause 22 in the IEEE Std. 802.3u.

The MII interface is a Reconciliation Sublayer interface which allows a variety of PHYs to be attached to the EMAC Ethernet MAC without future upgrade problems.

19.8.1 MII Station Management Interface

The EMAC MII station management unit (STA) implements a specific protocol and a special packet format to exchange management packets with the registers of the attached PHY device. EMAC automatically generates MII management packets, which conform to Clause 22 in IEEE Std. 802.3u. EMAC uses the EMAC0_STACR for generation of the management packet.

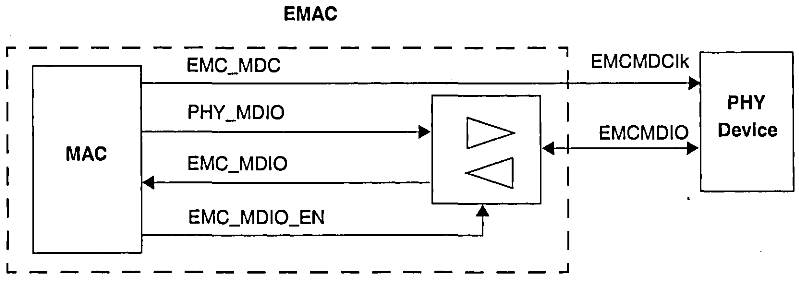


Figure 19-37. Management Interface with PHY

19.8.2 EMAC – MII Interface

See PPC405GP Data Sheet for information.

19.9 MAL – EMAC Packet Transfer Flow

The packet transfer flow consists of three phases. These three phases are used to define the details of the EMAC-MAL protocol.

1. Packet phase - EMAC initiates a packet transfer operation. The packet transfer is started by a command write. During command write MAL provides control information for EMAC on a per-packet basis. Following the command write, MAL begins the data transfer, during which MAL transfers data between the buffers located in the system's memory and EMAC. In transmit, the data is transferred from the system's memory to EMAC, while in receive, the data is transferred from EMAC to the system's memory buffers.
   - EMAC remains in the packet phase until the data transfer has been completed or a ready status can be returned to MAL. The packet phase ends when EMAC deasserts the FRAME signal associated with the related channel (receive/transmit).
   - The packet phase is defined by activity of an appropriate FRAME signal.
2. Status phase - This is the second phase of the packet transfer. Following the de-assertion of the FRAME signal, EMAC switches to the status phase. At this stage, EMAC uses an appropriate signal as a request for service which is interpreted by MAL as a request for status read.
3. Idle phase - EMAC moves into the idle phase following a reset or after status was transferred (end of status phase). During the idle phase, EMAC cannot send any signals to MAL, nor can MAL send any active signals to EMAC. EMAC exits the idle phase by asserting the FRAME signal (and entering the packet phase described above). Idle phase can be skipped when EMAC operates in multiple transfer mode.

Figure 19-38 illustrates the different phases in the EMAC-MAL communication.

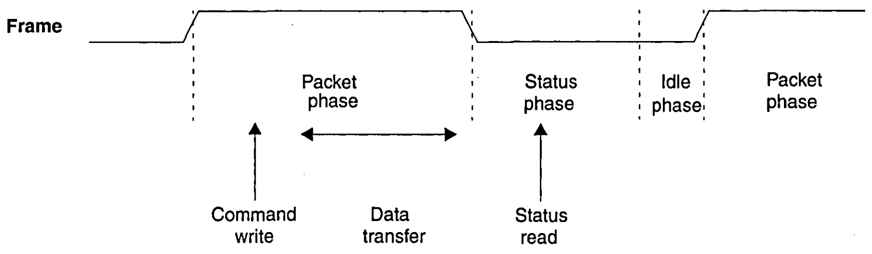


Figure 19-38. EMAC-MAL Communication Phases

During the packet and status phases EMAC signals a request for service by driving its arbitration level signal to a non-idle level.

19.10 Programming Notes

Certain combinations in device drivers are not allowed when writing to EMAC registers. When creating device drivers, ensure that the following guidelines are used:

- In dependent mode, EMAC0_MR1[TR0] must be equal EMAC0_MR1[TR1]
- When internal loopback is enabled (EMAC0_MR1[ILE] = 1), EMAC must be configured in full-duplex mode (EMAC0_MR1[FDE] =1)
- EMAC0_MR1[IST] =0 only when EMAC0_MR1[MF] = 10 and EMAC0_MR1[FDE] =0
- In dependent mode, EMAC0_ISER$_{24,25}$ must equal EMAC0_ISER$_{27,28}$
- EMAC0_MR1[EIFC] =0 if EMAC0_MR1[FDE] =0
- EMAC0_TMR1[TLR] must be greater than the MAL burst size in entities (6 for MAL)
- EMAC0_TMR1[TUR] must be greater or equal to EMAC0_TMR1[TLR] and less than the Transmit FIFO size in entries (EMAC0_MR1[TFS])
- To avoid deadlock, the sum of EMAC0_TMR1[TLR] and the EMAC0_TRTR[TRT] must be at least four less than the Transmit FIFO size specified in EMAC0_MR1[TFS]
- EMAC0_RWMR[RLWM] must be greater than the MAL burst size in entities (six for MAL)
- EMAC0_RWMR[RHWM] must be greater than EMAC0_RWMR[RLWM]
- EMAC0_RWMR[RHWM] must be less than the Receive FIFO size in entities (EMAC0_MR1[RFS])

19.10.1 Power-Up and Initialization

19.10.1.1 Reset Options

The EMAC must be reset before performing configuration changes. The following types of reset operations can be applied to EMAC.

- Hard Reset. When RESET input is asserted, EMAC aborts all on-going activities unconditionally, initializes all internal state machines, counters, registers, and flushes transmit and receive FIFOs. To be recognized, the reset signal must be asserted for at least two cycles of the slowest clock domain inside EMAC (indicating that the hard reset must be at least 800 ns).
- Soft Reset. Software first should reset the appropriate MAL channels and then begin a soft reset by setting EMAC0_MR0[SRST] = 1. In response to the soft reset, EMAC aborts all on-going activities unconditionally, initializes all internal state machines, counters, registers, and flushes transmit and receive FIFOs. After EMAC finishes all activities related to the soft reset processing, EMAC0_MR0[SRST] = 0.
- Smart Reset. The software initializes smart reset mode by writing 0 to EMAC0_MR0[TXE] or EMAC0_MR0[RXE], or to both. In this case, the Ethernet MAC sub-block completes on-going activity (receive, transmit, or both) and then goes to the related Idle state (indicated by setting either EMAC0_MR0[TXI] = 1 or EMAC0_MR0[RXI] = 1, or both). In this case, the control logic sub-block of EMAC is still accessible for OPB and MAL transactions.

Before performing the necessary configuration changes in EMAC, the software must follow one of the following scenarios. Then the EMAC can be properly configured.

19.10.1.2 Scenario 1

- Hard/soft reset was activated.
- During hard/soft reset, EMAC0_MR0[TXE] and EMAC0_MR0[RXE] are reset.
- Software detects that the EMAC0_MR0[SRST] is reset (after soft reset only).
- Software keeps EMAC0_TMR0[GNP0, GNP1] = 0.
- The software can change one or more fields in registers marked with a Reset write access mode in Table 19-5, "EMAC Register Summary," on page 19-23 (actually, all EMAC registers are accessible in this scenario).
- The software initializes EMAC0_TMR0[GNP0, GNP1] as appropriate.
- The software configures EMAC0_MR0[TXE, RXE].

19.10.1.3 Scenario 2

- Software sets EMAC0_MR0[TXE] = 0.
- The TXMAC component of the Ethernet MAC sub-block completes on-going activity and then sets EMAC0_MR0[TXI] = 1 to enter the related Idle state.
- Software detects EMAC0_MR0[TXI] = 1.
- Software performs the necessary EMAC configuration, keeping EMAC0_MR0[TXE] = 0. The software can access only part of the EMAC registers marked with write access mode T in Table 19-5, "EMAC Register Summary," on page 19-23.
- After all configuration is done, software can set EMAC0_MR0[TXE] = 1.

Note: When Scenario 2 occurs, EMAC can still receive packets if EMAC0_MR0[RXE] = 1. Scenarios 2 and 3 can occur simultaneously.

19.10.1.4 Scenario 3

- Software sets EMAC0_MR0[RXE] = 0.
- The RXMAC component of the Ethernet MAC sub-block completes on-going activity and then sets EMAC0_MR0[RXI] = 1 to enter the related Idle state.
- Software detects EMAC0_MR0[RXI] = 1.
- Software performs the necessary EMAC configuration, keeping EMAC0_MR0[RXE] = 0. The software can access only part of EMAC registers marked with write access mode R in Table 19-5, "EMAC Register Summary," on page 19-23.
- After all configuration is done, software can set EMAC0_MR0[RXE] = 1.

Note: When Scenario 3 occurs, EMAC can still transmit packets if EMAC0_MR0[TXE] = 1. Scenarios 2 and 3 can occur simultaneously.

Chapter 20. Memory Access Layer

The Memory Access Layer (MAL) is a hardware core that manages data transfers between packet-oriented communications cores, also known as COMMACs (communications media access controllers), and memory. In the PPC405GP, MAL manages the transfer of packets between the Ethernet Media Access Controller (EMAC) and memory attached to the PPC405GP (SDRAM or SRAM). The primary function of MAL is to move packets directly between memory and a COMMAC core to minimize involvement of the processor core.

MAL is comprised of multiple transmit (TX) and receive (RX) channels, each of which is dedicated to a specific COMMAC in the chip. In the PPC405GP, the EMAC utilizes MAL TX channels 0 and 1, and MAL RX channel 0.

To communicate with software device drivers, MAL utilizes a buffer descriptor ring structure in memory. A software device driver uses the buffer descriptor structure to inform MAL about buffer locations and packet or buffer status. MAL uses the buffer descriptors to convey packet transfer status from the COMMAC core back to the software device driver. Each MAL channel requires its own buffer descriptor table ring structure in memory.

MAL provides software device drivers a generic interface for control of:

- Configuration sequence
- Activation commands
- Deactivation commands
- Memory status handling

20.1 MAL Features

- No restrictions on buffer alignment
- Aligned bus accesses to enable burst operation with external memories
- Configurable receive buffer size (configurable per channel)
- No minimum transmit buffer size
- Maximum buffer sizes of 4095 bytes (TX) and 4080 bytes (RX)
- Up to 256 descriptors in the buffer descriptor table per channel
- Configures COMMAC according to commands specified in the descriptor status/control field
- Updates the descriptor status/control field at the end of packet transfer according to the status received from COMMAC
- Buffer-based interrupt capabilities for each channel
- Concurrent operation of RX and TX channels
- Configuration using Device Control Registers (DCRs)
- Programmable PLB arbitration priority
- PLB/OPB error detection

Figure 20-1 illustrates a general system structure overview of an embedded PowerPC processor core integrated with a packet oriented communication core. For the PPC405GP, the sole COMMAC is EMAC.

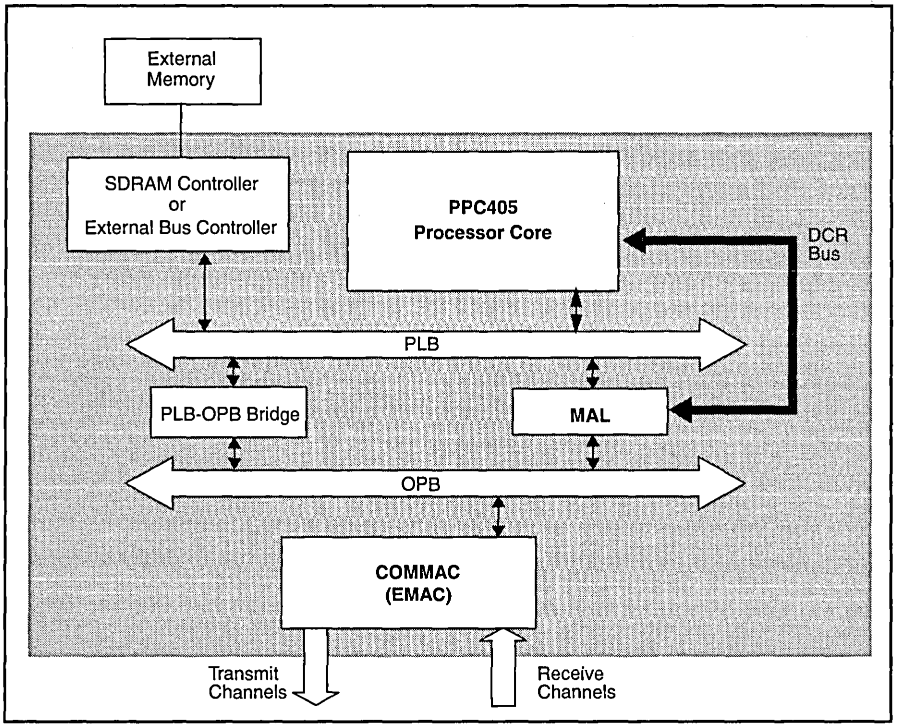


Figure 20-1. General PPC405GP Structure (Overview)

COMMACs are configured and controlled by the processor core using the OPB without MAL intervention. Packet data to be transmitted and received are stored in buffers in external memory. The MAL processes buffer descriptors and provides all data access facilities to the COMMACs.

The MAL is not aware of COMMACs such as EMAC as an entity. It is only aware of the COMMAC's channels. In the PPC405GP, EMAC contains two TX channels and one RX channel. Transmit and receive operations can be performed simultaneously by MAL (full duplex). When a channel wins arbitration, MAL transfers data between system memory and the COMMAC. MAL and the software driver maintain separate, dedicated buffer descriptor tables for each channel to maintain channel, packet, and buffer status. Packets can be constructed from one data buffer, or several data buffers (known as buffer chaining).

20.1.1 MAL – Internal Structure

Figure 20-2 illustrates the MAL internal structure.

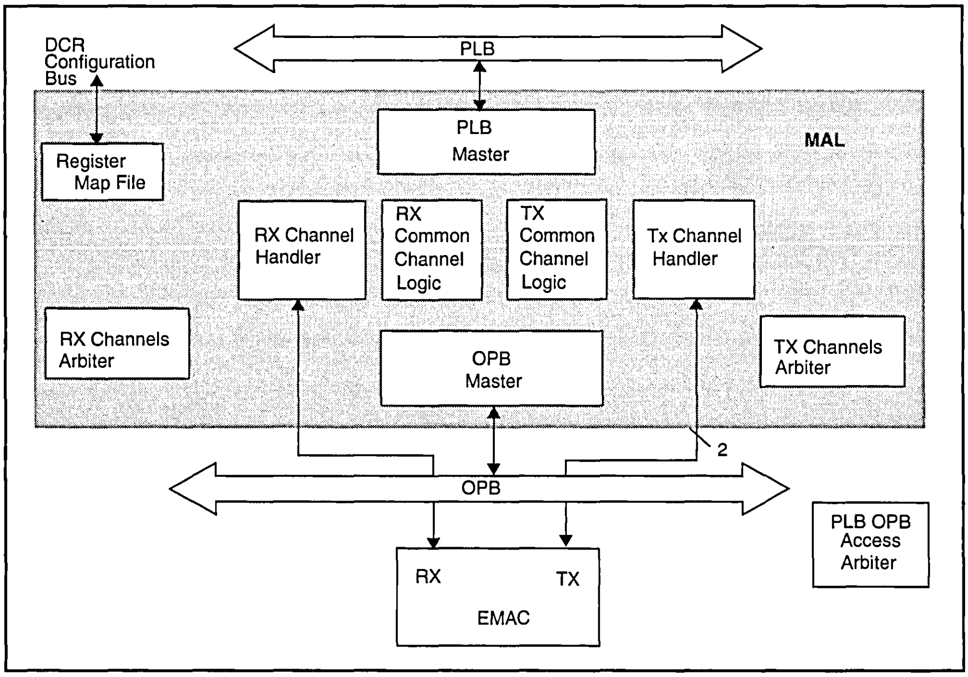


Figure 20-2. MAL Internal Structure

20.1.1.1 PLB Master

The PLB Master performs PLB transactions for MAL, and is used to transfer data between a COMMAC and memory, fetch buffer descriptors, and communicate status regarding data transfer.

20.1.1.2 OPB Master

The OPB Master performs OPB transactions for MAL, and is used to transfer data between a COMMAC and memory.

20.1.1.3 TX Channel Handler

The TX channel handler is a dedicated section for each TX channel. It keeps a record of the descriptor information and the current state of each channel.

20.1.1.4 RX Channel Handler

The RX channel handler is a dedicated section for each RX channel. It keeps a record of the descriptor information and the current state of each channel.

20.1.1.5 TX Channel Arbiter

The TX channel arbiter, connected to request lines from each TX channel, arbitrates between the TX channels and decides which channel gains access to the TX common channel logic.

20.1.1.6 RX Channel Arbiter

The RX channel arbiter, connected to request lines from each RX channel, arbitrates between the RX channels and decides which channel gains access to the RX common channel logic.

20.1.1.7 TX Common Channel Logic

The TX common channel logic is shared by all TX channels. It services a single TX channel at a time (selected by the TX arbiter). This logic activates the PLB and OPB masters for data and buffer descriptor transactions.

20.1.1.8 RX Common Channel Logic

The RX common channel logic is shared by all RX channels. It services a single RX channel at a time (selected by the RX arbiter). This logic activates the PLB and OPB masters for data and buffer descriptor transactions.

20.1.1.9 Register Map File

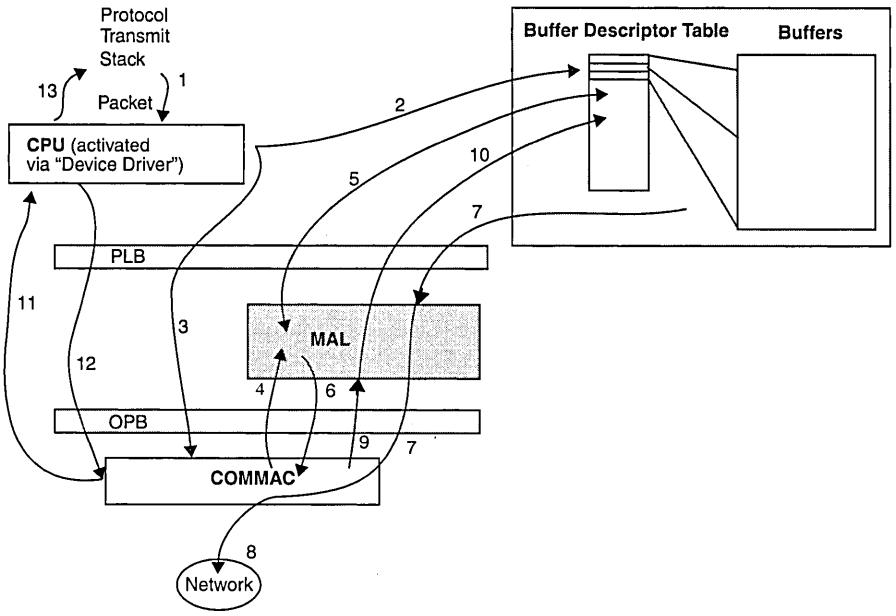
The register map file is used to configure MAL and read its status registers. Software accesses the MAL register file using the **mtdcr** and **mfdcr** instructions.

20.2 Transmit and Receive Operations

The device driver is responsible for configuring MAL before a COMMAC can begin requesting MAL to process packets of data. The device driver should ensure that channels are not enabled during reconfiguration; otherwise, fatal errors may occur.

For more information about the MAL software interface, see “MAL Programming Notes” on page 20-18.

Figure 20-3 describes the software and hardware operations involved in a typical transmit operation.

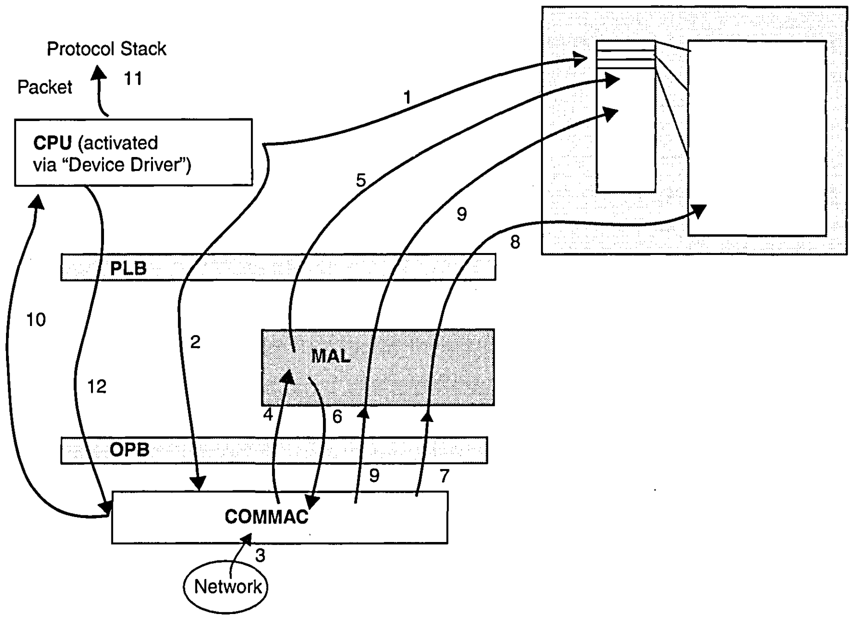


The numbered steps are described as follows:

1. The protocol stack (high-level software layer) initiates a packet transmit.
2. Software device driver parses the protocol stack buffer into descriptor table entries and buffers.
3. Software device driver instructs the COMMAC to process a new transmit packet.
4. The COMMAC channel requests MAL to process a new packet.
5. MAL fetches descriptor information.
6. MAL writes control information into the COMMAC and initiates the data move.
7. Packet data is transferred from memory into the COMMAC (the COMMAC controls the pace of the data transfer).
8. The packet is transmitted on the media (steps 7 and 8 can overlap).
9. The COMMAC requests that MAL read the packet status.
10. The status read by MAL is written back into a buffer descriptor.
11. Software is interrupted (if interrupt conditions are met) by the COMMAC or by the MAL end-of-buffer interrupt.
12. Software clears the interrupt status bits in the COMMAC and in MAL.
13. Software informs the protocol stack that transmission is complete.

Figure 20-3. Transmit Operation

Figure 20-4 describes the software and hardware operations when receiving a typical packet.



The numbered steps are described as follows:

1. Software device driver initializes the receive buffer descriptors.
2. Software device driver enables the COMMAC to process a new packet.
3. A packet is received from the network (steps 2 and 3 can change in order).
4. The COMMAC channel requests that MAL process a new packet.
5. MAL fetches receive buffer information from the descriptor table.
6. MAL writes the control word from the descriptor to the COMMAC and initiates the data transfer.
7. The COMMAC channel fills its FIFO storage.
8. MAL stores the packet in system memory buffers pointed to by the descriptors.
9. MAL reads status information from the COMMAC and writes it to the buffer descriptors.
10. Software device driver is interrupted (if interrupt conditions are met) by the COMMAC or by the MAL end-of-buffer interrupt.
11. The receive packet is passed to the protocol stack.
12. Software clears the receive buffer descriptor positions allowing them to be used again.

Figure 20-4. Receive Operation

Note: The description in Figure 20-4 is the general scheme for MAL operation in the software environment. The device driver should follow recommendations from "MAL Programming Notes" on page 20-18.

20.3 Buffer Descriptor Overview

The software interface for buffer descriptor (BD) processing consists of a set of registers within MAL and a set of circular queues in memory. Each transmit and receive COMMAC channel has a descriptor table that contains buffer location and status information allocated to the channel.

Note: Since MAL uses a flat addressing scheme on the PLB, the physical memory that holds descriptor tables and buffers can be allocated anywhere in the address space where memory is possible. Also, it is not necessary to place buffer descriptor tables and buffers in the same physical memory.

During its operation, MAL is able to modify the contents of memory directly without processor core knowledge. If the processor core does not provide hardware enforced data cache coherency or data cache snooping (the PPC405 CPU core does not), data cache coherency is the responsibility of the software device driver. To simplify device driver software, the MAL buffer descriptor tables should be placed in non-cached memory when possible. If this is not possible, the software driver must maintain cache coherency of the buffer descriptor tables by performing data cache flushes or invalidates when appropriate. When descriptors are in cached memory, the driver software must be aware that multiple descriptors are present in a single cache line and that cache invalidate or flush operations will be performed on multiple descriptors at the same time. This is significant because a cache line flush done by the driver to force a descriptor from the data cache to physical memory could corrupt another descriptor that occupies the same data cache line and is simultaneously being updated in physical memory by MAL.

Data buffers, in contrast, should be placed in cachable memory if possible. The software driver can easily maintain cache coherency of data buffers if:

- All buffers are aligned on a data cache line boundary
- All buffers are a multiple of a data cache line in size

Note: The data cache line size and alignment in the PPC405GP is 32 bytes.

Before using a received packet, the software driver must invalidate the memory occupied by the buffer in the data cache for the length specified in the RX buffer descriptor data length field. Before transmitting a packet the software driver must flush the data buffer from the data cache before setting the Ready bit in the TX buffer descriptor.

The software device driver fills the buffers pointed to by transmit buffer descriptors with packets to be transmitted, and/or provides empty buffers pointed to by receive buffer descriptors to be filled with received packets. Meanwhile, the hardware processes the descriptors, transfers the packet data to/from the COMMAC, and updates the status fields of the descriptors.

Each individual transmit or receive channel has its own buffer descriptor table. They are managed independently of each other. This section describes the individual transmit and receive interfaces.

Packet data associated with each transmit or receive channel is stored in buffers. Each buffer has an entry dedicated to it in one of the channel's buffer descriptor tables. MAL has a Channel Table Pointer Register for each of its channels. The COMMAC (EMAC in the PPC405GP) device driver sets the contents of these registers to point to the starting address of the buffer descriptor table for the associated channel.

Note: Buffer descriptor tables must start on a 4-KB boundary.

The buffer descriptor table forms a circular queue with a programmable length. The last descriptor in the table is defined by setting the Wrap bit in the status/control field (see "Status/Control Field Format" on page 20-14). If there is no Wrap bit set in the table, then MAL automatically wraps after processing 256 descriptors (the maximum number of descriptors allow per channel).

The format of the buffer descriptor (see Figure 20-5) is the same for all COMMACs, and has the same structure for both transmit and receive. The most significant halfword in each buffer descriptor contains a status/control halfword. This field contains two parts: the first part (6 bits) is BD handling information used by the MAL for descriptor processing, the second field (10 bits) is content specific for each COMMAC. The second halfword determines the data length referenced in this buffer descriptor. The second word in the buffer descriptor contains a 32 bit data buffer pointer that points to the actual data buffer in memory. It is suggested that each data buffer start on a cache line boundary and be a multiple of a cache line in size if it resides in cachable memory. (The cache line size in the PPC405 processor core is 32 bytes.)

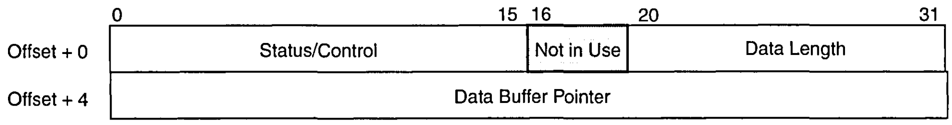


Figure 20-5. Buffer Descriptor Structure

A packet may reside in as many buffers as necessary (transmit or receive). Each buffer has a maximum length of (4KB – 16) bytes. In TX channels, the buffer descriptor length field is written by the device driver and defines the number of bytes in the data buffer that is identified by the data buffer pointer. In RX channels, the buffer descriptor length field is written by MAL and defines the number of bytes written by MAL to the buffer that is identified by the data buffer pointer (see "Receive Software Interface" on page 20-12).

When processing a packet, MAL does not assume that all buffers of the current packet are already valid. It expects the buffers to be ready in due time to be transmitted or received. Failure of the software to provide the descriptors in due time may result in an error.

Figure 20-6 describes the structure of the packet in memory.

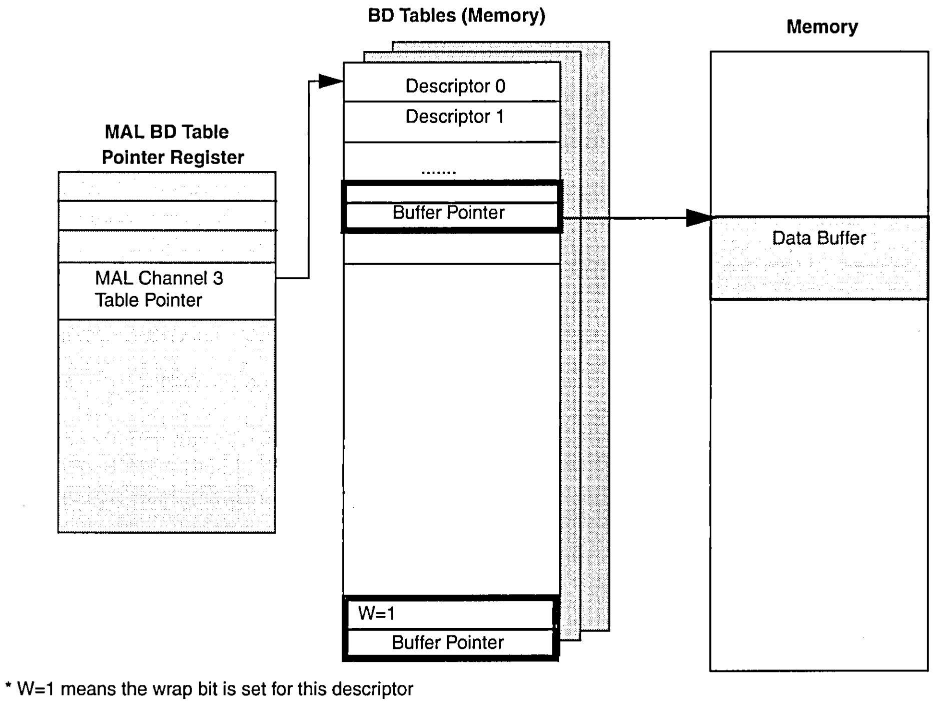

BD Tables (Memory)
Memory
MAL BD Table Pointer Register
MAL Channel 3 Table Pointer
Descriptor 0
Descriptor 1
....
Buffer Pointer
W=1
Buffer Pointer
Data Buffer
* W=1 means the wrap bit is set for this descriptor

Figure 20-6. Packet Memory Structure

20.4 Transmit Software Interface

Once a channel is enabled in MAL (this is done by setting the appropriate bit in the Channel Active Register), a channel may request service from MAL. When the first transmit request comes in from a COMMAC TX channel, MAL finds the starting address of the buffer descriptor table for the channel by looking in the corresponding Transmit Channel Table Pointer Register. If the first descriptor is marked as ready, MAL will start processing the associated buffer.

When MAL begins processing a packet, it writes the contents of the descriptor status/control field into the COMMAC. This information, (depending on communication core implementation), may be used by the communication core to configure each packet transfer.

Once all data from the current buffer has been transferred to the communication core on the channel, MAL moves on to the next buffer descriptor in the table.

If a given buffer descriptor indicates that it contains the last section of the current packet, MAL informs the channel that the last data transferred to the channel completed the transfer of a data packet. At this point, the COMMAC asks MAL to read the packet status. MAL then writes this information back into the status/control field of the last buffer descriptor of the packet.

The COMMAC channel may request that MAL process the next buffer descriptor and the same packet handling process will be initiated. The first descriptor in the next packet follows the descriptor marked "last" in the previous packet.

20.4.1 Wrapping the BD Table for Transmit

When MAL processes a buffer descriptor (while handling a packet for a COMMAC channel), it may encounter a Wrap indication within a buffer descriptor control field. This causes MAL to go back to the beginning of the buffer descriptor table for the next descriptor table entry. (This will also happen when MAL reaches the maximum number of descriptors.) The wrapping of the BD table, like all other BD table handling processes, is transparent to the COMMAC.

20.4.2 Continuous Mode for Transmit

After transmitting the data pointed to by a buffer descriptor, MAL clears the Ready bit in the buffer descriptor control/status field. In this way, MAL will not process the same buffer descriptor again until software has filled the buffer with valid data and set the Ready bit in the descriptor again. While the Continuous Mode (CM) bit is set in the status/control field, MAL will not clear the Ready bit. The Continuous Mode allows re-transmission of the current data buffer without software intervention. This mode is generally used by protocols in which frequent re-transmission is an integral part of the protocol itself. In such cases, re-transmission can be performed without software intervention.

20.4.3 Back Up a Packet for Transmit

MAL is capable of re-transmitting the last packet ("back up a packet") following a request from a COMMAC. If re-transmission is requested by the COMMAC, it must be assured that all the buffers of the re-transmitted packet are available and were not re-processed by the device driver. In regular operation, MAL resets the Ready bit of each buffer descriptor when finished processing the descriptor. When MAL is requested by the COMMAC to retransmit the last packet (the Back Up a Packet bit in the COMMAC TX channel Status Halfword is set), MAL doesn't reset the READY bit in the last processed buffer descriptor, activate the end of packet interrupt, or write the status back to the descriptor in the memory. MAL also doesn't consider this as an end of packet event.

On the next service request from the same channel, MAL will start transmitting the packet again, starting from the first descriptor.

Note: The last processed buffer descriptor can be either the last descriptor of the packet or, in case of early packet termination, the buffer descriptor that was being processed when the transmit channel initiated the early packet termination. MAL will retransmit the backed-up packet regardless of the Ready bit value.

During retransmission of a backed-up packet, MAL may use descriptors on which the Ready bit was already cleared. Therefore, the device driver should not reuse descriptors before the Ready bit of the last descriptor is cleared.

Note: In the case of descriptor not valid, which is the first one in TX channel, COMMAC is not allowed to return a status that contains a Back-Up a Packet request.

20.4.4 Descriptor Not Valid for Transmit

When MAL accesses a buffer descriptor, it checks whether or not the Ready bit is set. If the Ready bit is not set, two cases apply (special treatment of the READY bit is performed in the case of Back-up a packet):

For the case when the READY bit is not set:

- If the descriptor is the first descriptor of the packet MAL informs the channel that data is currently unavailable. Further handling of this scenario is COMMAC-specific. The channel might either instruct MAL to access the same buffer descriptor periodically (by keeping its service request to MAL active) until it becomes ready, or 'give up' on the descriptor, completing the end-of-packet protocol with MAL. The channel might also indicate the buffer descriptor status to the device driver via an interrupt. However, in this case the COMMAC should eventually complete the packet transfer protocol with MAL. Following a descriptor not valid indication, MAL's BD pointer continues pointing to the same location in the BD table. The next time a descriptor read is initiated by the COMMAC, MAL will search for the buffer in the same location.
- If the descriptor is not the first descriptor of the packet, it is considered a descriptor error. MAL deactivates the channel and from its point of view, the processing of the current packet has ended. Software may learn about this situation from one of two MAL interrupts (or from both). The first one is a nonmaskable interrupt that indicates the number of the TX channel, in which the descriptor error had occurred (interrupt bit for each TX channel, see "MAL Interrupt Enable Register (MAL0_IER)" on page 20-31). The second one is a maskable interrupt which indicates a descriptor error event, regardless the channel number (one interrupt bit for all the channels, see "MAL Error Status Register (MAL0_ESR)" on page 20-29). For more information about error handling, see "Error Handling" on page 20-19.

For the case of a back-up packet:

- When the current transmitted packet is a backed-up packet, all descriptors except the last, are valid even if the READY bit is not set. In this case, (not the last descriptor) MAL processes the packet descriptors regardless the READY bit value. If the READY bit of the last descriptor in the backed-up packet is not set, MAL treats it as a descriptor error. MAL handles the descriptor error as described above for the case when the packet isn't a backed-up packet.

20.4.5 Scroll Descriptors for Transmit

MAL may be configured by software, in the case of early packet termination, to scroll in the buffer descriptor table to the first descriptor of the next packet.

When a multiple-buffer packet is terminated early by the COMMAC, while MAL is processing a buffer which is not the last buffer in the packet, MAL can operate in one of the following ways:

The MAL Scroll Descriptor in the configuration register is set:

- In this case MAL will read the status word from the COMMAC channel. Then MAL will reset the READY bit in all the remaining buffer descriptors of the current packet. In addition, MAL will write the status to all the buffer descriptors. On the next service of this channel, MAL will fetch the first descriptor of the next packet.

The MAL Scroll Descriptor in the configuration register is clear:

- In this case MAL will read the Status word from the COMMAC channel. Then MAL will terminate the current channel service by resetting the READY bit of the last processed buffer descriptor (the one in which there was an early termination) and will write the status only to this descriptor. On the next service of this channel, MAL will fetch the next descriptor in the current packet. In this case, the software is responsible to monitor the MAL location in the buffer descriptor table.

In the case that the COMMAC requests a re-transmit of the early terminated packet (when the "backup" bit in the COMMAC status is set), MAL will re-transmit the packet regardless of the MAL Scroll Descriptor bit.

20.5 Receive Software Interface

MAL uses the RX channel buffer descriptors in a manner similar to that used for transmission. Once an RX COMMAC channel requests that a new packet be processed, MAL starts processing the channel's next buffer descriptor in the table. Once a channel is enabled in MAL, the channel may request MAL service. When it does, MAL accesses the first buffer descriptor (in that channel's buffer descriptor table) that is pointed to by the COMMAC channel table pointer register. If that descriptor is ready (empty for RX), MAL will start processing the buffer.

When it begins processing each packet, MAL writes the contents of the status/control field into the COMMAC. This information (defined by COMMAC's implementation) can be used by the COMMAC for a per-packet configuration.

Once data is received from the memory, MAL moves the data from the RX channel FIFO into the data buffer pointed to by the first buffer descriptor. The current buffer descriptor may be closed for two reasons: there is no more room left in the buffer, or the COMMAC channel indicated that the packet reception ended. If additional buffering space is needed for the current packet, MAL moves on to the next buffer descriptor. As each buffer descriptor is closed, MAL updates the length field with the actual amount of bytes written into the buffer. The maximal length of the buffers for each channel is defined by a configuration register. The maximal receive buffer length is defined per channel.

Once the COMMAC channel indicates that the packet reception has ended, it is expected to request that MAL update the received packet status in the BD status/control field. MAL updates the packet status and notifies the COMMAC. At this point the packet is considered received and the COMMAC may request that MAL begin the process of receiving a new packet. The first buffer of the next packet is the buffer in the BD table that followed the last descriptor of the previous packet.

20.5.1 Wrapping the BD Table for Receive

When MAL processes the buffer descriptor, it may encounter a Wrap indication within a buffer descriptor control field. This causes MAL to go back to the head of the channel's buffer descriptor for the next buffer descriptor. This also happens when MAL reaches the maximal number of descriptors.

20.5.2 Continuous Mode for Receive

After using a buffer descriptor, MAL sets the buffer descriptor control to the Not-Empty state. In this way, MAL will not use the same buffer descriptor a second time until the software has processed the not-empty buffer descriptor and set it to Empty again. MAL will not clear the Empty bit while the Continuous Mode (CM) bit is set in the status/control field. The Continuous Mode is generally used by protocols where frequent collisions are an integral part of the protocol itself (forcing the COMMAC to abort a reception process and restart). In such cases, re-reception can be performed without software intervention.

20.5.3 Descriptor Not Valid for Receive

When MAL accesses a buffer descriptor it may find that the Empty bit is not set. In the case of an RX channel descriptor, this situation is considered as a descriptor error. MAL deactivates the channel and from its point of view, the processing of the current packet has ended. Software may learn about this situation from one of two MAL interrupts (or from both):

- An RXDE interrupt with the MAL0_RXDEIR indicating which channel caused the interrupt
- An SERR interrupt (system error) with one interrupt bit for all channels in the MAL0_ESR

For more about error handling, see "Error Handling" on page 20-19.

20.5.4 Buffer Length for Receive

The maximum length of an RX buffer descriptor is predetermined for all RX descriptors in each channel. The data-length value is programmable through a set of MAL registers (see "MAL Registers" on page 20-24). The actual data length field within the RX buffer descriptor is written by MAL. If the buffer is completely filled up, the value written will match the value programmed into the matching RX-Channel-Descriptor data-length register. If the buffer is only partially filled up (for example, when the RX packet ended before running out of buffer space), the actual amount of space filled is written into the length field.

20.6 Descriptor Buffer Status/Control Fields

The following sections details the status/control field bits. The information fields within the status/control field can be divided as follows:

- Information from a software device driver directed to MAL and COMMAC
- Information from MAL and COMMAC directed to software
- Status/control field handling
- Status/control field format
- TX status/control field format
- RX status/control field format

20.6.1 Information from a Software Device Driver Directed To MAL and COMMAC

- MAL-related buffer descriptor processing information:
  - Buffer Ready/Not Ready (determines the buffer's validity).
  - Wrap to top of table or continue to next descriptor.
  - In a transmit buffer descriptor – Is the current buffer the last one in the packet?
  - Continuous or normal mode; that is, should MAL change the Ready/Not Ready value?

- COMMAC channel configuration information:
  - Should the channel generate an interrupt following the end of packet processing.
  - Protocol specific configuration.

20.6.2 Information from MAL and COMMAC Directed to Software

- MAL generated status information:
  - Buffer Ready/Not Ready (passes the buffer handling to software).
  - In receive buffer descriptor - Is the current buffer the first one in the packet?
  - In receive buffer descriptor - Is the current buffer the last one in the packet?
- COMMAC channel generated status information:
  - Protocol specific error and status information (transmit and receive).

20.6.3 Status/Control Field Handling

When MAL accesses a new buffer descriptor, the status/control word is written to the COMMAC channel. This allows the channel to configure itself for the current packet.

For all "intermediate" buffer descriptors (all descriptors that do not contain the packet's ending), the status/control field is written by MAL (rather than the COMMAC). In this case, the status/control field indicates that the current buffer is not the last one in the current packet.

As MAL finishes processing the last buffer descriptor in a given packet, it reads the channel's status (via an OPB transaction) and writes it into the buffer descriptor's status/control field.

In effect, since all of the various control and status fields do not overlap, the status/control halfword is read/written as a whole. Each agent (MAL, COMMAC channel, and software) reads the entire status/control halfword, relates to specific fields of interest, and updates another subset of fields within the same halfword. While an agent modifies its related fields, all other fields remain unchanged.

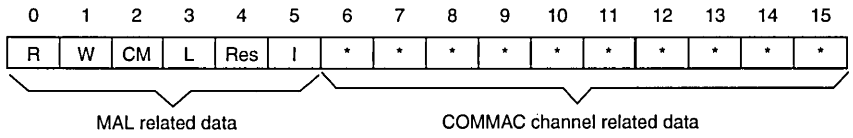
20.6.4 Status/Control Field Format

The status/control halfword is divided into COMMAC channel data and MAL related data. As explained above, the MAL related fields are either aimed at controlling MAL or written by MAL for use by the software. The MAL fields are of no interest to the COMMAC (except the Ready and Empty bits).

The same applies to the COMMAC channel fields. The COMMAC related fields are either aimed at controlling the COMMAC or written by COMMAC for use by the software. These fields are of no interest to MAL.

MAL will not manipulate the COMMAC related fields, and COMMAC is not allowed to manipulate the MAL related fields.

20.6.5 TX Status/Control Field Format


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
R W CM L Res I * * * * * * * * * *
MAL related data
COMMAC channel related data

* - COMMAC specific control or status fields

Figure 20-7. TX Status/Control Field

Note: The bit numbering in Figure 20-7 relates to the Buffer Descriptor's fullword which contains both the status/control and the length fields.

20.6.5.1 Bit 0 – R – Ready

This bit is set by the device driver and is cleared by MAL.

The device driver sets this bit after preparing the buffer for transmission.

MAL clears this bit when finish processing the buffer descriptor. MAL doesn't clear the Ready bit in the case of backing-up a packet request and in case of continuous mode (see "Back Up a Packet for Transmit" on page 20-10 and "Continuous Mode for Transmit" on page 20-10).

20.6.5.2 Bit 1 – W – Wrap

0 – This is not the last data buffer descriptor in the buffer descriptor table.

1 – This is the last data buffer descriptor in the buffer descriptor table. After this buffer has been used, MAL will transmit data from the first descriptor buffer in the table.

This bit is controlled by software only. It controls MAL activities, and does not affect the COMMAC channel.

20.6.5.3 Bit 2 – CM – Continuous Mode

0 – Normal Operation

1 – Continuous Operation. After this buffer descriptor is closed, the R-bit is not cleared by MAL. This ensures that the data buffer is ready for transmission when MAL next accesses this buffer descriptor. However, the R-bit is cleared if an error occurs during transmission.

This bit is controlled by software only. It controls MAL activities and does not affect the COMMAC channel.

20.6.5.4 Bit 3 – L – Last

0 – This is not the last buffer in the current packet.

1 – This is the last buffer in the current packet.

This bit is controlled by software only. It controls MAL activities, and does not affect the COMMAC channel.

20.6.5.5 Bit 4 – Reserved

This bit is reserved. It is assumed that this bit is set to zero by the software.

20.6.5.6 Bit 5 – I – Interrupt

1 – After finishing processing the current buffer, if this bit is 1, the end of buffer field in the End of Buffer Interrupt Status Register is set and the end of buffer interrupt is asserted.

0 – There is no action taken by MAL once it reaches the end of the current buffer.

MAL asserts the end of buffer interrupt after it updates the buffer descriptor's status field.

This bit is controlled by software only. It controls the MAL activities and does not affect the COMMAC.

20.6.5.7 Bits 6 to 15

These bits are COMMAC specific and may contain control fields generated by the software in order to control the COMMAC channel. They may also contain status fields, generated by the COMMAC channel, that will be processed by software.

20.6.6 RX Status/Control Field Format

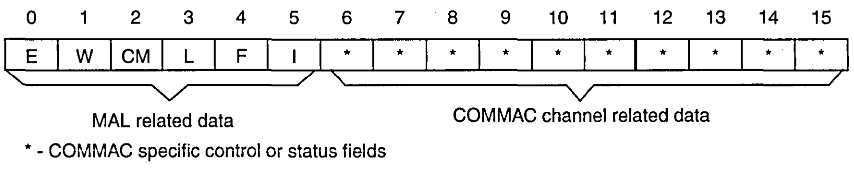

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
E W CM L F I * * * * * * * * * *
MAL related data
COMMAC channel related data
* - COMMAC specific control or status fields

Figure 20-8. RX Status/Control Field

Note: The bit numbering in Figure 20-8 relates to the buffer descriptor's fullword which contains both the status/control and the length fields.

20.6.6.1 Bit 0 – E – Empty

0 – The data buffer associated with this buffer descriptor has been filled with received data, or data reception has been aborted due to an error condition. Software is free to examine or write to any fields of this buffer descriptor. While this bit is set to Not Empty, MAL will not use this buffer descriptor again.

1 – The data buffer associated with this buffer descriptor is empty, or reception is currently in progress. This buffer descriptor and its associated receive buffer are owned by MAL. Once the E-bit is set, software should not write to any fields of this RX buffer descriptor.

MAL clears this bit after the buffer has been filled with received data **or** after an error is encountered. Software sets this bit to Empty after preparing the buffer for reception. This bit controls MAL and software activities. See "Bit 2 – CM – Continuous Mode" on page 20-17.

20.6.6.2 Bit 1 – W – Wrap

0 – This is not the last data buffer descriptor in the buffer descriptor table.

1 – This is the last data buffer descriptor in the buffer descriptor table. After this buffer has been used, MAL will transfer data to the first buffer descriptor in the table.

This bit is controlled by software only. It controls MAL activities and does not affect the COMMAC channel.

20.6.6.3 Bit 2 – CM – Continuous Mode

0 – Normal Operation

1 – Continuous Operation. After this buffer descriptor is closed, the E-bit is not cleared by MAL. This ensures that the data buffer is ready to receive data when MAL next accesses its buffer descriptor. However, the E-bit is cleared if an error occurs during reception.

This bit is controlled by software only. It controls MAL activities and does not affect the COMMAC channel.

20.6.6.4 Bit 3 – L – Last

0 – This is not the last buffer in the current packet.

1 – This is the last buffer in the current packet.

This bit is updated by MAL following the activity of the channel.

20.6.6.5 Bit 4 – F – First

0 – This is not the first buffer in the current packet.

1 – This is the first buffer in the current packet.

This bit is updated by MAL following the activity of the channel.

20.6.6.6 Bit 5 – I – Interrupt

1 – After finish processing the current buffer, if this bit is 1, the end of buffer field in the End of Buffer Interrupt Status Register is set and the end of buffer interrupt is asserted.

0 – No action is taken by MAL at the end of the current buffer.

MAL asserts the end of buffer interrupt after updating the buffer descriptor's status field.

This bit is controlled by software only. It controls MAL activities and does not affect the COMMAC.

20.6.6.7 Bits 6 to 15

These bits are COMMAC-specific and they may contain control fields generated by the software in order to control the COMMAC channel. They may also contain status fields generated by the COMMAC channel to be processed by software.

20.7 MAL Programming Notes

The following sections contain information about programming the MAL.

20.7.1 MAL Initialization

MAL initialization includes two parts: configuration and channel activation.

Configuration involves two steps:

- MAL configuration - This step is done only after a power on reset or after a MAL soft reset. The following registers are involved:
  - Configuration Register (MAL0_CFG). This register defines MAL operation on the PLB and OPB.
  - Interrupt Enable Register (MAL0_IER). This register is used to enable interrupts for various MAL error conditions.
- Channel specific configuration - This information may be changed only when the associated channel is not active. (The bit for the channel in the TX or RX Channel Active Set Register, is cleared.) The following registers are involved:
  - MAL0_RCBSx – RX Buffer Size (one register for each RX channel). This register defines the length of the RX buffers in memory.
  - MAL0_TXCTPxR or MAL0_RXCTPxR – Channel Table Pointer Register (one register for each channel). This register is programmed with the memory address of the first buffer descriptor table entry for the channel.

Setting the channel specific configuration can be done as part of MAL initialization or as part of the COMMAC initialization process. In order to activate a channel, the following actions should be taken:

- The channel has to be configured in MAL
- The related bit in Channel Active Set Register (MAL0_TXCASR or MAL0_RXCASR) has to be set
- The channel operation must be enabled (COMMAC configuration)

20.7.2 Interrupts

MAL has five interrupt lines (in the PPC405GP, all are connected to the UIC). Two interrupt lines, one for TX and one for RX, are used for interrupt events during packet transfer. An additional two interrupt lines, one for TX and one for RX, are used to report descriptor errors on a per-channel basis. The fifth interrupt is used to report MAL errors.

- TXEOB interrupt line is used to report end of buffer or end of packet for a specific TX channel. A bit for the related channel is set in the MAL0_TXEOBISR. See “End of Buffer Interrupt Status Registers” on page 20-28.
- RXEOB interrupt line is used to report end of buffer or end of packet for a specific RX channel. A bit for the related channel is set in the MAL0_RXEOBISR. See “End of Buffer Interrupt Status Registers” on page 20-28.
- TXDE interrupt line is used to indicate a descriptor error event in a specific TX channel descriptor table. A bit for the related channel is asserted in the MAL0_TXDEIR. See “Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)” on page 20-32.

- RXDE interrupt line is used to indicate a descriptor error event in a specific RX channel descriptor table. A bit for the related channel is asserted in the MAL0_RXDEIR. See “Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)” on page 20-32.
- SERR interrupt is used to report a system error indicated by MAL. For more information on handling the SERR interrupts, see “Error Handling” on page 20-19 and “Error Registers” on page 20-29.

20.7.3 Error Handling

MAL handles errors on a per-channel basis. Within a COMMAC channel, errors may arise from the COMMAC (detected as an OPB error), or from the memory access operations involved in MAL activity (detected as a PLB/descriptor error).

When a bus error occurs, MAL is notified by an OPB or PLB error signal. OPB errors are related to a specific channel and therefore channel operation is stopped. In the case of a PLB error, MAL cannot identify which channel is involved, therefore channel operation is not stopped. When a descriptor error occurs, MAL can again identify the channel involved, so channel operation is stopped. MAL stops channel operation by clearing the associated bit in the MAL0_TXCASR or MAL0_RSCASR register.

MAL keeps a record of the channels that experience errors and are made inactive. It also keeps a record of the characteristics of the first (or last) error detected (see “End of Buffer Interrupt Status Registers” on page 20-28).

20.7.3.1 Error Detection

The MAL communication, both with COMMACs and with memory, is carried out via the OPB or PLB. As long as this bus communication is error-free and no descriptor errors are detected, MAL maintains normal activity with the channels set by the processor as active in the Channel Active Registers.

When an error is detected while performing a transfer for a channel, MAL asserts a maskable interrupt signal. If the identity of the channel is known (as is the case for OPB errors or descriptor errors) then MAL immediately halts the dialogue with the channel. No further transactions are made, and that channel is registered by MAL as a nonactive channel. MAL resets the channel by resetting its active bit in the Channel Active Register. Software must access the Channel Active Register in order to reactivate the channel.

If the identity of the channel that caused the error is not known (as is the case for PLB errors) then MAL continues to work normally. Error resolution and channel de-activation are the responsibility of the software.

20.7.3.2 Indicated Errors

Error description is stored in the Error Status Register (MAL0_ESR), (see “MAL Error Status Register (MAL0_ESR)” on page 20-29).

- Descriptor Error

  A descriptor error is a data error recognized during access to the descriptor table. The error can occur during TX or RX transmission.

  For RX channels, a descriptor error occurs when MAL accesses a descriptor in which the Empty bit is cleared.

For TX channels, a descriptor error occurs when MAL accesses a descriptor in which the Ready bit is cleared. The following cases are exceptions.

- On access to the first buffer descriptor in a TX packet.
- On access to a buffer descriptor that is not the last descriptor in a backed-up packet.

As a result of this error, the following actions are taken by MAL:

- The Active bit of the related channel is reset and the channel activity is halted until software reactivates channel activity.
- The associated bit in the TX Descriptor Interrupt Error Register (MAL0_TXDEIR) or RX Descriptor Error Register (MAL0_RXDEIR) is set, causing a nonmaskable TXDE interrupt or RXDE interrupt respectively.
- When the channel is reactivated, MAL points to the descriptor at the head of the BD table.

- OPB Non-Fullword Error

  This error indicates that a non-fullword acknowledge was asserted by a slave.

  Following this error, the active bit of the associated channel is reset and channel activity is halted until it is reactivated by software. When the channel is reactivated, MAL points to the descriptor at the head of the BD table.

- OPB Time-Out Error

  This error indicates that an OPB time-out error was reported by the OPB arbiter.

  Following this error, the active bit of the associated channel is reset and channel activity is halted until reactivated by software. When the channel is reactivated, MAL points to the descriptor at the head of the BD table.

- OPB Error

  This error indicates that an OPB error was detected.

  Following this error, the active bit of the associated channel is reset and channel activity is halted until reactivated by software. When the channel is reactivated, MAL points to the descriptor at the head of the BD table.

- PLB Error

  This error indicates that a PLB error was detected (from the PLB slave).

  In this case, MAL cannot determine which channel caused the error. Therefore, operation is not halted for any of the channels.

20.7.3.3 Error Handling Registers

MAL error handling logic includes two registers.

- Error Status Register (ESR)

  This register holds information about the error that occurred and the interrupt status. The register includes the following fields:

  Error status – This field holds the error information. The information includes the number of the channel on which the error occurred (if known) and the type of the error. The error can be either the last detected error or a locked error if "Locked error mode" is active. See "Operational Error Modes" on page 20-21 for description of the Locked error mode.

The error status field includes an "Error Valid" bit which indicates whether there is valid error information in the error status field or not. The error status field is not valid when the "Error Valid" bit is cleared (by writing 1 to this bit).

Interrupt status – Every error detected by MAL sets a related bit in the interrupt status field. Software can clear an interrupt status bit by writing 1 to the bit to be cleared. The bits in this field are accumulative which allows more than one interrupt to be indicated in the register.

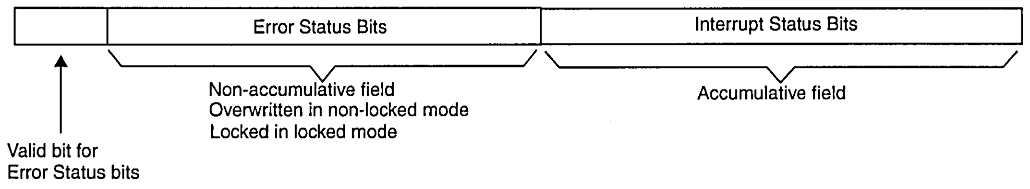


Figure 20-9. Error Status Register Field

20.7.3.4 Operational Error Modes

MAL can operate in two different error handling modes:

- Locked Error Mode: Information about the error is written to the Error Status Register, and the Valid bit in that register is set. Information in the Error Status field of the register stays locked until software unlocks it by resetting the error Valid bit. The Interrupt Status bits of the Error Status Register are not locked in this mode, so software can find out if more errors occur. However, the Error Status field applies only to the first error that is locked.
- Non-Locked Error Mode: Information about the error is written in the Error Status Register, and the error Valid bit is set. Each new error will be overwritten, so the information in the Error Status Field is valid only for the last error that occurred.

In both modes, each error written in the error description field will set the error Valid bit, and it is the responsibility of software to reset this bit.

The error handling mode is programmed in the MAL Configuration Register (see "MAL Configuration Register (MAL0_CFG)" on page 20-25).

20.7.3.5 Resolution of an Error Situation

When MAL encounters an error, it reacts as follows:

- Writes information about the error in the Error Status Register (ESR). This information includes the channel ID of the channel which caused the error (if known), the bus on which the error occurred, and the kind of error that occurred.
- Resets the channel that caused the error (if known) in the Channel Active Register.
- Updates the Interrupt Status bits in the MAL0_ESR. Then, depending on the mask defined in MAL0_IER (Interrupt Enable Register), it may send an interrupt to software (in PPC405GP, it sends it to the Universal Interrupt Controller).

After receiving an interrupt from MAL, software can analyze the error information read from the Error Status Register. Software can restart channel activity by setting the associated bit in the Channel Active Register.

When a channel is stopped and restarted, MAL starts processing descriptors from the first descriptor in the channel descriptor table. Therefore, software may also update the value of the other channel related registers (see "Channel Table Pointer Registers (MAL0_TXCTPxR, MAL0_RXCTP0R)" on page 20-33) in order to continue from the same buffer in memory.

In the case of PLB errors, MAL does not know which channel caused the error. It is the responsibility of the software to analyze the MAL error registers and the PLB slave error registers to determine which channel caused the error. Software should reset the channel within MAL, resolve the problem, and then reactivate the channel.

See Figure 20-10 on page 20-23 for a flow chart illustrating the steps MAL performs when resolving an error situation.

20.7.3.6 Interrupts To Software

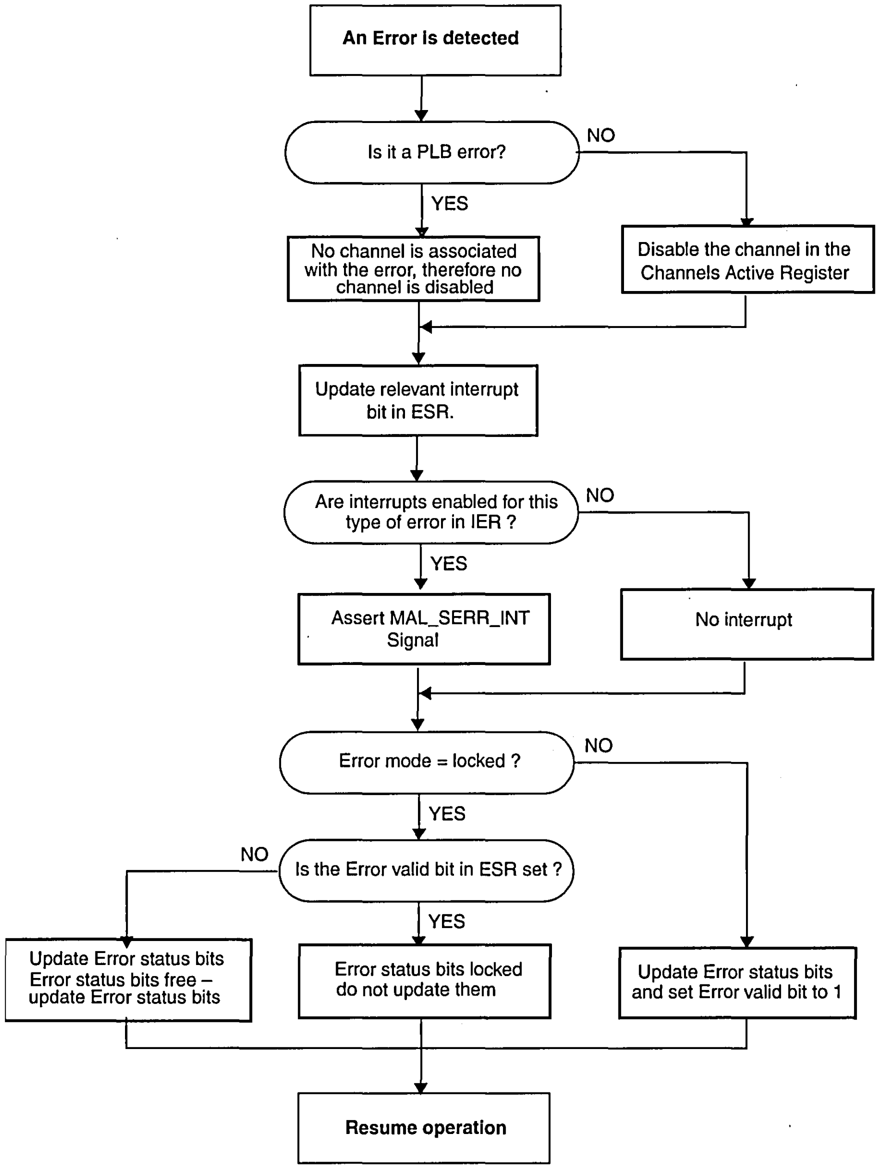

An Error Is detected
Is it a PLB error?
NO
YES
No channel is associated with the error, therefore no channel is disabled
Disable the channel in the Channels Active Register
Update relevant interrupt bit in ESR.
Are interrupts enabled for this type of error in IER ?
NO
YES
Assert MAL_SERR_INT Signal
No interrupt
Error mode = locked ?
NO
YES
NO
Is the Error valid bit in ESR set ?
YES
Update Error status bits Error status bits free – update Error status bits
Error status bits locked do not update them
Update Error status bits and set Error valid bit to 1
Resume operation

Figure 20-10. MAL Error Processing

Figure 20-10 on page 20-23 describes MAL actions once an error is detected. Note that the actual decisions MAL makes may be in a different order than represented by this figure. In any case, the device driver should consider that all of the MAL actions are performed at the same time.

20.8 MAL Registers

Access to MAL registers is through the DCR interface.

- Unless otherwise specified, all register fields are initialized at chip reset to 0.
- Reserved fields are read as undefined and must be written as 0s.

Table 20-1. MAL Registers

Register	DCR Number	Access	Description
MAL0_CFG	0x180	R/W	Configuration Register
MAL0_ESR	0x181	R/Clear	Error Status Register
MAL0_IER	0x182	R/W	Interrupt Enable Register
MALDBR	0x183	R	Debug Register
MAL0_TXCASR	0x184	R/W	TX Channel Active Set Register
MAL0_TXCARR	0x185	R/W	TX Channel Active Reset Register
MAL0_TXEOBISR	0x186	R/Clear	Tx End of Buffer Interrupt Status Register
MAL0_TXDEIR	0x187	R/Clear	Tx Descriptor Error Interrupt Register
MAL0_RXCASR	0x190	R/W	RX Channel Active Set Register
MAL0_RXCARR	0x191	R/W	RX Channel Active Reset Register
MAL0_RXEOBISR	0x192	R/Clear	Rx End of Buffer Interrupt Status Register
MAL0_RXDEIR	0x193	R/Clear	Rx Descriptor Error Interrupt Register
MAL0_TXCTP0R	0x1A0	R/W	Channel Tx 0 Channel Table Pointer Register
MAL0_TXCTP1R	0x1A1	R/W	Channel Tx 1 Channel Table Pointer Register
MAL0_RXCTP0R	0x1C0	R/W	Channel Rx 0 Channel Table Pointer Register
MAL0_RCBS0	0x1E0	R/W	Channel RX 0 Channel Buffer Size Register

20.8.1 MAL Configuration Register (MAL0_CFG)

This register defines the operational mode of MAL. Unless a configuration change is required during system operation, the configuration register needs to be set only during system initialization.

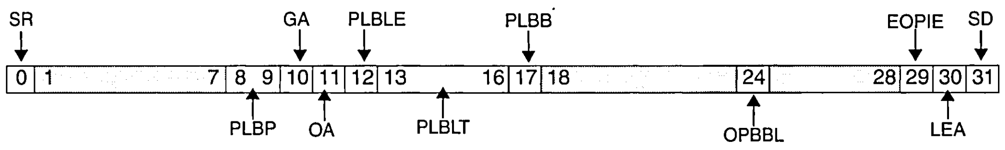


Figure 20-11. MAL Configuration Register (MAL0_CFG)

0	SR	MAL Software Reset 0 MAL reset is complete 1 Reset the MAL	This bit is used to generate a general reset to MAL through a software command. After setting this bit, MAL hardware (registers, interface and internal state machines) returns to the power-on reset value. The software writes 1 to this bit in order to drive MAL to the reset state. The bit is cleared by the hardware when the reset is completed (one system clock).
1:7		Reserved	
8:9	PLBP	PLB Priority 00 Lowest 01 10 11 Highest	Determines the priority of MAL requests on the PLB.
10	GA	Guarded Active 0 GUARDED signal not applied to the PLB slave 1 GUARDED signal applied to the PLB slave	When this bit is set, MAL applies the GUARDED signal to the PLB slave when it is the initiator on the bus. When set, the slave can access all the memory in the current page as well as the subsequent page.
11	OA	Ordered Active 0 ORDERED signal not applied to the PLB slave 1 ORDERED signal applied to the PLB slave	When this bit is set, MAL applies the ORDERED signal to the PLB slave when it is initiator on the bus during data write transactions. Note that the ORDERED signal is always driven active during status write transactions.
12	PLBLE	PLB Lock Error 0 LOCKERROR signal not applied to the PLB slave 1 LOCKERROR signal applied to the PLB slave	When this bit is set, MAL applies the LOCKERROR signal to the PLB slave when it is the initiator during PLB transactions.
13:16	PLBLT	PLB Latency Timer	Determines the number of cycles allowed for burst transactions on the PLB.
17	PLBB	PLB Burst 0 Burst transactions not allowed 1 Burst transactions allowed	When this bit is reset, MAL is not allowed to perform burst transactions.

24	OPBBL	OPB Bus Lock 0 OPB not locked 1 OPB locked	When this bit is set, MAL locks the OPB during data transfers to and from the COMMACs.
18:28	▨	Reserved	
29	EOPIE	End of Packet Interrupt Enable 0 Generate interrupt on every end-of-packet only if the buffers I bit is set 1 Generate interrupt is on every end-of-packet	When this bit is set, an interrupt is generated on every end of packet (both transmit and receive). When clear, end of packet/buffer interrupt is generated only if the buffers I bit is set (1). Note: An interrupt is generated for every descriptor on which the I bit is set, regardless of the state of the EOPIE bit.
30	LEA	Locked Error Active 0 Handle errors in a non-locked mode 1 Handle errors in locked mode	Determines MAL's error handling mode. When this bit is set, MAL will handle errors in the locked mode, otherwise it will handle errors in a non-locked mode.
31	SD	MAL Scroll Descriptor 0 Do not scroll to the first descriptor of the next packet 1 Scroll to the first descriptor of the next packet	Determines whether or not MAL should scroll to the first descriptor of the next packet, following an early packet termination initiated by the related COMMAC. When set, Scrolling mode is active.

20.8.2 Channel Active Set and Reset Registers

For the Channel Active Set/Reset Registers (MAL0_TXCASR, MAL0_TXCARR, MAL0_RXCASR, MAL0_RXCARR), each bit represents its associated channel (bit 0 for channel 0, etc.). When a bit is equal to 1, the channel has been enabled for operation. When a bit is equal to 0, the channel is disabled (MAL ignores any requests for service). If a channel is active when its enable bit is cleared, MAL stops processing the current packet. After the channel's enable bit is cleared, MAL goes back to the top of the channel descriptor table (pointed to by the Channel Table Pointer Register).

- To enable a channel:
  - Write a 1 to its corresponding bit in the Channel Active Set Register (CASR).
  - Multiple channels can be enabled with a single CASR register write.
- To stop and reset a channel:
  - Write a 1 to its corresponding bit in the Channel Active Reset Register (CARR).
  - Writing a 0 to bits in the CARR registers has no effect on the channels.
  - Multiple channels can be reset with a single CARR register write.

MAL also clears the enable bit of a channel following an indication of an error on the channel. The CASR or CARR register(s) can be read to determine which channels are currently active. The following figures describe these registers.

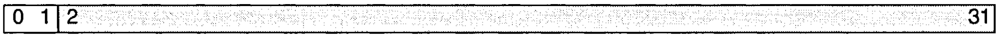


Figure 20-12. TX Channel_Active Set Register (MAL0_TXCASR)

0:1		Transmit Channel Active Set	Each bit represents its related channel (bit 0 for channel 0, etc.). When 1 is written to the bit, channel operation is enabled. There are only two TX channels in the PPC405GP.
2:31	▨	Reserved	

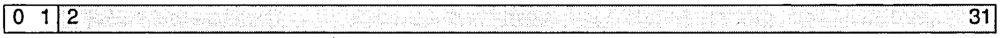


Figure 20-13. TX Channel_Active Reset Register (MAL0_TXCARR)

0:1		Transmit Channel Active Reset	Each bit represents its related channel (bit 0 for channel 0, etc.). When 1 is written to the bit, channel operation is disabled. There are only two TX channels in the PPC405GP.
2:31	▨	Reserved	

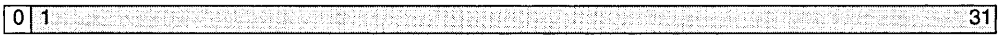


Figure 20-14. RX Channel_Active Set Register (MAL0_RXCASR)

0		Receive Channel Active Set	Each bit represents its related channel (bit 0 for channel 0 etc.). When 1 is written to the bit, channel operation is enabled. There is only one RX channel in the PPC405GP.
1:31	▨	Reserved	

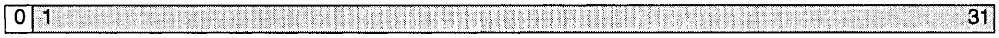


Figure 20-15. RX Channel_Active Reset Register (MAL0_RXCARR)

0		Receive Channel Active Reset	Each bit represents its related channel (bit 0 for channel 0, etc.). When 0 is written to the bit, channel operation is disabled. There is only one RX channel in the PPC405GP.
1:31	▨	Reserved	

20.8.2.1 End of Buffer Interrupt Status Registers

Each bit in the TX End-of-Buffer Interrupt Status and RX End-of-Buffer Interrupt Status registers is related to a channel's descriptor buffer table.

The TX End-of-Buffer Interrupt Status register contains the End-of-Buffer Status bits for each TX channel. The RX End-of-Buffer Interrupt Status register contains the End-of-Buffer Status bits for the RX channels. The mechanism (as described below) for both RX and TX registers is the same.

MAL sets a channel's bit in one of the following conditions:

- When MAL finishes the processing of a buffer (writes back the status to the current descriptor), the related bit in this register is set if the I bit in the descriptor status is set.
- When MAL finishes the processing of a packet (writes back the status of the packet's last buffer) and MAL0_MCR[EOPIE] is set.

Note: In case MAL finishes the processing of a packet which is backed up, MAL doesn't consider it as an end of packet. Therefore, MAL will not set the appropriate channel bit in the End-of-Buffer Register.

- When the Bad Packet bit is set in the COMMAC channel Status halfword.

The device driver resets the interrupt by writing a 1 to the related bit. Writing a 0 has no effect.

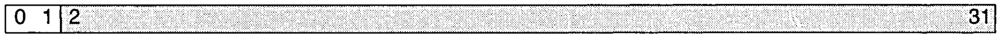


Figure 20-16. TX End of Buffer Interrupt Status Register (MAL0_TXEOBISR)

0:1		Transmit Channel End-of-Buffer Interrupt	Each bit represents its related channel (bit 0 for channel 0, etc.). Writing 1 to a bit resets it. There are only two TX channels in the PPC405GP.
2:31	▨	Reserved	

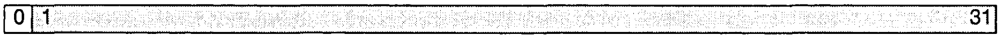


Figure 20-17. RX End of Buffer Interrupt Status Register (MAL0_RXEOBISR)

0		Receive Channel End-of-Buffer Interrupt	Each bit represents its related channel (bit 0 for channel 0, etc.). Writing 1 to a bit resets it. There is only one RX channel in the PPC405GP.
1:31		Reserved	

20.9 Error Registers

The following paragraphs describe MAL error registers. For more information about MAL errors see "Error Handling" on page 20-19.

20.9.1 MAL Error Status Register (MAL0_ESR)

This register holds the information about the error that occurred and the interrupts status. The register includes the following fields:

- **Error status bits** – This field holds the error information. The information includes the number of the channel on which the error occurred (if known) and the type of the error. The error can be either the last detected error or a locked error if "Locked error mode" is active. (See "Operational Error Modes" on page 20-21 for description of the Locked error mode.)

The error status field includes an "Error Valid" bit which indicates whether there is an error information in the error status field or not. The error status bits are not valid when the "Error Valid" bit is cleared (by writing 1 to this bit).

- **Interrupt status bits** – Every error detected by MAL sets a related bit in the interrupt status field. The interrupt status bits may be cleared by software by writing 1 to the bit to be cleared. The bits in this field are accumulative (more than one interrupt may be indicated here). These bits are masked by the IER (Interrupt Enable Register) to create a maskable interrupt, which is implemented by the MAL_SERR_INT signal (connected to the UIC on the PPC405GP).

Note: In order to reset the interrupt bits and the Error valid bit in the Error Status register, 1 must be written to the related bit. Writing 0 has no effect.

More than one bit can be cleared at a time and only R/W bits can be reset.

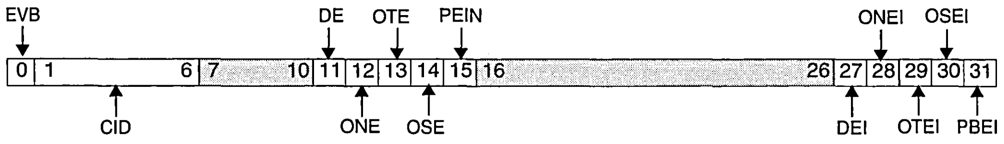


Figure 20-18. MAL Error Status Register (MAL0_ESR)

0	EVB	Error Valid Bit 0 Bit 1:15 are available for latching new error information. 1 Bits 1:15 contain last error. A new error cannot be latched.	When this bit is set, bits 1-6 include the ID of the erroneous channel (in case of OPB errors). Bits 11-15 indicate the type of error. In non-locked mode, the error indication describes the last error that had occurred. In locked mode, the error is the first one that had occurred after this bit was cleared. This bit is set when an error occurs and remains set until reset by the software. In locked mode, new errors cannot be latched in the error lock indication fields if this bit is set
1:6	CID	Channel ID	This field contains the number of the channel which caused the locked error. Bit 1 indicates whether the channel ID represents an RX channel (1) or a TX channel (0). Bits 2:6 indicates the number of the channel that caused the error. Note: An error on the PLB cannot be related to a channel. The error condition may be resolved by using the error information optionally locked in the PLB slave.
7:10		Reserved	
11	DE	Descriptor Error 0 No error 1 Non-valid descriptor	Indicates that the error is a non-valid descriptor, which is *not* the first descriptor in a TX packet.
12	ONE	OPB Non-fullword Error 0 No error 1 Non-fullword asserted	Indicates that the error is a non-fullword acknowledge asserted by an OPB slave.
13	OTE	OPB Timeout Error 0 No error 1 OPB timeout	Indicates the error is an OPB timeout.
14	OSE	OPB Slave Error 0 No error 1 OPB slave error	Indicates the error is an error indication asserted by an OPB slave.
15	PEIN	PLB Bus Error Indication 0 No error 1 PLB bus error	When this bit is set, the detected error is a PLB error. There is no meaning to the Channel ID field in this case.

16:26		Reserved	
27	DEI	Descriptor Error Interrupt 0 No error 1 Descriptor data error recognized	A descriptor data error is recognized during access to the descriptor table. This error indication is asserted when a non-valid descriptor is accessed, which is *not* the first descriptor in a TX packet. Set condition for this bit generates a maskable interrupt.
28	ONEI	OPB Non-fullword Error Interrupt 0 No error 1 Non-fullword acknowledgment from a slave	This bit is set following a non-fullword acknowledgment coming from a slave. Set condition for this bit generates a maskable interrupt.
29	OTEI	OPB Timeout Error Interrupt 0 No error 1 OPB time-out	This bit is set following an OPB time out error indication. Set condition for this bit generates a maskable interrupt.
30	OSEI	OPB Slave Error Interrupt 0 No error 1 OPB error from a slave	This bit is set following an OPB error indicated by the slave. Set condition for this bit generates a maskable interrupt.
31	PBEI	PLB Bus Error Interrupt 0 No error 1 PLB error indication	This bit is set following a PLB error indication (from the PLB slave). Set condition for this bit generates a maskable interrupt.

20.9.2 MAL Interrupt Enable Register (MAL0_IER)

Each bit in the following register, when it is set, enables assertion of the interrupt signal (MAL_SERR_INT) when the related bit in the MAL0_ESR (interrupt bit) is set.

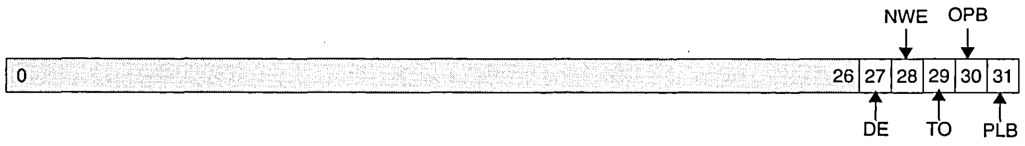


Figure 20-19. MAL Interrupt Enable Register (MAL0_IER)			
0:26		Reserved	
27	DE	Descriptor Error	When set, this bit enables the descriptor error (descriptor not valid) interrupt.
28	NWE	Non_W_Err_Int_Enable	When set, this bit enables OPB non-word transfer error interrupt.
29	TO	Time_Out_Int_Enable	When set, this bit enables OPB time-out error interrupt.
30	OPB	OPB_Err_Int_Enable	When set, this bit enables the OPB Slave error interrupt.
31	PLB	PLB_Err_Int_Enable	When set, this bit enables the PLB error interrupt.

20.9.3 Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)

Each bit in the following registers is related to a channel descriptor buffer table. Each bit indicates a descriptor data error related to a certain channel.

The TX Descriptor Error register contains the Descriptor errors bits of the TX channels. The RX Descriptor Error register contains the Descriptor errors bits of the RX channels. The mechanism (as described below) for both RX and TX registers is the same.

MAL sets a channel's bit when a descriptor data error was recognized during access to the descriptor table of a specific channel (see "Descriptor Error" on page 20-19).

The device driver resets the interrupt by writing a 1 to the related bit. Writing a 0 has no effect. When one or more of the TX Descriptor Error Interrupt bits is set, then the MAL_TX_DESC_ERR_INT bit is set. When one or more of the RX Descriptor Error Interrupt bits is set, then the MAL_RX_DESC_ERR_INT signal is set (attached to UIC on the PPC405GP).

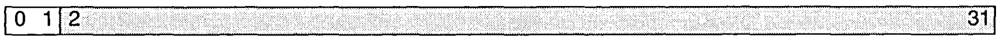


Figure 20-20. TX Descriptor Error Interrupt Register (MAL0_TXDEIR)

0:1		Transmit Descriptor Error Interrupt	Each bit represents its related channel (bit 0 for channel 0, etc.). When one or more bits are set, MAL_DESC_ERR_INT is set. Writing 1 to a bit resets it. There are only two TX channels in the PPC405GP.
2:31	▨	Reserved	

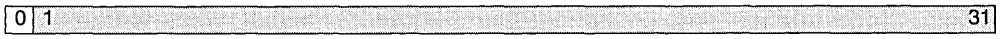


Figure 20-21. RX Descriptor Error Interrupt Register (MAL0_RXDEIR)

0		Receive Descriptor Error Interrupt	Each bit represents its related channel (bit 0 for channel 0 etc.). When one or more bits are set, MAL_DESC_ERR_INT is set. Writing 1 to a bit resets it. There is only one RX channel in the PPC405GP.
1:31	▨	Reserved	

20.9.4 Channel Table Pointer Registers (MAL0_TXCTPxR, MAL0_RXCTP0R)

MAL uses RX Channel table pointer registers, one for each RX channel, and TX Channel table pointer registers, one for each TX channel. The Channel Table Pointer Registers point to the base address, in memory, of the descriptor buffer table used by each channel.

Note 1: Bits 0 to 12 of all the TXCTPxR registers are mapped to the same physical register. Writing into any of these registers overwrites the value of bits 0 to 12 in all the TXCTPxR registers. Read operation has no effect. Bits 0 to 12 of all the RXCTPxR registers are mapped to the same physical register. Writing into any of these registers overwrites the value of bits 0 to 12 in all the RXCTPxR registers. Read operation has no effect.

Note 2: When changing the value of either of the MAL0_TXCTPxR registers, both TX channels must be idle. To verify a channel is idle, check the device's Transmit Idle bit. Another way to assure that the channels are idle is to disable the channels before changing the MAL0_TXCTPxR regsiter, and then re-enable them once the MAL0_TXCTPxR register is set to its new value.

The TX and RX Channel Table Pointer Registers have an identical format as shown in Figure 20-22 and Figure 20-23. There are two TX registers (0 and 1), and one RX register (0) in PPC405GP.

Figure 20-22. TX Channel Table Pointer x Register (MAL0_TXCTPxR)

0:31		Channel Table Pointer	Pointer to the base address in memory of the buffer descriptor table used by the channel. The value entered should be a pointer to a location in memory accommodating an aligned double fullword (this requires the three least significant bits of this pointer must be 000). There are two transmit channels (x = 0 and 1).

Figure 20-23. RX Channel Table Pointer x Register (MAL0_RXCTPxR)

0:31		Channel Table Pointer	Pointer to the base address in memory of the buffer descriptor table used by the channel. The value entered should be a pointer to a location in memory accommodating an aligned double fullword (this requires the three least significant bits of this pointer must be 000) There is one receive channel (x = 0).

The Table Pointer Registers retain their value following Soft Reset or Channel Reset.

Chapter 21. Serial Port Operations

The PPC405GP contains two universal asynchronous receiver/transmitters (UARTs) which provide two full-duplex serial interfaces to support communications with serial peripheral devices. Each UART is compatible with the National Semiconductor (NS) 16550 chip, and includes a 16-byte send and a 16-byte receive FIFO.

Features of the UART include:

- Compatible with the NS 16550
- 16-byte send FIFO, 16-byte receive FIFO
- Full duplex operation
- Programmable baud rate generator
- Supports 5- to 8-bit word size, 1 or 2 stop bits, even, odd, or no parity
- One 8-wire interface (UART0) and one 4-wire interface (UART1)

The UART performs serial-to-parallel conversion on data characters received from a peripheral device, and parallel-to-serial conversion on data characters received from the processor. The processor can read the complete status of the UART at any time during the functional operation. Status information reported includes the type and condition of the transfer operations being performed by the UART, as well as any error conditions, such as parity, overrun, framing, and break interrupt.

This UART is functionally identical to NS16450 in character mode (on power up it will be in this mode), and can be put into FIFO mode to relieve the processor of excessive software overhead. Here, internal FIFOs are activated allowing 16 bytes (plus 3 bits per byte of error data in the RCVR FIFO) to be stored in both receive and transmit modes.

The source of the UART serial clock input is selected in Chip Control Register 0 (CPC0_CR0[U0EC:U1EC]) bits 24 and 25. Either the internal serial clock or an external serial clock can be selected. A programmable baud rate generator is included that is capable of dividing the UART serial clock input by a divisor of 1 to ($2^{16} - 1$) and producing the 16× clock required for driving the UART internal transmitter/receiver logic. The internal serial clock input is derived from the CPU clock by a divisor specified in CPC0_CR0[UDIV].

The UART has an interrupt system that can be programmed to the user's requirements, helping to minimize the computing required to handle the communications link. UART interrupts are capable of triggering an interrupt request to the PPC405GP interrupt controller.

21.1 Functional Description

- Runs NS 16550 software
- Registers are identical to the NS16550 register set
- After reset, all registers are identical to the NS16450 register set
- Complete status reporting capability
- Transmitter and receiver are each buffered with 16-byte FIFOs when FIFO mode selected

- Can add/delete standard asynchronous communication bits such as start, stop, and parity to/from the serial data
- When in character mode, holding and shift registers eliminate the need for precise synchronization between the processor and serial data
- Full prioritized interrupt system controls
- Independently controlled transmit, receive, line status, and data set interrupts
- Programmable baud rate generator divides the UART serial clock input by 1 to (2^{16}–1) and generates the 16x clock:

  Baud rate (bps) = (Serial Clock Input) / (16 x Decimal Divisor)
- Receiver uses 5-way oversampling as follows: it samples each serial bit five times, and if at least three of the samples are 1's, the bit is determined to be a 1, otherwise it is a 0
- Fully programmable serial-interface characteristics:
  - 5-, 6-, 7-, or 8-bit characters
  - Even, odd, or no parity bit generation and detection
  - 1-,1.5-, or 2-stop bit generation
  - Variable baud rate
- Line break generation and detection, and false start bit detection
- Internal diagnostic capability:
  - Loopback controls for communications link fault isolation
  - Break, parity, overrun, framing error simulation

21.2 Serial Input Clocking

The two PPC405GP UARTs can be clocked individually from an external serial clock or from an internally generated serial clock. The internally generated serial clock is derived from the CPU clock, and is CPU/n, where n ranges from 1 to 32. The divisor n is programmed by setting a value of 0 to 31 in CPC0_CR0[UDIV] (see Chapter 7, "Clocking").

The choice of serial clock frequency affects the serial communications error rate. If an external clock of 1.8432 MHz (or some multiple of this frequency) is used, the error rate approaches zero. However, when using the internally generated clock only certain clock frequencies are possible, which results in a small, non-zero error rate in all cases, unless SysClk is chosen as an integer multiple of 1.8432 MHz.

The optimum serial clock frequency is determined from the following relationship:

Serial Clock = Baud Rate $\times$ 16 $\times$ UART Divisor

Acceptable baud rates are always integral multiples of 300 (for example, 1200 = 4 $\times$ 300). Table 21-1 shows optimum UART divisor and CPU divide ratios for a range of possible baud rates. This information is provided for four different CPU clock frequencies. The UART divisor is programmed in

UARTx_DLM and UARTx_DLL (see "Divisor Latch LSB and MSB Registers (UARTx_DLL, UARTx_DLM)" on page 21-14). The value range is 1 to (2^16-1)=65535.

Table 21-1. Baud Rate Settings

Desired Baud Rate (bps)	CPU:UART Divide Ratio	Serial Clock Frequency (MHz)	UART Divisor	Actual Baud Rate (bps)	Error (%)
CPU Clock = 150MHz					
1200	13	11.5380	601	1199.923205	0.0063996
2400	21	7.1429	186	2400.153610	0.0064004
4800	21	7.1429	93	4800.307220	0.0064004
9600	16	9.3750	61	9605.532787	0.0576332
19200	14	10.7140	35	19132.653061	0.3507653
28800	13	11.5380	25	28846.153846	0.1602564
33600	9	16.6670	31	33602.150538	0.0064004
38400	9	16.6670	27	38580.246914	0.4693930
57600	18	8.3333	9	57870.370370	0.4693930
115200	9	16.6670	9	115740.740741	0.4693930
307200	30	5.0000	1	312500.000000	1.7252604
CPU Clock = 166.66MHz					
1200	31	5.3763	280	1200.076800	0.0064000
2400	31	5.3763	140	2400.153600	0.0064000
4800	31	5.3763	70	4800.307200	0.0064000
9600	31	5.3763	35	9600.614401	0.0064000
19200	32	5.2083	17	19148.284237	0.2693529
28800	19	8.7719	19	28855.032202	0.1910840
33600	31	5.3763	10	33602.150403	0.0064000
38400	17	9.8039	16	38296.568474	0.2693529
57600	30	5.5556	6	57870.370139	0.4693926
115200	30	5.5556	3	115740.740278	0.4693926
307200	17	9.8039	2	306372.547794	0.2693529
CPU Clock = 200MHz					
1200	11	18.1820	947	1199.961601	0.0031999
2400	28	7.1429	186	2400.153610	0.0064004
4800	28	7.1429	93	4800.307220	0.0064004
9600	14	14.2860	93	9600.614439	0.0064004
19200	31	6.4516	21	19201.228879	0.0064004
28800	31	6.4516	14	28801.843318	0.0064004
33600	31	6.4516	12	33602.150538	0.0064004
38400	25	8.0000	13	38461.538462	0.1602564
57600	31	6.4516	7	57603.686636	0.0064004
115200	12	16.6670	9	115740.740741	0.4693930
307200	20	10.0000	2	312500.000000	1.7252604

21.3 UART Registers

UART registers are accessed via memory locations 0xEF60_0XYY where X=3 for UART 0 and X=4 for UART 1.

Table 21-2. UART Configuration Registers

UART Config Register	Address	R/W	Description	Reset
UARTx_RBR	EF60_0X00 [1]	R	UART x Receiver Buffer Register	
UARTx_THR	EF60_0X00 [1]	W	UART x Transmitter Holding Register	
UARTx_IER	EF60_0X01 [1]	R/W	UART x Interrupt Enable Register	0000 0000
UARTx_IIR	EF60_0X02	R	UART x Interrupt Identification Register	0000 0001
UARTx_FCR	EF60_0X02	W	UART x FIFO Control Register	0000 0000
UARTx_LCR	EF60_0X03	R/W	UART x Line Control Register	0000 0000
UARTx_MCR	EF60_0X04	R/W	UART x Modem Control Register	0000 0000
UARTx_LSR	EF60_0X05	R/W	UART x Line Status Register	0110 0000
UARTx_MSR	EF60_0X06	R/W	UART x Modem Status Register	xxxx 0000
UARTx_SCR	EF60_0X07	R/W	UART x Scratch Register	
UARTx_DLL	EF60_0X00 [1]	R/W	UART x Divisor Latch (LSB)	
UARTx_DLM	EF60_0X01 [1]	R/W	UART x Divisor Latch (MSB)	

1. UARTx_LCR[DLAB] controls the function accessed through registers EF60_0X00 and EF60_0X01. When UARTx_LCR[DLAB] is 0, access is enabled to the Receiver/Transmitter registers and the Interrupt Enable register. When UARTx_LCR[DLAB] is a 1, access is enabled to the Divisor Latch registers.

The system programmer may access any of the UART registers via the processor. These registers control all UART operations including transmission and reception of data. In PPC405GP there are two UARTs, designated 0 (8-wire interface) and 1 (4-wire interface). In the following sections, the registers are specified with a generic name where x represents 0 or 1. For example, the Line Control Register appears as a UARTx_LCR.

For UART1, two of the four wires are TX and RX. The remaining two wires can be programmed as a combination of DTR and DSR, or CTS and RTS in CPC0_CR0(DCS:RDS). DCD and RI are not available on the 4-wire interface.

21.3.1 Receiver Buffer Registers (UARTx_RBR)

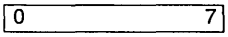


Figure 21-1. UART Receiver Buffer Registers (UARTx_RBR)

0:7		Data bit

Note: UARTx_RBR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.2 Transmitter Holding Registers (UARTx_THR)

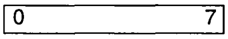


Figure 21-2. UART Transmitter Holding Registers (UARTx_THR)

0:7		Data bit

Note: UARTx_THR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.3 Interrupt Enable Registers (UARTx_IER)

Five UART interrupts on four priority levels are enabled via the Interrupt Enable Register, UARTx_IER. Any of the five interrupts can be used to surface a UART interrupt to the PPC405GP interrupt controller. Each interrupt can be enabled by setting its appropriate bit. Resetting UARTx_IER[4:7] totally disables the UART interrupt system. Disabling an interrupt prevents it from being shown as active in the UARTx_IIR and prevents it from signaling a UART interrupt to the PPC405GP interrupt controller. See Table 21-3, "Interrupt Priority Level," on page 21-6.

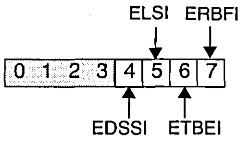


Figure 21-3. UART Interrupt Enable Registers (UARTx_IER)

0:3	▨	Reserved Always 0.
4	EDSSI	Enable Modem Status Interrupt
5	ELSI	Receiver Line Status Interrupt enable 0 Enable receiver line status interrupt 1 Disable receiver line status interrupt

6	ETBEI	Transmitter Holding Register Empty Interrupt enable 0 Enable transmitter holding register empty interrupt 1 Disable transmitter holding register empty interrupt	
7	ERBFI	Received Data Available Interrupt enable 0 Disable received data available interrupt 1 Enable received data available interrupt	In FIFO mode, timeout interrupts follow the enable/disable state of ERBFI.

Note: UARTx_IER is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.4 Interrupt Identification Registers (UARTx_IIR)

The UART prioritizes interrupts into four levels which are recorded in the Interrupt Identification Register. The interrupt types in the order of their priority are as follows:

1. Receiver line status.
2. Received data available and character timeout indication
3. Transmitter holding register empty
4. Modem status

Table 21-3 lists the interrupt priority levels.

Table 21-3. Interrupt Priority Level

IIR Bit 4	IIR Bit 5	IIR Bit 6	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	1	1	1	Receiver Line Status	Overrun, Parity or Framing Error, or Break Interrupt.	Read LSR.
0	1	0	2	Received Data Available	Receiver data available or trigger level reached.	Read RBR, or FIFO drops below trigger level.
1	1	0	2	Character Timeout Indication	No characters have been removed from or input to the receiver FIFO during the last four character times and it contains at least one character during this time.	Read RBR.
0	0	1	3	Transmitter Holding Register Empty	Transmitter Holding Register Empty.	Read IIR (if source of interrupt) or write THR.
0	0	0	4	Modem Status	Clear to Send, Data Set Ready, Ring Indicator or Data Carrier Detect.	Read MSR.

When the processor accesses UARTx_IIR, the UART records new interrupts, but does not change its current contents until the access by the processor is complete. The UART indicates the highest priority interrupt pending to the PPC405GP interrupt controller via the IIR.

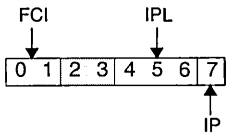


Figure 21-4. UART Interrupt Identification Registers (UARTx_IIR)

0:1	FCI	FIFO Control Indicator 00 FIFOs disabled (UARTx_FCR[FC] = 0) 01 Reserved 10 Reserved 11 FIFOs enabled (UARTx_FCR[FC] = 1)	
2:3		Reserved	
4:6	IPL	Interrupt Priority Level 000 Priority level 4 001 Priority level 3 010 Priority level 2 011 Priority level 1 100 Reserved 101 Reserved 110 Priority level 2 111 Reserved	See Table 21-3. Note: Priority 1 is highest priority.
7	IP	Interrupt Pending 0 Interrupt is pending 1 No interrupt pending	When set to 0, IIR contents can be used as a pointer to the appropriate interrupt service routine.

Note: UARTx_IIR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.5 FIFO Control Registers (UARTx_FCR)

The FIFO control register has the same address as the IIR and is a write-only register. This register is used to perform FIFO control operations such as selecting the type of DMA signaling, setting the receiver FIFO trigger levels, clearing the FIFOs, and enabling the FIFO.

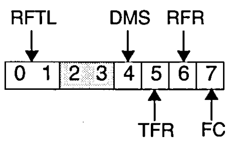


Figure 21-5. UART FIFO Control Registers (UARTx_FCR)

0:1	RFTL	Receiver FIFO Trigger Level 00 1 byte 01 4 bytes 10 8 bytes 11 14 bytes	
2:3		Reserved	
4	DMS	DMA Mode Select 0 Mode 0 = single transfer 1 Mode 1 = multiple transfers	Select single or multiple transfer mode if UARTx_FCR[7] = 1.
5	TFR	Transmitter FIFO Reset 0 Operation complete 1 Reset the transmitter FIFO	A 1 written to this bit clears all bytes in the transmitter FIFO and resets all of its counter logic to 0. The transmitter shift register is not cleared. This bit is self-clearing.
6	RFR	Receiver FIFO Reset 0 Operation complete 1 Reset the receiver FIFO	A 1 written to this bit clears all bytes in the receiver FIFO and resets all of its counter logic to 0. The receiver shift register is not cleared. This bit is self-clearing.
7	FE	FIFO Enable 0 Disable FIFOs 1 Enable FIFOs	When set to 1, both the receiver and transmitter FIFOs are enabled. When set to 0, both receiver and transmitter FIFOs are reset. Data is automatically cleared from both FIFOs when changing to and from FIFO and 16450 modes. Programming other bits will be ignored if this bit is not a 1.

Note: UARTx_FCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.6 Line Control Registers (UARTx_LCR)

The system programmer uses the line control register (LCR) to specify the format of the asynchronous data communications exchange and to set the Divisor Latch Access bit. The contents of the LCR can also be read by the processor. The read capability simplifies system programming, and eliminates the need for separate storage of the line characteristics in system memory.

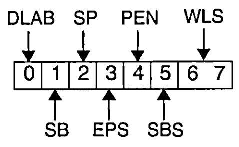


Figure 21-6. UART Line Control Registers (UARTx_LCR)

0	DLAB	Divisor Latch Access Bit 0 Address RBR, THR and IER with LTADR2-0 for read or write operation 1 Address Divisor Latches with LTADR2-0 for read or write operation	
1	SB	Set Break 0 Disable Break 1 Enable Break	Causes a break condition to be transmitted to the UART when the core is receiving. SOUT is forced to the spacing state (0). This bit acts only on SOUT and has no effect on the transmitter logic.
2	SP	Sticky Parity 0 Disable sticky parity 1 Enable sticky parity	If UARTx_LCR[EPS] = 1 and UARTx_LCR[PE] = 1, the parity bit is transmitted and checked as 0. If UARTx_LCR [EPS] = 0 and UARTx_LCR[PE] = 1,the parity bit is transmitted and checked as 1.
3	EPS	Even Parity Select 0 Generate odd parity 1 Generate even parity	This bit is significant only if UARTx_LCR[PE] = 1.
4	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	
5	SBS	Stop Bit Select 0 Characters have 1 stop bit 1 Characters have 1.5 or 2 stop bits	If UARTx_LCR[CL] = 00, characters have 1.5 stop bits. For any other value of UARTx_LCR[CL], characters have 2 stop bits. The receiver checks the first stop bit only, regardless of how many stop bits are selected.
6:7	WLS0, WLS1	Word Length Select Bits 0,1 00 Use 5-bit characters 01 Use 6-bit characters 10 Use 7-bit characters 11 Use 8-bit characters	

Note: UARTx_LCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.7 Modem Control Registers (UARTx_MCR)

The interface between the modem, data set, or peripheral device emulating a modem, and the UART, is controlled by the Modem Control Register (UARTx_MCR).

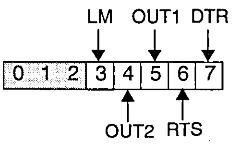


Figure 21-7. UART Modem Control Registers (UARTx_MCR)

0:2		Reserved	Always 0.
3	LM	Loopback Mode 0 Disabled 1 Enabled	Provides a local loopback feature for diagnostic testing of the UART. The following occurs: 1. SOUT is set to the marking state (logic 1) SIN is disconnected. 2. The output of the transmitter shift register feeds the input of the receiver shift register. 3. The four modem control inputs $\overline{\text{DSR}}$, $\overline{\text{CTS}}$, $\overline{\text{RI}}$, and $\overline{\text{DCD}}$ are disconnected. 4. The four modem control outputs $\overline{\text{DTR}}$, $\overline{\text{RTS}}$, $\overline{\text{OUT1}}$, and $\overline{\text{OUT2}}$ are set to a logic 1 (their inactive state). 5. The four modem control outputs are connected internally to the four modem control inputs. Transmitted data is immediately received to verify the UART transmit and receive data paths. Receiver and transmitter interrupts are operational. Their sources are external to the UART. Also operational are the modem control interrupts, but their source is the low-order 4 bits of UARTx_MCR instead of the modem control inputs to the UART. UARTx_IER still controls the interrupts.
4	OUT2	User Output 2 0 $\overline{\text{OUT2}}$ inactive (1) 1 $\overline{\text{OUT2}}$ active (0)	Auxiliary user designated output.
5	OUT1	User Output 1 0 $\overline{\text{OUT1}}$ inactive (1) 1 $\overline{\text{OUT1}}$ active (0)	Auxiliary user designated output.
6	RTS	Request To Send 0 $\overline{\text{RTS}}$ inactive (1) 1 $\overline{\text{RTS}}$ active (0)	
7	DTR	Data Terminal Ready 0 $\overline{\text{DTR}}$ inactive (1) 1 $\overline{\text{DTR}}$ active (0)	

Note: UARTx_MCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.8 Line Status Registers (UARTx_LSR)

Information concerning the data transfer is held for the processor in this register. Bits 3 through 6 are conditions that produce a receiver line status interrupt whenever the condition corresponding to the active bit is detected and the interrupt is enabled. This register is intended for read operations only and writing is not recommended.

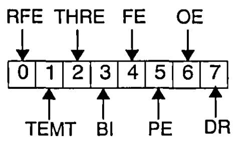


Figure 21-8. UART Line Status Registers (UARTx_LSR)

0	RFE	Receiver FIFO Error Indicator 0 In FIFO mode, reset to 0 when the processor reads the UARTx_LSR, provided there are no subsequent errors in the FIFO. 1 There are one or more instances of parity error, framing error or break indication in the FIFO.	Always 0 in 16450 mode.
1	TEMT	Transmitter Empty Indicator 0 Reset to 0 whenever the THR or the transmitter shift register contain a character. In FIFO mode, it is reset to 0 whenever the transmitter FIFO or the transmitter shift register contain a character. 1 Set to 1 when the THR and the Transmitter shift register are both empty. In FIFO mode, it is set to 1 when the transmitter FIFO and the transmitter shift register are both empty.	
2	THRE	Transmitter Holding Register Empty Indicator 0 Concurrent reset to 0 with the loading of the THR by the processor. In FIFO mode it is reset to 0 when at least one byte is written to the transmitter FIFO. 1 Set to 1 when the UART is ready to accept a new character for transmission. In FIFO mode, this bit is set when the transmitter FIFO is empty.	When UARTx_IER[THRE] = 1, the UART issues an interrupt to the PPC405GP interrupt controller. This bit is set to 1 when a character is transferred from the THR to the transmitter shift register.

3	BI	Break Interrupt Indicator. 0 Reset to 0 whenever processor reads Line Status Register (LSR). 1 Set to 1 whenever the received data input is held at the spacing level (0) for longer than a full word transmission time.	The full word transmission time is the time required for the start bit, data bits (can be 5–8 bits), parity and stop bits. In FIFO mode, this error is revealed to the processor when the character this error is associated with is at the top of the FIFO. Only one 0 character is loaded into the receiver FIFO when a break occurs. After the next valid start bit is received and has gone into the marking state, the next character transfer is enabled. Error causes a Receiver Line Status Interrupt.
4	FE	Framing Error Indicator. 0 Reset to 0 whenever processor reads LSR. 1 Set to 1 whenever stop bit following the last data bit or parity bit is detected as 0 (spacing level).Indicates that a valid stop bit was not found in the received character.	Error causes a Receiver Line Status Interrupt.
5	PE	Parity Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Indicates that the received data character does not have the correct parity as determined by the even parity select bit (UARTx_LCR.[EPS]). Set to 1 upon detection of a parity error.	In FIFO mode, this error is revealed to the processor when the character this error is associated with is at the top of the FIFO. Error causes a Receiver Line Status Interrupt.
6	OE	Overrun Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Data in the RBR was read by the processor before the next character was transferred into the UARTx_RBR, hence the original data was lost.	In FIFO mode, if the incoming data continues to fill the FIFO beyond the trigger level, an OE occurs only after the FIFO is completely full and the entire next character has been received in the receiver shift register. The processor is informed of the OE immediately upon occurrence. The character in the shift register will be overwritten and will not be transferred to the FIFO. Error causes a Receiver Line Status Interrupt.
7	DR	Receiver Data Ready Indicator. 0 Reset to 0 when all data has been read from the receiver FIFO or the UARTx_RBR. 1 An entire incoming character has been received into the UARTx_RBR or receiver FIFO.	

Note: UARTx_LSR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.9 Modem Status Registers (UARTx_MSR)

The processor can monitor the present state of the modem (or peripheral device) control lines by reading the Modem Status Register (UARTx_MSR). In addition, the UARTx_MSR has four bits to indicate if any of the modem (or peripheral device) control lines have changed state.

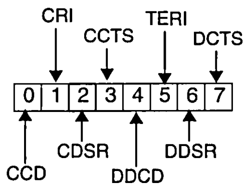


Figure 21-9. UART Modem Status Registers (UARTx_MSR)

0	DCD	Data Carrier Detect	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT2].
1	CRI	Complement of Ring Indicator	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT1].
2	CDSR	Complement of Data Set Ready	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[DTR].
3	CCTS	Complement of Clear To Send	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[RTS].
4	DDCD	Delta Data Carrier Detect 0 Set when processor reads the Modem Status Register 1 $\overline{\text{DCD}}$ input changed state	Indicates that the $\overline{\text{DCD}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
5	TERI	Trailing Edge of Ring Indicator 0 Set when processor reads the Modem Status Register 1 $\overline{\text{RI}}$ input changed from 0 to 1	Indicates that the $\overline{\text{RI}}$ input to the UART changed from 0 to 1 since the processor last read the Modem Status Register. A modem status interrupt is generated.
6	DDSR	Delta Data Set Ready 0 Set when processor reads the Modem Status Register 1 $\overline{\text{DSR}}$ input changed state	Indicates that the $\overline{\text{DSR}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
7	DCTS	Delta Clear To Send 0 Set when processor reads the Modem Status Register 1 $\overline{\text{CTS}}$ input changed state	Indicates that the $\overline{\text{CTS}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.

Note: UARTx_MSR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.10 Scratchpad Registers (UARTx_SCR)

A scratchpad register intended for use by the programmer as a temporary data location is provided in this UART. It does not control the UART operation in any way.

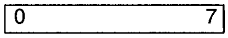


Figure 21-10. Scratchpad Registers (UARTx_SCR)

0:7		Data bits

Note: UARTx_SCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

21.3.11 Divisor Latch LSB and MSB Registers (UARTx_DLL, UARTx_DLM)

The divisor latches are used to program the UART divisor used in generating the baud clock. A 16-bit divisor may be programmed through these registers. Access to these registers is provided by setting UARTx_LCR[DLAB] = 1. These registers have a power-on reset value of 0.

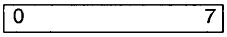


Figure 21-11. UART Baud-Rate Divisor Latch (MSB) Registers (UARTx_DLM)

0:7		Data bits

Note: UARTx_DLM is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

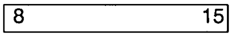


Figure 21-12. UART Baud-Rate Divisor Latch (LSB) Registers (UARTx_DLL)

8:15		Data bits

Note: UARTx_DLL is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

The UART divisor is calculated using the following formula:

UART Divisor = Serial Input Clock/(16 x Baud Rate)

For example, if the serial input clock= 11.0592MHz and a baud rate of 9600bps is required:

UART Divisor = Serial Input Clock/(16 x Baud Rate)

= 11,059,200/(16x9600)

= 72 = 0x48

For this example, UARTx_DLM should be programmed to 0 and UARTx_DLL register should be programmed to 0x48. Due to the error introduced by rounding, some baud rates cannot be generated at certain serial input clock frequencies. Table 21-4 lists some common baud rates and their corresponding Divisor Latch register values with a serial input clock of 11.0592MHz.

Table 21-4. Divisor Latch Settings for Certain Baud Rates

Baud Rate (bps)	Divisor Latch MSB	Divisor Latch LSB
9600	0x00	0x48
19200	0x00	0x24
28800	0x00	0x18
38400	0x00	0x12
57600	0x00	0x0C

21.4 FIFO Operation

21.4.1 Interrupt Mode

21.4.1.1 Receiver

Receiver interrupts occur as described below when the receiver FIFO and receiver interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ERBFI] = 1.

The received data available interrupt is issued when the number of characters in the FIFO has reached the trigger level programmed into UARTx_FCR. This interrupt is reset to 0 when the FIFO character count drops below this trigger level.

The received data available indicator is issued when the number of characters in the FIFO has reached the trigger level programmed into UARTx_FCR. This indicator is reset to 0 when the FIFO character count drops below this trigger level.

The receiver line status interrupt (UARTx_IIR = 0xC6) is a top priority interrupt, whereas the received data available interrupt (UARTx_IIR = 0xC4) is a second priority interrupt.

Data Ready (UARTx_LSR[DR]) is set as soon as a character is transferred from the shift register to the receiver FIFO. This bit is reset when the FIFO is empty.

Receiver timeout interrupts will occur as described below when the receiver FIFO and receiver interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ERBFI] = 1.

A FIFO timeout will occur when:

At least one character is in the receiver FIFO, no serial characters have been received for four serial character time periods, and the processor has not read the FIFO for four serial character time periods. A serial character time period is as follows:

1/(baud rate) x (# start bits + word length + # parity bits + # stop bits)

For example, the serial character time period for an 8-bit word with one parity bit, two stop bits at 56K baud is as follows:

$1/(56000) \times (1 + 8 + 1 + 2) = 214.3\mu s$

So the timeout would occur after 857.1 µs, if the above conditions hold.

When a timeout interrupt has occurred, it is cleared and the timer is reset when the processor reads one character from the receiver FIFO.

When a timeout interrupt has not occurred, the timer is reset after a new serial character is received or the processor reads the receiver FIFO.

21.4.1.2 Transmitter

Transmitter interrupts occur, as described below, when the transmitter FIFO and transmitter interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ETBEI] = 1.

The transmitter holding register interrupt (UARTx_IIR = 0xC2) occurs when transmit FIFO is empty, and is cleared as soon as the transmitter holding register is written to or the IIR is read. One to 16 characters may be written to the transmitter FIFO while servicing this interrupt.

The transmitter FIFO empty indications are delayed by one character time minus the last stop bit time whenever the following event occurs: UARTx_LSR[THRE] = 1 and there were less than two bytes simultaneously present in the transmit FIFO since the last UARTx_LSR[THRE] = 1. If UARTx_FCR[FE] = 1 (FIFOs enabled), the first transmitter interrupt after changing UARTx_FCR[FE] is immediate.

Receiver FIFO trigger level interrupts, received data available interrupts, and character timeouts all have equivalent second interrupt priority. Current transmitter holding register empty interrupt and Transmit FIFO empty have equivalent third interrupt priority.

21.4.2 Polled Mode

When UARTx_FCR[FE] = 1 (FIFOs enabled), and UARTx_IER[5:7] are all set to 0 (interrupts disabled), the UART is in FIFO polled mode of operation. The receiver and transmitter are controlled separately, so either can be in polled mode of operation. In polled mode, the user program must check the UARTx_LSR to see the status of the receiver and/or transmitter.

UARTx_LSR3:6 specifies which errors (if any) have occurred. Character status errors are handled in the same way as in interrupt mode. Since UARTx_IER[ELSI] = 0, the IIR is not affected. UARTx_LSR[DR] is set as long as there is at least one character in the receiver FIFO. UARTx_LSR[THRE] indicates if the transmitter FIFO is empty. UARTx_LSR[TEMT] indicates if the transmitter FIFO and the transmitter shift register are empty. UARTx_LSR[RFE] indicates if there are any errors in the receiver FIFO.

In FIFO polled mode, there are no character timeout or trigger levels; however, the FIFOs are still capable of holding characters.

21.5 UART and Sleep Mode

Both UARTs can be placed in sleep mode via the UART sleep bits in the CPC0_ER register (CPC0_ER[UART0:UART1]). The most common usage would be to save a little power if one or both of the UARTs were not going to be used.

Using sleep mode dynamically requires careful software control to make sure the UARTs are idle before putting them to sleep.

21.6 DMA Operation

The DMA controller can be configured to perform DMA operations using UART0, which appears as an 8-bit peripheral to the DMA controller. When selected, the UART receiver is internally wired to the DMAReq and DMAAck signals of DMA channel 2, and the transmitter is internally wired to the DMAReq and DMAAck signals of DMA channel 3.

The UART can be operated in FIFO mode or non-FIFO mode. In FIFO mode, the transfers can be done as single transfers (DMA mode 0) or multiple transfers (DMA mode 1), depending on the setting of the DMS field in the FIFO Control Register (FCR). In non-FIFO mode, DMA transfers are performed using single transfers, using the UART's DMA mode 0. This section describes proper UART0 DMA programming. For more information on general DMA programming, see Chapter 18, "Direct Memory Access Controller," on page 18-1.

21.6.1 Chip Control Register 0 (CPC0_CR0)

Only CPC0_CR0 fields related to UART are shown in Figure 21-13. Other non-related functions are not shown here.

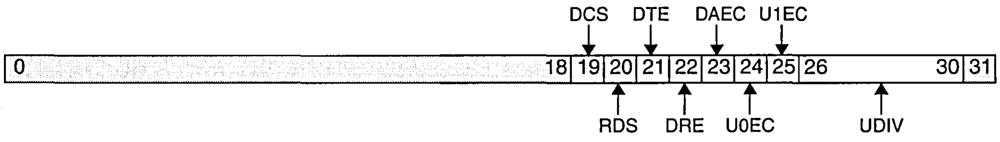


Figure 21-13. Chip Control Register 0 (CPC0_CR0)

0:18		Reserved.
19	DCS	DSR/CTS select 0 DSR is selected. 1 CTS is selected.
20	RDS	RTS/DTR select 0 RTS is selected. 1 DTR is selected.
21	DTE	DMA Transmit Enable for UART0 0 DMA transmit channel is disabled. 1 DMA transmit channel is enabled.
22	DRE	DMA Receive Enable for UART0 0 DMA receive channel is disabled. 1 DMA receive channel is enabled.

23	DAEC	DMA Allow Enable Clear for UART0 0 DTE and DRE for UART0 are not cleared when the UART receives a corresponding terminal count. 1 DTE and DRE for UART0 are cleared when the UART receives a corresponding terminal count.	
24	U0EC	Select External Clock for UART0 0 UART0 uses the internally derived serial clock. 1 UART0 uses the UARTSerClk external serial clock input.	
25	U1EC	Select External Clock for UART1 0 UART1 uses the internally derived serial clock. 1 UART1 uses the UARTSerClk external serial clock input.	
26:30	UDIV	UART Divisor 00000 Divide by 1 00001 Divide by 2 00010 Divide by 3 . . . 11110 Divide by 31 11111 Divide by 32	UDIV specifies the divider ratio between the CPU and UART serial clock frequencies. UART0 and UART1 can use a serial clock frequency derived from the CPU clock frequency divided by UDIV, or use the UARTSerClk external serial clock input. For example, if the CPU runs at 200MHz, a UDIV of 20 sets the serial clock frequency at 10MHz. **Note:** Maximum serial clock frequency is slightly less than 1/2x OPB frequency.
31		Reserved.	

21.6.2 Transmitter DMA Mode

The UART0 Transmit Channel Enable field of the Chip Control Register 0, CPC0_CR0[DTE], controls the use of the serial port transmitter as a DMA destination. For the transmitter in DMA mode 0, when the FIFOs are disabled or the FIFOs are enabled and there are no characters in the TX FIFO or Transmit Holding Register (THR), the DMA request goes active. Once activated, the DMA request goes inactive after the first character is loaded into the TX FIFO or THR. For the transmitter in DMA mode 1, when FIFOs are enabled and there is at least one unfilled position in the TX FIFO, the DMA request goes active. This signal will become inactive when the TX FIFO is completely full. To operate in this mode, DMA Channel Control Register 3 (DMA0_CR3) must be configured to accept DMA requests from an internal source. Setting the Peripheral Location (PL) bit of DMA0_CR3 to a logic 1 configures DMA channel to accept DMA requests from UART0. Table 21-5 lists required register settings for UART0 transmit transfers. Other DMA registers and register fields must be programmed appropriately, see Chapter 18, "Direct Memory Access Controller," on page 18-1 for more information.

Table 21-5. UART0 Transmitter DMA Mode Register Field Settings

Register [Field]	Meaning
CPC0_CR0[DTE]=1	UART0 DMA Transmit channel is enabled using DMA channel 3.
CPC0_CR0[DAEC]	Set to 0 to not clear CPC0_CR0[DTE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA0_CR3[TD]=0	DMA Channel 3 transfer direction is from memory to peripheral.
DMA0_CR3[PL]=1	DMA Channel 3 peripheral is on the OPB (UART0).
DMA0_CR3[PW]=00	Peripheral width is byte (8 bits).
DMA0_CR3[TM]=00	DMA Channel 3 is in peripheral mode.
DMA0_CR3[PWC]=000010	Peripheral Wait cycles, how long the internal DMAck is active. Three cycles are required.
DMA0_CR3[PHC]=000	Peripheral Hold Cycles are 0.
DMA0_CR3[ETD]=1	EOT/TC is programmed as terminal count output.
UART0_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.

Note: When using DMA Channel 3 for UART0 transmitter transfers, external DMA transfers cannot be performed on this channel.

21.6.3 Receiver DMA Mode

The UART0 Receive Enable field of the Chip Control Register 0, CPC0_CR0[DRE], controls the use of the serial port receiver as a DMA source. For the receiver in DMA mode 0, when there is at least one character in the RX FIFO or Receive Buffer Register, RBR, the DMA request goes active. Once activated, the DMA request goes inactive when there are no more characters in the FIFO or RBR. For the receiver in DMA mode 1, when the FIFOs are enabled and the trigger level or the timeout has been reached, the DMA request goes active. Once activated, it will go inactive when there are no more characters in the RX FIFO or RBR. To operate in this mode, DMA Channel Control Register 2 (DMA0_CR2) must be configured to accept DMA requests from an internal source. Setting the Peripheral Location (PL) bit of DMA0_CR2 to a logic 1 configures DMA channel to accept DMA requests from UART0. Table 21-6 lists required register settings for UART0 receiver transfers. Other DMA registers and register fields must be programmed appropriately, see Chapter 18, "Direct Memory Access Controller," on page 18-1 for more information.

Table 21-6. UART0 Receiver DMA Mode Register Field Settings

Register [Field]	Meaning
CPC0_CR0[DRE]=1	UART0 DMA Receiver channel is enabled using DMA channel 2.
CPC0_CR0[DAEC]	Set to 0 to not clear CPC0_CR0[DRE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA0_CR2[TD]=1	DMA Channel 2 transfer direction is from peripheral to memory.
DMA0_CR2[PL]=1	DMA Channel 2 peripheral is on the OPB (UART0).
DMA0_CR2[PW]=00	Peripheral width is byte (8 bits).
DMA0_CR2[TM]=00	DMA Channel 2 is in peripheral mode.
DMA0_CR3[PWC]=000010	Peripheral Wait cycles, how long the internal DMAAck is active. Three cycles are required.
DMA0_CR2[PHC]=000	Peripheral Hold Cycles are 0.
DMA0_CR2[ETD]=1	EOT/TC is programmed as terminal count output.
UART0_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.

Note: When using DMA Channel 2 for UART0 receiver transfers, external DMA transfers cannot be performed on this channel.

Chapter 22. IIC Bus Interface

The PPC405GP provides an inter-integrated circuit (IIC) bus interface complying with specifications contained in the Philips® Semiconductors document *The I^2C-bus and how to use it (including specifications)* (1995 update).

The IIC bus is a two-wire, bi-directional, open-drain, low-speed serial interface. The serial clock (IICSCL) and serial data (IICSDA) lines are bidirectional, to support multiple bus masters and to mix high- and low-speed devices on the same bus.

The IIC interface (referred to as IIC to distinguish it from the Philips I^2C bus) supports the following standard and enhanced features:

- 100-kHz and 400-kHz operation
- 8-bit data transfers
- 7-bit and 10-bit addressing
- Slave transmitter and receiver
- Master transmitter and receiver
- Multiple bus masters

The IIC interface can switch between 7-bit and 10-bit addressing under program control.

22.1 Addressing

The IIC interface supports 7-bit and 10-bit addressing for master and slave transfers.

Addressing is described in detail in "IIC0 Low Master Address Register" on page 22-5, "IIC0 High Master Address Register" on page 22-6, "IIC0 Low Slave Address Register" on page 22-14, and "IIC0 High Slave Address Register" on page 22-14.

Descriptions of addressing modes and address formats follow.

22.1.1 Addressing Modes

For master transfers, the address mode (AMD) field of the IIC Control register (IIC0_CNTL) controls whether 7-bit or 10-bit addresses are used. If IIC0_CNTL[AMD] = 0, addresses contain 7 bits; if IIC0_CNTL[AMD] = 1, addresses contain 10 bits.

For slave transfers, the contents of the IIC0 High Slave Address register (IIC0_HSADR) determines whether 7-bit or 10-bit addressing is used. If IIC0_HSADR = 0b00000000, 7-bit addressing is used. If 10-bit addressing is to be used for slave transfers, IIC0_HSADR = 0b11110*yyx*, where *yy* contains the high-order bits of the 10-bit address, and x is a don't care.

Programming Note: For slave transfers, IIC0_CNTL[AMD] does not control addressing mode.

22.1.2 Seven-Bit Addresses

Figure 22-1 illustrates a 7-bit address. For master transfers, the address bits 0 through 6 (A0:A6) are read from IIC0_LMADR. For slave transfers, A0:A6 are read from IIC0_LSADR. Bit 7 of address byte 0 contains a transfer type bit provided by the IIC interface.

	MSb							LSb
Address Byte 0	A0	A1	A2	A3	A4	A5	A6	R/W
	Bit 0							Bit 7

Figure 22-1. 7-Bit Addressing

22.1.3 Ten-Bit Addresses

Figure 22-2 illustrates a 10-bit address. A0:A1 of address byte 0 are read from IIC0_HMADR[A6:A7] (for master transfers) or IIC0_HSADR[A6:A7] (for slave transfers). These are the two highest-order address bits transmitted on the IIC bus. Bit 7 of address byte 0 contains a transfer type bit provided by the IIC interface.

For 10-bit addressing for master or slave transfers, respectively, IIC0_HMADR[A0:A4] and IIC0_HSADR [A0:A4] must contain 0b11110.

The low-order byte of the 10-bit address, contained in A0:A7 of address byte 1, are read from IIC0_LMADR or IIC0_LSADR for master or slave transfers, respectively.

	MSb							LSb
Address Byte 0	1	1	1	1	0	A0	A1	R/W
	Bit 0							Bit 7

	MSb							LSb
Address Byte 1	A2	A3	A4	A5	A6	A7	A8	A9
	Bit 0							Bit 7

Figure 22-2. 10-Bit Addressing

22.2 IIC Registers

IIC registers are accessed at memory locations 0xEF600500–0xEF600510 in the PPC405GP.

Table 22-1 lists the IIC registers. Descriptions of the registers, in the listed order, follow in "IIC Register Descriptions" on page 22-3.

Table 22-1. IIC Registers

Register	Mnemonic	PPC405GP Memory Map	Address	Access	Effect of Reset	Bits
IIC0 Master Data Buffer	IIC0_MDBUF	0xEF60 0500	0x0	R/W	Cleared	8,16
Reserved		0xEF60 0501	0x1			

Table 22-1. IIC Registers

Register	Mnemonic	PPC405GP Memory Map	Address	Access	Effect of Reset	Bits
IIC0 Slave Data Buffer	IIC0_SDBUF	0xEF60 0502	0x2	R/W	Cleared	8,16
IIC0 Reserved		0xEF60 0503	0x3			
IIC0 Low Master Address	IIC0_LMADR	0xEF60 0504	0x4	R/W	No	8
IIC0 High Master Address	IIC0_HMADR	0xEF60 0505	0x5	R/W	No	8
IIC0 Control	IIC0_CNTL	0xEF60 0506	0x6	R/W	Cleared	8
IIC0 Mode Control	IIC0_MDCNTL	0xEF60 0507	0x7	R/W	Cleared	8
IIC0 Status	IIC0_STS	0xEF60 0508	0x8	R/W	Cleared	8
IIC0 Extended Status	IIC0_EXTSTS	0xEF60 0509	0x9	R/W	Cleared	8
IIC0 Low Slave Address	IIC0_LSADR	0xEF60 050A	0xA	R/W	No	8
IIC0 High Slave Address	IIC0_HSADR	0xEF60 050B	0xB	R/W	No	8
IIC0 Clock Divide	IIC0_CLKDIV	0xEF60 050C	0xC	R/W	Cleared	8
IIC0 Interrupt Mask	IIC0_INTRMSK	0xEF60 050D	0xD	R/W	Cleared	8
IIC0 Transfer Count	IIC0_XFRCNT	0xEF60 050E	0xE	R/W	Cleared	8
IIC0 Extended Control and Slave Status	IIC0_XTCNTLSS	0xEF60 050F	0xF	R/W	Cleared	8
IIC0 Direct Control	IIC0_DIRECTCNTL	0xEF60 0510	0x10	R/W	0x0f	4

22.3 IIC Register Descriptions

The following sections contains the bit definitions for the various registers in the IIC interface.

22.3.1 IIC0 Master Data Buffer

The IIC0 Master Data Buffer (IIC0_MDBUF) is a 1-byte × 4-byte first-in/first-out (FIFO) buffer. A byte written to IIC0_MDBUF is placed into the fourth FIFO stage. If the third FIFO stage is empty, the data is moved into the third stage at the next OPB clock. This process is repeated for the second and first FIFO stages at each successive OPB clock. The byte moves through the buffer until it reaches the deepest unoccupied stage of the FIFO. The buffer data is either written on the IIC bus when the IIC interface performs a write, or is received from the IIC bus when the IIC interface performs a read.

Figure 22-3 illustrates the IIC0_MDBUF.

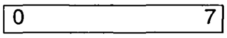


Figure 22-3. IIC0 Master Data Buffer (IIC0_MDBUF)

0		Data bit
1		Data bit
2		Data bit
3		Data bit
4		Data bit

5		Data bit
6		Data bit
7		Data bit

IIC0_MDBUF is cleared (flushed and set to empty) whenever the IIC interface is reset, or IIC0_MDCNTL[FMB] = 1. Figure 22-4 shows the four FIFO stages.

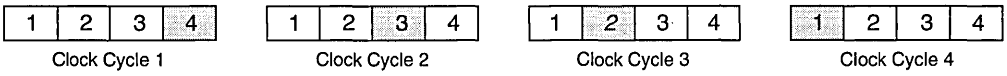


Figure 22-4. FIFO Stages

When IIC0_MDBUF is written with a byte, the byte is placed in the FIFO. The hardware pushes the byte into the deepest unoccupied stage in the FIFO and advances one FIFO stage per clock. Thus, if the FIFO is empty, four clocks are needed (one per stage) for the byte to walk to the first stage of the FIFO. This timing is important to consider when reading the IIC0_MDBUF immediately after data is written. When a master transfer is requested, the IIC interface handles this latency.

If a byte is written to IIC0_MDBUF while the FIFO is full, the byte is discarded and not placed into the FIFO.

If IIC0_MDBUF is written with two bytes in a halfword access, and there is space in the FIFO, byte 0 of the halfword is placed ahead of byte 1 in the FIFO. The MSB, byte 0, is written to the IIC bus first, followed by the LSB, byte 1.

IIC0__MDBUF receives data from the IIC bus when the requested master transfer is a read. The first byte received is the first byte read by software from IIC0_MDBUF.

For halfword reads the first byte received is MSB ,byte 0, and the following byte is LSB, byte 1. When an empty FIFO is read, the byte (or halfword) most recently read is returned.

Care must be taken not to start a requested master operation while there is data in IIC0_MDBUF. If, for example, a master read transfer is requested and obsolete data is in IIC0_MDBUF, the obsolete data would be presented, to the requesting software, as data read by the requested transfer.

22.3.2 IIC0 Slave Data Buffer

The IIC0 Slave Data Buffer (IIC0_SDBUF) is a copy of IIC0_MDBUF. The data contained in the slave buffer is either received from the IIC bus when the IIC interface is addressed as a slave during a write, or is written on the IIC bus when the IIC interface is addressed as a slave during a read operation.

IIC0_SDBUF works in the same way as IIC0_MDBUF, except that IIC0_SDBUF is used only to store data sent or received in slave transfers on the IIC bus. This enables overlapping slave and master transfers on the IIC bus.

Bit assignments for the IIC0_MDBUF and IIC0_SDBUF are identical, as illustrated in Figure 22-5.

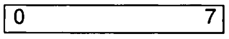


Figure 22-5. IIC0 Slave Data Buffer (IIC0_SDBUF)

0		Data bit
1		Data bit
2		Data bit
3		Data bit
4		Data bit
5		Data bit
6		Data bit
7		Data bit

IIC0_SDBUF is cleared (flushed and set to empty) whenever the IIC interface is reset, or IIC0_MDCNTL[FSB] = 1.

22.3.3 IIC0 Low Master Address Register

The IIC0 Low Master Address (IIC0_LMADR) and IIC0 High Master Address Register (IIC0_HMADR) form addresses that the IIC interface transmits on the IIC bus.

Programming Note: IIC0_HMADR is used only for 10-bit addressing.

When IIC0_CNTL[AMD] = 0 (7-bit addressing), only IIC0_LMADR is written. IIC0_LMADR[A0:A6] form the address transmitted on the IIC bus; IIC0_LMADR[A7] is a don't care. When IIC0_CNTL[AMD] = 1 (10-bit addressing), IIC0_LMADR[A0:A7] form the second byte address transmitted on the IIC bus.

Figure 22-6 illustrates the IIC0_LMADR.

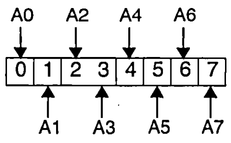


Figure 22-6. IIC0 Low Master Address Register (IIC0_LMADR)

0	A0	Address bit 0
1	A1	Address bit 1
2	A2	Address bit 2
3	A3	Address bit 3
4	A4	Address bit 4
5	A5	Address bit 5

6	A6	Address bit 6	LSb for 7-bit addresses
7	A7	Address bit 7	LSb for 10-bit addresses; don't care for 7-bit addresses

22.3.4 IIC0 High Master Address Register

IIC0 High Master Address Register (IIC0_HMADR) is not used for 7-bit addressing.

When IIC0_CNTL[AMD] = 1 (10-bit addressing), IIC0_HMADR must be programmed to 0b1111 0*yyx*, where *yy* are the high-order bits of a 10-bit address and *x* is a don't care.

Thus, in 10-bit address mode, IIC0_HMADR[A5:A6] are the two highest-order bits of the 10-bit address and IIC0_HMADR[A7] is a don't care. IIC0_LMADR contains the low-order byte of the 10-bit address.

Figure 22-7 illustrates the IIC0_HMADR.

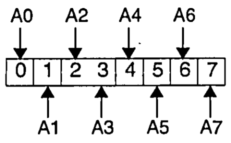


Figure 22-7. IIC0 High Master Address Register (IIC0_HMADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	MSb for 10-bit addresses
6	A6	Address bit 6	Next to MSb for 10-bit addresses
7	A7	Address bit 7	Don't care for 10-bit addresses

22.3.5 IIC0 Control Register

The IIC0 Control Register (IIC0_CNTL) starts and stops IIC interface master transfers on the IIC bus. When a transfer begins, the IIC interface uses the values in IIC0_CNTL to determine the type and size of the transfer.

Programming Note: IIC0_CNTL *must* be the last register programmed. Whenever IIC0_CNTL[PT] = 1, the IIC interface attempts to perform the requested transfer, using values set in other registers. Note that not all IIC registers must be programmed before performing each transfer.

During transfers, and after transfers finish, software can read the IIC0_STS and IIC0_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Only IIC0_CNTL[PT] is cleared when a requested master transfer is complete; the remaining bits are not affected.

Figure 22-8 illustrates the IIC0_CNTL.

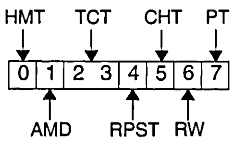


Figure 22-8. IIC0 Control Register (IIC0_CNTL)

0	HMT	Halt Master Transfer 0 Normal transfer operation. 1 Issue Stop signal on the IIC bus as soon as possible to halt master transfer.	If no transfer is in progress, no action is taken. IIC0_CNTL[PT] needs not be set. If IIC0_MDCNTL[EINT] = 1, an interrupt is generated.
1	AMD	Addressing Mode 0 Use 7-bit addressing. 1 Use 10-bit addressing.	Does not affect slave transfers.
2:3	TCT	Transfer Count 00 Transfer one byte. 01 Transfer two bytes. 10 Transfer three bytes. 11 Transfer four bytes.	
4	RPST	Repeated Start 0 Normal start operation 1 Use repeated Start function to start transfer.	
5	CHT	Chain Transfer 0 Transfer is only or last transfer. 1 Transfer is one of a sequence of transfers (but not last in sequence).	Completion of a requested transfer causes a Stop signal to be issued on the IIC bus.
6	RW	Read/Write 0 Transfer is a write. 1 Transfer is a read.	
7	PT	Pending Transfer 0 Most recent requested transfer is complete. 1 Start transfer if bus is free.	

Table 22-2 summarizes IIC interface operation for settings of IIC0_CNTL[HMT, RPST, CHT, PT] x is a don't care.

Table 22-2. IIC Response to IIC0_CNTL Field Settings

IIC0_CNTL Fields				Resulting Action on IIC Bus and Inside IIC Interface
HMT	RPST	CHT	PT	
0	x	x	0	No action taken
0	0	1	1	Start, Transfer, ACK on last byte, Pause
0	0	0	1	Start, Transfer, NACK on last byte, Stop
1	x	x	x	NACK on current byte, Stop
0	1	x	1	Start, Transfer, NACK on last byte, Wait

Settings of IIC0_CNTL[HMT, RPST, CHT, PT] result in the following actions.

Start	IIC Start condition generated, if the IIC interface was stopped or waiting.
Stop	IIC Stop condition generated; IIC interface enters the Stop condition.
ACK	IIC Acknowledge condition generated.
NACK	IIC Not Acknowledge condition generated, if performing a read.
Transfer	Requested bytes are transferred.
Pause	IIC interface enters the Pause state.
Wait	IIC interface enters the Wait state.

IIC0_CNTL[HMT] overrides IIC0_CNTL[RPST, CHT].

IIC0_CNTL[RPST, PT] overrides IIC0_CNTL[CHT].

22.3.6 IIC0 Mode Control Register

The IIC0 Mode Control Register (IIC0_MDCNTL) sets the major modes of operation on the IIC bus. In addition, IIC0_MDCNTL can force the data buffers into the empty state.

In typical applications, IIC0_MDCNTL is configured once, during software initialization. Applications providing complex error handling may reconfigure this register more often.

Programming Note: IIC0_CLKDIV must be initialized before IIC0_MDCNTL. IIC0_LSADR and IIC0_HSADR should also be configured before IIC0_MDCNTL.

Note that the IIC hardware does not implement time-out functions on the IIC bus. Such functions must be implemented, in software, by setting IIC0_CNTL[HMT] = 1, or setting IIC0_XTCNTLSS[SRST] = 1.

Regarding IIC0_MDCNTL[HSCL], a "slave not ready" condition occurs during a slave receive operation, if a slave has no free space in its slave data buffer at the start of a write operation, or if the slave data buffer fills during the write. In a slave transmit operation, a slave not ready condition occurs if a slave has no data in its slave data buffer at the start of a read operation, or if the slave data buffer becomes empty during the read.

Using IIC0_MDCNTL[HSCL] to handle slave not ready conditions can affect system performance. A slave holding the IICSCL signal low guarantees data delivery to or from the requesting master, but prevents other masters from performing transfers over the IIC bus. There is no general rule for handling slave not ready conditions; each system has its own requirements.

Figure 22-9 illustrates the IIC0_MDCNTL.

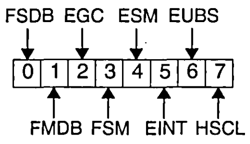


Figure 22-9. IIC0 Mode Control Register (IIC0_MDCNTL)

0	FSDB	Flush Slave Data Buffer 0 Normal operation 1 Set slave data buffer to empty.	Cleared after buffer is emptied.
1	FMDB	Flush Master Data Buffer 0 Normal operation 1 Set master data buffer to empty.	Cleared after buffer is emptied.
2	EGC	Enable General Call 0 Ignore general call on IIC bus. 1 Respond to general call on IIC bus.	IIC0_MDCNTL[ESM] overrides this field; if IIC0_MDCNTL[ESM] = 1, a general call is ignored.
3	FSM	Fast/Standard Mode 0 IIC transfers run at 100 kHz (standard mode). 1 IIC transfers run at 400 kHz (fast mode).	
4	ESM	Enable Slave Mode 0 Slave transfers are ignored. 1 Slave transfers are enabled.	Program IIC0_LSADR and IIC0_HSADR before setting this field.
5	EINT	Enable Interrupt 0 Interrupts are disabled. 1 Enables interrupts for interrupts enabled in IIC0_INTRMSK.	
6	EUBS	Exit Unknown IIC Bus State 0 Normal operation. 1 IIC bus control state machine exits unknown bus state, if in an unknown state.	If the IIC bus control state machine is in a known state, setting IIC0_MDCNTL[EUBS] = 1 has no effect.
7	HSCL	Hold IIC Serial Clock Low 0 If slave is not ready, issue a NACK in response to slave transfer request. 1 If slave is not ready, hold the IICSCL signal low until slave is ready.	This field is used only when in slave mode.

22.3.7 IIC0 Status Register

The IIC0 Status register (IIC0_STS) provides a summary of the state of the IIC interface and the status of any previously requested master transfer.

During and after transfers, software can read the IIC0_STS and IIC0_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Programming Note: IIC0_STS should be the first register read by an interrupt or error handler routine. IIC0_STS can also be read in a polling loop if software does not use the IIC interrupts.

Software must clear IIC0_STS before requesting another master transfer, except for IIC0_STS[SSS]. Because IIC0_STS[SSS] involves slave transfers, it can remain set.

Figure 22-10 illustrates IIC0_STS.

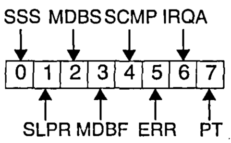


Figure 22-10. IIC0 Status Register (IIC0_STS)

0	SSS	Slave Status Set 0 No slave operations are in progress. 1 Slave operation is in progress.	Read-only; this field is set when any of the following fields are set: IIC0_XTCNTLSS[SRC, SRRS, SWC, SWRS].
1	SLPR	Sleep Request 0 Normal operation. 1 Sleep mode (CPC0_ER[IIC] = 1).	Read-only. The IIC interface is awakened when a start signal is detected on the IIC bus or when the CPC0_ER[IIC] is cleared.
2	MDBS	Master Data Buffer Status 0 Master data buffer is empty. 1 Master data buffer contains data.	Read-only.
3	MDBF	Master Data Buffer Full 0 Master data buffer is not full. 1 Master data buffer is full.	Read-only.
4	SCMP	Stop Complete 0 No request to halt transfer, or master data transfer, is complete. 1 Request to halt transfer, or master data transfer, is complete.	To clear IIC0_STS[SCMP], set IIC0_STS[SCMP] = 1.
5	ERR	Error 0 No error has occurred. 1 One of the following fields is set: IIC0_EXTSTS[LA, ICT, XFRA] = 1.	Read-only.

6	IRQA	IRQ Active 0 No IIC interrupt has been sent to the universal interrupt controller (UIC). 1 An IIC interrupt has been sent to the UIC.	To clear IIC0_STS[IRQA], set IIC0_STS[IRQA] = 1. If IIC0_MDCNTL[EINT] = 0, then IIC0_STS[IRQA] is not set.
7	PT	Pending Transfer 0 No transfer is pending, or transfer is in progress. 1 Transfer is pending.	Read-only.

The Error and Pending Transfer, IIC0_STS[ERR, PT], bit fields indicate the success or failure of the requested transfer. Table 22-3 interprets the transfer status for all possible combinations of the IIC0__STS[ERR,PT] bit fields.

Table 22-3. IIC0_STS[ERR, PT] Decoding

ERR	PT	Status
0	0	Requested transfer completed without errors
0	1	Requested transfer is in progress; no errors were detected
1	0	Requested transfer is complete, but not all data was transferred
1	1	Requested transfer is in progress; but an error was detected

Programming Note: Software should not take any action regarding a master transfer unless all pending transfers are completed, IIC0_STS[PT] = 0.

If an error requires the IIC interface to send a Stop, the Stop Complete bit field is set, IIC0_STS[SCMP] = 1. Note that slave operations should be serviced regardless of the state of a requested master transfer.

The IIC interface is placed in sleep mode by setting the CPC0_ER[IIC] via software. Awaking the IIC interface is possible directly through software by clearing the CPC0_ER[IIC] or indirectly by detecting a Start condition on the IIC bus. When a Start condition is detected, the IIC interface is awakened, clearing the CPC0_ER[IIC] and the IIC0__STS[SLPR].

The IIC0_STS[MDBS, MDBF] contain the current status of the Master Data Buffer, IIC0_MDBUF. When the IIC0_MDBUF contains data, IIC0_STS[MDBS] is set. When the IIC0__MDBUF is full, IIC0_STS[MDBF] is set.

The state of the IIC0_MDBUF is not instantly recorded by the IIC0_STS[MDBS, MDBF]. The delay depends on the size of the buffer access. For halfword accesses, these fields are valid on the third OPB clock following the transfer. For byte accesses, these fields are valid on the second OPB clock following the transfer.

22.3.8 IIC0 Extended Status Register

The IIC0 Extended Status register (IIC0_EXTSTS) reports additional IIC status.

During and after transfers, software can read the IIC0_STS and IIC0_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Figure 22-11 illustrates the IIC0_EXTSTS.

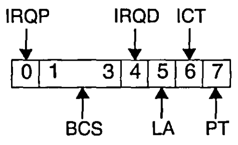


Figure 22-11. IIC0 Extended Status Register (IIC0_EXTSTS)

0	IRQP	IRQ Pending 0 No IRQ is pending. 1 An IRQ is active, another IRQ is on-deck, and another interrupt-generating condition has occurred.	• IIC0_EXTSTS[IRQP] might be set momentarily while an IRQ moves from the Pending to the On-deck state. • An interrupt remains pending, IIC0_EXTSTS[IRQP]=1, until the current on-deck interrupt becomes active, IIC0_STS[IRQD]=0 and IIC0_STS[IRQA]]=1. • Writing 1 to IIC0_EXTSTS[IRQP] clears the field. • When the IIC interrupt is disabled, IIC0_MDCNTL[IRQP] = 0, IIC0_EXTSTS[IRQP] should be ignored.
1:3	BCS	Bus Control State 000 Unused; if this value is read, a major IIC hardware problem occurred. 001 Slave-selected state; the IIC interface has detected and decoded a slave transfer request on the IIC bus. 010 Slave Transfer state; the IIC interface has detected but has not decoded a slave transfer request on the IIC bus. 011 Master Transfer state; entered after a master transfer request has started on the IIC bus. 100 Free Bus state; the bus is free and no transfer request is pending. 101 Busy Bus state; the bus is busy. 110 Unknown state; value after IIC reset. 111 Unused; if this value is read, a major IIC hardware problem occurred.	Read-only.

4	IRQD	IRQ On-Deck 0 No IRQ is on-deck. 1 An interrupt is active, and another interrupt-generating condition has occurred.	• IIC0_EXTSTS[IRQD] might be set momentarily while an IRQ moves from the On-deck to the Active state. • An interrupt remains on-deck, IIC0_EXTSTS[IRQD = 1, until the current active interrupt is no longer active, IIC0_STS[IRQA] = 0. • If IIC0_EXTSTS[IRQP] = 1, IIC0_EXTSTS[IRQD] is set on the next OPB clock. • Writing 1 to IIC0_EXTSTS[IRQD] clears the field. • When the IIC interrupt is disabled, IIC0_MDCNTL[IRQP]=0, IIC0_EXTSTS[IRQD] should be ignored.
5	LA	Lost Arbitration 0 Normal operation. 1 Loss of arbitration has ended the requested master transfer.	• If arbitration is lost, any requested master transaction may have terminated prematurely. Read data may be incomplete and not all write data may have been written. • If arbitration is lost during a repeat start, the master may not own the IIC bus.
6	ICT	Incomplete Transfer 0 Normal operation. 1 Some of the bytes of the requested master transfer were not transferred.	For an incomplete transfer, read the transfer count, IIC0_XFRCNT, to determine how bytes were transferred.
7	XFRA	Transfer Aborted 0 No transfer is pending, or transfer is in progress. 1 A requested master transfer was aborted by a NACK during the transfer of the address byte, or was aborted because arbitration was lost. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.	Transfer aborted. When set to a 1, a requested master transfer was aborted by a NOT acknowledge during the transfer of the address byte. It is also set to a 1 when a requested master transfer loses data. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.

IIC0_EXTSTS[IRQP, IRQD] and IIC0_STS[IRQA] provide a FIFO for storing interrupts. A new interrupt is considered pending, and remains pending while an on-deck interrupt is present. Once the on-deck interrupt becomes active, the pending interrupt moves on-deck, and remains on-deck until there is no active interrupt. When the active interrupt is cleared, the on-deck (initially pending) interrupt becomes active.

Programming Note: An active interrupt remains active until software clears it.

IIC0_EXTSTS[BCS] indicates the state of the IIC interface. The field is read-only.

IIC0_EXTSTS[LA, ICT, XFRA] are cleared when IIC0_EXTSTS[PT] = 1.

Writing 1 to IIC0_EXTSTS[IRQP, IRQD, LA, ICT, XFRA] clears these fields.

22.3.9 IIC0 Low Slave Address Register

The IIC0 Low Slave Address Register (IIC0_LSADR) and IIC0_ High Slave Address Register (IIC0_HSADR) program the slave address of the IIC interface. IIC0_HSADR is used only for 10-bit addressing, and is not programmed in 7-bit addressing mode.

When 7-bit addressing is used, IIC0_LSADR is written with the slave address; IIC0_HSADR must be written with zeros. For 7-bit addressing, IIC0_LSADR[A0:A6] contain the address transmitted on the IIC bus; IIC0_LSADR[A7] is a don't care.

When 10-bit addressing is used, IIC0_LSADR[A0:A7] contain the second address byte transmitted on the IIC bus.

Figure 22-6 illustrates the IIC0_LSADR.

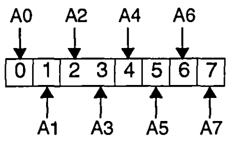


Figure 22-12. IIC0 Low Slave Address Register (IIC0_LSADR)

0	A0	Address bit 0	
1	A1	Address bit 1	
2	A2	Address bit 2	
3	A3	Address bit 3	
4	A4	Address bit 4	
5	A5	Address bit 5	
6	A6	Address bit 6	LSb for 7-bit addresses
7	A7	Address bit 7	LSb for 10-bit addresses; don't care for 7-bit addresses

22.3.10 IIC0 High Slave Address Register

For 7-bit addressing, set IIC0 High Slave Address Register (IIC0_HSADR) to 0.

To enable 10-bit slave addressing, IIC0_HSADR must be programmed to 0b1111 0*yyx*, where *yy* are the high-order bits of a 10-bit address and *x* is a don't care.

Programming Note: IIC0_HSADR is used only for 10-bit addressing, and should be set to 0 for 7-bit addressing mode.

Thus, in 10-bit address mode, IIC0_HSADR[A6:A7] contain the two highest -order bits of the 10-bit address; IIC0_HSADR[A7] is a don't care. IIC0_LSADR contains the low-order byte of the 10-bit address.

Figure 22-13 illustrates the IIC0_HSADR.

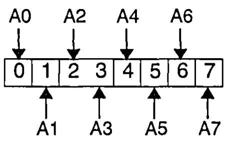


Figure 22-13. IIC0 High Slave Address Register (IIC0_HSADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	MSb for 10-bit addresses
6	A6	Address bit 6	Next to MSb for 10-bit addresses
7	A7	Address bit 7	Don't care for 10-bit addresses

Thus, in 10-bit address mode, bits 0:6 are used to decode the first address byte that was transmitted on the IIC bus, and bit 7 is in a *don't care* state.

22.3.11 IIC0 Clock Divide Register

The IIC0 Clock Divide Register (IIC0_CLKDIV) establishes a reference between the OPB clock and the IIC bus serial clock.

Programming Note: IIC0_CLKDIV must be initialized before IIC0_MDCTRL. Until IIC0_CLKDIV is initialized, all IIC bus activity is ignored.

Figure 22-14 illustrates the IIC0_CLKDIV.

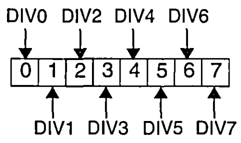


Figure 22-14. IIC0 Clock Divide Register (IIC0_CLKDIV)

0	DIV0	Divisor bit 0
1	DIV1	Divisor bit 1
2	DIV2	Divisor bit 2
3	DIV3	Divisor bit 3
4	DIV4	Divisor bit 4
5	DIV5	Divisor bit 5

6	DIV6	Divisor bit 6
7	DIV7	Divisor bit 7

IIC0_CLKDIV divides PPC405GP's on-chip peripheral bus (OPB) clock to form the base clock for the IIC bus.

Table 22-4 lists the divisor values for several OPB frequency ranges. These divisor values apply for standard and fast mode. Select the divisor value by matching the OPB clock frequency to the corresponding frequency range in Table 22-4. For example, if the OPB clock frequency is 50MHz, select a divisor value of 0x4.

Table 22-4. IIC0 Clock Divide Programming

OPB Frequency Range (MHz)	Divisor Value
20	0x1
$20 < f \leq 30$	0x2
$30 < f \leq 40$	0x3
$40 < f \leq 50$	0x4
$50 < f \leq 60$	0x5

22.3.12 IIC0 Interrupt Mask Register

The IIC0 Interrupt Mask Register (IIC0_INTRMSK) specifies which conditions can generate an IIC interrupt when the IIC interrupt is enabled, IIC0_MDCNTL[EINT]=1.

Figure 22-15 illustrates the IIC0_INTRMSK.

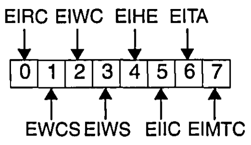


Figure 22-15. IIC0 Interrupt Mask Register (IIC0_INTRMSK)

0	EIRC	Enable IRQ on Slave Read Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave read on the IIC bus. IIC0_XTCNTLSS[SRC] = 1 indicates a Slave Read Complete.
1	EIRS	Enable IRQ on Slave Read Needs Service 0 Disable 1 Enable	The interrupt is activated upon receipt of a slave read on the IIC bus and the slave buffer was empty or went empty and more data was requested on the IIC bus. **Note:** IIC0_XTCNTLSS[SRS] = 1 indicates a Slave Read Needs Service.

2	EIWC	Enable IRQ on Slave Write Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave write on the IIC bus. **Note:** IIC0XTCNTLSS[SWC] = 1 indicates a Slave Write Compete.
3	EIWS	Enable IRQ on Slave Write Needs Service 0 Disable 1 Enable	The interrupt is activated when the slave buffer becomes full during a slave write on the IIC bus. **Note:** IIC0_XTCNTLSS[SWS] = 1 indicates a Slave Write Needs Service.
4	EIHE	Enable IRQ on Halt Executed 0 Disable 1 Enable	
5	EIIC	Enable IRQ on Incomplete Transfer 0 Disable 1 Enable	
6	EITA	Enable IRQ on Transfer Aborted 0 Disable 1 Enable	
7	EIMTC	Enable IRQ on Requested Master Transfer Complete 0 Disable 1 Enable	

22.3.13 IIC0 Transfer Count Register

The IIC0 Transfer Count Register (IIC0_XFRCNT) reports the number of bytes transferred on the IIC bus during a master or a slave operation.

Figure 22-16 illustrates the IIC0_XFRCNT.

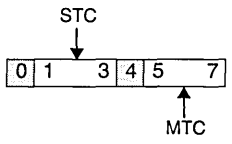


Figure 22-16. IIC0 Transfer Count Register (IIC0_XFRCNT)

0		Reserved
1:3	STC	Slave Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved

4		Reserved
5:7	MTC	Master Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved

IIC0_XFRCNT[MTC] is cleared when there is a pending transfer, IIC0_CNTL[PT] = 1.

IIC0_XFRCNT[STC] is cleared when:

- A slave operation starts on the IIC bus
- Software indicates the slave does not need service by clearing IIC0_XTCNTLSS[SRS] or IIC0_XTCNTLSS[SWS].

22.3.14 IIC0 Extended Control and Slave Status Register

The IIC0 Extended Control and Slave Status Register (IIC0_XTCNTLSS) provides additional control of IIC interface functions and reports the status of slave operations.

Figure 22-17 illustrates the IIC0_XTCNTLSS.

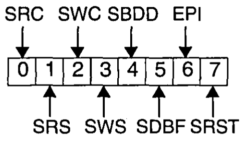


Figure 22-17. IIC0 Extended Control and Slave Status Register (IIC0_XTCNTLSS)

0	SRC	Slave Read Complete 0 Normal operation, or IIC0_MDCNTL[HSCL] = 0, IIC0_SDBUF is empty, and a read operation is in progress. 1 A NACK or Stop condition was received over the IIC bus, or a repeated Start condition ended a read operation.	Check whether the read operation emptied IIC0_SDBUF.

1	SRS	Slave Read Needs Service 0 Normal operation or slave read does not need service. 1 IIC0_SDBUF is empty, and a read operation was requested on the IIC bus. The set condition may also indicate that IIC0_SDBUF is empty due to a slave read and additional data is requested by the master.	1. If IIC0_MDCNTL[HSCL]=0 and IIC0_SDBUF contains no data, the slave will issue a NACK and IIC0_XTCNTLSS[SRS] is set. 2. If IC0MDCNTL[HSCL]=0, and IIC0_SDBUF contains data, the slave will send the data. IIC0_XTCNTLSS[SRS] is not set unless the master request additional data. 3. If IIC0_MDCNTL[HSCL]=0, and IIC0_SDBUF contains no data, the slave will hold IICSCL low to indicate the slave is busy. IIC0_XTCNTLSS[SRS] is set until the IIC0_SDBUF is filled. Once filled, IICSCL is released, IIC0_XTCNTLSS[SRS] is cleared, and the slave sends the data. 4. If IIC0_MDCNTL[HSCL]=1, and IIC0_SDBUF contains data, the slave will send the data. IIC0_XTCNTLSS[SRS] is not set unless the master requests additional data.
2	SWC	Slave Write Complete 0 Normal operation or slave write in progress. 1 A Stop signal was received during a write operation, or a repeated Start condition ended a write operation.	
3	SWS	Slave Write Needs Service 0 Normal operation or slave write does not need service. 1 IIC0_SDBUF is full during a slave write.	1. If IIC0_MDCNTL[HSCL] = 1 and IIC0_SDBUF is full, the slave will hold IICSCL low to indicate the slave is busy. IIC0_XTCNTLSS[SWS] is set until IIC0_SDBUF is empty. Once empty, IICSCL is released, IIC0_XTCNTLSS[SWS] is cleared, and the slave receives the data. 2. If IIC0_MDCNTL[HSCL] = 0 and IIC0_SDBUF is full, the slave will issue a NACK and IIC0_XTCNTLSS[SWS] is set.
4	SBDD	Slave Data Buffer Has Data 0 IIC0_SDBUF is empty 1 IIC0_SDBUF contains data	Read-only
5	SDBF	Slave Data Buffer Full 0 IIC0_SDBUF is not full 1 IIC0_SDBUF is full	Read-only

6	EPI	Enable Pulsed IRQ on Transfer Aborted 0 The internal IIC interrupt signal to the UIC remains active until the status is cleared, IIC0STS[IRQA] =0. 1 The internal IIC interrupt signal to the PPC405GP UIC is active for one OPB clock cycle.
7	SRST	Soft Reset 0 Normal operation 1 Soft reset

Writing a 1 to IIC0_XTCNTLSS[SRC, SRS, SWC, SWS] clears these fields.

Care must be used when changing IIC0_XTCNTLSS[EPI]. If this field changes (from 1 to 0 or from 0 to 1) while an interrupt is active, the IIC interrupt signal is asserted to the universal interrupt controller (UIC).

The IIC0_XTCNTLSS[SBSS, SDBF] contain the current status of the Slave Data Buffer, IIC0_SDBUF. When the IIC0_SDBUF contains data, IIC0_XTCNTLSS[SBDD] is set. When the IIC0_SDBF is full, IIC0_XTCNTLSS[SDBF] is set.

The state of the IIC0_SDBUF is not instantly recorded by the IIC0_XTCNTL[SBSS, SDBF]. The delay depends on the size of the buffer access. For half-word accesses, these fields are valid on the third OPB clock following the transfer. For byte accesses, these fields are valid on the second OPB clock following the transfer.

If any of the following fields: IIC0_XTCNTLSS[SRC, SRS, SWC, SWS] = 1 and IIC0_MDCNTL[HSCL] = 0; no new slave operations will be accepted over the IIC bus. A NACK is issued until IIC0_XTCNTLSS[SRS] or IIC0_XTCNTLSS[SRS], and IIC0_XTCNTLSS[SRS] or IIC0_XTCNTLSS[SRS], are cleared.

Soft reset, IIC0_XTCNTLSS[SRST], provides a last means of recovery from IIC interface or IIC bus failure. Once enabled, soft reset completely resets the IIC interface. All IIC registers are affected. All transmissions from the IIC interface are terminated. Enabling soft reset during an IIC transmission may improperly terminate the transmission and hang the IIC bus.

22.3.15 IIC0 Direct Control Register

The IIC0 Direct Control Register (IIC0_DIRECTCNTL), which controls and monitors the IIC serial clock (IICSCL) and serial data (IICSDA) signal, is used for error recovery when a malfunction is detected on the IIC interface.

Figure 22-18 illustrates the IIC0_DIRECTCNTL.

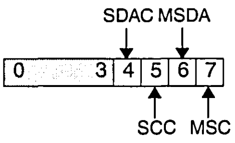


Figure 22-18. IIC0 Direct Control Register (IIC0_DIRECTCNTL)

0:3		Reserved	
4	SDAC	IICSDA Output Control Directly controls the IICSDA output. 0 IICSDA is a logic 0 1 IICSDA is a logic 1	
5	SCC	IICSCL Output Control Directly controls the IICSCL output 0 IICSCL is a logic 0 1 IICSCL is a logic 1	
6	MSDA	Monitor IICSDA Used to monitor the IICSDA input 0 IICSDA is a logic 0 1 IICSDA is a logic 1	Read-only
7	MSC	Monitor IICSCL. Used to monitor the IICSCL input. 0 IICSCL is a logic 0 1 IICSCL is a logic 1	Read-only

IIC0_DIRECTCNTL[SDAC, SCC] can be written to control the IICSDA and IICSCL signals. When controlling the IICSDA and IICSCL signals directly, the IIC controller must be placed in the reset state, IIC0_XTCNTL[SRST] = 1.

IIC0_DIRECTCNTL[MSDA, MSC] are used to verify that IIC0_DIRECTCNTL[SDAC, SCC] were written successfully, and that the IICSCL signals can be controlled. If IIC0_DIRECTCNTL[MSDA, MSC] do not correspond to IIC0_DIRECTCNTL[SDAC, SCC], respectively, toggle IICSCL repeatedly to regain control.

IIC0_DIRECTCNTL[SDAC, SCC, MDSA, MSC] = 1 after a chip or system reset. A Soft Reset, IIC0_XTCNTLSS[SRST] = 1, does not affect the state IIC0_DIRECRCNTL.

22.4 Interrupt Handling

The IIC interface can handle interrupts in two ways. The processor can poll IIC0_STS[SSS, PT]. Alternatively, software can use IIC interface interrupts.

IIC0_MDCNTL[EINT] and the interrupt mask bits in the IIC0_INTRMSK control interrupts. IIC0_XTCNTLSS[EPI] controls whether the IIC IRQ is set at a 1-clock pulse or at a constant level.

The IIC controller guarantees that the interrupt is activated after the state of the has been set to ensure that interrupt handlers read stable values from the IIC interface registers. IIC interrupts can generate interrupt requests to the UIC. If enabled properly in the UIC, an IIC interrupt can interrupt the processor core. See “Interrupt Handling in the Processor Core” on page 10-22 for more information.

Because a master operation can have one interrupt and a slave operation can have two interrupts, the IIC interface can queue up to three interrupts. The current interrupt is referred to as the *active* interrupt. The first interrupt in the queue is referred to as the *on-deck* interrupt; the second queued interrupt is called the *pending* interrupt. The queue holds multiple interrupts until the active interrupt is cleared by writing a 1 to IIC0_STS[IRQA]. When an active interrupt is cleared, the on-deck interrupt becomes the active interrupt and the pending interrupt becomes the on-deck interrupt.

Status associated with an IIC interrupt is immediately set in its corresponding register. Thus, an interrupt handler can see the status for the current and the queued interrupts. This can occur if, for example, the interrupt handler is itself interrupted before reading any status in the IIC registers. If a second or third interrupt occurs before the interrupt handler regains control, the status seen at this time will include status associated with all three interrupting conditions. In this case, the interrupt handler cannot determine the first interrupting condition; all the interrupts will appear to have been merged. Note that if the interrupt handler is intelligent enough to handle and clear all conditions currently set in the status, instead of choosing only one interrupt and servicing it, the routine would not see status for the subsequent interrupts.

A more typical situation involves the case where an interrupt handler has read the IIC status for the active interrupt and a second (on-deck) interrupt occurs. In this case, the status changed after it was first read by the interrupt handler. Because status bits are cleared by writing a 1, no ill effects, such as lost status information, result from this case. To illustrate, consider that the interrupt handler might have read IIC0_XTCNTLSS = 0x10 when the first interrupt occurred. Then, sometime later, the routine clears this status by writing 0x10 back to the IIC0_XTCNTLSS. If the slave write operation completes while the register is read and written, the register will contain a 0x30 when the 0x10 is written. Because writing 0 to a bit has no effect after it is set to 1, the slave write complete status is not lost.

Under certain conditions, the IIC interface merges slave read (write) needs service and slave read (write) complete interrupts into one interrupt. If a slave read (write) needs service interrupt is active, or queued, and a slave read (write) complete interrupt occurs, and IIC0_XTCNTLSS has not yet been read, the two interrupts are merged into a single interrupt. This merge function is performed in the IIC interface logic, and is not under software control.

22.5 General Considerations

1. After a reset, the IIC interface enters the unknown IIC bus state. This state is exited when either activity is seen on the bus or when the exit unknown IIC bus state bit, in the mode control register, is set to a 1. If the IIC interface is being used in a single master system as the master, then the exit unknown IIC bus state bit must be used to force the logic out of the unknown state.
2. Once a byte is written into either the master or slave buffer, a total of four OPB clock periods must occur before the data can be read. Flushing the master or slave buffer also requires four OPB clock periods to complete.

3. The master and slave buffers are actually 4×1 byte-wide FIFOs. Exercise care when using master and slave buffers. As an example, consider the case where one byte of data is written on the IIC bus. The data is first written into the master or slave buffer. After four OPB clock cycles the data is placed on the IIC bus. There is no way to verify data in the buffer without disturbing the IIC transaction. The act of verification requires removing the data. If the data is removed, invalid data is placed on the IIC bus.
4. Use care when monitoring the IIC0_XCNTLSS[SBDD] or to IIC0_STS[MDBS] to determine when data is present. These bits are set to 1 when the buffer contains data in any stage. Consider the case where the master buffer is empty prior to being loaded with a byte received over the IIC bus. The byte enters the fourth stage of the buffer and the IIC0_STS[MDBS] is set to 1. Stages 1, 2, and 3 do not contain data. Therefore, the data is not available for four OPB clock cycles. Any attempt to prematurely read the data yields invalid data.
5. When responding to a slave needs service request, manage the data first. Read data out of the slave buffer, IIC0_SDB, for slave reads or write data into the IIC0_SDB for slave writes. Next clear the slave needs service request. For reads, clear IIC0_XTCNTLSS[SRS]. For writes, clear IIC0_XTCNTLSS[SWS]. Last, clear the active interrupt, IIC0_STS[IRQA] =0.
6. There is no timeout function implemented in the IIC interface. If this type of error recovery function is needed, it must be implemented in software.
7. Avoid the situations listed in Section 7.2 of the *Philips Semiconductors I^2C Specification,* dated 1995. For your convenience, the section is summarized as follows:

   If multiple masters can be simultaneously involved in a transfer to the same address, or device, then the design of the system must be done in such a way that arbitration between:

   – A repeated Start condition and a data bit does not occur.

   – A Stop condition and a data bit does not occur.

   – A repeated Start condition and a Stop condition does not occur.

One example of a not allowed case would be if one master were to write 1 byte while another device wrote 2 bytes to the same device. In this situation, the first master would issue a Stop while the second master sends the MSb of the second data byte.

Chapter 23. GPIO Operations

This chapter describes the General Purpose I/O (GPIO) controller located on the on-chip peripheral bus (OPB) of the PPC405GP. The GPIO controller allows flexible control of up to 23 multiplexed I/Os with user-defined functions.

23.1 GPIO Controller Overview

The GPIO Controller is an OPB macro that controls up to 23 bidirectional module I/O pins with user-programmable functions. Each of these IOs are multiplexed with other signals to reduce the quantity of module I/O on the PPC405GP package. Control for this multiplexing is performed using the CPC0_CR0 register. See "External Module Signals" on page 23-3 for more information.

The GPIO outputs can be programmed to emulate an open drain driver.

All module I/O inputs are synchronized to the OPBClk before being stored in the Input register.

All registers, except the Input Register, are both read and write accessible. The Input register is read-only. Registers provide direct control of all GPIO Controller functions.

All register bits and core input and output signals maintain a bit-for-bit correspondence. For example, GPIO_Out[20] is controlled by GPIO0_OR Register Bit 20.

23.2 Features

The GPIO Controller has the following features:

- Direct control of all functions from registers programmed via memory-mapped addresses
- Control of 23 bidirectional GPIO module pins
  - Each GPIO output has programmable three-state control
  - Each GPIO output is separately programmable to emulate an open drain driver
  - Each GPIO input is observable from a register bit

23.3 GPIO Interface Signals

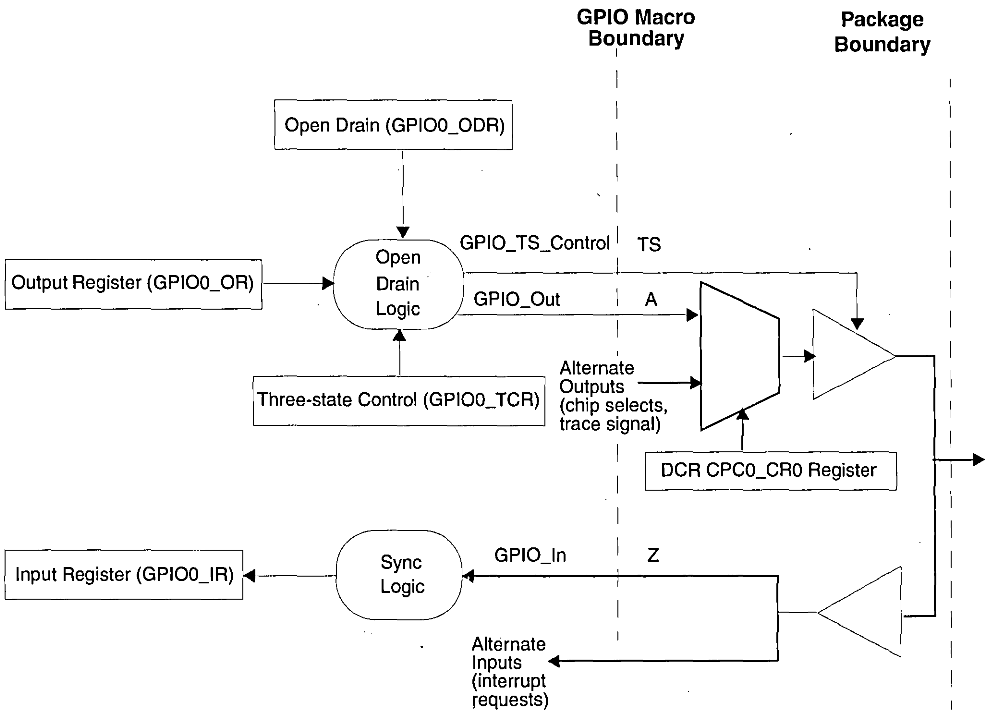

GPIO Macro Boundary
Package Boundary
Open Drain (GPIO0_ODR)
Output Register (GPIO0_OR)
Open Drain Logic
GPIO_TS_Control
TS
GPIO_Out
A
Three-state Control (GPIO0_TCR)
Alternate Outputs (chip selects, trace signal)
DCR CPC0_CR0 Register
Input Register (GPIO0_IR)
Sync Logic
GPIO_In
Z
Alternate Inputs (interrupt requests)

Figure 23-1. GPIO Functional Block Diagram

The following sections discuss the GPIO interface signals.

23.3.1 External Macro Signals

All registers within the GPIO Controller are synchronous to OPBClk, and all GPIO_In inputs are synchronized to OPBClk before being stored.

Table 23-1. Macro I/O Interface Signals

Signal Name	I/O	Function
GPIO_In(0:23)	I	Input to the GPIO Controller directly from Pin Z (Receiver Output). The bi-directional three-state driver books must be in the high impedance state in order to receive valid data as input from off chip.
GPIO_Out(0:23)	O	Data outputs from the GPIO Controller. These signals are designed to tie directly to Pin A (Driver Data Input).
GPIO_TS_Control(0:23)	O	Three-state control outputs from the GPIO Controller macro. These signals are designed to tie directly to each corresponding module I/O Pin TS (Driver Three-State Control). 0 = High Impedance. 1 = Active (drive GPIO_Out signal)

23.3.2 External Module Signals

The GPIO signals do not have dedicated module pins. The module pins, used by the GPIO signals, are shared (multiplexed) with other signals. The Chip Control Register 0 (CPC0_CR0) controls the way in which the signal on a shared pin is interpreted. The following sections provide details of the CPC0_CR0 register bit settings and the function of the GPIO registers.

The GPIO signals are multiplexed with the nine instruction trace signals, seven of the eight EBC chip selects, and the seven external interrupt request inputs. Multiplexing is controlled by the setting of CPC0_CR0[4:18] as follows:

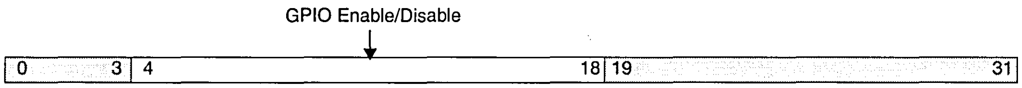


Figure 23-2. CPC0_CR0 Bits Controlling GPIO

4	TRE	CPU Trace Enable 0 GPIO1-9 are enabled 1 GPIO1-9 are disabled	Trace interface cannot be used when GPIO is enabled.
5	G10E	GPIO 10 Enable 0 Enable $\overline{\text{PerCS1}}$ as a chip select 1 Enable $\overline{\text{PerCS1}}$ as GPIO10	
6	G11E	GPIO 11 Enable 0 Enable $\overline{\text{PerCS2}}$ as a chip select 1 Enable $\overline{\text{PerCS2}}$ as GPIO11	
7	G12E	GPIO 12 Enable 0 Enable $\overline{\text{PerCS3}}$ as a chip select 1 Enable $\overline{\text{PerCS3}}$ as GPIO12	
8	G13E	GPIO 13 Enable 0 Enable $\overline{\text{PerCS4}}$ as a chip select 1 Enable $\overline{\text{PerCS4}}$ as GPIO13	
9	G14E	GPIO 14 Enable 0 Enable $\overline{\text{PerCS5}}$ as a chip select 1 Enable $\overline{\text{PerCS5}}$ as GPIO14	
10	G15E	GPIO 15 Enable 0 Enable $\overline{\text{PerCS6}}$ as a chip select 1 Enable $\overline{\text{PerCS6}}$ as GPIO15	
11	G16E	GPIO 16 Enable 0 Enable $\overline{\text{PerCS7}}$ as a chip select 1 Enable $\overline{\text{PerCS7}}$ as GPIO16	

Figure 23-2. CPC0_CR0 Bits Controlling GPIO (continued)

12	G17E	GPIO 17 Enable 0 Enable interrupt IRQ0 as an interrupt 1 Enable interrupt IRQ0 as GPIO17	The purpose of GPIO_17_EN through GPIO_23_EN is to isolate the interrupt controller from activity on a shared pin when that pin is being used as a GPIO. For instance, when G17E is set to a 1, IRQ0 at the UIC will always be forced to a zero. **Note:** Setting G17E to a 0 will not prevent GPIO channel 17 (if configured as an output) from creating contention with the off-chip source of the IRQ input. Therefore, be sure to leave the shared GPIO channel disabled when using the pin as an interrupt input.
13	G18E	GPIO 18 Enable 0 Enable interrupt IRQ1 as an interrupt 1 Enable interrupt IRQ1 as GPIO18	
14	G19E	GPIO 19 Enable 0 Enable interrupt IRQ2 as an interrupt 1 Enable interrupt IRQ2 as GPIO19	
15	G20E	GPIO 20 Enable 0 Enable interrupt IRQ3 as an interrupt 1 Enable interrupt IRQ3 as GPIO20	
16	G21E	GPIO 21 Enable 0 Enable interrupt IRQ4 as an interrupt 1 Enable interrupt IRQ4 as GPIO21	
17	G22E	GPIO 22 Enable 0 Enable interrupt IRQ5 as an interrupt 1 Enable interrupt IRQ5 as GPIO22	
18	G23E	GPIO 23 Enable 0 Enable interrupt IRQ6 as an interrupt 1 Enable interrupt IRQ6 as GPIO23	

In all cases when a pin is used as a GPIO, it is necessary to configure the appropriate bit in this register as well as the GPIO registers located in the GPIO controller.

23.4 Clock and Power Management

The GPIO Controller macro supports Clock and Power Management. Unconditional Sleep (Class 1) power management is implemented, and is enabled by setting the CPC0_ER[GPIO] bit.

23.5 GPIO Register Overview

The following table contains a summary of the GPIO registers.

Table 23-2. GPIO Register Summary

MMIO Address	Mnemonic	Description	Access Mode
EF600700	GPIO0_OR	GPIO Output	R/W
EF600704	GPIO0_TCR	GPIO Three-State Control	R/W
EF600718	GPIO0_ODR	GPIO Open Drain	R/W
EF60071C	GPIO0_IR	GPIO Input	R
Note: All GPIO registers are memory-mapped and accessed via load/store instructions at the address of the register.			

23.5.1 GPIO Register Reset Values

When a system reset occurs, each register in the GPIO macro, except GPIO0_IR, is reset to 0. All outputs are in the high-impedance state. GPIO0_IR is not reset because it is always clocked and always follows the GPIO_In state.

23.5.2 Detailed Register Description

The following sections provide a bit description of the GPIO registers. All registers are accessed from the OPB. The GPIO0_IR register is read-only; all other registers are both read and write accessible.

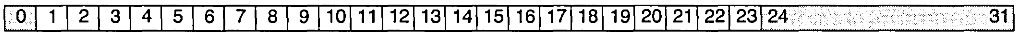


Figure 23-3. GPIO Registers

0		Reserved
1:23		GPIO register bits
24:31		Reserved

23.5.2.1 GPIO Output Register (GPIO0_OR)

The state of each bit in the GPIO0_OR Register may be reflected in the corresponding GPIO Controller macro output signal GPIO_Out.

23.5.2.2 GPIO Three-State Control Register (GPIO0_TCR)

Each bit in the GPIO0_TCR Register controls the corresponding GPIO_TS_Control macro output signal.Each bit in the GPIO0_TCR Register controls the corresponding GPIO_TS_Control macro output signal.

When 0, GPIO_TS_Control forces the module I/O drivers into the high-impedance state.

The state of the GPIO_Out signal driving Pin A (Driver Data Input) of the module I/O three-state driver is irrelevant when the driver is in the high impedance state. High impedance take precedence over data output signals.

23.5.2.3 GPIO Open Drain Register (GPIO0_ODR)

The GPIO Open Drain Register configures the module I/O three-state driver to emulate open drain drivers on a bit-by-bit basis. This is done by controlling the GPIO_Out and GPIO_TS_Control signals, via the GPIO0_OR and GPIO0_TCR registers, respectively.

When programmed to 1, each bit in the GPIO0_ODR register forces the corresponding GPIO_Out signal to 0 and the corresponding GPIO_TS_Control signal to the inverted GPIO_Out signal. In this case, the value of the corresponding bit in the GPIO0_TCR register is ignored. The open drain logic is shown in the table below.

When emulating an open drain driver, the module I/O driver never drives a 1 level. It either drives a 0 level or it is in the high impedance state emulating an open drain 1 level.

Table 23-3. GPIO0_ODR Control Logic

GPIO0_ODR Bit	GPIO0_OR Bit	GPIO_Out Macro Output	GPIO0_TCR Bit	GPIO_TS_Control Macro Output	Module I/O Three-State Driver
0	x	x	0	0	Forced to high-impedance state
0	0	0	1	1	Driving 0
0	1	1	1	1	Driving 1
1	0	0	x	1	Driving 0
1	1	0	x	0	Forced to high-impedance state

23.5.2.4 GPIO Input Register (GPIO0_IR)

The state of each bit in the GPIO0_IR Register reflects the corresponding GPIO Controller macro input signal GPIO_In. All GPIO_In signals are synchronized to OPBClk before being stored in the GPIO0_IR Register. The GPIO0_IR Register is read-only and does not change during a read access.

All bi-directional three-state drivers must be in the high-impedance state in order to receive valid data as input from off chip.

Index

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY
12533-6531

The IBM home page can be found at www.ibm.com

The IBM Microelectronics Division home page can be found at www.chips.ibm.com

Address technical queries about this product to ppcsupp@us.ibm.com

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY
12533-6531

The IBM home page can be found at
www.ibm.com

The IBM Microelectronics Division home page can be found at
www.chips.ibm.com

GK10-3118-03