PowerPC 405 Processor Block Reference Guide

Embedded Development Kit

EDK (v3.2.1) April 1, 2003

XILINX®

PowerPC 405 Processor Block Reference Guide
EDK (v3.2.1) April 1, 2003

The following table shows the revision history for this document..

Date	Version	Revision
04/01/03	3.2.1	EDK 3.2.1 Version Release.
02/20/03	3.2	EDK 3.2 Version Release.
09/16/02	1.0	Initial Embedded Develpment Kit (EDK) release.

Table of Contents

Chapter 3: PowerPC® 405 OCM Controller

About This Guide

This guide serves as a technical reference describing the hardware interface to the PowerPC™ PPC405x3 processor block. It contains information on input/output signals, timing relationships between signals, and the mechanisms software can use to control the interface operation. The document is intended for use by FPGA and system hardware designers and by system programmers who need to understand how certain operations affect hardware external to the processor.

Document Organization

- Chapter 1, **Introduction to the PowerPC® 405 Processor**, provides an overview of the PowerPC embedded-environment architecture and the features supported by the PPC405x3.
- Chapter 2, **Input/Output Interfaces**, describes the interface signals into and out of the PPC405x3 processor block. Where appropriate, timing diagrams are provided to assist in understanding the functional relationship between multiple signals.
- Chapter 3, **PowerPC® 405 OCM Controller**, describes the features, interface signals, timing specifications, and programming model for the PPC405x3 on-chip memory (OCM) controller. The OCM controller serves as a dedicated interface between the block RAMs in the FPGA and OCM signals available on the embedded PPC405x3 core.
- Appendix A, **RISCWatch and RISCTrace Interfaces**, describes the interface requirements between the PPC405x3 processor block and the RISCWatch and RISCTrace tools.
- Appendix B, **Signal Summary**, lists all PPC405x3 interface signals in alphabetical order.
- Appendix C, **Processor Block Timing Model**, explains all of the timing parameters associated with the IBM PPC405 Processor Block.

Document Conventions

General Conventions

Table 2-1 lists the general notational conventions used throughout this document.

Table 2-1: **General Notational Conventions**

Convention	Definition
mnemonic	Instruction mnemonics are shown in lower-case bold.
variable	Variable items are shown in italic.
$\overline{\text{ActiveLow}}$	An overbar indicates an active-low signal.
n	A decimal number
0x*n*	A hexadecimal number
0b*n*	A binary number
OBJECT_{b}	A single bit in any object (a register, an instruction, an address, or a field) is shown as a subscripted number or name
$\text{OBJECT}_{b:b}$	A range of bits in any object (a register, an instruction, an address, or a field)
$\text{OBJECT}_{b,b,\ldots}$	A list of bits in any object (a register, an instruction, an address, or a field)
REGISTER[FIELD]	Fields within any register are shown in square brackets
REGISTER[FIELD, FIELD . . .]	A list of fields in any register
REGISTER[FIELD:FIELD]	A range of fields in any register

Registers

Table 2-2 lists the PPC405x3 registers used in this document and their descriptive names.

Table 2-2: **PPC405x3 Registers**

Register	Descriptive Name
CCR0	Core-configuration register 0
DBCR*n*	Debug-control register *n*
DBSR	Debug-status register
ESR	Exception-syndrome register
MSR	Machine-state register
PIT	Programmable-interval timer
TBL	Time-base lower
TBU	Time-base upper
TCR	Timer-control register
TSR	Timer-status register

Terms

active	As applied to signals, this term indicates a signal is in a state that causes an action to occur in the receiving device, or indicates an action occurred in the sending device. An *active-high* signal drives a logic 1 when active. An *active-low* signal drives a logic 0 when active.
assert	As applied to signals, this term indicates a signal is driven to its active state.
atomic access	A memory access that attempts to read from and write to the same address uninterrupted by other accesses to that address. The term refers to the fact that such transactions are indivisible.
big endian	A memory byte ordering where the address of an item corresponds to the most-significant byte.
Book-E	An version of the PowerPC architecture designed specifically for embedded applications.
cache block	Synonym for *cache line*.
cache line	A portion of a cache array that contains a copy of contiguous system-memory addresses. Cache lines are 32-bytes long and aligned on a 32-byte address.
cache set	Synonym for *congruence class*.
clear	To write a bit value of 0.
clock	Unless otherwise specified, this term refers to the PPC405x3 processor clock.
congruence class	A collection of cache lines with the same index.
cycle	The time between two successive rising edges of the associated clock.
dead cycle	A cycle in which no useful activity occurs on the associated interface.
deassert	As applied to signals, this term indicates a signal is driven to its inactive state.
dirty	An indication that cache information is more recent than the copy in memory.
doubleword	Eight bytes, or 64 bits.
effective address	The untranslated memory address as seen by a program.
exception	An abnormal event or condition that requires the processor's attention. They can be caused by instruction execution or an external device. The processor records the occurrence of an exception and they often cause an *interrupt* to occur.

fill buffer	A buffer that receives and sends data and instructions between the processor and PLB. It is used when cache misses occur and when access to non-cacheable memory occurs.
flush	A cache operation that involves writing back a modified entry to memory, followed by an invalidation of the entry.
GB	Gigabyte, or one-billion bytes.
halfword	Two bytes, or 16 bits.
hit	An indication that requested information exists in the accessed cache array, the associated fill buffer, or on the corresponding OCM interface.
inactive	As applied to signals, this term indicates a signal is in a state that does not cause an action to occur, nor does it indicate an action occurred. An *active-high* signal drives a logic 0 when inactive. An *active-low* signal drives a logic 1 when inactive.
interrupt	The process of stopping the currently executing program so that an exception can be handled.
invalidate	A cache or TLB operation that causes an entry to be marked as invalid. An invalid entry can be subsequently replaced.
KB	Kilobyte, or one-thousand bytes.
line buffer	A buffer located in the cache array that can temporarily hold the contents of an entire cache line. It is loaded with the contents of a cache line when a cache hit occurs.
line fill	A transfer of the contents of the instruction or data line buffer into the appropriate cache.
line transfer	A transfer of an aligned, sequentially addressed 4-word or 8-word quantity (instructions or data) across the PLB interface. The transfer can be from the PLB slave (read) or to the PLB slave (write).
little endian	A memory byte ordering where the address of an item corresponds to the least-significant byte.
logical address	Synonym for *effective address*.
MB	Megabyte, or one-million bytes.
memory	Collectively, cache memory and system memory.
miss	An indication that requested information does not exist in the accessed cache array, the associated fill buffer, or on the corresponding OCM interface.
OEA	The PowerPC operating-environment architecture, which defines the memory-management model, supervisor-level registers and instructions, synchronization requirements, the exception model, and the time-base resources as seen by supervisor programs.

on chip	In system-on-chip implementations, this indicates on the same FPGA chip as the processor core, but external to the processor core.
pending	As applied to interrupts, this indicates that an exception occurred, but the interrupt is disabled. The interrupt occurs when it is later enabled.
physical address	The address used to access physically-implemented memory. This address can be translated from the effective address. When address translation is not used, this address is equal to the effective address.
PLB	Processor local bus.
privileged mode	The operating mode typically used by system software. Privileged operations are allowed and software can access all registers and memory.
problem state	Synonym for *user mode*.
process	A program (or portion of a program) and any data required for the program to run.
real address	Synonym for *physical address*.
scalar	Individual data objects and instructions. Scalars are of arbitrary size.
set	To write a bit value of 1.
sleep	A state in which the PPC405x3 processor clock is prevented from toggling. The execution state of the PPC405x3 does not change when in the sleep state.
sticky	A bit that can be set by software, but cleared only by the processor. Alternatively, a bit that can be cleared by software, but set only by the processor.
string	A sequence of consecutive bytes.
supervisor state	Synonym for *privileged mode*.
system memory	Physical memory installed in a computer system external to the processor core, such RAM, ROM, and flash.
tag	As applied to caches, a set of address bits used to uniquely identify a specific cache line within a congruence class. As applied to TLBs, a set of address bits used to uniquely identify a specific entry within the TLB.
UISA	The PowerPC user instruction-set architecture, which defines the base user-level instruction set, registers, data types, the memory model, the programming model, and the exception model as seen by user programs.
user mode	The operating mode typically used by application software. Privileged operations are not allowed in user mode, and software can access a restricted set of registers and memory.

VEA	The PowerPC virtual-environment architecture, which defines a multi-access memory model, the cache model, cache-control instructions, and the time-base resources as seen by user programs.
virtual address	An intermediate address used to translate an effective address into a physical address. It consists of a process ID and the effective address. It is only used when address translation is enabled.
wake up	The transition of the PPC405x3 out of the sleep state. The PPC405x3 processor clock begins toggling and the execution state of the PPC405x3 advances from that of the sleep state.
word	Four bytes, or 32 bits.

Additional Reading

The following documents contain additional information of potential interest to readers of this manual:

- XILINX *PowerPC Processor Reference Guide*
- XILINX *Virtex-II Pro Platform FPGA Handbook*

Chapter 1

Introduction to the PowerPC® 405 Processor

The PPC405x3 is a 32-bit implementation of the *PowerPC™ embedded-environment architecture* that is derived from the PowerPC architecture. Specifically, the PPC405x3 is an embedded PowerPC 405D5 processor core (PPC405D5). The term *processor block* is used throughout this document to refer to the combination of PPC405D5 core, on-chip memory logic (OCM), and the gasket logic and interface.

The PowerPC architecture provides a software model that ensures compatibility between implementations of the PowerPC family of microprocessors. The PowerPC architecture defines parameters that guarantee compatible processor implementations at the application-program level, allowing broad flexibility in the development of derivative PowerPC implementations that meet specific market requirements.

This chapter provides an overview of the PowerPC architecture and an introduction to the features of the PPC405x3 core.

PowerPC Architecture

The PowerPC architecture is a 64-bit architecture with a 32-bit subset. The various features of the PowerPC architecture are defined at three levels. This layering provides flexibility by allowing degrees of software compatibility across a wide range of implementations. For example, an implementation such as an embedded controller can support the user instruction set, but not the memory management, exception, and cache models where it might be impractical to do so.

The three levels of the PowerPC architecture are defined in Table 1-1.

Table 1-1: **Three Levels of PowerPC Architecture**

User Instruction-Set Architecture (UISA)	Virtual Environment Architecture (VEA)	Operating Environment Architecture (OEA)
• Defines the architecture level to which user-level (sometimes referred to as problem state) software should conform • Defines the base user-level instruction set, user-level registers, data types, floating-point memory conventions, exception model as seen by user programs, memory model, and the programming model **Note:** All PowerPC implementations adhere to the UISA.	• Defines additional user-level functionality that falls outside typical user-level software requirements • Describes the memory model for an environment in which multiple devices can access memory • Defines aspects of the cache model and cache-control instructions • Defines the time-base resources from a user-level perspective **Note:** Implementations that conform to the VEA level are guaranteed to conform to the UISA level.	• Defines supervisor-level resources typically required by an operating system • Defines the memory-management model, supervisor-level registers, synchronization requirements, and the exception model • Defines the time-base resources from a supervisor-level perspective **Note:** Implementations that conform to the OEA level are guaranteed to conform to the UISA and VEA levels.

The PowerPC architecture requires that all PowerPC implementations adhere to the UISA, offering compatibility among all PowerPC application programs. However, different versions of the VEA and OEA are permitted.

Embedded applications written for the PPC405x3 are compatible with other PowerPC implementations. Privileged software generally is not compatible. The migration of privileged software from the PowerPC architecture to the PPC405x3 is in many cases straightforward because of the simplifications made by the PowerPC embedded-environment architecture. Refer to *PowerPC Processor Reference Guide* for more information on programming the PPC405x3.

PowerPC Embedded-Environment Architecture

The PPC405x3 is an implementation of the PowerPC embedded-environment architecture. This architecture is optimized for embedded controllers and is a forerunner to the PowerPC Book-E architecture. The PowerPC embedded-environment architecture provides an alternative definition for certain features specified by the PowerPC VEA and OEA. Implementations that adhere to the PowerPC embedded-environment architecture also adhere to the PowerPC UISA. PowerPC embedded-environment processors are 32-bit only implementations and thus do not include the special 64-bit extensions to the PowerPC UISA. Also, floating-point support can be provided either in hardware or software by PowerPC embedded-environment processors.

The following are features of the PowerPC embedded-environment architecture:

- Memory management optimized for embedded software environments.
- Cache-management instructions for optimizing performance and memory control in complex applications that are graphically and numerically intensive.
- Storage attributes for controlling memory-system behavior.
- Special-purpose registers for controlling the use of debug resources, timer resources,

interrupts, real-mode storage attributes, memory-management facilities, and other architected processor resources.

- A device-control-register address space for managing on-chip peripherals such as memory controllers.
- A dual-level interrupt structure and interrupt-control instructions.
- Multiple timer resources.
- Debug resources that enable hardware-debug and software-debug functions such as instruction breakpoints, data breakpoints, and program single-stepping.

Virtual Environment

The virtual environment defines architectural features that enable application programs to create or modify code, to manage storage coherency, and to optimize memory-access performance. It defines the cache and memory models, the timekeeping resources from a user perspective, and resources that are accessible in user mode but are primarily used by system-library routines. The following summarizes the virtual-environment features of the PowerPC embedded-environment architecture:

- Storage model:
  - Storage-control instructions as defined in the PowerPC virtual-environment architecture. These instructions are used to manage instruction caches and data caches, and for synchronizing and ordering instruction execution.
  - Storage attributes for controlling memory-system behavior. These are: write-through, cacheability, memory coherence (optional), guarded, and endian.
  - Operand-placement requirements and their effect on performance.
- The time-base function as defined by the PowerPC virtual-environment architecture, for user-mode read access to the 64-bit time base.

Operating Environment

The operating environment describes features of the architecture that enable operating systems to allocate and manage storage, to handle errors encountered by application programs, to support I/O devices, and to provide operating-system services. It specifies the resources and mechanisms that require privileged access, including the memory-protection and address-translation mechanisms, the exception-handling model, and privileged timer resources. Table 1-2 summarizes the operating-environment features of the PowerPC embedded-environment architecture.

Table 1-2: **OEA Features of the PowerPC Embedded-Environment Architecture**

Operating Environment	Features
Register model	• Privileged special-purpose registers (SPRs) and instructions for accessing those registers • Device control registers (DCRs) and instructions for accessing those registers
Storage model	• Privileged cache-management instructions • Storage-attribute controls • Address translation and memory protection • Privileged TLB-management instructions
Exception model	• Dual-level interrupt structure supporting various exception types • Specification of interrupt priorities and masking • Privileged SPRs for controlling and handling exceptions • Interrupt-control instructions • Specification of how partially executed instructions are handled when an interrupt occurs
Debug model	• Privileged SPRs for controlling debug modes and debug events • Specification for seven types of debug events • Specification for allowing a debug event to cause a reset • The ability of the debug mechanism to freeze the timer resources
Time-keeping model	• 64-bit time base • 32-bit decrementer (the programmable-interval timer) • Three timer-event interrupts: - Programmable-interval timer (PIT) - Fixed-interval timer (FIT) - Watchdog timer (WDT) • Privileged SPRs for controlling the timer resources • The ability to freeze the timer resources using the debug mechanism
Synchronization requirements	• Requirements for special registers and the TLB • Requirements for instruction fetch and for data access • Specifications for context synchronization and execution synchronization
Reset and initialization requirements	• Specification for two internal mechanisms that can cause a reset: - Debug-control register (DBCR) - Timer-control register (TCR) • Contents of processor resources after a reset • The software-initialization requirements, including an initialization code example

PPC405x3 Software Features

The PPC405x3 processor core is an implementation of the PowerPC embedded-environment architecture. The processor provides fixed-point embedded applications with high performance at low power consumption. It is compatible with the PowerPC UISA.

Much of the PPC405x3 VEA and OEA support is also available in implementations of the PowerPC Book-E architecture. Key software features of the PPC405x3 include:

- A fixed-point execution unit fully compliant with the PowerPC UISA:
  - 32-bit architecture, containing thirty-two 32-bit general purpose registers (GPRs).
- PowerPC embedded-environment architecture extensions providing additional support for embedded-systems applications:
  - True little-endian operation
  - Flexible memory management
  - Multiply-accumulate instructions for computationally intensive applications
  - Enhanced debug capabilities
  - 64-bit time base
  - 3 timers: programmable interval timer (PIT), fixed interval timer (FIT), and watchdog timer (all are synchronous with the time base)
- Performance-enhancing features, including:
  - Static branch prediction
  - Five-stage pipeline with single-cycle execution of most instructions, including loads and stores
  - Multiply-accumulate instructions
  - Hardware multiply / divide for faster integer arithmetic (4-cycle multiply, 35-cycle divide)
  - Enhanced string and multiple-word handling
  - Support for unaligned loads and unaligned stores to cache arrays, main memory, and on-chip memory (OCM)
  - Minimized interrupt latency
- Integrated instruction-cache:
  - 16 KB, 2-way set associative
  - Eight words (32 bytes) per cache line
  - Fetch line buffer
  - Instruction-fetch hits are supplied from the fetch line buffer
  - Programmable prefetch of next-sequential line into the fetch line buffer
  - Programmable prefetch of non-cacheable instructions: full line (eight words) or half line (four words)
  - Non-blocking during fetch line fills
- Integrated data-cache:
  - 16 KB, 2-way set associative
  - Eight words (32 bytes) per cache line
  - Read and write line buffers
  - Load and store hits are supplied from / to the line buffers
  - Write-back and write-through support
  - Programmable load and store cache line allocation
  - Operand forwarding during cache line fills
  - Non-blocking during cache line fills and flushes
- Support for on-chip memory (OCM) that can provide memory-access performance identical to a cache hit

- Flexible memory management:
  - Translation of the 4 GB logical-address space into the physical-address space
  - Independent control over instruction translation and protection, and data translation and protection
  - Page-level access control using the translation mechanism
  - Software control over the page-replacement strategy
  - Write-through, cacheability, user-defined 0, guarded, and endian (WIU0GE) storage-attribute control for each virtual-memory region
  - WIU0GE storage-attribute control for thirty-two 128 MB regions in real mode
  - Additional protection control using zones
- Enhanced debug support with logical operators:
  - Four instruction-address compares
  - Two data-address compares
  - Two data-value compares
  - JTAG instruction for writing into the instruction cache
  - Forward and backward instruction tracing
- Advanced power management support

The following sections describe the software resources available in the PPC405x3. Refer to the *PowerPC Processor Reference Guide* for more information on using these resources.

Privilege Modes

Software running on the PPC405x3 can do so in one of two privilege modes: privileged and user.

Privileged Mode

Privileged mode allows programs to access all registers and execute all instructions supported by the processor. Normally, the operating system and low-level device drivers operate in this mode.

User Mode

User mode restricts access to some registers and instructions. Normally, application programs operate in this mode.

Address Translation Modes

The PPC405x3 also supports two modes of address translation: real and virtual.

Real Mode

In *real mode*, programs address physical memory directly.

Virtual Mode

In *virtual mode*, programs address virtual memory and virtual-memory addresses are translated by the processor into physical-memory addresses. This allows programs to access much larger address spaces than might be implemented in the system.

Addressing Modes

Whether the PPC405x3 is running in real mode or virtual mode, data addressing is supported by the load and store instructions using one of the following addressing modes:

- Register-indirect with immediate index—A base address is stored in a register, and a displacement from the base address is specified as an immediate value in the instruction.
- Register-indirect with index—A base address is stored in a register, and a displacement from the base address is stored in a second register.
- Register indirect—The data address is stored in a register.

Instructions that use the two indexed forms of addressing also allow for automatic updates to the base-address register. With these instruction forms, the new data address is calculated, used in the load or store data access, and stored in the base-address register.

With sequential instruction execution, the next-instruction address is calculated by adding four bytes to the current-instruction address. In the case of branch instructions, the next-instruction address is determined using one of four branch-addressing modes:

- Branch to relative—The next-instruction address is at a location relative to the current-instruction address.
- Branch to absolute—The next-instruction address is at an absolute location in memory.
- Branch to link register—The next-instruction address is stored in the link register.
- Branch to count register—The next-instruction address is stored in the count register.

Data Types

PPC405x3 instructions support byte, halfword, and word operands. Multiple-word operands are supported by the load / store multiple instructions and byte strings are supported by the load / store string instructions. Integer data are either signed or unsigned, and signed data is represented using two's-complement format.

The address of a multi-byte operand is determined using the lowest memory address occupied by that operand. For example, if the four bytes in a word operand occupy addresses 4, 5, 6, and 7, the word address is 4. The PPC405x3 supports both big-endian (an operand's *most significant* byte is at the lowest memory address) and little-endian (an operand's *least significant* byte is at the lowest memory address) addressing.

Register Set Summary

Figure 1-1 shows the registers contained in the PPC405x3. Descriptions of the registers are in the following sections.

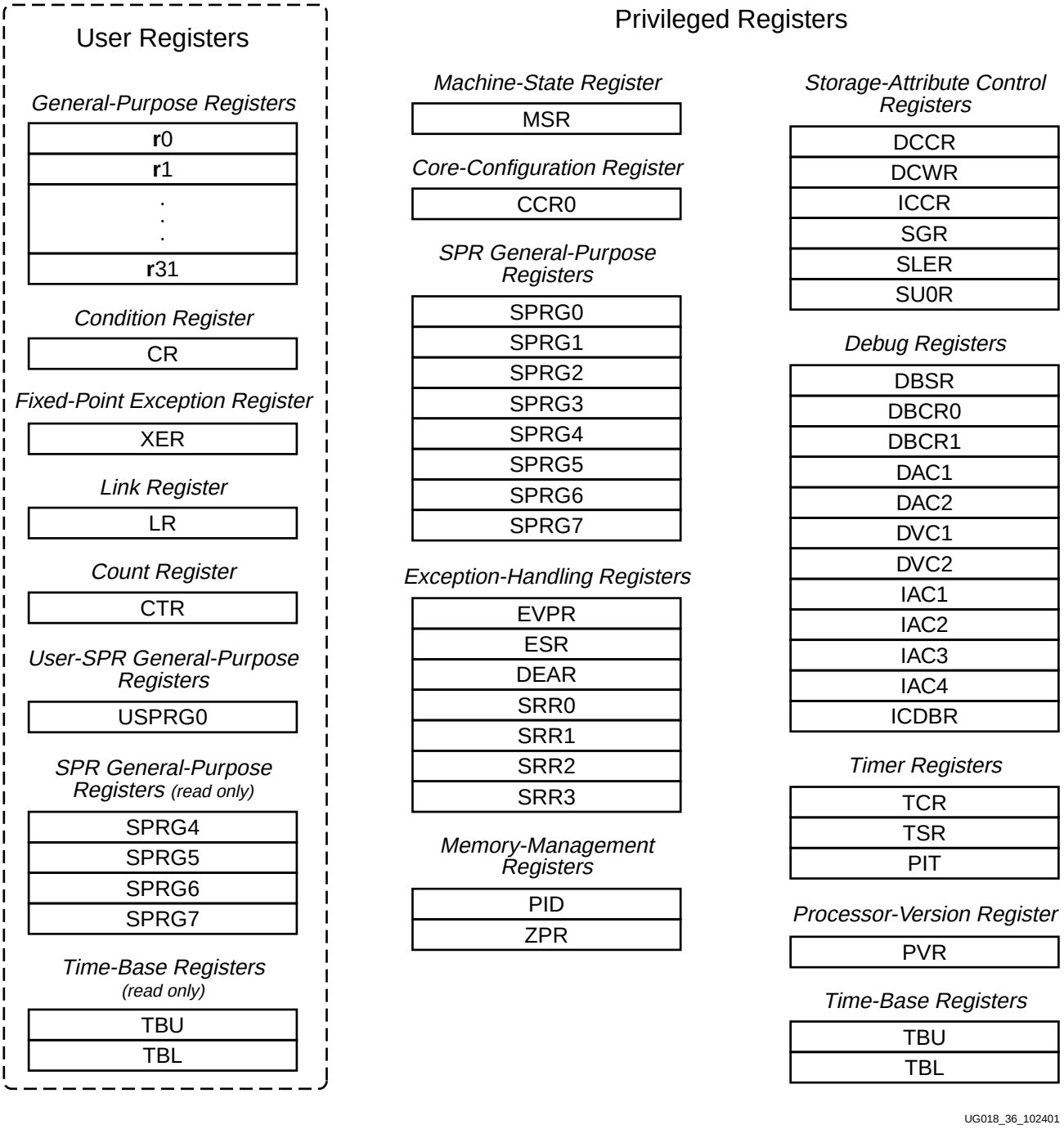

Figure 1-1: PPC405x3 **Registers**

General-Purpose Registers

The processor contains thirty-two 32-bit *general-purpose registers* (GPRs), identified as **r**0 through **r**31. The contents of the GPRs are read from memory using load instructions and written to memory using store instructions. Computational instructions often read

operands from the GPRs and write their results in GPRs. Other instructions move data between the GPRs and other registers. GPRs can be accessed by all software.

Special-Purpose Registers

The processor contains a number of 32-bit *special-purpose registers* (SPRs). SPRs provide access to additional processor resources, such as the count register, the link register, debug resources, timers, interrupt registers, and others. Most SPRs are accessed only by privileged software, but a few, such as the count register and link register, are accessed by all software.

Machine-State Register

The 32-bit *machine-state register* (MSR) contains fields that control the operating state of the processor. This register can be accessed only by privileged software.

Condition Register

The 32-bit *condition register* (CR) contains eight 4-bit fields, CR0–CR7. The values in the CR fields can be used to control conditional branching. Arithmetic instructions can set CR0 and compare instructions can set any CR field. Additional instructions are provided to perform logical operations and tests on CR fields and bits within the fields. The CR can be accessed by all software.

Device Control Registers

The 32-bit *device control registers* (not shown) are used to configure, control, and report status for various external devices that are not part of the PPC405x3 processor. The OCM controllers are examples of devices that contain DCRs. Although the DCRs are not part of the PPC405x3 implementation, they are accessed using the **mtdcr** and **mfdcr** instructions. The DCRs can be accessed only by privileged software.

PPC405x3 Hardware Organization

As shown in Figure 1-2, the PPC405x3 processor contains the following elements:

- A 5-stage pipeline consisting of fetch, decode, execute, write-back, and load write-back stages
- A virtual-memory-management unit that supports multiple page sizes and a variety of storage-protection attributes and access-control options
- Separate instruction-cache and data-cache units
- Debug support, including a JTAG interface
- Three programmable timers

The following sections provide an overview of each element. Refer to the *PowerPC Processor Reference Guide* for more information on how software interacts with these elements.

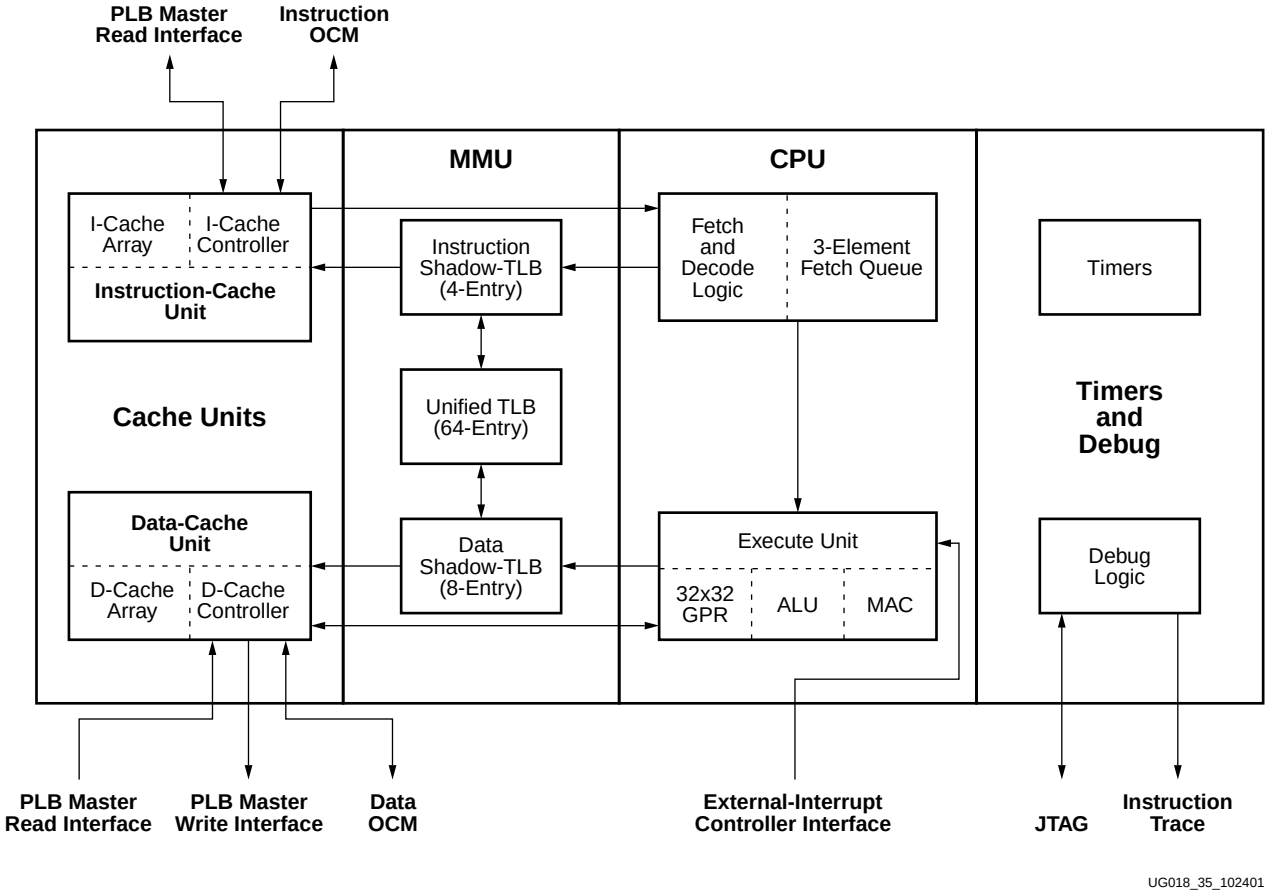

Figure 1-2: PPC405x3 **Organization**

Central-Processing Unit

The PPC405x3 central-processing unit (CPU) implements a 5-stage instruction pipeline consisting of fetch, decode, execute, write-back, and load write-back stages.

The fetch and decode logic sends a steady flow of instructions to the execute unit. All instructions are decoded before they are forwarded to the execute unit. Instructions are queued in the fetch queue if execution stalls. The fetch queue consists of three elements: two prefetch buffers and a decode buffer. If the prefetch buffers are empty instructions flow directly to the decode buffer.

Up to two branches are processed simultaneously by the fetch and decode logic. If a branch cannot be resolved prior to execution, the fetch and decode logic predicts how that branch is resolved, causing the processor to speculatively fetch instructions from the predicted path. Branches with negative-address displacements are predicted as taken, as are branches that do not test the condition register or count register. The default prediction can be overridden by software at assembly or compile time.

The PPC405x3 has a single-issue execute unit containing the general-purpose register file (GPR), arithmetic-logic unit (ALU), and the multiply-accumulate unit (MAC). The GPRs consist of thirty-two 32-bit registers that are accessed by the execute unit using three read ports and two write ports. During the decode stage, data is read out of the GPRs for use by the execute unit. During the write-back stage, results are written to the GPR. The use of five

read / write ports on the GPRs allows the processor to execute load / store operations in parallel with ALU and MAC operations.

The execute unit supports all 32-bit PowerPC UISA integer instructions in hardware, and is compliant with the PowerPC embedded-environment architecture specification. Floating-point operations are not supported.

The MAC unit supports implementation-specific multiply-accumulate instructions and multiply-halfword instructions. MAC instructions operate on either signed or unsigned 16-bit operands, and they store their results in a 32-bit GPR. These instructions can produce results using either modulo arithmetic or saturating arithmetic. All MAC instructions have a single cycle throughput.

Exception Handling Logic

Exceptions are divided into two classes: critical and noncritical. The PPC405x3 CPU services exceptions caused by error conditions, the internal timers, debug events, and the external interrupt controller (EIC) interface. Across the two classes, a total of 19 possible exceptions are supported, including the two provided by the EIC interface.

Each exception class has its own pair of save / restore registers. SRR0 and SRR1 are used for noncritical interrupts, and SRR2 and SRR3 are used for critical interrupts. The exception-return address and the machine state are written to these registers when an exception occurs, and they are automatically restored when an interrupt handler exits using the return-from-interrupt (**rfi**) or return-from critical-interrupt (**rfci**) instruction. Use of separate save / restore registers allows the PPC405x3 to handle critical interrupts independently of noncritical interrupts.

Memory Management Unit

The PPC405x3 supports 4 GB of flat (non-segmented) address space. The memory-management unit (MMU) provides address translation, protection functions, and storage-attribute control for this address space. The MMU supports demand-paged virtual memory using multiple page sizes of 1 KB, 4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB and 16 MB. Multiple page sizes can improve memory efficiency and minimize the number of TLB misses. When supported by system software, the MMU provides the following functions:

- Translation of the 4 GB logical-address space into a physical-address space.
- Independent enabling of instruction translation and protection from that of data translation and protection.
- Page-level access control using the translation mechanism.
- Software control over the page-replacement strategy.
- Additional protection control using zones.
- Storage attributes for cache policy and speculative memory-access control.

The translation look-aside buffer (TLB) is used to control memory translation and protection. Each one of its 64 entries specifies a page translation. It is fully associative, and can simultaneously hold translations for any combination of page sizes. To prevent TLB contention between data and instruction accesses, a 4-entry instruction and an 8-entry data shadow-TLB are maintained by the processor transparently to software.

Software manages the initialization and replacement of TLB entries. The PPC405x3 includes instructions for managing TLB entries by software running in privileged mode. This capability gives significant control to system software over the implementation of a page replacement strategy. For example, software can reduce the potential for TLB

thrashing or delays associated with TLB-entry replacement by reserving a subset of TLB entries for globally accessible pages or critical pages.

Storage attributes are provided to control access of memory regions. When memory translation is enabled, storage attributes are maintained on a page basis and read from the TLB when a memory access occurs. When memory translation is disabled, storage attributes are maintained in storage-attribute control registers. A zone-protection register (ZPR) is provided to allow system software to override the TLB access controls without requiring the manipulation of individual TLB entries. For example, the ZPR can provide a simple method for denying read access to certain application programs.

Instruction and Data Caches

The PPC405x3 accesses memory through the instruction-cache unit (ICU) and data-cache unit (DCU). Each cache unit includes a PLB-master interface, cache arrays, and a cache controller. Hits into the instruction cache and data cache appear to the CPU as single-cycle memory accesses. Cache misses are handled as requests over the PLB bus to another PLB device, such as an external-memory controller.

The PPC405x3 implements separate instruction-cache and data-cache arrays. Each is 16 KB in size, is two-way set-associative, and operates using 8 word (32 byte) cache lines. The caches are non-blocking, allowing the PPC405x3 to overlap instruction execution with reads over the PLB (when cache misses occur).

The cache controllers replace cache lines according to a least-recently used (LRU) replacement policy. When a cache line fill occurs, the most-recently accessed line in the cache set is retained and the other line is replaced. The cache controller updates the LRU during a cache line fill.

The ICU supplies up to two instructions every cycle to the fetch and decode unit. The ICU can also forward instructions to the fetch and decode unit during a cache line fill, minimizing execution stalls caused by instruction-cache misses. When the ICU is accessed, four instructions are read from the appropriate cache line and placed temporarily in a line buffer. Subsequent ICU accesses check this line buffer for the requested instruction prior to accessing the cache array. This allows the ICU cache array to be accessed as little as once every four instructions, significantly reducing ICU power consumption.

The DCU can independently process load / store operations and cache-control instructions. The DCU can also dynamically reprioritize PLB requests to reduce the length of an execution stall. For example, if the DCU is busy with a low-priority request and a subsequent storage operation requested by the CPU is stalled, the DCU automatically increases the priority of the current (low-priority) request. The current request is thus finished sooner, allowing the DCU to process the stalled request sooner. The DCU can forward data to the execute unit during a cache line fill, further minimizing execution stalls caused by data-cache misses.

Additional features allow programmers to tailor data-cache performance to a specific application. The DCU can function in write-back or write-through mode, as determined by the storage-control attributes. Loads and stores that do not allocate cache lines can also be specified. Inhibiting certain cache line fills can reduce potential pipeline stalls and unwanted external-bus traffic.

Timer Resources

The PPC405x3 contains a 64-bit time base and three timers. The time base is incremented synchronously using the CPU clock or an external clock source. The three timers are

incremented synchronously with the time base. The three timers supported by the PPC405x3 are:

- Programmable Interval Timer
- Fixed Interval Timer
- Watchdog Timer

Programmable Interval Timer

The *programmable interval timer* (PIT) is a 32-bit register that is decremented at the time-base increment frequency. The PIT register is loaded with a delay value. When the PIT count reaches 0, a PIT interrupt occurs. Optionally, the PIT can be programmed to automatically reload the last delay value and begin decrementing again.

Fixed Interval Timer

The *fixed interval timer* (FIT) causes an interrupt when a selected bit in the time-base register changes from 0 to 1. Programmers can select one of four predefined bits in the time-base for triggering a FIT interrupt.

Watchdog Timer

The *watchdog timer* causes a hardware reset when a selected bit in the time-base register changes from 0 to 1. Programmers can select one of four predefined bits in the time-base for triggering a reset, and the type of reset can be defined by the programmer.

Debug

The PPC405x3 debug resources include special debug modes that support the various types of debugging used during hardware and software development. These are:

- *Internal-debug mode* for use by ROM monitors and software debuggers
- *External-debug* mode for use by JTAG debuggers
- *Debug-wait mode*, which allows the servicing of interrupts while the processor appears to be stopped
- *Real-time trace mode*, which supports event triggering for real-time tracing

Debug events are supported that allow developers to manage the debug process. Debug modes and debug events are controlled using debug registers in the processor. The debug registers are accessed either through software running on the processor or through the JTAG port.

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

PPC405x3 Interfaces

The PPC405x3 provides the following set of interfaces that support the attachment of cores and user logic:

- Processor local bus interface
- Device control register interface
- Clock and power management interface
- JTAG port interface
- On-chip interrupt controller interface
- On-chip memory controller interface

Processor Local Bus

The *processor local bus (PLB) interface* provides a 32-bit address and three 64-bit data buses attached to the instruction-cache and data-cache units. Two of the 64-bit buses are attached to the data-cache unit, one supporting read operations and the other supporting write operations. The third 64-bit bus is attached to the instruction-cache unit to support instruction fetching.

Device Control Register

The *device control register (DCR) bus interface* supports the attachment of on-chip registers for device control. Software can access these registers using the **mfdcr** and **mtdcr** instructions.

Clock and Power Management

The *clock and power-management interface* supports several methods of clock distribution and power management.

JTAG Port

The *JTAG port interface* supports the attachment of external debug tools. Using the JTAG test-access port, a debug tool can single-step the processor and examine internal-processor state to facilitate software debugging.

On-Chip Interrupt Controller

The *on-chip interrupt controller interface* is an external interrupt controller that combines asynchronous interrupt inputs from on-chip and off-chip sources and presents them to the core using a pair of interrupt signals (critical and noncritical). Asynchronous interrupt sources can include external signals, the JTAG and debug units, and any other on-chip peripherals.

On-Chip Memory Controller

An *on-chip memory* (OCM) *interface* supports the attachment of additional memory to the instruction and data caches that can be accessed at performance levels matching the cache arrays.

PPC405x3 Performance

The PPC405x3 executes instructions at sustained speeds approaching one cycle per instruction. Table 1-3 lists the typical execution speed (in processor cycles) of the instruction classes supported by the PPC405x3.

Instructions that access memory (loads and stores) consider only the “first order” effects of cache misses. The performance penalty associated with a cache miss involves a number of second-order effects. This includes PLB contention between the instruction and data caches and the time associated with performing cache-line fills and flushes. Unless stated otherwise, the number of cycles described applies to systems having zero-wait-state memory access.

Table 1-3: **PPC405x3 Cycles per Instruction**

Instruction Class	Execution Cycles
Arithmetic	1
Trap	2
Logical	1
Shift and Rotate	1
Multiply (32-bit, 48-bit, 64-bit results, respectively)	1, 2, 4
Multiply Accumulate	1
Divide	35
Load	1
Load Multiple and Load String (cache hit)	1 per data transfer
Store	1
Store Multiple and Store String (cache hit or miss)	1 per data transfer
Move to / from device-control register	3
Move to / from special-purpose register	1
Branch known taken	1 or 2
Branch known not taken	1
Predicted taken branch	1 or 2
Predicted not-taken branch	1
Mispredicted branch	2 or 3

XILINX®

Chapter 2

Input/Output Interfaces

The processor block (PPC405x3, OCM controllers, and gasket logic) provides input/output (I/O) signals that are grouped functionally into the following interfaces:

Each section within this chapter provides the following information:

- An overview summarizing the purpose of the interface.
- An I/O symbol providing a quick view of the signal names and the direction of information flow with respect to the processor block.
- A signal table that summarizes the function of each signal. The I/O column in these tables specifies the direction of information flow with respect to the processor block.
- Detailed descriptions for each signal.

Detailed timing diagrams (where appropriate) that more clearly describe the operation of the interface. The diagrams typically illustrate best-case performance when the core is attached to the FPGA processor local bus (PLB) core, or to custom bus interface unit (BIU) designs.

The instruction-side and data-side OCM controller interfaces are described separately in Chapter 3, **PowerPC® 405 OCM Controller**.

Appendix B, **Signal Summary**, alphabetically lists the signals described in this chapter. The l/O designation and a description summary are included for each signal.

Signal Naming Conventions

The following convention is used for signal names throughout this document:

PREFIX1PREFIX2SIGNAME1[SIGNAME1][NEG][(m:n)]

The components of a signal name are as follows:

- PREFIX1 is an uppercase prefix identifying the source of the signal. This prefix specifies either a unit (for example, CPU) or a type of interface (for example, DCR). If PREFIX1 specifies the processor block, the signal is considered an output signal. Otherwise, it is an input signal.
- PREFIX2 is an uppercase prefix identifying the destination of the signal. This prefix specifies either a unit (for example, CPU) or a type of interface (for example, DCR). If PREFIX2 specifies the processor block, the signal is considered an input signal. Otherwise, it is an output signal.
- SIGNAME1 is an uppercase name identifying the primary function of the signal.
- SIGNAME1 is an uppercase name identifying the primary function of the signal.
- [NEG] is an optional notation that indicates a signal is active low. If this notation is not use, the signal is active high.
- [m:n] is an optional notation that indicates a bussed signal. “m” designates the most-significant bit of the bus and “n” designates the least-significant bit of the bus.

Table 2-1 defines the prefixes used in the signal names. The last column in the table identifies whether the functional unit resides *inside* the processor block or *outside* the processor block.

Table 2-1: **Signal Name Prefix Definitions**

Prefix1 or Prefix2	Definition	Location
CPM	Clock and power management	Outside
C405	Processor block	Inside
DBG	Debug unit	Inside
DCR	Device control register	Outside
DSOCM	Data-side on-chip memory (DSOCM)	Outside(1)
EIC	External interrupt controller	Outside
ISOCM	Instruction-side on-chip memory (ISOCM)	Outside(1)
JTG	JTAG	Inside
PLB	Processor local bus	Inside
RST	Reset	Inside
TIE	TIE (signal tied statically to GND or V_{DD})	Outside
TRC	Trace	Inside
XXX	Unspecified FPGA unit	Outside

Notes:
1. OCM controllers are located in the Processor Block.

Clock and Power Management Interface

The clock and power management (CPM) interface enables power-sensitive applications to control the processor clock using external logic. The OCM controllers are clocked separately from the processor. Two types of processor clock control are possible:

- *Global local enables* control a clock zone within the processor. These signals are used to disable the clock splitters within a zone so that the clock signal is prevented from propagating to the latches within the zone. The PPC405x3 is divided into three clock zones: core, timer, and JTAG. Control over a zone is exercised as follows:
  - The core clock zone contains most of the logic comprising the PPC405x3. It does not contain logic that belongs to the timer or JTAG zones, or other logic within the processor block. The core zone is controlled by the CPMC405CPUCLKEN signal.
  - The timer clock zone contains the PPC405x3 timer logic. It does not contain logic that belongs to the core or JTAG zones, or other logic within the processor block. This zone is separated from the core zone so that timer events can be used to "wake up" the core logic if a power management application has put it to sleep. The timer zone is controlled by the CPMC405TIMERCLKEN signal.
  - The JTAG clock zone contains the PPC405x3 JTAG logic. It does not contain logic that belongs to the core or timer zones, or other logic within the processor block. The JTAG zone is controlled by the CPMC405JTAGCLKEN signal. Although an enable is provided for this zone, the JTAG standard does not allow local gating of the JTAG clock. This enables basic JTAG functions to be maintained when the rest of the chip (including the CPM FPGA macro) is not running.
- *Global gating* controls the toggling of the PPC405x3 clock, CPMC405CLOCK. Instead of using the global-local enables to prevent the clock signal from propagating through a zone, CPM logic can stop the PPC405x3 clock input from toggling. If this method of power management is employed, the clock signal should be held active (logic 1). The CPMC405CLOCK is used by the core and timer zones, but not the JTAG zone.

CPM logic should be designed to wake the PPC405x3 from sleep mode when any of the following occurs:

- A timer interrupt or timer reset is asserted by the PPC405x3.
- A chip-reset or system-reset request is asserted (this request comes from a source other than the PPC405x3).
- An external interrupt or critical interrupt input is asserted and the corresponding interrupt is enabled by the appropriate machine-state register (MSR) bit.
- The DBGC405DEBUGHALT chip-input signal (if provided) is asserted. Assertion of this signal indicates that an external debug tool wants to control the PPC405x3 processor. See page 126 for more information.

CPM Interface I/O Signal Summary

Figure 2-1 shows the block symbol for the CPM interface. The signals are summarized in Table 2-2.

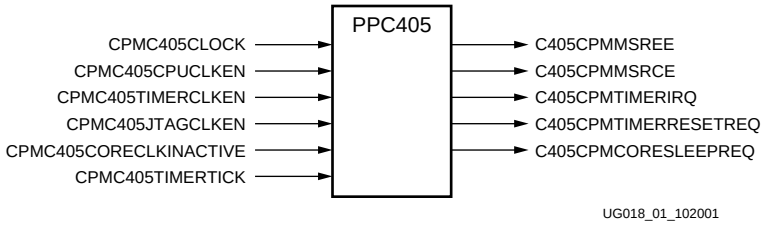

Figure 2-1: **CPM Interface Block Symbol**

Table 2-2: **CPM Interface I/O Signals**

Signal	I/O Type	If Unused	Function
CPMC405CLOCK	I	Required	PPC405x3 clock input (for all non-JTAG logic, including timers).
CPMC405CPUCLKEN	I	1	Enables the core clock zone.
CPMC405TIMERCLKEN	I	1	Enables the timer clock zone.
CPMC405JTAGCLKEN	I	1	Enables the JTAG clock zone.
CPMC405CORECLKINACTIVE	I	0	Indicates the CPM logic disabled the clocks to the core.
CPMC405TIMERTICK	I	1	Increments or decrements the PPC405x3 timers every time it is active with the CPMC405CLOCK.
C405CPMMSREE	O	No Connect	Indicates the value of MSR[EE].
C405CPMMSRCE	O	No Connect	Indicates the value of MSR[CE].
C405CPMTIMERIRQ	O	No Connect	Indicates a timer-interrupt request occurred.
C405CPMTIMERRESETREQ	O	No Connect	Indicates a watchdog-timer reset request occurred.
C405CPMCORESLEEPREQ	O	No Connect	Indicates the core is requesting to be put into sleep mode.

CPM Interface I/O Signal Descriptions

The following sections describe the operation of the CPM interface I/O signals.

CPMC405CLOCK (input)

This signal is the source clock for all PPC405x3 logic (including timers). It is not the source clock for the JTAG logic. External logic can implement a power management mode that stops toggling of this signal. If such a method is employed, the clock signal should be held active (logic 1).

CPMC405CPUCLKEN (input)

Enables the core clock zone when asserted and disables the zone when deasserted. If logic is not implemented to control this signal, it must be held active (tied to 1).

CPMC405TIMERCLKEN (input)

Enables the timer clock zone when asserted and disables the zone when deasserted. If logic is not implemented to control this signal, it must be held active (tied to 1).

CPMC405JTAGCLKEN (input)

Enables the JTAG clock zone when asserted and disables the zone when deasserted. CPM logic should not control this signal. The JTAG standard requires that it be held active (tied to 1).

CPMC405CORECLKINACTIVE (input)

This signal is a status indicator that is latched by an internal PPC405x3 register (JDSR). An external debug tool (such as RISCWatch) can read this register and determine that the PPC405x3 is in sleep mode. This signal should be asserted by the CPM when it places the PPC405x3 in sleep mode using either of the following methods:

- Deasserting CPMC405CPUCLKEN to disable the core clock zone.
- Stopping CPMC405CLOCK from toggling by holding it active (logic 1).

CPMC405TIMERTICK (input)

This signal is used to control the update frequency of the PPC405x3 time base and PIT (the FIT and WDT are timer events triggered by the time base). The time base is incremented and the PIT is decremented every cycle that CPMC405TIMERTICK and CPMC405CLOCK are both active. CPMC405TIMERTICK should be synchronous with CPMC405CLOCK for the timers to operate predictably. The timers are updated at the PPC405x3 clock frequency if CPMC405TIMERTICK is held active.

C405CPMMSREE (output)

This signal indicates the state of the MSR[EE] (external-interrupt enable) bit. When asserted, external interrupts are enabled (MSR[EE]=1). When deasserted, external interrupts are disabled (MSR[EE]=0). The CPM can use this signal to wake the processor from sleep mode when an external noncritical interrupt occurs.

When the processor wakes up, it deasserts the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals one processor clock cycle before it deasserts the C405CPMCORESLEEPREQ signal. For this reason, the CPM should latch the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals before using them to control the processor clocks.

C405CPMMSRCE (output)

This signal indicates the state of the MSR[CE] (critical-interrupt enable) bit. When asserted, critical interrupts are enabled (MSR[CE]=1). When deasserted, critical interrupts are disabled (MSR[CE]=0). The CPM can use this signal to wake the processor from sleep mode when an external critical interrupt occurs.

When the processor wakes up, it deasserts the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals one processor clock cycle before it deasserts the C405CPMCORESLEEPREQ signal. For this reason, the CPM should latch the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals before using them to control the processor clocks.

C405CPMTIMERIRQ (output)

When asserted, this signal indicates a timer exception occurred within the PPC405x3 and an interrupt request is pending to handle the exception. When deasserted, no timer-interrupt request is pending. This signal is the logical OR of interrupt requests from the programmable-interval timer (PIT), the fixed-interval timer (FIT), and the watchdog timer

(WDT). The CPM can use this signal to wake the processor from sleep mode when an internal timer exception occurs.

When the processor wakes up, it deasserts the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals one processor clock cycle before it deasserts the C405CPMCORESLEEPREQ signal. For this reason, the CPM should latch the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals before using them to control the processor clocks.

C405CPMTIMERRESETREQ (output)

When asserted, this signal indicates a watchdog time-out occurred and a reset request is pending. When deasserted, no reset request is pending. This signal is the logical OR of the core, chip, and system reset modes that are programmed using the watchdog timer mechanism. The CPM can use this signal to wake the processor from sleep mode when a watchdog time-out occurs.

C405CPMCORESLEEPREQ (output)

When asserted, this signal indicates the PPC405x3 has requested to be put into sleep mode. When deasserted, no request exists. This signal is asserted after software enables the wait state by setting the MSR[WE] (wait-state enable) bit to 1. The processor completes execution of all prior instructions and memory accesses before asserting this signal. The CPM can use this signal to place the processor in sleep mode at the request of software.

When the processor gets out of sleep mode at a later time, it deasserts the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals one processor clock cycle before it deasserts the C405CPMCORESLEEPREQ signal. For this reason, the CPM should latch the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals before using them to control the processor clocks.

CPU Control Interface

The CPU control interface is used primarily to provide CPU setup information to the PPC405x3. It is also used to report the detection of a machine check condition within the PPC405x3.

CPU Control Interface I/O Signal Summary

Figure 2-2 shows the block symbol for the CPU control interface. The signals are summarized in Table 2-3.

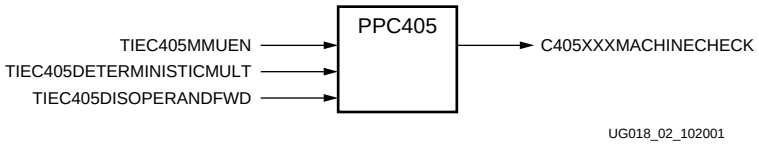

Figure 2-2: **CPU Control Interface Block Symbol**

Table 2-3: **CPU Control Interface I/O Signals**

Signal	I/O Type	If Unused	Function
TIEC405MMUEN	I	Required	Enables the memory-management unit (MMU)
TIEC405DETERMINISTICMULT	I	Required	Specifies whether all multiply operations complete in a fixed number of cycles or have an early-out capability
TIEC405DISOPERANDFWD	I	Required	Disables operand forwarding for load instructions
C405XXXMACHINECHECK	O	No Connect	Indicates a machine-check error has been detected by the PPC405x3

CPU Control Interface I/O Signal Descriptions

The following sections describe the operation of the CPU control-interface I/O signals.

TIEC405MMUEN (input)

When held active (tied to logic 1), this signal enables the PPC405x3 memory-management unit (MMU). When held inactive (tied to logic 0), this signal disables the MMU. The MMU is used for virtual to address translation and for memory protection. Its operation is described in the *PowerPC Processor Reference Guide*. Disabling the MMU can improve the performance (clock frequency) of the PPC405x3.

TIEC405DETERMINISTICMULT (input)

When held active (tied to logic 1), this signal disables the hardware multiplier *early-out* capability. All multiply instructions have a 4-cycle reissue rate and a 5-cycle latency rate. When held inactive (tied to logic 0), this signal enables the hardware multiplier early-out capability. If early out is enabled, multiply instructions are executed in the number of cycles specified in Table 2-4. The performance of multiply instructions is described in the *PowerPC Processor Reference Guide*.

Table 2-4: **Multiply and MAC Instruction Timing**

Operations	Issue-Rate Cycles	Latency Cycles
MAC and Negative MAC	1	2
Halfword × Halfword (32-bit result)	1	2
Halfword × Word (48-bit result)	2	3
Word × Word (64-bit result)	4	5

Notes:

For the purposes of this table, words are treated as halfwords if the upper 16 bits of the operand contain a sign extension of the lower 16 bits. For example, if the upper 16 bits of a word operand are zero, the operand is considered a halfword when calculating execution time.

TIEC405DISOPERANDFWD (input)

When held active (tied to logic 1), this signal disables operand forwarding. When held inactive (tied to logic 0), this signal enables operand forwarding. The processor uses operand forwarding to send load-instruction data from the data cache to the execution units as soon as it is available. Operand forwarding often saves a clock cycle when instructions following the load require the loaded data. Disabling operand forwarding can improve the performance (clock frequency) of the PPC405x3.

C405XXXMACHINECHECK (output)

When asserted, this signal indicates the PPC405x3 detected an instruction machine-check error. When deasserted, no error exists. This signal is asserted when the processor attempts to execute an instruction that was transferred to the PPC405x3 with the PLBC405ICUERR signal asserted. This signal remains asserted until software clears the instruction machine-check bit in the exception-syndrome register (ESR[MCI]).

Reset Interface

A reset causes the processor block to perform a hardware initialization. It always occurs when the processor block is powered-up and can occur at any time during normal operation. If it occurs during normal operation, instruction execution is immediately halted and all processor state is lost.

The processor block recognizes three types of reset:

- A *processor reset* affects the processor block only, including PPC405x3 execution units, cache units, and the on-chip memory controller (OCM). External devices (on-chip and off-chip) are not affected. This type of reset is also referred to as a *core reset*.
- A *chip reset* affects the processor block and all other devices or peripherals located on the same chip as the processor.
- A *system reset* affects the processor chip and all other devices or peripherals external to the processor chip that are connected to the same system-reset network. The scope of a system reset depends on the system implementation. Power-on reset (POR) is a form of system reset.

Input signals are provided to the processor block for each reset type. The signals are used to reset the processor block and to record the reset type in the debug-status register (DBSR[MRR]). The processor block can produce reset-request output signals for each reset type. External reset logic can process these output signals and generate the appropriate reset input signals to the processor block. Reset activity does not occur when the processor block requests the reset. Reset activity only occurs when external logic asserts the appropriate reset input signal.

Reset Requirements

FPGA logic (external to the processor block) is required to generate the reset input signals to the processor block. The reset input signals can be based on the reset-request output signals from the processor block, system-specific reset-request logic, or some combination of the two. Reset input signals must meet the following minimum requirements:

- The reset input signals must be synchronized with the PPC405x3 clock.
- The reset input signals must be asserted for at least eight clock cycles.
- Only the combinations of signals shown in Table 2-5 are used to cause a reset.

POR (power-on reset) is handled by logic within the processor block. This logic asserts the RSTC405RESETCORE, RSTC405RESETCHIP, RSTC405RESETSYS, and JTGC405TRSTNEG signals for at least eight clock cycles. FPGA designers cannot modify the processor block power-on reset mechanism.

The reset logic is not required to support all three types of reset. However, distinguishing resets by type can make it easier to isolate errors during system debug. For example, a system could reset the core to recover from an external error that affects software operation. Following the chip reset, a debugger could be used to locate the external error source which is preserved because neither a chip or system reset occurred.

Table 2-5 shows the valid combinations of reset signals and their effect on the DBSR[MRR] field following reset.

Table 2-5: **Valid Reset Signal Combinations and Effect on DBSR(MRR)**

Reset Input Signal	Reset Type				
	None	Core	Chip	System	Power-On[1]
RSTC405RESETCORE	Deassert	Assert	Assert	Assert	Assert
RSTC405RESETCHIP	Deassert	Deassert	Assert	Assert	Assert
RSTC405RESETSYS	Deassert	Deassert	Deassert	Assert	Assert
JTGC405TRSTNEG	Deassert	Deassert	Deassert	Deassert	Assert
Value of DBSR[MRR] following reset	Previous DBSR[MRR]	0b01	0b10	0b11	0b11

Notes:
1. Handled automatically by logic within the processor block.

Reset Interface I/O Signal Summary

Figure 2-3 shows the block symbol for the reset interface. The signals are summarized in Table 2-6.

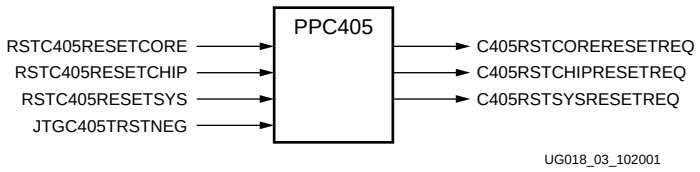

Figure 2-3: **Reset Interface Block Symbol**

Table 2-6: **Reset Interface I/O Signals**

Signal	I/O Type	If Unused	Function
C405RSTCORERESETREQ	O	Required	Indicates a core-reset request occurred.
C405RSTCHIPRESETREQ	O	Required	Indicates a chip-reset request occurred.
C405RSTSYSRESETREQ	O	Required	Indicates a system-reset request occurred.
RSTC405RESETCORE	I	Required	Resets the processor block, including the PPC405x3 core logic, data cache, instruction cache, and the on-chip memory controller (OCM).
RSTC405RESETCHIP	I	Required	Indicates a chip-reset occurred.
RSTC405RESETSYS	I	Required	Indicates a system-reset occurred. Resets the logic in the PPC405x3 JTAG unit.
JTGC405TRSTNEG	I	Required	Performs a JTAG test reset (TRST).

Reset Interface I/O Signal Descriptions

The following sections describe the operation of the reset interface I/O signals.

C405RSTCORERESETREQ (output)

When asserted, this signal indicates the processor block is requesting a core reset. If this signal is asserted, it remains active until two clock cycles after external logic asserts the RSTC405RESETCORE input to the processor block. When deasserted, no core-reset request exists. The processor asserts this signal when one of the following occurs:

- A JTAG debugger sets the reset field in the debug-control register 0 (DBCR0[RST]) to 0b01.
- Software sets the reset field in the debug-control register 0 (DBCR0[RST]) to 0b01.
- The timer-control register watchdog-reset control field (TCR[WRC]) is set to 0b01 and a watchdog time-out causes the watchdog-event state machine to enter the reset state.

C405RSTCHIPRESETREQ (output)

When asserted, this signal indicates the processor block is requesting a chip reset. If this signal is asserted, it remains active until two clock cycles after external logic asserts the RSTC405RESETCHIP input to the processor block. When deasserted, no chip-reset request exists. The processor asserts this signal when one of the following occurs:

- A JTAG debugger sets the reset field in the debug-control register 0 (DBCR0[RST]) to 0b10.
- Software sets the reset field in the debug-control register 0 (DBCR0[RST]) to 0b10.
- The timer-control register watchdog-reset control field (TCR[WRC]) is set to 0b10 and a watchdog time-out causes the watchdog-event state machine to enter the reset state.

C405RSTSYSRESETREQ (output)

When asserted, this signal indicates the processor block is requesting a system reset. If this signal is asserted, it remains active until two clock cycles after external logic asserts the RSTC405RESETSYS input to the processor block. When deasserted, no system-reset request exists. The processor asserts this signal when one of the following occurs:

- A JTAG debugger sets the reset field in the debug-control register 0 (DBCR0[RST]) to 0b11.
- Software sets the reset field in the debug-control register 0 (DBCR0[RST]) to 0b11.
- The timer-control register watchdog-reset control field (TCR[WRC]) is set to 0b11 and a watchdog time-out causes the watchdog-event state machine to enter the reset state.

RSTC405RESETCORE (input)

External logic asserts this signal to reset the processor block (core). This includes the PPC405x3 core logic, data cache, instruction cache, and the on-chip memory controller (OCM). The PPC405x3 also uses this signal to record a core reset type in the DBSR[MRR] field. This signal should be asserted for at least eight clock cycles to guarantee that the processor block initiates its reset sequence. No reset occurs and none is recorded in DBSR[MRR] when this signal is deasserted.

Table 2-5, page 40 shows the valid combinations of the RSTC405RESETCORE, RSTC405RESETCHIP, and RSTC405RESETSYS signals and their effect on the DBSR[MRR] field following reset.

RSTC405RESETCHIP (input)

External logic asserts this signal to reset the chip. A chip reset involves the FPGA logic, on-chip peripherals, and the processor block (the PPC405x3 core logic, data cache, instruction cache, and the OCM). The signal does not reset logic in the processor block. The PPC405x3

uses this signal only to record a chip reset type in the DBSR[MRR] field. The RSTC405RESETCORE signal must be asserted with this signal to cause a core reset. Both signals must be asserted for at least eight clock cycles to guarantee that the processor block recognizes the reset type and initiates the core-reset sequence. The PPC405x3 does not record a chip reset type in DBSR[MRR] when this signal is deasserted.

Table 2-5, page 40 shows the valid combinations of the RSTC405RESETCORE, RSTC405RESETCHIP, and RSTC405RESETSYS signals and their effect on the DBSR[MRR] field following reset.

RSTC405RESETSYS (input)

External logic asserts this signal to reset the system. A system reset involves logic external to the FPGA, the FPGA logic, on-chip peripherals, and the processor block (the PPC405x3 core logic, data cache, instruction cache, and the OCM). This signal resets the logic in the PPC405x3 JTAG unit, but it does not reset any other processor block logic. The PPC405x3 uses this signal to record a system reset type in the DBSR[MRR] field. The RSTC405RESETCORE signal must be asserted with this signal to cause a core reset. The RSTC405RESETCORE, RSTC405RESETCHIP, and RSTC405RESETSYS signals must be asserted for at least eight clock cycles to guarantee that the processor block recognizes the reset type and initiates the core-reset sequence. The PPC405x3 does not record a system reset type in DBSR[MRR] when this signal is deasserted.

This signal must be asserted during a power-on reset to properly initialize the JTAG unit.

Table 2-5, page 40 shows the valid combinations of the RSTC405RESETCORE, RSTC405RESETCHIP, and RSTC405RESETSYS signals and their effect on the DBSR[MRR] field following reset.

JTGC405TRSTNEG (input)

This input is the JTAG test reset ($\overline{\text{TRST}}$) signal. It can be connected to the chip-level $\overline{\text{TRST}}$ signal. Although optional in IEEE Standard 1149.1, this signal is automatically used by the processor block during power-on reset to properly reset all processor block logic, including the JTAG and debug logic. When deasserted, no JTAG test reset exists.

This is a negative active signal.

Instruction-Side Processor Local Bus Interface

The instruction-side processor local bus (ISPLB) interface enables the PPC405x3 instruction cache unit (ICU) to fetch (read) instructions from any memory device connected to the processor local bus (PLB). The ICU cannot write to memory. This interface has a dedicated 30-bit address bus output and a dedicated 64-bit read-data bus input. The interface is designed to attach as a master to a 64-bit PLB, but it also supports attachment as a master to a 32-bit PLB. The interface is capable of one transfer (64 or 32 bits) every PLB cycle.

At the chip level, the ISPLB can be combined with the data-side read-data bus (also a PLB master) to create a shared read-data bus. This is done if a single PLB arbiter services both PLB masters and the PLB arbiter implementation only returns data to one PLB master at a time.

Refer to the *PowerPC Processor Reference Guide* for more information on the operation of the PPC405x3 ICU.

Instruction-Side PLB Operation

Fetch requests are produced by the ICU and communicated over the PLB interface. Fetch requests occur when an access misses the instruction cache or the memory location that is accessed is non-cacheable. A fetch request contains the following information:

- A fetch request is indicated by C405PLBICUREQUEST (page 47).
- The target address of the instruction to be fetched is specified by the address bus, C405PLBICUABUS[0:29] (page 48). Bits 30:31 of the 32-bit instruction-fetch address are always zero and must be tied to zero at the PLB arbiter. The ICU always requests an aligned doubleword of data, so the byte enables are not used.
- The transfer size is specified as four words (quadword) or eight words (cache line) using C405PLBICUSIZE[2:3] (page 48). The remaining bits of the transfer size (0:1) must be tied to zero at the PLB arbiter.
- The cacheability storage attribute is indicated by C405PLBICUCACHEABLE (page 48). Cacheable transfers are always performed with an eight-word transfer size.
- The user-defined storage attribute is indicated by C405PLBICUU0ATTR (page 49).
- The request priority is indicated by C405PLBICUPRIORITY[0:1] (page 49). The PLB arbiter uses this information to prioritize simultaneous requests from multiple PLB masters.

The processor can abort a PLB fetch request using C405PLBICUABORT (page 49). This can occur when a branch instruction is executed or when an interrupt occurs.

Fetched instructions are returned to the ICU by a PLB slave device over the PLB interface. A fetch response contains the following information:

- The fetch-request address is acknowledged by the PLB slave using PLBC405ICUADDRACK (page 50).
- Instructions sent from the PLB slave to the ICU during a line transfer are indicated as valid using PLBC405ICURDDACK (page 51).
- The PLB-slave bus width, or size (32-bit or 64-bit), is specified by PLBC405ICUSSIZE1 (page 50). The PLB slave is responsible for packing data bytes from non-word devices so that the information sent to the ICU is presented appropriately, as determined by the transfer size.
- The instructions returned to the ICU by the PLB slave are sent using four-word or eight-word line transfers, as specified by the transfer size in the fetch request. These

instructions are returned over the ICU read-data bus, PLBC405ICURDDBUS[0:63] (page 51). Line transfers operate as follows:

- A four-word line transfer returns the quadword aligned on the address specified by C405PLBICUABUS[0:27]. This quadword contains the target instruction requested by the ICU. The quadword is returned using two doubleword or four word transfer operations, depending on the PLB slave bus width (64-bit or 32-bit, respectively).
- An eight-word line transfer returns the eight-word cache line aligned on the address specified by C405PLBICUABUS[0:26]. This cache line contains the target instruction requested by the ICU. The cache line is returned using four doubleword or eight word transfer operations, depending on the PLB slave bus width (64-bit or 32-bit, respectively).

• The words returned during a line transfer can be sent from the PLB slave to the ICU in any order (target-word-first, sequential, other). This transfer order is specified by PLBC405ICURDWDADDR[1:3] (page 52).

Interaction with the ICU Fill Buffer

As mentioned above, the PLB slave can transfer instructions to the ICU in any order (target-word-first, sequential, other). When instructions are received by the ICU from the PLB slave, they are placed in the ICU fill buffer. When the ICU receives the target instruction, it forwards it immediately from the fill buffer to the instruction-fetch unit so that pipeline stalls due to instruction-fetch delays are minimized. This operation is referred to as a *bypass*. The remaining instructions are received from the PLB slave and placed in the fill buffer. Subsequent instruction fetches read from the fill buffer if the instruction is already present in the buffer. For the best possible software performance, the PLB slave should be designed to return the target word first.

Non-cacheable instructions are transferred using a four-word or eight-word line-transfer size. Software controls this transfer size using the *non-cacheable request-size* bit in the core-configuration register (CCR0[NCRS]). This enables non-cacheable transfers to take advantage of the PLB line-transfer protocol to minimize PLB-arbitration delays and bus delays associated with multiple, single-word transfers. The transferred instructions are placed in the ICU fill buffer, but not in the instruction cache. Subsequent instruction fetches from the same non-cacheable line are read from the fill buffer instead of requiring a separate arbitration and transfer sequence across the PLB. Instructions in the fill buffer are fetched with the same performance as a cache hit. The non-cacheable line remains in the fill buffer until the fill buffer is needed by another line transfer.

Cacheable instructions are always transferred using an eight-word line-transfer size. The transferred instructions are placed in the ICU fill buffer as they are received from the PLB slave. Subsequent instruction fetches from the same cacheable line are read from the fill buffer during the time the line is transferred from the PLB slave. When the fill buffer is full, its contents are transferred to the instruction cache. Software can prevent this transfer by setting the *fetch without allocate* bit in the core-configuration register (CCR0[FWOA]). In this case, the cacheable line remains in the fill buffer until the fill buffer is needed by another line transfer. An exception is that the contents of the fill buffer are always transferred if the line was fetched because an **icbt** instruction was executed.

Prefetch and Address Pipelining

A *prefetch* is a request for the eight-word cache line that sequentially follows the current eight-word fetch request. Prefetched instructions are fetched before it is known that they are needed by the sequential execution of software.

The ICU can overlap a single prefetch request with the prior fetch request. This process, known as *address pipelining*, enables a second address to be presented to a PLB slave while the slave is returning data associated with the first address. Address pipelining can occur if a prefetch request is produced before all instructions from the previous fetch request are transferred by the slave. This capability maximizes PLB-transfer throughput by reducing dead cycles between instruction transfers associated with the two requests. The ICU can pipeline the prefetch with any combination of sequential, branch, and interrupt fetch requests. A prefetch request is communicated over the PLB two or more cycles after the prior fetch request is acknowledged by the PLB slave.

Address pipelining of prefetch requests never occurs under any one of the following conditions:

- The PLB slave does not support address pipelining.
- The prefetch address falls outside the 1 KB physical page holding the current fetch address. This limitation avoids potential problems due to protection violations or storage-attribute mismatches.
- Non-cacheable transfers are programmed to use a four-word line-transfer size (CCR0[NCRS]=0).
- For non-cacheable transfers, prefetching is disabled (CCR0[PFNC]=0).
- For cacheable transfers, prefetching is disabled (CCR0[PFC]=0).

Address pipelining of non-cacheable prefetch requests can occur if all of the following conditions are met:

- Address pipelining is supported by the PLB slave.
- The ICU is not already involved in an address-pipelined PLB transfer.
- A branch or interrupt does not modify the sequential execution of the current (first) instruction-fetch request.
- Non-cacheable prefetching is enabled (CCR0[PFNC]=1).
- A non-cacheable instruction-prefetch is requested, and the instruction is not in the fill buffer or being returned over the ISOCM interface.
- The prefetch address does not fall outside the current 1 KB physical page.

Address pipelining of cacheable prefetch requests can occur if all of the following conditions are met:

- Address pipelining is supported by the PLB slave.
- The ICU is not already involved in an address-pipelined PLB transfer.
- A branch or interrupt does not modify the sequential execution of the current (first) instruction-fetch request.
- Cacheable prefetching is enabled (CCR0[PFC]=1).
- A cacheable instruction-prefetch is requested, and the instruction is not in the instruction cache, the fill buffer, or being returned over the ISOCM interface.
- The prefetch address does not fall outside the current 1 KB physical page.

Guarded Storage

Accesses to guarded storage are not indicated by the ISPLB interface. This is because the PowerPC Architecture allows instruction prefetching when:

- The processor is in real mode (instruction address translation is disabled).
- The fetched instruction is located in the same physical page (1 KB) as an instruction that is required by the sequential execution model, or

- The fetched instruction is located in the next physical page (1 KB) as an instruction that is required by the sequential execution model.

Memory should be organized such that real-mode instruction prefetching from the same or next 1 KB page does not affect sensitive addresses, such as memory-mapped I/O devices.

If the processor is in virtual mode, an attempt to prefetch from guarded storage causes an instruction-storage interrupt. In this case, the prefetch never appears on the ISPLB.

Instruction-Side PLB I/O Signal Table

Figure 2-4 shows the block symbol for the instruction-side PLB interface. The signals are summarized in Table 2-7.

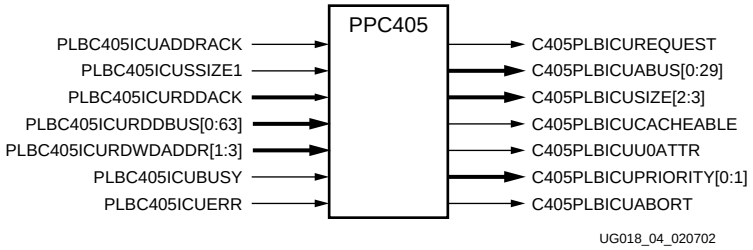

Figure 2-4: **Instruction-Side PLB Interface Block Symbol**

Table 2-7: **Instruction-Side PLB Interface Signal Summary**

Signal	I/O Type	If Unused	Function
C405PLBICUREQUEST	O	No Connect	Indicates the ICU is making an instruction-fetch request.
C405PLBICUABUS[0:29]	O	No Connect	Specifies the memory address of the instruction-fetch request. Bits 30:31 of the 32-bit address are assumed to be zero.
C405PLBICUSIZE[2:3]	O	No Connect	Specifies a four word or eight word line-transfer size.
C405PLBICUCACHEABLE	O	No Connect	Indicates the value of the cacheability storage attribute for the target address.
C405PLBICUU0ATTR	O	No Connect	Indicates the value of the user-defined storage attribute for the target address.
C405PLBICUPRIORITY[0:1]	O	No Connect	Indicates the priority of the ICU fetch request.
C405PLBICUABORT	O	No Connect	Indicates the ICU is aborting an unacknowledged fetch request.
PLBC405ICUADDRACK	I	0	Indicates a PLB slave acknowledges the current ICU fetch request.
PLBC405ICUSSIZE1	I	0	Specifies the bus width (size) of the PLB slave that accepted the request.
PLBC405ICURDDACK	I	0	Indicates the ICU read-data bus contains valid instructions for transfer to the ICU.

Table 2-7: **Instruction-Side PLB Interface Signal Summary *(Continued)***

Signal	I/O Type	If Unused	Function
PLBC405ICURDDBUS[0:63]	I	0x0000_0000 _0000_0000	The ICU read-data bus used to transfer instructions from the PLB slave to the ICU.
PLBC405ICURDWDADDR[1:3]	I	0b000	Indicates which word or doubleword of a four-word or eight-word line transfer is present on the ICU read-data bus.
PLBC405ICUBUSY	I	0	Indicates the PLB slave is busy performing an operation requested by the ICU.
PLBC405ICUERR	I	0	Indicates an error was detected by the PLB slave during the transfer of instructions to the ICU.

Instruction-Side PLB Interface I/O Signal Descriptions

The following sections describe the operation of the instruction-side PLB interface I/O signals.

Throughout these descriptions and unless otherwise noted, the term *clock* refers to the PLB clock signal, PLBCLK (see page 133 for information on this clock signal). The term *cycle* refers to a PLB cycle. To simplify the signal descriptions, it is assumed that PLBCLK and the PPC405x3 clock (CPMC405CLOCK) operate at the same frequency.

C405PLBICUREQUEST (output)

When asserted, this signal indicates the ICU is requesting instructions from a PLB slave device. The PLB slave asserts PLBC405ICUADDRACK to acknowledge the request. The request can be acknowledged in the same cycle it is presented by the ICU. The request is deasserted in the cycle after it is acknowledged by the PLB slave. When deasserted, no unacknowledged instruction-fetch request exists.

The following output signals contain information for the PLB slave device and are valid when the request is asserted. The PLB slave must latch these signals by the end of the same cycle it acknowledges the request:

- C405PLBICUABUS[0:31] contains the word address of the instruction-fetch request.
- C405PLBICUSIZE[2:3] indicates the instruction-fetch line-transfer size.
- C405PLBICUCACHEABLE indicates whether the instruction-fetch address is cacheable.
- C405PLBICUU0ATTR indicates the value of the user-defined storage attribute for the instruction-fetch address.

C405PLBICUPRIORITY[0:1] is also valid when the request is asserted. This signal indicates the priority of the instruction-fetch request. It is used by the PLB arbiter to prioritize simultaneous requests from multiple PLB masters.

The ICU supports two outstanding fetch requests over the PLB. The ICU can make a second fetch request (a prefetch) after the current request is acknowledged. The ICU deasserts C405PLBICUREQUEST for at least one cycle after the current request is acknowledged and before the subsequent request is asserted.

If the PLB slave supports address pipelining, it must respond to the two fetch requests in the order they are presented by the ICU. All instructions associated with the first request must be returned before any instruction associated with the second request is returned. The ICU cannot present a third fetch request until the first request is completed by the PLB

slave. This third request can be presented two cycles after the last read acknowledge (PLBC405ICURDDACK) is sent from the PLB slave to the ICU, completing the first request.

The ICU can abort a fetch request if it no longer requires the requested instruction. The ICU removes a request by asserting C405PLBICUABORT while the request is asserted. In the next cycle the request is deasserted and remains deasserted for at least one cycle.

C405PLBICUABUS[0:29] (output)

This bus specifies the memory address of the instruction-fetch request. Bits 30:31 of the 32-bit address are assumed to be zero so that all fetch requests are aligned on a word boundary. The fetch address is valid during the time the fetch request signal (C405PLBICUREQUEST) is asserted. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405ICUADDRACK to acknowledge the request).

C405PLBICUSIZE[2:3] indicates the instruction-fetch line-transfer size. The PLB slave uses memory-address bits [0:27] to specify an aligned four-word address for a four-word transfer size. Memory-address bits [0:26] are used to specify an aligned eight-word address for an eight-word transfer size.

C405PLBICUSIZE[2:3] (output)

These signals are used to specify the line-transfer size of the instruction-fetch request. A four-word transfer size is specified when C405PLBICUSIZE[2:3]=0b01. An eight-word transfer size is specified when C405PLBICUSIZE[2:3]=0b10. The transfer size is valid during the cycles the fetch-request signal (C405PLBICUREQUEST) is asserted. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405ICUADDRACK to acknowledge the request).

A four-word line transfer returns the quadword aligned on the address specified by C405PLBICUABUS[0:27]. This quadword contains the target instruction requested by the ICU. The quadword is returned using two doubleword or four word transfer operations, depending on the PLB slave bus width (64-bit or 32-bit, respectively).

An eight-word line transfer returns the eight-word cache line aligned on the address specified by C405PLBICUABUS[0:26]. This cache line contains the target instruction requested by the ICU. The cache line is returned using four doubleword or eight word transfer operations, depending on the PLB slave bus width (64-bit or 32-bit, respectively).

The words returned during a line transfer can be sent from the PLB slave to the ICU in any order (target-word-first, sequential, other). This transfer order is specified by PLBC405ICURDWDADDR[1:3].

C405PLBICUCACHEABLE (output)

This signal indicates whether the requested instructions are cacheable. It reflects the value of the cacheability storage attribute for the target address. The requested instructions are non-cacheable when the signal is deasserted (0). They are cacheable when the signal is asserted (1). This signal is valid during the time the fetch-request signal (C405PLBICUREQUEST) is asserted. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405ICUADDRACK to acknowledge the request).

Non-cacheable instructions are transferred using a four-word or eight-word line-transfer size. Software controls this transfer size using the *non-cacheable request-size* bit in the core-configuration register (CCR0[NCRS]). This enables non-cacheable transfers to take

advantage of the PLB line-transfer protocol to minimize PLB-arbitration delays and bus delays associated with multiple, single-word transfers. The transferred instructions are placed in the ICU fill buffer, but not in the instruction cache. Subsequent instruction fetches from the same non-cacheable line are read from the fill buffer instead of requiring a separate arbitration and transfer sequence across the PLB. Instructions in the fill buffer are fetched with the same performance as a cache hit. The non-cacheable line remains in the fill buffer until the fill buffer is needed by another line transfer.

Cacheable instructions are always transferred using an eight-word line-transfer size. The transferred instructions are placed in the ICU fill buffer as they are received from the PLB slave. Subsequent instruction fetches from the same cacheable line are read from the fill buffer during the time the line is transferred from the PLB slave. When the fill buffer is full, its contents are transferred to the instruction cache. Software can prevent this transfer by setting the *fetch without allocate* bit in the core-configuration register (CCR0[FWOA]). In this case, the cacheable line remains in the fill buffer until the fill buffer is needed by another line transfer. An exception is that the contents of the fill buffer are always transferred if the line was fetched because an **icbt** instruction was executed.

C405PLBICUU0ATTR (output)

This signal reflects the value of the user-defined (U0) storage attribute for the target address. The requested instructions are not in memory locations characterized by this attribute when the signal is deasserted (0). They are in memory locations characterized by this attribute when the signal is asserted (1). This signal is valid during the time the fetch-request signal (C405PLBICUREQUEST) is asserted. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405ICUADDRACK to acknowledge the request).

The system designer can use this signal to assign special behavior to certain memory addresses. Its use is optional.

C405PLBICUABORT (output)

When asserted, this signal indicates the ICU is aborting the current fetch request. It is used by the ICU to abort a request that has not been acknowledged, or is in the process of being acknowledged by the PLB slave. The fetch request continues normally if this signal is not asserted. This signal is only valid during the time the fetch-request signal (C405PLBICUREQUEST) is asserted. It must be ignored by the PLB slave if the fetch-request signal is not asserted. In the cycle after the abort signal is asserted, the fetch-request signal is deasserted and remains deasserted for at least one cycle.

If the abort signal is asserted in the same cycle that the fetch request is acknowledged by the PLB slave (PLBC405ICUADDRACK is asserted), the PLB slave is responsible for ensuring that the transfer does not proceed further. The PLB slave cannot assert the ICU read-data bus acknowledgement signal (PLBC405ICURDDACK) for an aborted request.

The ICU can abort an address-pipelined fetch request while the PLB slave is responding to a previous fetch request. The PLB slave is responsible for completing the previous fetch request and aborting the new (pipelined) request.

C405PLBICUPRIORITY[0:1] (output)

These signals are used to specify the priority of the instruction-fetch request. Table 2-8 shows the encoding of the 2-bit PLB-request priority signal. The priority is valid during the cycles the fetch-request signal (C405PLBICUREQUEST) is asserted. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405ICUADDRACK to acknowledge the request).

Table 2-8: **PLB-Request Priority Encoding**

Bit 0	Bit 1	Definition
0	0	Lowest PLB-request priority.
0	1	Next-to-lowest PLB-request priority.
1	0	Next-to-highest PLB-request priority.
1	1	Highest PLB-request priority.

Software establishes the instruction-fetch request priority by writing the appropriate value into the *ICU PLB-priority bits 0:1* of the core-configuration register (CCR0[IPP]). After a reset, the priority is set to the highest level (CCR0[IPP]=0b11).

PLBC405ICUADDRACK (input)

When asserted, this signal indicates the PLB slave acknowledges the ICU fetch request (indicated by the ICU assertion of C405PLBICUREQUEST). When deasserted, no such acknowledgement exists. A fetch request can be acknowledged by the PLB slave in the same cycle the request is asserted by the ICU. The PLB slave must latch the following fetch-request information in the same cycle it asserts the fetch acknowledgement:

- C405PLBICUABUS[0:29], which contains the word address of the instruction-fetch request.
- C405PLBICUSIZE[2:3], which indicates the instruction-fetch line-transfer size.
- C405PLBICUCACHEABLE, which indicates whether the instruction-fetch address is cacheable.
- C405PLBICUU0ATTR, which indicates the value of the user-defined storage attribute for the instruction-fetch address (use of this signal is optional).

During the acknowledgement cycle, the PLB slave must return its bus width indicator (32 bits or 64 bits) using the PLBC405ICUSSIZE1 signal.

The acknowledgement signal remains asserted for one cycle. In the next cycle, both the fetch request and acknowledgement are deasserted. Instructions can be returned to the ICU from the PLB slave beginning in the cycle following the acknowledgement. The PLB slave must abort an ICU fetch request (return no instructions) if the ICU asserts C405PLBICUABORT in the same cycle the PLB slave acknowledges the request.

The ICU supports two outstanding fetch requests over the PLB. The ICU can make a second fetch request after the current request is acknowledged. The ICU deasserts C405PLBICUREQUEST for at least one cycle after the current request is acknowledged and before the subsequent request is asserted.

If the PLB slave supports address pipelining, it must respond to the two fetch requests in the order they are presented by the ICU. All instructions associated with the first request must be returned before any instruction associated with the second request is returned. The ICU cannot present a third fetch request until the first request is completed by the PLB slave. This third request can be presented two cycles after the last read acknowledge (PLBC405ICURDDACK) is sent from the PLB slave to the ICU, completing the first request.

PLBC405ICUSSIZE1 (input)

This signal indicates the bus width (size) of the PLB slave device that acknowledged the ICU fetch request. A 32-bit PLB slave responded when the signal is deasserted (0). A 64-bit

PLB slave responded when the signal is asserted (1). This signal is valid during the cycle the acknowledge signal (PLBC405ICUADDRACK) is asserted.

The size signal is used by the ICU to determine how instructions are read from the 64-bit PLB interface during a transfer cycle (a transfer occurs when the PLB slave asserts PLBC405ICURDDACK). The ICU uses the size signal as follows:

- When a 32-bit PLB slave responds, an aligned word is sent from the slave to the ICU during each transfer cycle. The 32-bit PLB slave bus should be connected to both the high and low 32 bits of the 64-bit ICU read-data bus (see Figure 2-5, page 52). This type of connection duplicates the word returned by the slave across the 64-bit bus. The ICU reads either the low 32 bits or the high 32 bits of the 64-bit interface, depending on the order of the transfer (PLBC405ICURDWDADDR[1:3]).
- When a 64-bit PLB slave responds, an aligned doubleword is sent from the slave to the ICU during each transfer cycle. Both words are read from the 64-bit interface by the ICU in this cycle.

Table 2-10, page 53 shows the location of instructions on the ICU read-data bus as a function of PLB-slave size, line-transfer size, and transfer order.

PLBC405ICURDDACK (input)

When asserted, this signal indicates the ICU read-data bus contains valid instructions sent by the PLB slave to the ICU (read data is acknowledged). The ICU latches the data from the bus at the end of the cycle this signal is asserted. The contents of the ICU read-data bus are not valid when this signal is deasserted.

Read-data acknowledgement is asserted for one cycle per transfer. There is no limit to the number of cycles between two transfers. The number of transfers (and the number of read-data acknowledgements) depends on the following:

- The PLB slave size (bus width) specified by PLBC405ICUSSIZE1.
- The line-transfer size specified by C405PLBICUSIZE[2:3].
- The cacheability of the fetched instructions specified by C405PLBICUCACHEABLE.
- The value of the *non-cacheable request-size* bit (CCR0[NCRS]).

Table 2-9 summarizes the effect these parameters have on the number of transfers.

Table 2-9: **Number of Transfers Required for Instruction-Fetch Requests**

PLB-Slave Size	Line-Transfer Size	Instruction Cacheability	CCR0[NCRS]	Number of Transfers
32-Bit	Four Words	Non-Cacheable	0	4
	Eight Words		1	8
	Eight Words	Cacheable	—	8
64-Bit	Four Words	Non-Cacheable	0	2
	Eight Words		1	4
	Eight Words	Cacheable	—	4

PLBC405ICURDDBUS[0:63] (input)

This read-data bus contains the instructions transferred from a PLB slave to the ICU. The contents of the bus are valid when the read-data acknowledgement signal

(PLBC405ICURDDACK) is asserted. This acknowledgment is asserted for one cycle per transfer. There is no limit to the number of cycles between two transfers. The bus contents are not valid when the read-data acknowledgement signal is deasserted.

The PLB slave returns either a single instruction (an aligned word) or two instructions (an aligned doubleword) per transfer. The number of instructions sent per transfer depends on the PLB slave size (bus width), as follows:

- When a 32-bit PLB slave responds, an aligned word is sent from the slave to the ICU during each transfer cycle. The 32-bit PLB slave bus should be connected to both the high and low 32 bits of the 64-bit read-data bus, as shown in Figure 2-5 below. This type of connection duplicates the word returned by the slave across the 64-bit bus. The ICU reads either the low 32 bits or the high 32 bits of the 64-bit interface, depending on the value of PLBC405ICURDWDADDR[1:3].
- When a 64-bit PLB slave responds, an aligned doubleword is sent from the slave to the ICU during each transfer cycle. Both words are read from the 64-bit interface by the ICU in this cycle.

Table 2-10 shows the location of instructions on the ICU read-data bus as a function of PLB-slave size, line-transfer size, and transfer order.

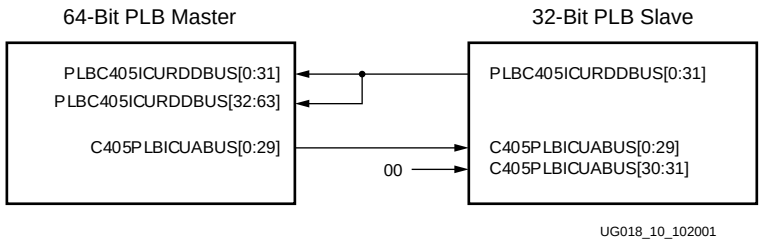

Figure 2-5: **Attachment of ISPLB Between 32-Bit Slave and 64-Bit Master**

PLBC405ICURDWDADDR[1:3] (input)

These signals are used to specify the transfer order. They identify which word or doubleword of a line transfer is present on the ICU read-data bus when the PLB slave returns instructions to the ICU. The words returned during a line transfer can be sent from the PLB slave to the ICU in any order (target-word-first, sequential, other). The transfer-order signals are valid when the read-data acknowledgement signal (PLBC405ICURDDACK) is asserted. This acknowledgment is asserted for one cycle per transfer. There is no limit to the number of cycles between two transfers. The transfer-order signals are not valid when the read-data acknowledgement signal is deasserted.

Table 2-10 shows the location of instructions on the ICU read-data bus as a function of PLB-slave size, line-transfer size, and transfer order. In this table, the *Transfer Order* column contains the possible values of PLBC405ICURDWDADDR[1:3]. For 64-bit PLB slaves, PLBC405ICURDWDADDR[3] should always be 0 during a transfer. In this case, the transfer order is invalid if this signal asserted. The entries for a 32-bit PLB slave assume the connection to a 64-bit master shown in Figure 2-5, page 52.

Table 2-10: **Contents of ICU Read-Data Bus During Line Transfer**

PLB-Slave Size	Line-Transfer Size	Transfer Order[1]	ICU Read-Data Bus [0:31][2]	ICU Read-Data Bus [32:63][2]
32-Bit	Four Words	x00	Instruction 0	*Instruction 0*
		x01	*Instruction 1*	Instruction 1
		x10	Instruction 2	*Instruction 2*
		x11	*Instruction 3*	Instruction 3
	Eight Words	000	Instruction 0	*Instruction 0*
		001	*Instruction 1*	Instruction 1
		010	Instruction 2	*Instruction 2*
		011	*Instruction 3*	Instruction 3
		100	Instruction 4	*Instruction 4*
		101	*Instruction 5*	Instruction 5
		110	Instruction 6	*Instruction 6*
		111	*Instruction 7*	Instruction 7
64-Bit	Four Words	x00	Instruction 0	Instruction 1
		x10	Instruction 2	Instruction 3
		xx1	Invalid	
	Eight Words	000	Instruction 0	Instruction 1
		010	Instruction 2	Instruction 3
		100	Instruction 4	Instruction 5
		110	Instruction 6	Instruction 7
		xx1	Invalid	

Notes:
1. "x" indicates a don't-care value in PLBC405ICURDWDADDR[1:3].
2. An instruction shown in *italics* is ignored by the ICU during the transfer.

PLBC405ICUBUSY (input)

When asserted, this signal indicates the PLB slave acknowledged and is responding to (is busy with) an ICU fetch request. When deasserted, the PLB slave is not responding to an ICU fetch request.

This signal should be asserted in the cycle after an ICU fetch request is acknowledged by the PLB slave and remain asserted until the request is completed by the PLB slave. It should be deasserted in the cycle after the last read-data acknowledgement signal is asserted by the PLB slave, completing the transfer. If multiple fetch requests are initiated and overlap, the busy signal should be asserted in the cycle after the first request is acknowledged and remain asserted until the cycle after the final read-data acknowledgement is completed for the last request.

Following reset, the processor block prevents the ICU from fetching instructions until the busy signal is deasserted for the first time. This is useful in situations where the processor

block is reset by a core reset, but PLB devices are not reset. Waiting for the busy signal to be deasserted prevents fetch requests following reset from interfering with PLB activity that was initiated before reset.

PLBC405ICUERR (input)

When asserted, this signal indicates the PLB slave detected an error when attempting to access or transfer the instructions requested by the ICU. This signal should be asserted with the read-data acknowledgement signal that corresponds to the erroneous transfer. The error signal should be asserted for only one cycle. When deasserted, no error is detected.

If a cacheable instruction is transferred with an error indication, it is loaded into the ICU fill buffer. However, the cache line held in the fill buffer is not transferred to the instruction cache.

The PLB slave must not terminate instruction transfers when an error is detected. The processor block is responsible for responding to any error detected by the PLB slave. A machine-check exception occurs if the PPC405x3 attempts to *execute* an instruction that was transferred to the ICU with an error indication. If an instruction is transferred with an error indication but is never executed, no machine-check exception occurs.

The PLB slave should latch error information in DCRs so that software diagnostic routines can attempt to report and recover from the error. A bus-error address register (BEAR) should be implemented for storing the address of the access that caused the error. A bus-error syndrome register (BESR) should be implemented for storing information about cause of the error.

Instruction-Side PLB Interface Timing Diagrams

The following timing diagrams show typical transfers that can occur on the ISPLB interface between the ICU and a bus-interface unit (BIU). These timing diagrams represent the optimal timing relationships supported by the processor block. The BIU can be implemented using the FPGA processor local bus (PLB) or using customized hardware. Not all BIU implementations support these optimal timing relationships.

The ICU only performs reads (fetches) when accessing instructions across the ISPLB interface.

ISPLB Timing Diagram Assumptions

The following assumptions and simplifications were made in producing the optimal timing relationships shown in the timing diagrams:

- Fetch requests are acknowledged by the BIU in the same cycle they are presented by the ICU. This represents the earliest cycle a BIU can acknowledge a fetch request.
- The first read-data acknowledgement for a line transfer is asserted in the cycle immediately following the fetch-request acknowledgement. This represents the earliest cycle a BIU can begin transferring instructions to the ICU in response to a fetch request. However, the earliest the FPGA PLB begins transferring instructions is *two cycles* after the fetch request is acknowledged.
- Subsequent read-data acknowledgements for a line transfer are asserted in the cycle immediately following the prior read-data acknowledgement. This represents the fastest rate at which a BIU can transfer instructions to the ICU (there is no limit to the number of cycles between two transfers).
- All line transfers assume the target instruction (word) is returned first. Subsequent instructions in the line are returned sequentially by address, wrapping as necessary to the lower addresses in the same line.
- The rate at which the ICU makes instruction-fetch requests to the BIU is not limited by the rate instructions are executed.
- An ICU fetch request to the BIU occurs two cycles after a miss is determined by the ICU.
- The ICU latches instructions into the fill buffer in the cycle after the instructions are received from the BIU on the PLB.
- The transfer of instructions from the fill buffer to the instruction cache takes three cycles. This transfer takes place after all instructions are read into the fill buffer from the BIU.
- The BIU size (bus width) is 64 bits, so PLBC405ICUSSIZE1 is not shown.
- No instruction-access errors occur, so PLBC405ICUERR is not shown.
- The abort signal, C405PLBICUABORT is shown only in the last example.
- The storage attribute signals are not shown.
- The ICU activity is shown only as an aide in describing the examples. The occurrence and duration of this activity is not observable on the ISPLB.

The following abbreviations appear in the timing diagrams:

Table 2-11: **Key to ISPLB Timing Diagram Abbreviations**

Abbreviation[1]	Description	Where Used	
rl#	Fetch-request identifier	Request Request acknowledge Read-data acknowledge	(C405PLBICUREQUEST) (PLBC405ICUADDRACK) (PLBC405ICURDDACK)
adr#	Fetch-request address	Request address	(C405PLBICUABUS[0:29])
d#$_{\#}$	Doublewords (two instructions) transferred as a result of a fetch request	ICU read-data bus	(PLBC405ICURDDBUS[0:63])
miss#	The ICU detects a cache miss that causes a fetch request on the PLB	ICU	
fill#	The ICU is busy performing a fill operation	ICU	
byp#	The ICU forwards instructions to the PPC405x3 instruction-fetch unit from the fill buffer as they become available (bypass)	ICU	
prefetch#	The ICU speculatively prefetches instructions from the BIU	ICU	
Subscripts	Used to identify the instruction words returned by a transfer	Read-data acknowledge ICU read-data bus ICU forward (bypass)	(PLBC405ICURDDACK) (PLBC405ICURDDBUS[0:63])
#	Used to identify the order doublewords are sent to the ICU	Transfer order	(PLBC405ICURDWDADDR[1:3])

Notes:
1. "#" indicates a number.

ISPLB Non-Pipelined Cacheable Sequential Fetch (Case 1)

The timing diagram in Figure 2-6 shows two consecutive eight-word line fetches that are not address pipelined. The example assumes instructions are fetched sequentially from the beginning of the first line through the end of the second line.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). Instructions are sent from the BIU to the ICU fill buffer in cycles 4 through 7. Instructions in the fill buffer are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp1 transaction in cycles 5 through 8). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache. This is represented by the fill1 transaction in cycles 9 through 11.

After the last instruction in the line is fetched, a sequential fetch from the next cache line causes a miss in cycle 13 (miss2). The second line read (rl2) is requested by the ICU in cycle 15 in response to the cache miss. Instructions are sent from the BIU to the ICU fill buffer in cycles 16 through 19. Instructions in the fill buffer are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp2 transaction in cycles 17 through 20). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache (not shown).

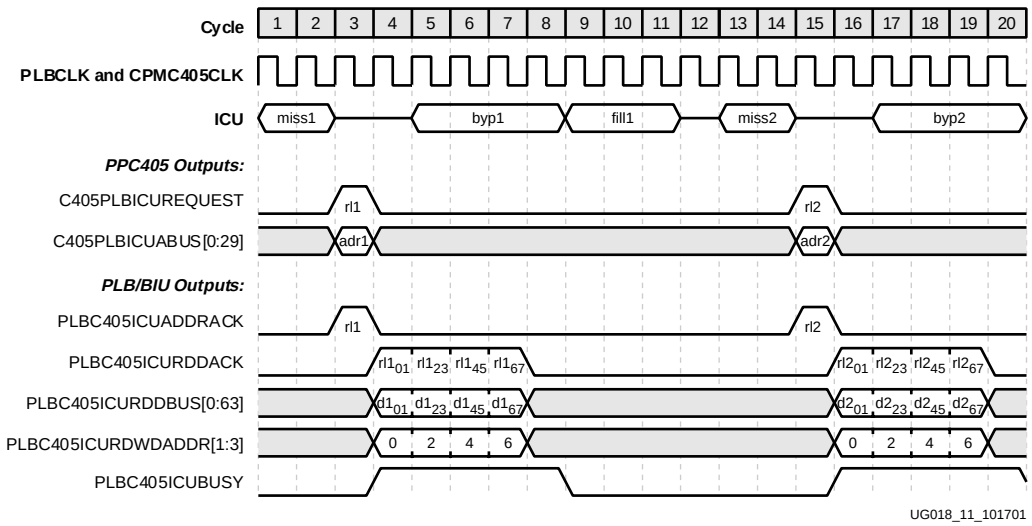

Figure 2-6: **ISPLB Non-Pipelined Cacheable Sequential Fetch (Case 1)**

ISPLB Non-Pipelined Cacheable Sequential Fetch (Case 2)

The timing diagram in Figure 2-7 shows two consecutive eight-word line fetches that are not address pipelined. The example assumes instructions are fetched sequentially from the end of the first line through the end of the second line. It provides an illustration of a transfer where the target instruction returned first by the BIU is not located at the start of the cache line.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). Instructions are sent from the BIU to the ICU fill buffer in cycles 4 through 7. The target instruction is bypassed to the instruction fetch unit in cycle 5 (byp1). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache. This is represented by the fill1 transaction in cycles 8 through 10.

After the target instruction is bypassed, a sequential fetch from the next cache line causes a miss in cycle 6 (miss2). The second line read (rl2) is requested by the ICU in cycle 8 in response to the cache miss. After the first line is read from the BIU, instructions for the second line are sent from the BIU to the ICU fill buffer. This occurs in cycles 9 through 12. Instructions in the fill buffer are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp2 transaction in cycles 11 through 13). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache (represented by the fill2 transaction in cycles 14 through 16).

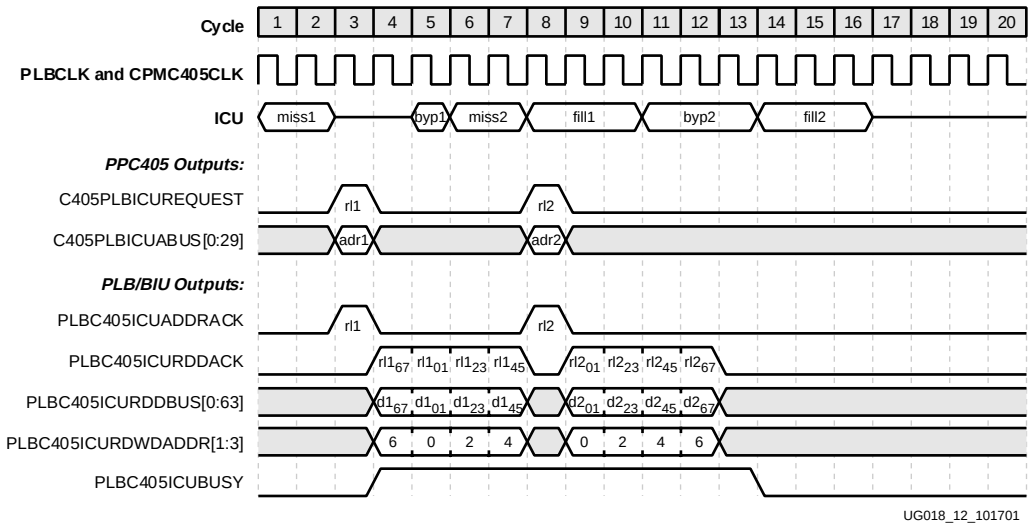

Figure 2-7: **ISPLB Non-Pipelined Cacheable Sequential Fetch (Case 2)**

ISPLB Pipelined Cacheable Sequential Fetch (Case 1)

The timing diagram in Figure 2-8 shows two consecutive eight-word line fetches that are address pipelined. The example assumes instructions are fetched sequentially from the beginning of the first line through the end of the second line. It shows the fastest speed at which the ICU can request and receive instructions over the PLB.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). Instructions are sent from the BIU to the ICU fill buffer in cycles 4 through 7. Instructions in the fill buffer are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp1 transaction in cycles 5 through 8). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache. This is represented by the fill1 transaction in cycles 9 through 11.

After the first miss is detected, the ICU performs a prefetch in anticipation of requiring instructions from the next cache line (represented by the prefetch2 transaction in cycles 3 and 4). The second line read (rl2) is requested by the ICU in cycle 5 in response to the prefetch. After the first line is read from the BIU, instructions for the second line are sent from the BIU to the ICU fill buffer. This occurs in cycles 8 through 11. After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache (represented by the fill2 transaction in cycles 13 through 15). Instructions from this second line are not bypassed because the fill buffer is transferred to the cache before the instructions are required.

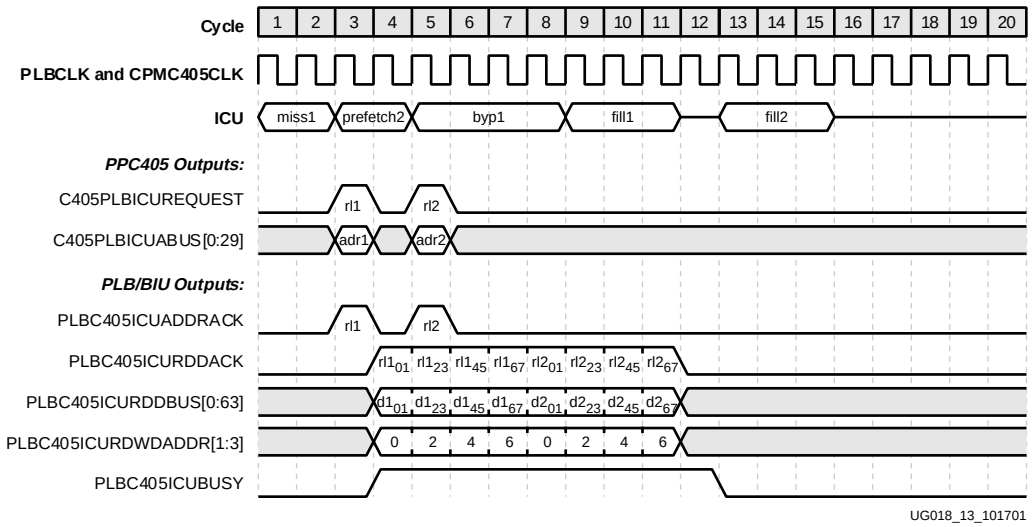

Figure 2-8: **ISPLB Pipelined Cacheable Sequential Fetch (Case 1)**

ISPLB Pipelined Cacheable Sequential Fetch (Case 2)

The timing diagram in Figure 2-9 shows two consecutive eight-word line fetches that are address pipelined. The example assumes instructions are fetched sequentially from the end of the first line through the end of the second line. As with the previous example, it shows the fastest speed at which the ICU can request and receive instructions over the PLB. It also illustrates a transfer where the target instruction returned first by the BIU is not located at the start of the cache line.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). Instructions are sent from the BIU to the ICU fill buffer in cycles 4 through 7. The target instruction is bypassed to the instruction fetch unit in cycle 5 (byp1). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache. This is represented by the fill1 transaction in cycles 8 through 10.

After the first miss is detected, the ICU performs a prefetch in anticipation of requiring instructions from the next cache line (represented by the prefetch2 transaction in cycles 3 and 4). The second line read (rl2) is requested by the ICU in cycle 5 in response to the prefetch. After the first line is read from the BIU, instructions for the second line are sent from the BIU to the ICU fill buffer. This occurs in cycles 8 through 11. Instructions in the fill buffer are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp2 transaction in cycles 11 through 12). After all instructions are received, they are transferred by the ICU from the fill buffer to the instruction cache (represented by the fill2 transaction in cycles 13 through 15).

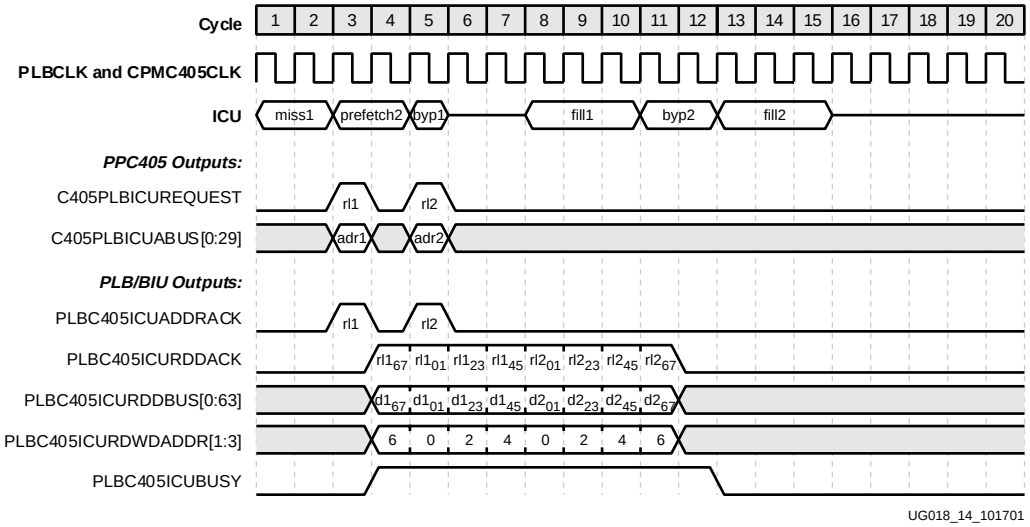

Figure 2-9: **ISPLB Pipelined Cacheable Sequential Fetch (Case 2)**

ISPLB Non-Pipelined Non-Cacheable Sequential Fetch

The timing diagram in Figure 2-10 shows two consecutive eight-word line fetches that are not address pipelined. The example assumes the instructions are not cacheable. It also assumes the instructions are fetched sequentially from the end of the first line through the end of the second line. It provides an illustration of how all instructions in a line must be transferred even though some of the instructions are discarded.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). Instructions are sent from the BIU to the ICU fill buffer in cycles 4 through 7. The target instruction is bypassed to the instruction fetch unit in cycle 5 (byp1). Because the instructions are executing sequentially, the target instruction is the only instruction in the line that is executed. The line is not cacheable, so instructions are not transferred from the fill buffer to the instruction cache.

After the target instruction is bypassed, a sequential fetch from the next cache line causes a miss in cycle 6 (miss2). The second line read (rl2) is requested by the ICU in cycle 8 in response to the cache miss. After the first line is read from the BIU, instructions for the second line are sent from the BIU to the ICU fill buffer. This occurs in cycles 9 through 12. These instructions overwrite the instructions from the previous line. After loading into the fill buffer, instructions from the second line are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp2 transaction in cycles 10 through 15). The line is not cacheable, so instructions are not transferred from the fill buffer to the instruction cache.

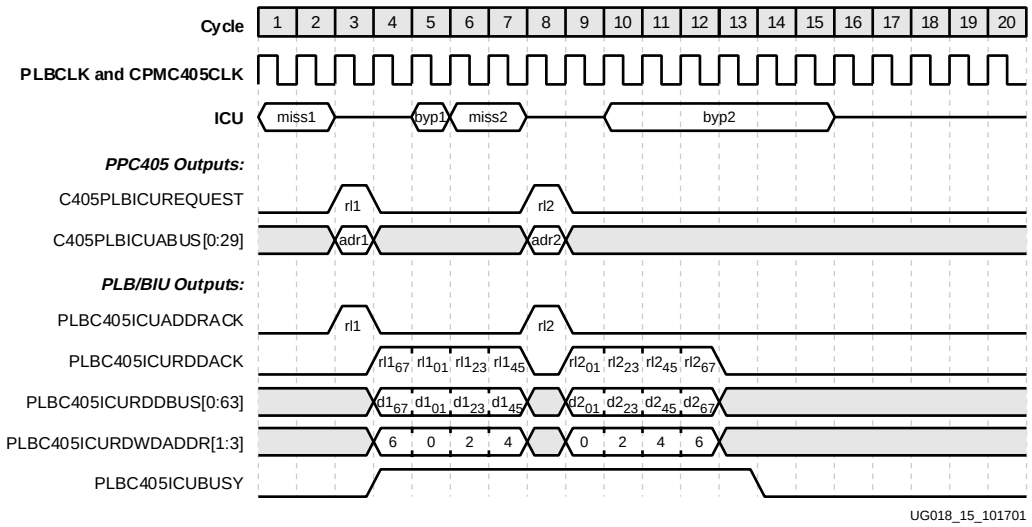

Figure 2-10: **ISPLB Non-Pipelined Non-Cacheable Sequential Fetch**

ISPLB Pipelined Non-Cacheable Sequential Fetch

The timing diagram in Figure 2-11 shows two consecutive eight-word line fetches that are address pipelined. The example assumes the instructions are not cacheable. It also assumes the instructions are fetched sequentially from the end of the first line through the end of the second line. As with the previous example, it provides an illustration of how all instructions in a line must be transferred even though some of the instructions are discarded.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). Instructions are sent from the BIU to the ICU fill buffer in cycles 4 through 7. The target instruction is bypassed to the instruction fetch unit in cycle 5 (byp1). Because the instructions are executing sequentially, the target instruction is the only instruction in the line that is executed. The line is not cacheable, so instructions are not transferred from the fill buffer to the instruction cache.

After the first miss is detected, the ICU performs a prefetch in anticipation of requiring instructions from the next cache line (represented by the prefetch2 transaction in cycles 3 and 4). The second line read (rl2) is requested by the ICU in cycle 5 in response to the prefetch. After the first line is read from the BIU, instructions for the second line are sent from the BIU to the ICU fill buffer. This occurs in cycles 8 through 11. These instructions overwrite the instructions from the previous line. After loading into the fill buffer, instructions from the second line are bypassed to the instruction fetch unit to prevent a processor stall during sequential execution (represented by the byp2 transaction in cycles 9 through 14). The line is not cacheable, so instructions are not transferred from the fill buffer to the instruction cache.

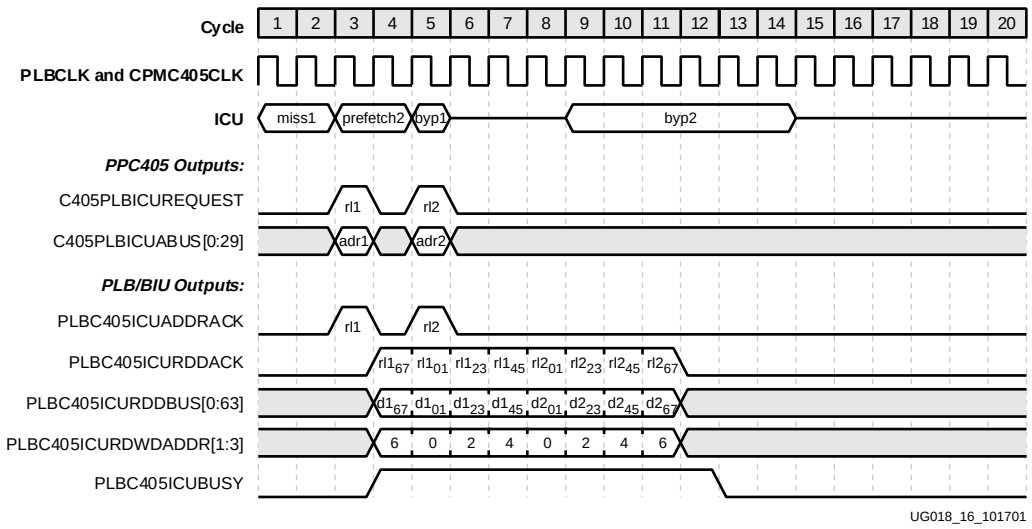

Figure 2-11: **ISPLB Pipelined Non-Cacheable Sequential Fetch**

ISPLB 2:1 Core-to-PLB Line Fetch

The timing diagram in Figure 2-12 shows an eight-word line fetch in a system with a PLB clock that runs at one half the frequency of the PPC405x3 clock.

The line read (rl1) is requested by the ICU in PLB cycle 2, which corresponds to PPC405x3 cycle 3. The BIU responds in the same cycle. Instructions are sent from the BIU to the ICU fill buffer in PLB cycles 3 through 6 (PPC405x3 cycles 5 through 12). After all instructions associated with this line are read, the line is transferred by the ICU from the fill buffer to the instruction cache (not shown).

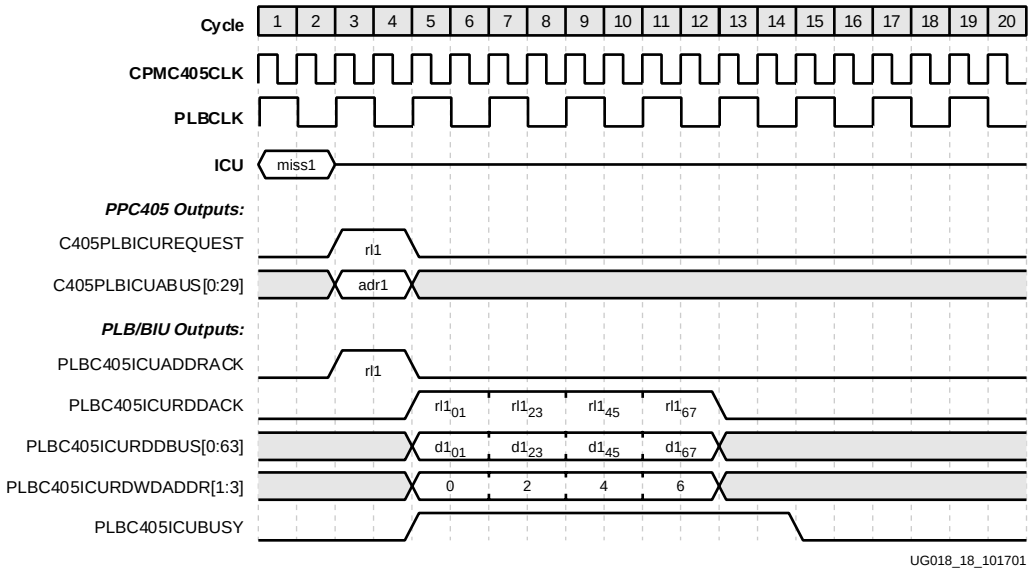

Figure 2-12: **ISPLB 2:1 Core-to-PLB Line Fetch**

ISPLB 3:1 Core-to-PLB Line Fetch

The timing diagram in Figure 2-13 shows an eight-word line fetch in a system with a PLB clock that runs at one third the frequency of the PPC405x3 clock.

The line read (rl1) is requested by the ICU in PLB cycle 2, which corresponds to PPC405x3 cycle 4. The BIU responds in the same cycle. Instructions are sent from the BIU to the ICU fill buffer in PLB cycles 3 through 6 (PPC405x3 cycles 7 through 18). After all instructions associated with this line are read, the line is transferred by the ICU from the fill buffer to the instruction cache (not shown).

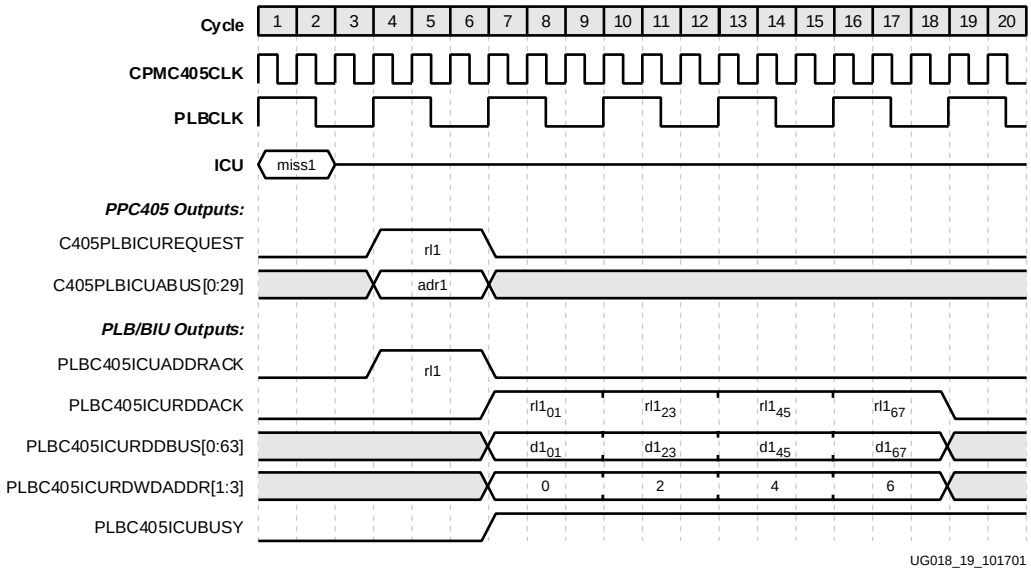

Figure 2-13: **ISPLB 3:1 Core-to-PLB Line Fetch**

ISPLB Aborted Fetch Request

The timing diagram in Figure 2-14 shows an aborted fetch request. The request is aborted because of an instruction-flow change, such as a taken branch or an interrupt. It shows the earliest-possible subsequent fetch-request that can be produced by the ICU.

The first line read (rl1) is requested by the ICU in cycle 3 in response to a cache miss (represented by the miss1 transaction in cycles 1 and 2). The BIU responds in the same cycle the request is made by the ICU. However, the processor also aborts the request in cycle 3, possibly because a branch was mispredicted or an interrupt occurred. Therefore, the BIU ignores the request and does not transfer instructions associated with the request.

The change in control flow causes the ICU to fetch instructions from a non-sequential address. The second line read (rl2) is requested by the ICU in cycle 7 in response to a cache miss of the new instructions. (represented by the miss2 transaction in cycles 5 and 6). Instructions are sent from the BIU to the ICU fill buffer in cycles 8 through 11.

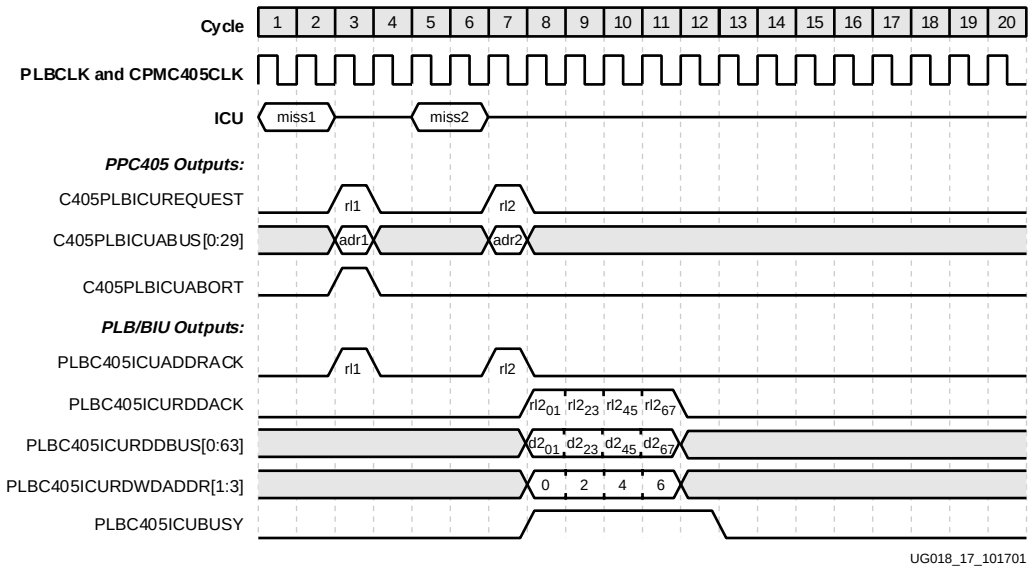

Figure 2-14: **ISPLB Aborted Fetch Request**

Data-Side Processor Local Bus Interface

The data-side processor local bus (DSPLB) interface enables the PPC405x3 data cache unit (DCU) to load (read) and store (write) data from any memory device connected to the processor local bus (PLB). This interface has a dedicated 32-bit address bus output, a dedicated 64-bit read-data bus input, and a dedicated 64-bit write-data bus output. The interface is designed to attach as a master to a 64-bit PLB, but it also supports attachment as a master to a 32-bit PLB. The interface is capable of one data transfer (64 or 32 bits) every PLB cycle.

At the chip level, the DSPLB can be combined with the instruction-side read-data bus (also a PLB master) to create a shared read-data bus. This is done if a single PLB arbiter services both PLB masters and the PLB arbiter implementation only returns data to one PLB master at a time.

Refer to *PowerPC Processor Reference Guide* for more information on the operation of the PPC405x3 DCU.

Data-Side PLB Operation

Data-access (read and write) requests are produced by the DCU and communicated over the PLB interface. A request occurs when an access misses the data cache or the memory location that is accessed is non-cacheable. A data-access request contains the following information:

- The request is indicated by C405PLBDCUREQUEST (page 70).
- The type of request (read or write) is indicated by C405PLBDCURNW (page 71).
- The target address of the data to be accessed is specified by the address bus, C405PLBDCUABUS[0:31] (page 71).
- The transfer size is specified as a single word or as eight words (cache line) using C405PLBDCUSIZE2 (page 71). The remaining bits of the transfer size (0, 1, and 3) must be tied to zero at the PLB arbiter.
- The byte enables for single-word accesses are specified using C405PLBDCUBE[0:7] (page 73). The byte enables specify one, two, three, or four contiguous bytes in either the upper or lower four byte word of the 64-bit data bus. The byte enables are not used by the processor during line transfers and must be ignored by the PLB slave.
- The cacheability storage attribute is indicated by C405PLBDCUCACHEABLE (page 72). Cacheable transfers are performed using word or line transfer sizes.
- The write-through storage attribute is indicated by C405PLBDCUWRITETHRU (page 72).
- The guarded storage attribute is indicated by C405PLBDCUGUARDED (page 73).
- The user-defined storage attribute is indicated by C405PLBDCUU0ATTR (page 73).
- The request priority is indicated by C405PLBDCUPRIORITY[0:1] (page 75). The PLB arbiter uses this information to prioritize simultaneous requests from multiple PLB masters.

The processor can abort a PLB data-access request using C405PLBDCUABORT (page 75). This occurs only when the processor is reset.

Data is returned to the DCU by a PLB slave device over the PLB interface. The response to a data-access request contains the following information:

- The address of the data-access request is acknowledged by the PLB slave using PLBC405DCUADDRACK (page 77).

- Data sent during a read transfer from the PLB slave to the DCU over the read-data bus are indicated as valid using PLBC405DCURDDACK (page 79). Data sent during a write transfer from the DCU to the PLB slave over the write-data bus are indicated as valid using PLBC405DCUWRDACK (page 80).
- The PLB-slave bus width, or size (32-bit or 64-bit), is specified by PLBC405DCUSSIZE1 (page 78). The PLB slave is responsible for packing (during reads) or unpacking (during writes) data bytes from non-word devices so that the information sent to the DCU is presented appropriately, as determined by the transfer size.
- The data transferred between the DCU and the PLB slave is sent as a single word or as an eight-word line transfer, as specified by the transfer size in the data-access request. Data reads are transferred from the PLB slave to the DCU over the DCU read-data bus, PLBC405DCURDDBUS[0:63] (page 79). Data writes are transferred from the DCU to the PLB slave over the DCU write-data bus, PLBC405DCUWRDBUS[0:63] (page 76). Data transfers operate as follows:
  - A word transfer moves the entire word specified by the address of the data-access request. The specific bytes being accessed are indicated by the byte enables, C405PLBDCUBE[0:7] (page 73). The word is transferred using one transfer operation.
  - An eight-word line transfer moves the eight-word cache line aligned on the address specified by C405PLBDCUABUS[0:26]. This cache line contains the target data accessed by the DCU. The cache line is transferred using four doubleword or eight word transfer operations, depending on the PLB slave bus width (64-bit or 32-bit, respectively). The byte enables are not used by the processor for this type of transfer and they must be ignored by the PLB slave.
- The words read during a data-read transfer can be sent from the PLB slave to the DCU in any order (target-word-first, sequential, other). This transfer order is specified by PLBC405DCURDWDADDR[1:3] (page 79). For data-write transfers, data is transferred from the DCU to the PLB slave in ascending-address order.

Interaction with the DCU Fill Buffer

As mentioned above, the PLB slave can transfer data to the DCU in any order (target-word-first, sequential, other). When data is received by the DCU from the PLB slave, it is placed in the DCU fill buffer. When the DCU receives the target (requested) data, it forwards it immediately from the fill buffer to the load / store unit so that pipeline stalls due to load-miss delays are minimized. This operation is referred to as a *bypass*. The remaining data is received from the PLB slave and placed in the fill buffer. Subsequent data is read from the fill buffer if the data is already present in the buffer. For the best possible software performance, the PLB slave should be designed to return the target word first.

Non-cacheable data is usually transferred as a single word. Software can indicate that non-cacheable reads be loaded using an eight-word line transfer by setting the *load-word-as-line bit* in the core-configuration register (CCR0[LWL]) to 1. This enables non-cacheable reads to take advantage of the PLB line-transfer protocol to minimize PLB-arbitration delays and bus delays associated with multiple, single-word transfers. The transferred data is placed in the DCU fill buffer, but not in the data cache. Subsequent data reads from the same non-cacheable line are read from the fill buffer instead of requiring a separate arbitration and transfer sequence across the PLB. Data in the fill buffer is read with the same performance as a cache hit. The non-cacheable line remains in the fill buffer until the fill buffer is needed by another line transfer.

Non-cacheable reads from guarded storage and all non-cacheable writes are transferred as a single word, regardless of the value of CCR0[LWL].

Cacheable data is transferred as a single word or as an eight-word line, depending on whether the transfer allocates a cache line. Transfers that allocate cache lines use eight-word transfer sizes. Transfers that do not allocate cache lines use a single-word transfer size. Line allocation of cacheable data is controlled by the core-configuration register. The *load without allocate* bit CCR0[LWOA] controls line allocation for cacheable loads and the *store without allocate* bit CCR0[SWOA] controls line allocation for cacheable stores. Clearing the appropriate bit to 0 enables line allocation (this is the default) and setting the bit to 1 disables line allocation. The **dcbt** and **dcbtst** instructions always allocate a cache line and ignore the CCR0 bits.

Data read during an eight-word line transfer (one that allocates a cache line) is placed in the DCU fill buffer as it is received from the PLB slave. Cacheable writes that allocate a cache line also cause an eight-word read transfer from the PLB slave. The cacheable write replaces the appropriate bytes in the fill buffer after they are read from the PLB. Subsequent data accesses to and from the same cacheable line access the fill buffer during the time the remaining bytes are transferred from the PLB slave. When the fill buffer is full, its contents are transferred to the data cache.

An eight-word line-write transfer occurs when the fill buffer replaces an existing data-cache line containing modified data. The existing cache line is written to memory before it is replaced with the fill-buffer contents. The write is performed using a separate PLB transaction than the previous transfer that caused the replacement. Execution of the **dcbf** and **dcbst** instructions also cause an eight-word line write.

Address Pipelining

The DCU can overlap a data-access request with a previous request. This process, known as *address pipelining*, enables a second address to be presented to a PLB slave while the slave is transferring data associated with the first address. Address pipelining can occur if a data-access request is produced before all data from a previous request are transferred by the slave. This capability maximizes PLB-transfer throughput by reducing dead cycles between multiple requests. The DCU can pipeline up to two read requests and one write request (multiple write requests cannot be pipelined). A pipelined request is communicated over the PLB two or more cycles after the prior request is acknowledged by the PLB slave.

Unaligned Accesses

If necessary, the processor automatically decomposes accesses to unaligned operands into two data-access requests that are presented separately to the PLB. This occurs if an operand crosses a word boundary (for a word transfer) or a cache line boundary (for an eight-word line transfer). For example, assume software reads the unaligned word at address 0x1F. This word crosses a cache line boundary: the byte at address 0x1F is in one cache line and the bytes at addresses 0x20:0x22 are in another cache line. If neither cache line is in the data cache, two consecutive read requests are presented by the DCU to the PLB slave. If one cache line is already in the data cache, only the missing portion is requested by the DCU.

Because write requests are not address pipelined by the DCU, writes to unaligned data that cross cache line boundaries can take significantly longer than aligned writes.

Guarded Storage

No bytes can be accessed speculatively from guarded storage. The PLB slave must return only the requested data when guarded storage is read and update only the specified memory locations when guarded storage is written. For single word transfers, only the

bytes indicated by the byte enables are transferred. For line transfers, all eight words in the line are transferred.

Data-Side PLB Interface I/O Signal Table

Figure 2-15 shows the block symbol for the data-side PLB interface. The signals are summarized in Table 2-12.

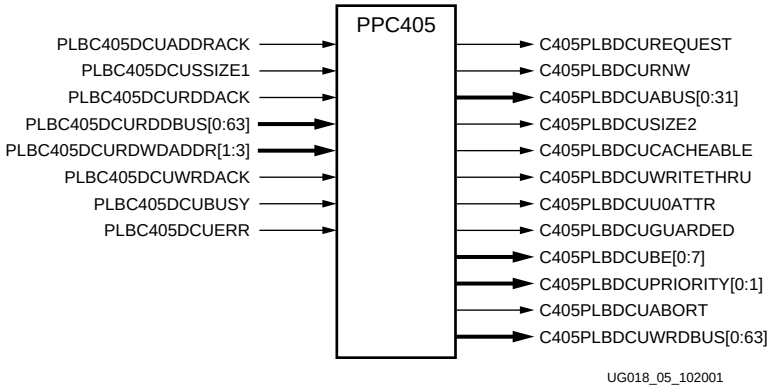

Figure 2-15: **Data-Side PLB Interface Block Symbol**

Table 2-12: **Data-Side PLB Interface I/O Signal Summary**

Signal	I/O Type	If Unused	Function
C405PLBDCUREQUEST	O	No Connect	Indicates the DCU is making a data-access request.
C405PLBDCURNW	O	No Connect	Specifies whether the data-access request is a read or a write.
C405PLBDCUABUS[0:31]	O	No Connect	Specifies the memory address of the data-access request.
C405PLBDCUSIZE2	O	No Connect	Specifies a single word or eight-word transfer size.
C405PLBDCUCACHEABLE	O	No Connect	Indicates the value of the cacheability storage attribute for the target address.
C405PLBDCUWRITETHRU	O	No Connect	Indicates the value of the write-through storage attribute for the target address.
C405PLBDCUU0ATTR	O	No Connect	Indicates the value of the user-defined storage attribute for the target address.
C405PLBDCUGUARDED	O	No Connect	Indicates the value of the guarded storage attribute for the target address.
C405PLBDCUBE[0:7]	O	No Connect	Specifies which bytes are transferred during single-word transfers.
C405PLBDCUPRIORITY[0:1]	O	No Connect	Indicates the priority of the data-access request.
C405PLBDCUABORT	O	No Connect	Indicates the DCU is aborting an unacknowledged data-access request.
C405PLBDCUWRDBUS[0:63]	O	No Connect	The DCU write-data bus used to transfer data from the DCU to the PLB slave.

Table 2-12: **Data-Side PLB Interface I/O Signal Summary *(Continued)***

Signal	I/O Type	If Unused	Function
PLBC405DCUADDRACK	I	0	Indicates a PLB slave acknowledges the current data-access request.
PLBC405DCUSSIZE1	I	0	Specifies the bus width (size) of the PLB slave that accepted the request.
PLBC405DCURDDACK	I	0	Indicates the DCU read-data bus contains valid data for transfer to the DCU.
PLBC405DCURDDBUS[0:63]	I	0x0000_0000 _0000_0000	The DCU read-data bus used to transfer data from the PLB slave to the DCU.
PLBC405DCURDWDADDR[1:3]	I	0b000	Indicates which word or doubleword of an eight-word line transfer is present on the DCU read-data bus.
PLBC405DCUWRDACK	I	0	Indicates the data on the DCU write-data bus is being accepted by the PLB slave.
PLBC405DCUBUSY	I	0	Indicates the PLB slave is busy performing an operation requested by the DCU.
PLBC405DCUERR	I	0	Indicates an error was detected by the PLB slave during the transfer of data to or from the DCU.

Data-Side PLB Interface I/O Signal Descriptions

The following sections describe the operation of the data-side PLB interface I/O signals.

Throughout these descriptions and unless otherwise noted, the term *clock* refers to the PLB clock signal, PLBCLK (see page 133 for information on this clock signal). The term *cycle* refers to a PLB cycle. To simplify the signal descriptions, it is assumed that PLBCLK and the PPC405x3 clock (CPMC405CLOCK) operate at the same frequency.

C405PLBDCUREQUEST (output)

When asserted, this signal indicates the DCU is presenting a data-access request to a PLB slave device. The PLB slave asserts PLBC405DCUADDRACK to acknowledge the request. The request can be acknowledged in the same cycle it is presented by the DCU. The request is deasserted in the cycle after it is acknowledged by the PLB slave. When deasserted, no unacknowledged data-access request exists.

The following output signals contain information for the PLB slave device and are valid when the request is asserted. The PLB slave must latch these signals by the end of the same cycle it acknowledges the request:

- C405PLBDCURNW, which specifies whether the data-access request is a read or a write.
- C405PLBDCUABUS[0:31], which contains the address of the data-access request.
- C405PLBDCUSIZE2, which indicates the transfer size of the data-access request.
- C405PLBDCUCACHEABLE, which indicates whether the data address is cacheable.
- C405PLBDCUWRITETHRU, which specifies the caching policy of the data address.
- C405PLBDCUU0ATTR, which indicates the value of the user-defined storage attribute for the instruction-fetch address.
- C405PLBDCUGUARDED, which indicates whether the data address is in guarded storage.

If the transfer size is a single word, C405PLBDCUBE[0:7] is also valid when the request is asserted. These signals specify which bytes are transferred between the DCU and PLB slave. If the transfer size is an eight-word line, C405PLBDCUBE[0:7] is not used and must be ignored by the PLB slave.

C405PLBDCUPRIORITY[0:1] is valid when the request is asserted. This signal indicates the priority of the data-access request. It is used by the PLB arbiter to prioritize simultaneous requests from multiple PLB masters.

The DCU supports up to three outstanding requests over the PLB (two reads and one write). The DCU can make a subsequent request after the current request is acknowledged. The DCU deasserts C405PLBDCUREQUEST for at least one cycle after the current request is acknowledged and before the subsequent request is asserted.

If the PLB slave supports address pipelining, it must respond to multiple requests in the order they are presented by the DCU. All data associated with a prior request must be transferred before any data associated with a subsequent request is transferred. Multiple write requests are not pipelined. The DCU does not present a second write request until at least two cycles after the last write acknowledge (PLBC405DCUWRDACK) is sent from the PLB slave to the DCU, completing the first request.

The DCU only aborts a data-access request if the processor is reset. The DCU removes a request by asserting C405PLBDCUABORT while the request is asserted. In the next cycle the request is deasserted and remains deasserted until after the processor is reset.

C405PLBDCURNW (output)

When asserted, this signal indicates the DCU is making a read request. When deasserted, this signal indicates the DCU is making a write request. This signal is valid when the DCU is presenting a data-access request to the PLB slave. The signal remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

C405PLBDCUABUS[0:31] (output)

This bus specifies the memory address of the data-access request. The address is valid during the time the data-access request signal (C405PLBDCUREQUEST) is asserted. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

C405PLBDCUSIZE2 indicates the data-access transfer size. If an eight-word transfer size is used, memory-address bits [0:26] specify the aligned eight-word cache line to be transferred. If a single word transfer size is used, the byte enables (C405PLBDCUBE[0:7]) specify which bytes on the data bus are involved in the transfer.

C405PLBDCUSIZE2 (output)

This signal specifies the transfer size of the data-access request. When asserted, an eight-word transfer size is specified. When deasserted, a single word transfer size is specified. This signal is valid when the DCU is presenting a data-access request to the PLB slave. The signal remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

A single word transfer moves one to four consecutive data bytes beginning at the memory address of the data-access request. For this transfer size, C405PLBDCUBE[0:7] specify which bytes on the data bus are involved in the transfer.

An eight-word line transfer moves the cache line aligned on the address specified by C405PLBDCUABUS[0:26]. This cache line contains the target data accessed by the DCU.

The cache line is transferred using four doubleword or eight word transfer operations, depending on the PLB slave bus width (64-bit or 32-bit, respectively).

The words moved during an eight-word line transfer can be sent from the PLB slave to the DCU in any order (target-word-first, sequential, other). This transfer order is specified by PLBC405DCURDWDADDR[1:3].

C405PLBDCUCACHEABLE (output)

This signal indicates whether the accessed data is cacheable. It reflects the value of the cacheability storage attribute for the target address. The data is non-cacheable when the signal is deasserted (0). The data is cacheable when the signal is asserted (1). This signal is valid when the DCU is presenting a data-access request to the PLB slave. The signal remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

Non-cacheable data is usually transferred as a single word. Software can indicate that non-cacheable reads be loaded using an eight-word line transfer by setting the *load-word-as-line bit* in the core-configuration register (CCR0[LWL]) to 1. This enables non-cacheable reads to take advantage of the PLB line-transfer protocol to minimize PLB-arbitration delays and bus delays associated with multiple, single-word transfers. The transferred data is placed in the DCU fill buffer, but not in the data cache. Subsequent data reads from the same non-cacheable line are read from the fill buffer instead of requiring a separate arbitration and transfer sequence across the PLB. Data in the fill buffer are read with the same performance as a cache hit. The non-cacheable line remains in the fill buffer until the fill buffer is needed by another line transfer.

Cacheable data is transferred as a single word or as an eight-word line, depending on whether the transfer allocates a cache line. Transfers that allocate cache lines use an eight-word transfer size. Transfers that do not allocate cache lines use a single-word transfer size. Line allocation of cacheable data is controlled by the core-configuration register. The *load without allocate* bit CCR0[LWOA] controls line allocation for cacheable loads and the *store without allocate* bit CCR0[SWOA] controls line allocation for cacheable stores. Clearing the appropriate bit to 0 enables line allocation (this is the default) and setting the bit to 1 disables line allocation. The **dcbt** and **dcbtst** instructions always allocate a cache line and ignore the CCR0 bits.

C405PLBDCUWRITETHRU (output)

This signal indicates whether the accessed data is in write-through or write-back cacheable memory. It reflects the value of the write-through storage attribute which controls the caching policy of the target address. The data is in write-back memory when the signal is deasserted (0). The data is in write-through memory when the signal is asserted (1). This signal is valid when the DCU is presenting a data-access request to the PLB slave and when the data cacheability signal is asserted. The signal remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

The system designer can use this signal in systems that require shared memory coherency. Stores to write-through memory update both the data cache and system memory. Stores to write-back memory update the data cache but not system memory. Write-back memory locations are updated in system memory when a cache line is flushed due to a line replacement or by executing a **dcbf** or **dcbst** instruction. See the *PowerPC Processor Reference Guide* for more information on memory coherency and caching policy.

C405PLBDCUU0ATTR (output)

This signal reflects the value of the user-defined (U0) storage attribute for the target address. The accessed data is not in a memory location characterized by this attribute when the signal is deasserted (0). It is in a memory location characterized by this attribute when the signal is asserted (1). This signal is valid when the DCU is presenting a data-access request to the PLB slave. The signal remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

The system designer can use this signal to assign special behavior to certain memory addresses. Its use is optional.

C405PLBDCUGUARDED (output)

This signal indicates whether the accessed data is in guarded storage. It reflects the value of the guarded storage attribute for the target address. The data is not in guarded storage when the signal is deasserted (0). The data is in guarded storage when the signal is asserted (1). This signal is valid when the DCU is presenting a data-access request to the PLB slave. The signal remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

No bytes are accessed speculatively from guarded storage. The PLB slave must return only the requested data when guarded storage is read and update only the specified memory locations when guarded storage is written. For single word transfers, only the bytes indicated by the byte enables are transferred. For line transfers, all eight words in the line are transferred.

C405PLBDCUBE[0:7] (output)

These signals, referred to as byte enables, indicate which bytes on the DCU read-data bus or write-data bus are valid during a word transfer. The byte enables are not used by the DCU during line transfers and must be ignored by the PLB slave. The byte enables are valid when the DCU is presenting a data-access request to the PLB slave. They remain valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

Attachment of a 32-bit PLB slave to the DCU (a 64-bit PLB master) requires the connections shown in Figure 2-16. These connections enable the byte enables to be presented properly to the 32-bit slave. Address bit 29 is used to select between the upper byte enables [0:3] and the lower byte enables [4:7] when making a request to the 32-bit slave. Words are always transferred to the 32-bit PLB slave using write-data bus bits [0:31], so bits [32:63] are not connected. The 32-bit read-data bus from the PLB slave is attached to both the high and low words of the 64-bit read-data bus into the DCU.

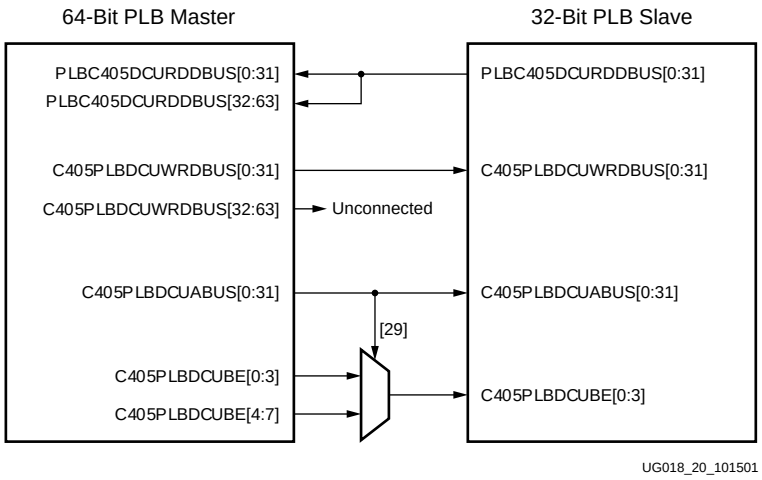

Figure 2-16: **Attachment of DSPLB Between 32-Bit Slave and 64-Bit Master**

Table 2-13 shows the possible values that can be presented by the byte enables and how they are interpreted by the PLB slave. All encodings of the byte enables not shown are invalid and are not generated by the DCU. The column headed "32-Bit PLB Slave Data Bus" assumes an attachment to a 64-bit PLB master as shown in Figure 2-16.

Table 2-13: **Interpretation of DCU Byte Enables During Word Transfers**

Byte Enables [0:7]	32-Bit PLB Slave Data Bus		64-Bit PLB Slave Data Bus	
	Valid Bytes	Bits	Valid Bytes	Bits
1000_0000	Byte 0	0:7	Byte 0	0:7
1100_0000	Bytes 0:1 (Halfword 0)	0:15	Bytes 0:1 (Halfword 0)	0:15
1110_0000	Bytes 0:2	0:23	Bytes 0:2	0:23
1111_0000	Bytes 0:3 (Word 0)	0:31	Bytes 0:3 (Word 0)	0:31
0100_0000	Byte 1	8:15	Byte 1	8:15
0110_0000	Bytes 1:2	8:23	Bytes 1:2	8:23
0111_0000	Bytes 1:3	8:31	Bytes 1:3	8:31
0010_0000	Byte 2	16:23	Byte 2	16:23
0011_0000	Bytes 2:3 (Halfword 1)	16:31	Bytes 2:3 (Halfword 1)	16:31
0001_0000	Byte 3	24:31	Byte 3	24:31
0000_1000	Byte 0	0:7	Byte 4	32:39
0000_1100	Bytes 0:1 (Halfword 0)	0:15	Bytes 4:5 (Halfword 2)	32:47
0000_1110	Bytes 0:2	0:23	Bytes 4:6	32:55
0000_1111	Bytes 0:3 (Word 0)	0:31	Bytes 4:7 (Word 1)	32:63
0000_0100	Byte 1	8:15	Byte 5	40:47
0000_0110	Bytes 1:2	8:23	Bytes 5:6	40:55

Table 2-13: **Interpretation of DCU Byte Enables During Word Transfers *(Continued)***

Byte Enables [0:7]	32-Bit PLB Slave Data Bus		64-Bit PLB Slave Data Bus	
	Valid Bytes	Bits	Valid Bytes	Bits
0000_0111	Bytes 1:3	8:31	Bytes 5:7	40:63
0000_0010	Byte 2	16:23	Byte 6	48:55
0000_0011	Bytes 2:3 (Halfword 1)	16:31	Bytes 6:7 (Halfword 3)	48:63
0000_0001	Byte 3	24:31	Byte 7	56:63

C405PLBDCUPRIORITY[0:1] (output)

These signals are used to specify the priority of the data-access request. Table 2-8 shows the encoding of the 2-bit PLB-request priority signal. The priority is valid when the DCU is presenting a data-access request to the PLB slave. It remains valid until the cycle following acknowledgement of the request by the PLB slave (the PLB slave asserts PLBC405DCUADDRACK to acknowledge the request).

Table 2-14: **PLB-Request Priority Encoding**

Bit 0	Bit 1	Definition
0	0	Lowest PLB-request priority.
0	1	Next-to-lowest PLB-request priority.
1	0	Next-to-highest PLB-request priority.
1	1	Highest PLB-request priority.

Bit 1 of the request priority is controlled by the DCU. It is asserted whenever a data-read request is presented on the PLB. The DCU can also assert this bit if the processor stalls due to an unacknowledged request. Software controls bit 0 of the request priority by writing the appropriate value into the *DCU PLB-priority bit 1* of the core-configuration register (CCR0[DPP1]).

If the least significant bits of the DCU and ICU PLB priority signals are 1 and the most significant bits are equal, the PLB arbiter should let the DCU win the arbitration. This generally results in better processor performance.

C405PLBDCUABORT (output)

When asserted, this signal indicates the DCU is aborting the current data-access request. It is used by the DCU to abort a request that has not been acknowledged, or is in the process of being acknowledged by the PLB slave. The data-access request continues normally if this signal is not asserted. This signal is only valid during the time the data-access request signal is asserted. It must be ignored by the PLB slave if the data-access request signal is not asserted. In the cycle after the abort signal is asserted, the data-access request signal is deasserted and remains deasserted for at least one cycle.

If the abort signal is asserted in the same cycle that the data-access request is acknowledged by the PLB slave (PLBC405DCUADDRACK is asserted), the PLB slave is responsible for ensuring that the transfer does not proceed further. The PLB slave must not assert the DCU read-data bus acknowledgement signal for an aborted request. It is possible for a PLB slave to return the first write acknowledgement when acknowledging an aborted data-write request. In this case, memory must not be updated by the PLB slave

and no further write acknowledgements can be presented by the PLB slave for the aborted request.

The DCU only aborts a data-access request when the processor is reset. Such an abort can occur during an address-pipelined data-access request while the PLB slave is responding to a previous data-access request. If the PLB is not also reset (as is the case during a core reset), the PLB slave is responsible for completing the previous request and aborting the new (pipelined) request.

PLBC405DCUWRDBUS[0:63] (output)

This write-data bus contains the data transferred from the DCU to a PLB slave during a write transfer. The operation of this bus depends on the transfer size, as follows:

- During a single word write, the write-data bus is valid when the write request is presented by the DCU. The data remains valid until the PLB slave accepts the data. The PLB slave asserts the write-data acknowledgement signal when it latches data transferred on the write-data bus, indicating that it accepts the data. This completes the word write.

  The DCU replicates the data on the high and low words of the write data bus (bits [0:31] and [32:63], respectively) during a single word write. The byte enables indicate which bytes on the high word or low word are valid and should be latched by the PLB slave.
- During an eight-word line transfer, the write-data bus is valid when the write request is presented by the DCU. The data remains valid until the PLB slave accepts the data. The PLB slave asserts the write-data acknowledgement signal when it latches data transferred on the write-data bus, indicating that it accepts the data. In the cycle after the PLB slave accepts the data, the DCU presents the next word or doubleword of data (depending on the PLB slave size). Again, the PLB slave asserts the write-data acknowledgement signal when it latches data transferred on the write-data bus, indicating that it accepts the data. This continues until all eight words are transferred to the PLB slave.

  Data is transferred from the DCU to the PLB slave in ascending address order. Word 0 (lowest address of the cache line) is transferred first and word 7 (highest address) is transferred last. The byte enables are not used during a line transfer and must be ignored by the PLB slave.

  The location of data on the write-data bus depends on the size of the PLB slave, as follows:

  - If the slave has a 64-bit bus, the DCU transfers even words (words 0, 2, 4, and 6) on write-data bus bits [0:31] and odd words (words 1, 3, 5, and 7) on write-data bus bits [32:63]. Four doubleword writes are required to complete the eight-word line transfer. The first transfer writes words 0 and 1, the second transfer writes words 2 and 3, and so on.
  - If the slave has a 32-bit bus, the DCU transfers all words on write-data bus bits [0:31]. Eight doubleword writes are required to complete the eight-word line transfer. The first transfer writes word 0, the second transfer writes word 1, and so on.

  Table 2-15 summarizes the location of words on the write-data bus during an eight-word line transfer.

Table 2-15: **Contents of DCU Write-Data Bus During Eight-Word Line Transfer**

PLB-Slave Size	Transfer	DCU Write-Data Bus [0:31]	DCU Write-Data Bus [32:63]
32-Bit	First	Word 0	Not Applicable
	Second	Word 1	
	Third	Word 2	
	Fourth	Word 3	
	Fifth	Word 4	
	Sixth	Word 5	
	Seventh	Word 6	
	Eighth	Word 7	
64-Bit	First	Word 0	Word 1
	Second	Word 2	Word 3
	Third	Word 4	Word 5
	Fourth	Word 6	Word 7

PLBC405DCUADDRACK (input)

When asserted, this signal indicates the PLB slave acknowledges the DCU data-access request (indicated by the DCU assertion of C405PLBDCUREQUEST). When deasserted, no such acknowledgement exists. A data-access request can be acknowledged by the PLB slave in the same cycle the request is asserted by the DCU. The PLB slave must latch the following data-access request information in the same cycle it asserts the request acknowledgement:

- C405PLBDCURNW, which specifies whether the data-access request is a read or a write.
- C405PLBDCUABUS[0:31], which contains the address of the data-access request.
- C405PLBDCUSIZE2, which indicates the transfer size of the data-access request.
- C405PLBDCUCACHEABLE, which indicates whether the data address is cacheable.
- C405PLBDCUWRITETHRU, which specifies the caching policy of the data address.
- C405PLBDCUU0ATTR, which indicates the value of the user-defined storage attribute for the instruction-fetch address.
- C405PLBDCUGUARDED, which indicates whether the data address is in guarded storage.

During the acknowledgement cycle, the PLB slave must return its bus width indicator (32 bits or 64 bits) using the PLBC405DCUSSIZE1 signal.

The acknowledgement signal remains asserted for one cycle. In the next cycle, both the data-access request and acknowledgement are deasserted. The PLB slave can begin receiving data from the DCU in the same cycle the address is acknowledged. Data can be sent to the DCU beginning in the cycle after the address acknowledgement. The PLB slave must abort a DCU request (move no data) if the DCU asserts C405PLBDCUABORT in the same cycle the PLB slave acknowledges the request.

The DCU supports up to three outstanding requests over the PLB (two read and one write). The DCU can make a subsequent request after the current request is acknowledged. The DCU deasserts C405PLBDCUREQUEST for at least one cycle after the current request is acknowledged and before the subsequent request is asserted.

If the PLB slave supports address pipelining, it must respond to multiple requests in the order they are presented by the DCU. All data associated with a prior request must be moved before data associated with a subsequent request is accessed. The DCU cannot present a third read request until the first read request is completed by the PLB slave, or a second write request until the first write request is completed. Such a request (third read or second write) can be presented two cycles after the last acknowledge is sent from the PLB slave to the DCU, completing the first request (read or write, respectively).

PLBC405DCUSSIZE1 (input)

This signal indicates the bus width (size) of the PLB slave device that acknowledged the DCU request. A 32-bit PLB slave responded when the signal is deasserted (0). A 64-bit PLB slave responded when the signal is asserted (1). This signal is valid during the cycle the acknowledge signal (PLBC405DCUADDRACK) is asserted.

A 32-bit PLB slave must be attached to a 64-bit PLB master as shown in Figure 2-16, page 74. In this figure, the 32-bit read-data bus from the PLB slave is attached to both the high word and low word of the 64-bit read-data bus at the PLB master. The 32-bit write-data bus into the PLB slave is attached to the high word of the 64-bit write-data bus at the PLB master. The low word of the 64-bit write-data bus is not connected. When a 64-bit PLB master recognizes a 32-bit PLB slave (the size signal is deasserted), data transfers operate as follows:

- During a single word read, data is received by the 64-bit master over the high word (bits 0:31) or the low word (bits 32:63) of the read-data bus as specified by the byte enable signals.
- During an eight-word line read, data is received by the 64-bit master over the high word (bits 0:31) or the low word (bits 32:63) of the read-data bus as specified by bit 3 of the transfer order (PLBC405DCURDWDADDR[1:3]). Table 2-10, page 53, shows the location of data on the DCU read-data bus as a function of transfer order when an eight-word line read from a 32-bit PLB slave occurs.
- During a single word write or an eight-word line write, data is sent by the 64-bit master over the high word (bits 0:31) of the write-data bus. Table 2-15, page 77, shows the order data is transferred to a 32-bit PLB slave during an eight-word line write.

All bits of the read-data bus and write-data bus are directly connected between a 64-bit PLB slave and a 64-bit PLB master. When a 64-bit PLB master recognizes a 64-bit PLB slave (the size signal is asserted), data transfers operate as follows:

- During a single word read, data is received by the 64-bit master over the high word (bits 0:31) or the low word (bits 32:63) of the read-data bus as specified by the byte enable signals.
- During an eight-word line read, data is received by the 64-bit master over the entire read-data bus. Table 2-10, page 53, shows the location of data on the DCU read-data bus as a function of transfer order when an eight-word line read from a 64-bit PLB slave occurs.
- During a single word write, the DCU replicates the data on the high and low words of the write data bus. The byte enables indicate which bytes on the high word or low word are valid and should be latched by the PLB slave.
- During an eight-word line write, data is sent by the 64-bit master over the entire

write-data bus. Table 2-15, page 77, shows the order data is transferred to a 64-bit PLB slave during an eight-word line write. Data is written in order of ascending address, so the transfer order signals are not used during a line write.

PLBC405DCURDDACK (input)

When asserted, this signal indicates the DCU read-data bus contains valid data sent by the PLB slave to the DCU (read data is acknowledged). The DCU latches the data from the bus at the end of the cycle this signal is asserted. The contents of the DCU read-data bus are not valid when this signal is deasserted.

Read-data acknowledgement is asserted for one cycle per transfer. There is no limit to the number of cycles between two transfers. The number of transfers (and the number of read-data acknowledgements) depends on the PLB slave size (specified by PLBC405DCUSSIZE1) and the line-transfer size (specified by C405PLBDCUSIZE2). The number of transfers are summarized as follows:

- Single word reads require one transfer, regardless of the PLB slave size.
- Eight-word line reads require eight transfers when sent from a 32-bit PLB slave.
- Eight-word line reads require four transfers when sent from a 64-bit PLB slave.

PLBC405DCURDDBUS[0:63] (input)

This read-data bus contains the data transferred from a PLB slave to the DCU. The contents of the bus are valid when the read-data acknowledgement signal is asserted. This acknowledgment is asserted for one cycle per transfer. There is no limit to the number of cycles between two transfers. The bus contents are not valid when the read-data acknowledgement signal is deasserted.

The PLB slave returns data as an aligned word or an aligned doubleword. This depends on the PLB slave size (bus width), as follows:

- When a 32-bit PLB slave responds, an aligned word is sent from the slave to the DCU during each transfer cycle. The 32-bit PLB slave bus should be connected to both the high and low 32 bits of the 64-bit read-data bus (see Figure 2-16, page 74). This type of connection duplicates the word returned by the slave across the 64-bit bus. The DCU reads either the low 32 bits or the high 32 bits of the 64-bit interface, depending on the value of PLBC405DCURDWDADDR[1:3].
- When a 64-bit PLB slave responds, an aligned doubleword is sent from the slave to the DCU during each transfer cycle. Both words are read from the 64-bit interface by the DCU in this cycle.

For a single word transfer, the bytes enables are used to select the valid data bytes from the aligned word or doubleword. Table 2-13, page 74 shows how the byte enables are interpreted by the processor when reading data during single word transfers from 32-bit and 64-bit PLB slaves. Table 2-10 shows the location of data on the DCU read-data bus as a function of PLB-slave size and transfer order when an eight-word line read occurs.

PLBC405DCURDWDADDR[1:3] (input)

These signals are used to specify the transfer order. They identify which word or doubleword of an eight-word line transfer is present on the DCU read-data bus when the PLB slave returns instructions to the DCU. The words returned during a line transfer can be sent from the PLB slave to the DCU in any order (target-word-first, sequential, other). The transfer-order signals are valid when the read-data acknowledgement signal (PLBC405DCURDDACK) is asserted. This acknowledgment is asserted for one cycle per

transfer. There is no limit to the number of cycles between two transfers. The transfer-order signals are not valid when the read-data acknowledgement signal is deasserted.

These signals are ignored by the processor during single word transfers.

Table 2-10 shows the location of data on the DCU read-data bus as a function of PLB-slave size and transfer order when an eight-word line read occurs. In this table, the "Transfer Order" column contains the possible values of PLBC405DCURDWDADDR[1:3]. For 64-bit PLB slaves, PLBC405DCURDWDADDR[3] should always be 0 during a transfer. In this case, the transfer order is invalid if this signal asserted. For 32-bit slaves, the connection to a 64-bit master shown in Figure 2-16, page 74 is assumed.

Table 2-16: **Contents of DCU Read-Data Bus During Eight-Word Line Transfer**

PLB-Slave Size	Transfer Order[1]	DCU Read-Data Bus [0:31][2]	DCU Read-Data Bus [32:63][2]
32-Bit	000	Word 0	*Word 0*
	001	*Word 1*	Word 1
	010	Word 2	*Word 2*
	011	*Word 3*	Word 3
	100	Word 4	*Word 4*
	101	*Word 5*	Word 5
	110	Word 6	*Word 6*
	111	*Word 7*	Word 7
64-Bit	000	Word 0	Word 1
	010	Word 2	Word 3
	100	Word 4	Word 5
	110	Word 6	Word 7
	xx1	Invalid	

Notes:
1. "x" indicates a don't-care value in PLBC405DCURDWDADDR[1:3].
2. A word shown in *italics* is ignored by the DCU during the transfer.

PLBC405DCUWRDACK (input)

When asserted, this signal indicates the PLB slave latched the data on the write-data bus sent from the DCU (write data is acknowledged). The DCU holds this data valid until the end of the cycle this signal is asserted. In the following cycle, the DCU presents new data and holds it valid until acknowledged by the PLB slave. This continues until all write data is transferred from the DCU to the PLB slave. If this signal is deasserted, valid data on the write data bus has not been latched by the PLB slave.

Write-data acknowledgement is asserted for one cycle per transfer. There is no limit to the number of cycles between two transfers. The number of transfers (and the number of write-data acknowledgements) depends on the PLB slave size (specified by PLBC405DCUSSIZE1 and the line-transfer size (specified by C405PLBDCUSIZE2). The number of transfers are summarized as follows:

- Single word writes require one transfer, regardless of the PLB slave size.

- Eight-word line writes require eight transfers when sent to a 32-bit PLB slave.
- Eight-word line writes require four transfers when sent to a 64-bit PLB slave.

PLBC405DCUBUSY (input)

When asserted, this signal indicates the PLB slave acknowledged and is responding to (is busy with) a DCU data-access request. When deasserted, the PLB slave is not responding to a DCU data-access request.

This signal should be asserted in the cycle after a DCU request is acknowledged by the PLB slave and remain asserted until the request is completed by the PLB slave. For read requests, it should be deasserted in the cycle after the last read-data acknowledgement. For write requests, it should be deasserted in the cycle after the target memory device is updated by the PLB slave. If multiple requests are initiated and overlap, the busy signal should be asserted in the cycle after the first request is acknowledged and remain asserted until the cycle after the last request is completed.

The processor monitors the busy signal when executing a **sync** instruction. The **sync** instruction requires that all storage operations initiated prior to the **sync** be completed before subsequent instructions are executed. Storage operations are considered complete when there are no pending DCU requests and the busy signal is deasserted.

Following reset, the processor block prevents the DCU from accessing data until the busy signal is deasserted for the first time. This is useful in situations where the processor block is reset by a core reset, but PLB devices are not reset. Waiting for the busy signal to be deasserted prevents data accesses following reset from interfering with PLB activity that was initiated before reset.

PLBC405DCUERR (input)

When asserted, this signal indicates the PLB slave detected an error when attempting to transfer data to or from the DCU. The error signal should be asserted for only one cycle. When deasserted, no error is detected.

For read operations, this signal should be asserted with the read-data acknowledgement signal that corresponds to the erroneous transfer. For write operations, it is possible for the error to not be detected until some time after the data is accepted by the PLB slave. Thus, the signal can be asserted independently of the write-data acknowledgement signal that corresponds to the erroneous transfer. However, it must be asserted while the busy signal is asserted.

The PLB slave must not terminate data transfers when an error is detected. The processor block is responsible for responding to any error detected by the PLB slave. A machine-check exception occurs if the exception is enabled by software (MSR[ME]=1) and data is transferred between the processor block and a PLB slave while the error signal is asserted.

The PLB slave should latch error information in DCRs so that software diagnostic routines can attempt to report and recover from the error. A bus-error address register (BEAR) should be implemented for storing the address of the access that caused the error. A bus-error syndrome register (BESR) should be implemented for storing information about cause of the error.

Data-Side PLB Interface Timing Diagrams

The following timing diagrams show typical transfers that can occur on the DSPLB interface between the DCU and a bus-interface unit (BIU). These timing diagrams represent the optimal timing relationships supported by the processor block. The BIU can be implemented using the FPGA processor local bus (PLB) or using customized hardware. Not all BIU implementations support these optimal timing relationships.

DSPLB Timing Diagram Assumptions

The following assumptions and simplifications were made in producing the optimal timing relationships shown in the timing diagrams:

- Requests are acknowledged by the BIU in the same cycle they are presented by the DCU if the BIU is not busy. This represents the earliest cycle a BIU can acknowledge a request. If the BIU is busy, the request is acknowledged in a later cycle.
- The first read-data acknowledgement for a data read is asserted in the cycle immediately following the read-request acknowledgement. This represents the earliest cycle a BIU can begin transferring data to the DCU in response to a read request. However, the earliest the FPGA PLB begins transferring data is *two cycles* after the read request is acknowledged.
- Subsequent read-data acknowledgements for eight-word line transfers are asserted in the cycle immediately following the prior read-data acknowledgement. This represents the fastest rate at which a BIU can transfer data to the DCU (there is no limit to the number of cycles between two transfers).
- The first write-data acknowledgement for a data write is asserted in the same cycle as the write-request acknowledgement. This represents the earliest cycle a BIU can begin accepting data from the DCU in response to a write request.
- Subsequent write-data acknowledgements for eight-word line transfers are asserted in the cycle immediately following the prior write-data acknowledgement. This represents the fastest rate at which the DCU can transfer data to the BIU (there is no limit to the number of cycles between two transfers).
- All eight-word line reads assume the target data (word) is returned first. Subsequent data in the line is returned sequentially by address, wrapping as necessary to the lower addresses in the same line.
- The transfer of read data from the fill buffer to the data cache (fill operation) takes three cycles. This transfer takes place after all data is read into the fill buffer from the BIU.
- The queuing of data flushed from the data cache (flush operation) takes two cycles. The PPC405x3 can queue up to two flush operations.
- The BIU size (bus width) is 64 bits, so PLBC405DCUSSIZE1 is not shown.
- No data-access errors occur, so PLBC405DCUERR is not shown.
- The abort signal, C405PLBDCUABORT is shown only in the last example.
- The storage attribute signals are not shown.
- The DCU activity is shown only as an aide in describing the examples. The occurrence and duration of this activity is not observable on the DSPLB.

The following abbreviations appear in the timing diagrams:

Table 2-17: **Key to DSPLB Timing Diagram Abbreviations**

Abbreviation[1]	Description	Where Used	
rl#, wl#	Eight-word line read-request or write-request identifier, respectively	Request Request acknowledge Read-data acknowledge Write-data acknowledge	(C405PLBDCUREQUEST) (PLBC405DCUADDRACK) (PLBC405DCURDDACK) (PLBC405DCUWRDACK)
rw#, ww#	Single word read-request or write-request identifier, respectively	Request Request acknowledge Read-data acknowledge Write-data acknowledge	(C405PLBDCUREQUEST) (PLBC405DCUADDRACK) (PLBC405DCURDDACK) (PLBC405DCUWRDACK)
adr#	Data-access request address	Request address	(C405PLBDCUABUS[0:31])
$d\#_{\#}$	A doubleword (eight data bytes) transferred as a result of an eight-word line transfer request	DCU read-data bus DCU write-data bus	(PLBC405DCURDDBUS[0:63]) (PLBC405DCUWRDBUS[0:63])
d#	A word (four data bytes) transferred as a result of a single word transfer request	DCU read-data bus DCU write-data bus	(PLBC405DCURDDBUS[0:63]) (PLBC405DCUWRDBUS[0:63])
val	Byte enables are valid	Byte enables	(C405PLBDCUBE[0:7])
flush#	The DCU is busy performing a flush operation	DCU	
fill#	The DCU is busy performing a fill operation	DCU	
Subscripts	Used to identify the data words transferred between the BIU and DCU	Read-data acknowledge DCU read-data bus Write-data acknowledge DCU write-data bus	(PLBC405DCURDDACK) (PLBC405DCURDDBUS[0:63]) (PLBC405DCUWRDACK) (PLBC405DCUWRDBUS[0:63])
#	Used to identify the order doublewords are sent to the DCU	Transfer order	(PLBC405DCURDWDADDR[1:3])

Notes:
1. "#" indicates a number.

DSPLB Three Consecutive Line Reads

The timing diagram in Figure 2-17 shows three consecutive eight-word line reads that are address-pipelined between the DCU and BIU. It provides an example of the fastest speed at which the DCU can request and receive data over the PLB. All reads are cacheable.

The first line read (rl1) is requested by the DCU in cycle 2. Data is sent from the BIU to the DCU fill buffer in cycles 3 through 6. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill1 transaction in cycles 7 through 9.

The second line read (rl2) is requested by the DCU in cycle 4. The BIU responds to this request after it has completed all transactions associated with the first request (rl1). Data is sent from the BIU to the DCU fill buffer in cycles 7 through 10. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill2 transaction in cycles 11 through 13.

The third line read (rl3) cannot be requested until the first request (rl1) is complete. The earliest this request can occur is in cycle 7. However, the request is delayed to cycle 10 because the DCU is busy transferring the fill buffer to the data cache in cycles 7 through 9 (fill1). The BIU responds to the rl3 request after it has completed all transactions associated with the second request (rl2). Data is sent from the BIU to the DCU fill buffer in cycles 11 through 14. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill3 transaction in cycles 15 through 17.

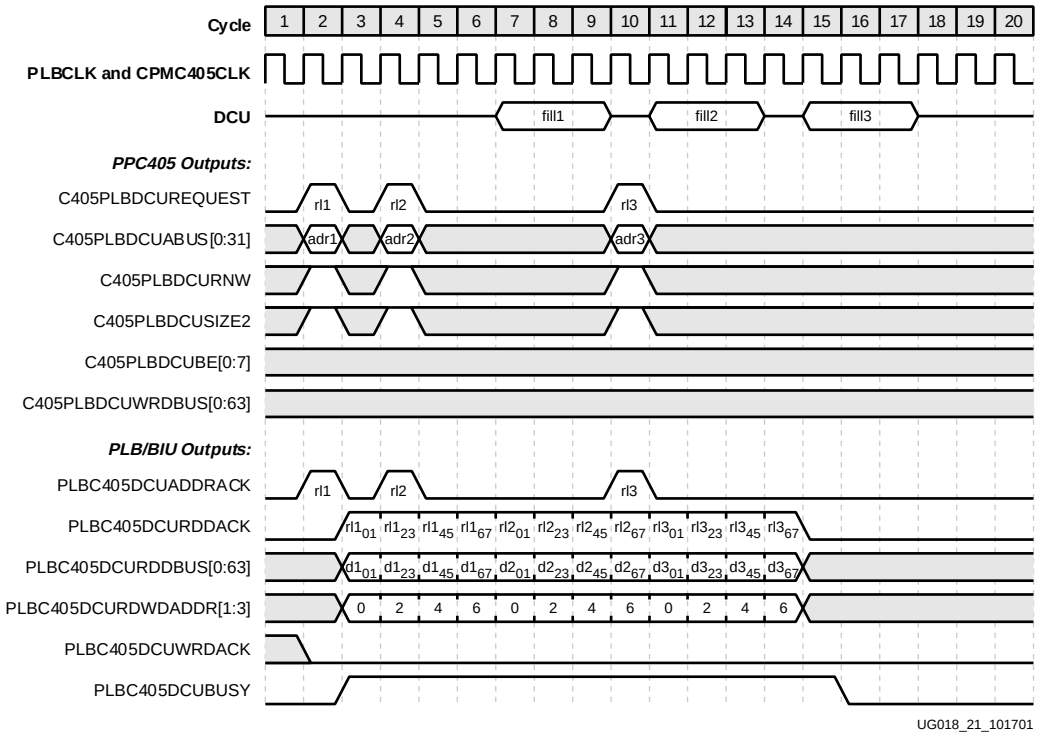

Figure 2-17: **DSPLB Three Consecutive Line Reads**

DSPLB Line Read/Word Read/Line Read

The timing diagram in Figure 2-18 shows a sequence involving an eight-word line read, a word read, and another an eight-word line read. These requests are address-pipelined between the DCU and BIU. The line reads are cacheable and the word read is not cacheable.

The first line read (rl1) is requested by the DCU in cycle 2 and the BIU responds in the same cycle. Data is sent from the BIU to the DCU fill buffer in cycles 3 through 6. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill1 transaction in cycles 7 through 9.

The word read (rw2) is requested by the DCU in cycle 4. The BIU responds to this request after it has completed all transactions associated with the first request (rl1). A single word is sent from the BIU to the DCU fill buffer in cycle 7. The DCU uses the byte enables to select the appropriate bytes from the read-data bus. The data is not cacheable, so the fill buffer is not transferred to the data cache after this transaction is completed.

The third line read (rl3) cannot be requested until the first request (rl1) is complete. The earliest this request can occur is in cycle 7. However, the request is delayed to cycle 10 because the DCU is busy transferring the fill buffer to the data cache in cycles 7 through 9 (fill1). The BIU can respond immediately to the rl3 request because all transactions associated with the second request (rw2) are complete. Data is sent from the BIU to the DCU fill buffer in cycles 11 through 14. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill3 transaction in cycles 15 through 17.

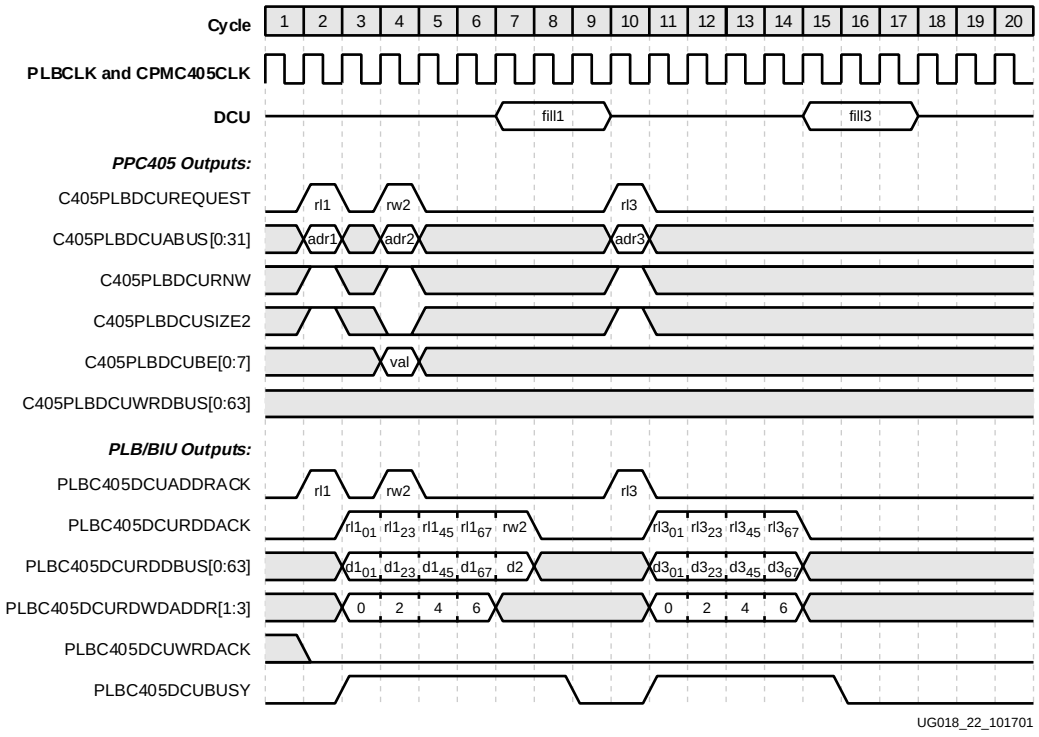

Figure 2-18: **DSPLB Line Read/Word Read/Line Read**

DSPLB Three Consecutive Word Reads

The timing diagram in Figure 2-19 shows three consecutive word reads. The word reads could be in response to non-cacheable loads or cacheable loads that do not allocate a cache line.

Figure 2-19 provides an example of the fastest speed at which the PPC405x3 DCU can request and receive single words over the PLB. The DCU is designed to wait for the current single-word read request to be satisfied before making a subsequent request. This requirement results in the delay between requests shown in the figure. It is possible for other PLB masters to request and receive single words at a faster rate than shown in this example.

The first word read (rw1) is requested by the DCU in cycle 2 and the BIU responds in the same cycle. A single word is sent from the BIU to the DCU in cycle 3. The DCU uses the byte enables to select the appropriate bytes from the read-data bus.

The second word read (rw2) is requested by the DCU in cycle 7 and the BIU responds in the same cycle. A single word is sent from the BIU to the DCU in cycle 8. The DCU uses the byte enables to select the appropriate bytes from the read-data bus.

The third word read (rw3) is requested by the DCU in cycle 12 and the BIU responds in the same cycle. A single word is sent from the BIU to the DCU in cycle 13. The DCU uses the byte enables to select the appropriate bytes from the read-data bus.

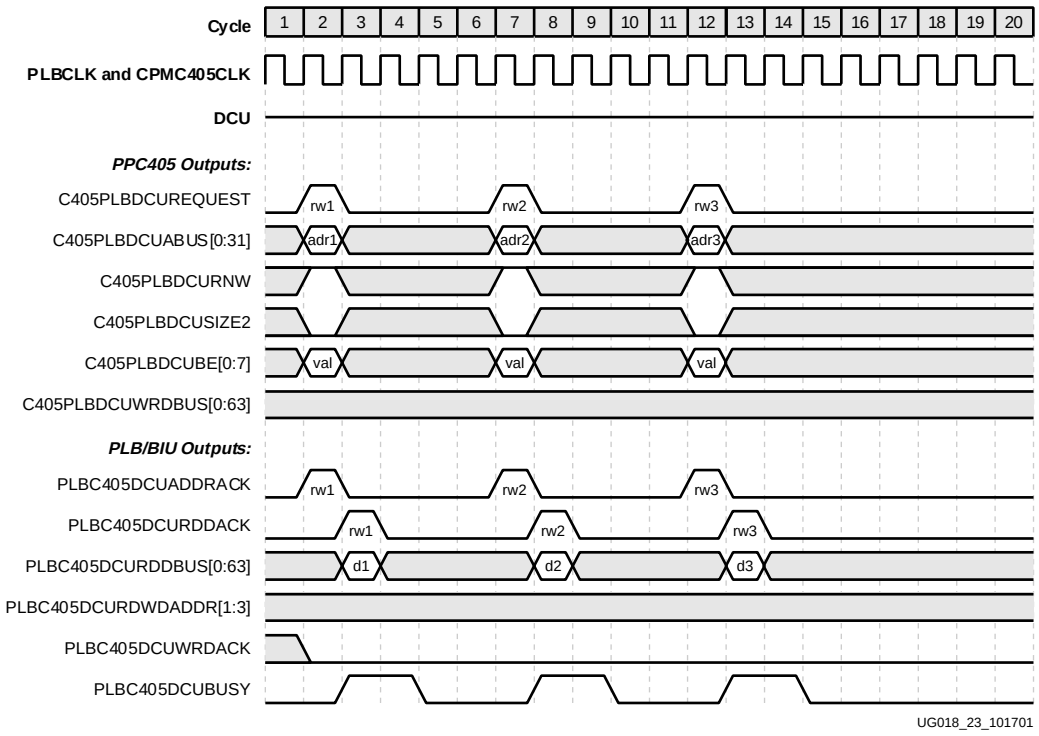

Figure 2-19: **DSPLB Three Consecutive Word Reads**

DSPLB Three Consecutive Line Writes

The timing diagram in Figure 2-20 shows three consecutive eight-word line writes. It provides an example of the fastest speed at which the DCU can request and send data over the PLB. All writes are cacheable. Consecutive writes cannot be address pipelined between the DCU and BIU.

The first line write (wl1) is requested by the DCU in cycle 3 in response to a cache flush (represented by the flush1 transaction in cycles 1 through 2). The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 3 through 6.

The second line write (wl2) cannot be started until the first request is complete. This request is made by the DCU in cycle 8 in response to the cache flush in cycles 3 through 4 (flush2). The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 8 through 11.

The DCU can queue two outstanding data-cache flush requests. In this example, a third flush request cannot be queued until the first is complete. The third flush request (flush3) is queued in cycles 8 and 9.

The third line write (wl3) cannot be started until the second request (wl2) is complete. This request is made by the DCU in cycle 13 in response to the flush3 request. The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 13 through 16.

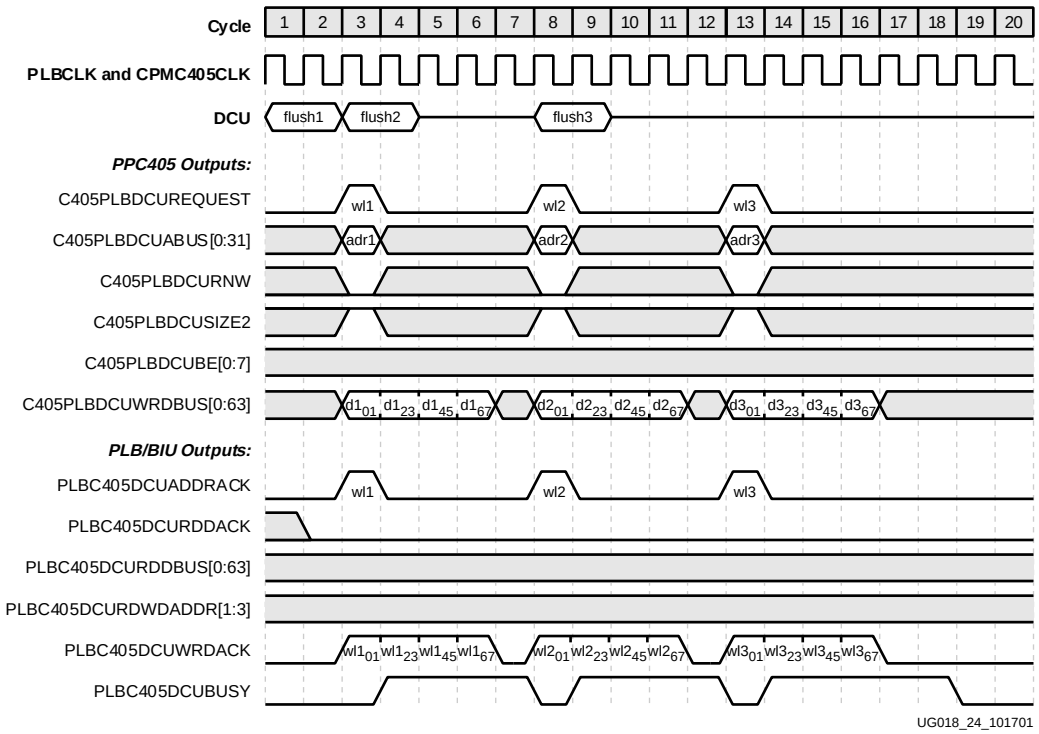

Figure 2-20: **DSPLB Three Consecutive Line Writes**

DSPLB Line Write/Word Write/Line Write

The timing diagram in Figure 2-21 shows a sequence involving an eight-word line write, a word write, and another an eight-word line write. Consecutive writes cannot be address pipelined between the DCU and BIU. The line writes are cacheable. The word writes could be in response to non-cacheable stores, cacheable stores to write-through memory, or cacheable stores that do not allocate a cache line.

The first line write (wl1) is requested by the DCU in cycle 3 in response to a cache flush (represented by the flush1 transaction in cycles 1 through 2). The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 3 through 6.

The word write (ww2) cannot be started until the first request is complete. This request is made by the DCU in cycle 8 and the BIU responds in the same cycle. A single word is sent from the DCU to the BIU in cycle 8. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

The DCU queues the second flush request, flush3. The second line write (wl3) cannot be started until the second request (ww2) is complete. This request is made by the DCU in cycle 10 in response to the flush3 request. The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 10 through 13.

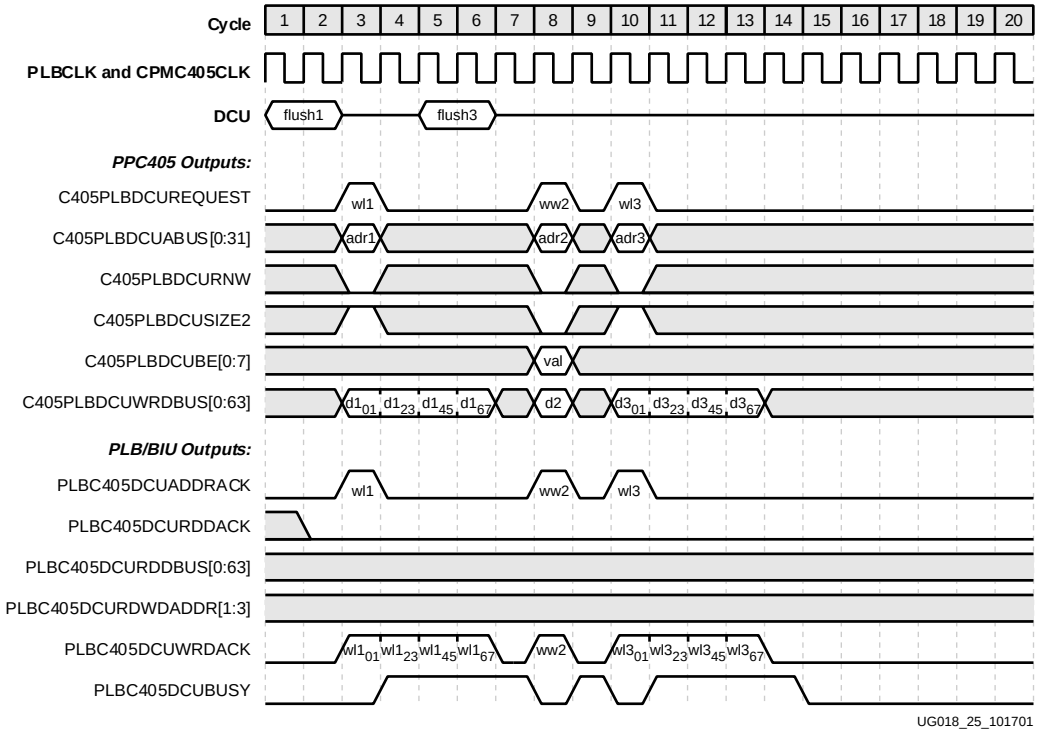

Figure 2-21: **DSPLB Line Write/Word Write/Line Write**

DSPLB Three Consecutive Word Writes

The timing diagram in Figure 2-22 shows three consecutive word writes. It provides an example of the fastest speed at which the DCU can request and send single words over the PLB. The word writes could be in response to non-cacheable stores, cacheable stores to write-through memory, or cacheable stores that do not allocate a cache line. Consecutive writes cannot be address pipelined between the DCU and BIU.

The first word write (ww1) is requested by the DCU in cycle 2. The BIU responds in the same cycle the request is made by the DCU. A single word is sent from the DCU to the BIU in cycle 2. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

The second word write (ww2) is requested after the first write is complete. The DCU makes the request in cycle 4 and the BIU responds in the same cycle. A single word is sent from the DCU to the BIU in cycle 4. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

The third word write (ww3) is requested after the second write is complete. The DCU makes the request in cycle 6 and the BIU responds in the same cycle. A single word is sent from the DCU to the BIU in cycle 6. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

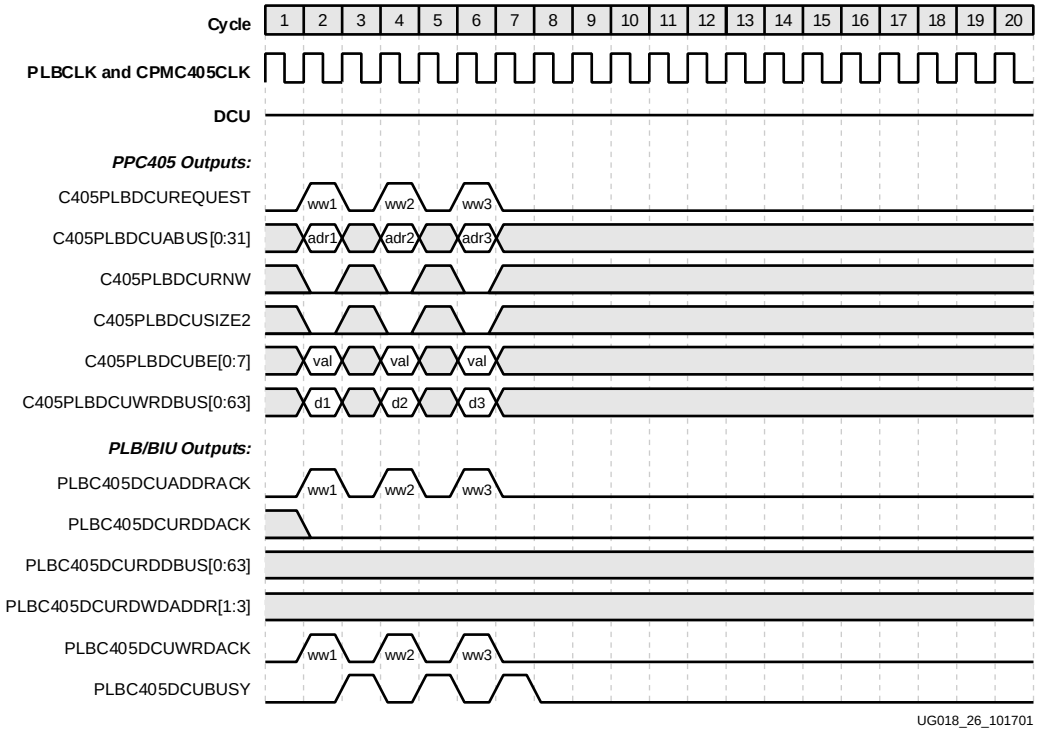

Figure 2-22: **DSPLB Three Consecutive Word Writes**

DSPLB Line Write/Line Read/Word Write

The timing diagram in Figure 2-23 shows a sequence involving an eight-word line write, an eight-word line read, and a word write. It provides an example of address pipelining involving writes and reads. It also demonstrates how read and write operations can overlap due to the split read-data and write-data busses.

The first line write (wl1) is requested by the DCU in cycle 3 in response to a cache flush (represented by the flush1 transaction in cycles 1 through 2). The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 3 through 6.

The first line read (rl2) is address pipelined with the previous line write. The rl2 request is made by the DCU in cycle 5 and the BIU responds in the same cycle. Data is sent from the BIU to the DCU fill buffer in cycles 6 through 9. Because of the split data bus, a read operation overlaps with a previous write operation in cycle 6. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill2 transaction in cycles 10 through 12.

The word write (ww3) cannot be requested until the first write request (wl1) is complete because address pipelining of multiple write requests is not supported. However, this request is address pipelined with the previous line read request (rl2). The ww3 request is made by the DCU in cycle 8 and the BIU responds in the same cycle. A single word is sent from the DCU to the BIU in cycle 8. The BIU uses the byte enables to select the appropriate bytes from the write-data bus. Because of the split data bus, this write operation overlaps with a read operation from the previous read request (rl2).

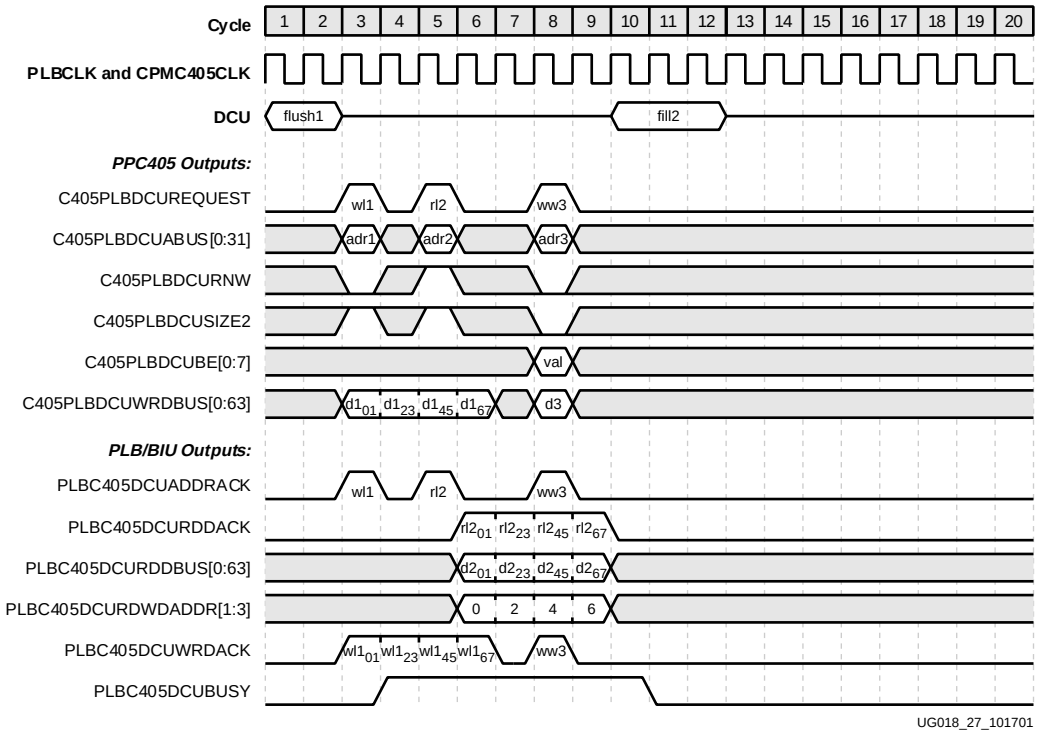

Figure 2-23: **DSPLB Line Write/Line Read/Word Write**

DSPLB Word Write/Word Read/Word Write/Line Read

The timing diagram in Figure 2-24 shows a sequence involving a word write, a word read, another word write, and an eight-word line read.

The first word write (ww1) is requested by the DCU in cycle 2 and the BIU responds in the same cycle. A single word is sent from the DCU to the BIU in cycle 2. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

The first word read (rw2) is requested by the DCU in cycle 4. Even though the previous request is completed in cycle 2, this is the earliest an address pipelined request can be started by the DCU. The BIU responds in the same cycle the rw2 request is made by the DCU. A single word is sent from the BIU to the DCU in cycle 5. The DCU uses the byte enables to select the appropriate bytes from the write-data bus.

The second word write (ww3) is requested by the DCU in cycle 6. Again, this is the earliest an address pipelined request can be started by the DCU. The BIU responds in the same cycle the ww3 request is made by the DCU. A single word is sent from the DCU to the BIU in cycle 6. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

The line read (rl4) is address pipelined with the word write. The rl4 request is made by the DCU in cycle 8 and the BIU responds in the same cycle. Data is sent from the BIU to the DCU fill buffer in cycles 9 through 12. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill4 transaction in cycles 13 through 15.

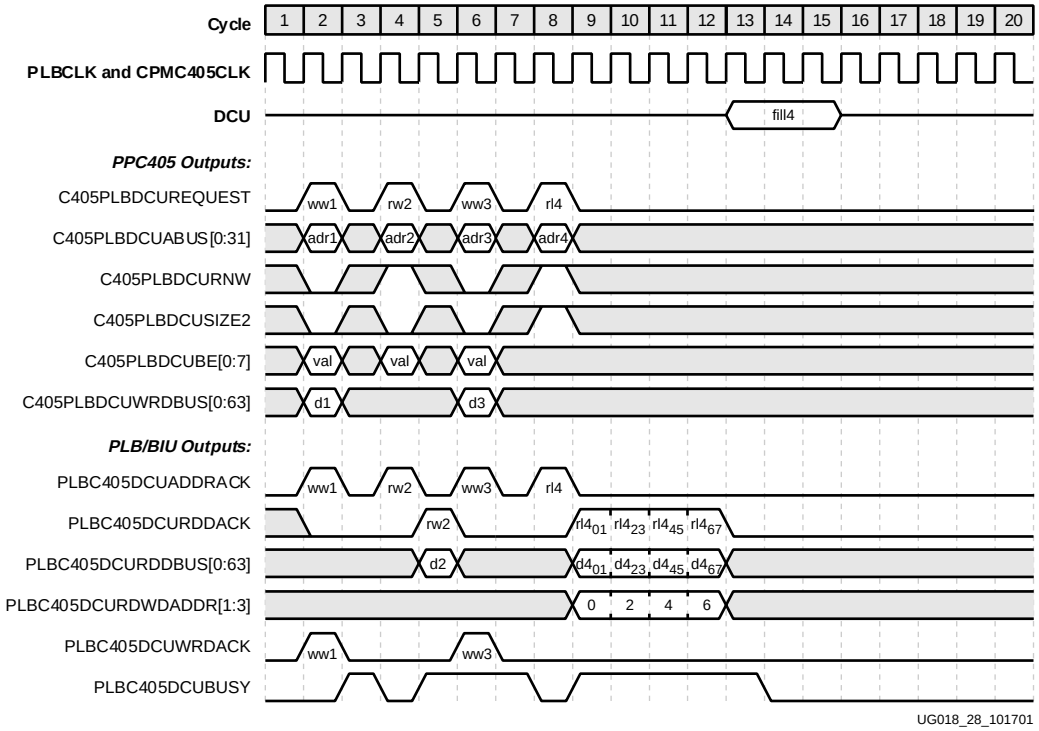

Figure 2-24: **DSPLB Word Write/Word Read/Word Write/Line Read**

DSPLB Word Write/Line Read/Line Write

The timing diagram in Figure 2-25 shows a sequence involving a word write, an eight-word line read, and an eight-word line write. It demonstrates how read and write operations can overlap due to the split read-data and write-data busses.

The word write (ww1) is requested by the DCU in cycle 2 and the BIU responds in the same cycle. A single word is sent from the DCU to the BIU in cycle 2. The BIU uses the byte enables to select the appropriate bytes from the write-data bus.

The line read (rl2) is address pipelined with the previous word write. The rl2 request is made by the DCU in cycle 4 and the BIU responds in the same cycle. Data is sent from the BIU to the DCU fill buffer in cycles 5 through 8. After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill2 transaction in cycles 9 through 11.

The line write (wl3) is address pipelined with the previous line read. The wl3 request is made by the DCU in cycle 6 in response to the cache flush in cycles 4 through 5 (flush3). The BIU responds to the wl3 request in the same cycle it is asserted by the DCU. Data is sent from the DCU to the BIU in cycles 6 through 9. Because of the split data bus, the write operations in cycles 6 through 8 overlap read operations from the previous read request (rl2).

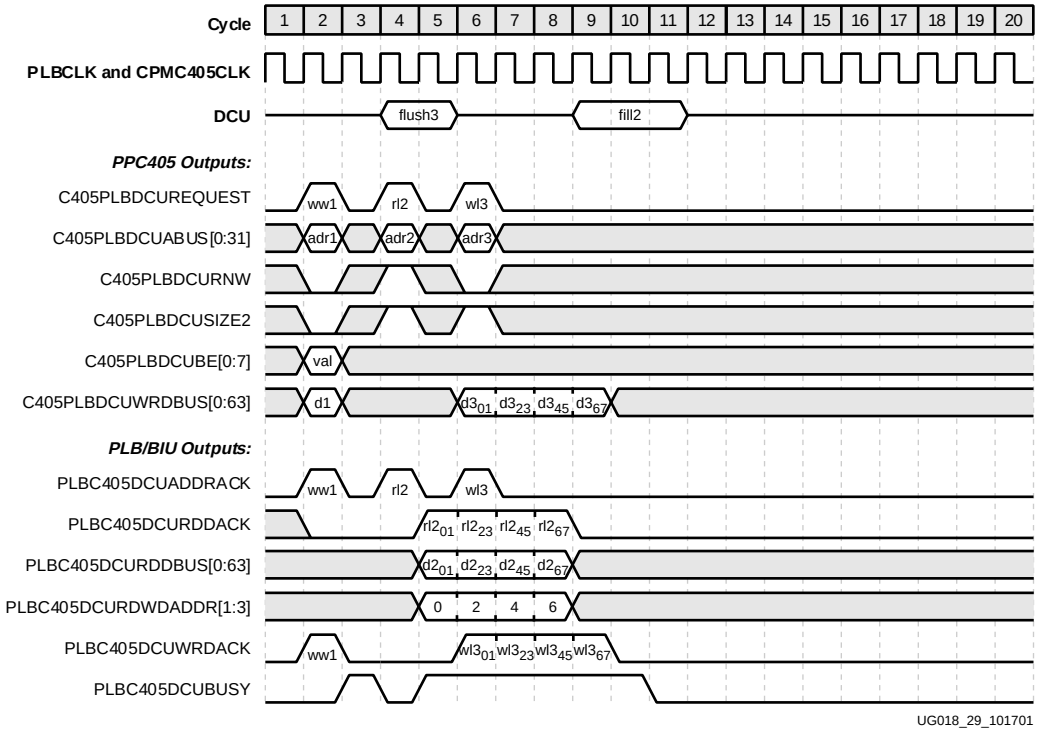

Figure 2-25: **DSPLB Word Write/Line Read/Line Write**

DSPLB 2:1 Core-to-PLB Line Read

The timing diagram in Figure 2-26 shows a line read in a system with a PLB clock that runs at one half the frequency of the PPC405x3 clock.

The line read (rl1) is requested by the DCU in PLB cycle 2, which corresponds to PPC405x3 cycle 3. The BIU responds in the same cycle. Data is sent from the BIU to the DCU fill buffer in PLB cycles 3 through 6 (PPC405x3 cycles 5 through 12). After all data associated with this line is read, it is transferred by the DCU from the fill buffer to the data cache. This is represented by the fill1 transaction in PPC405x3 cycles 13 through 15.

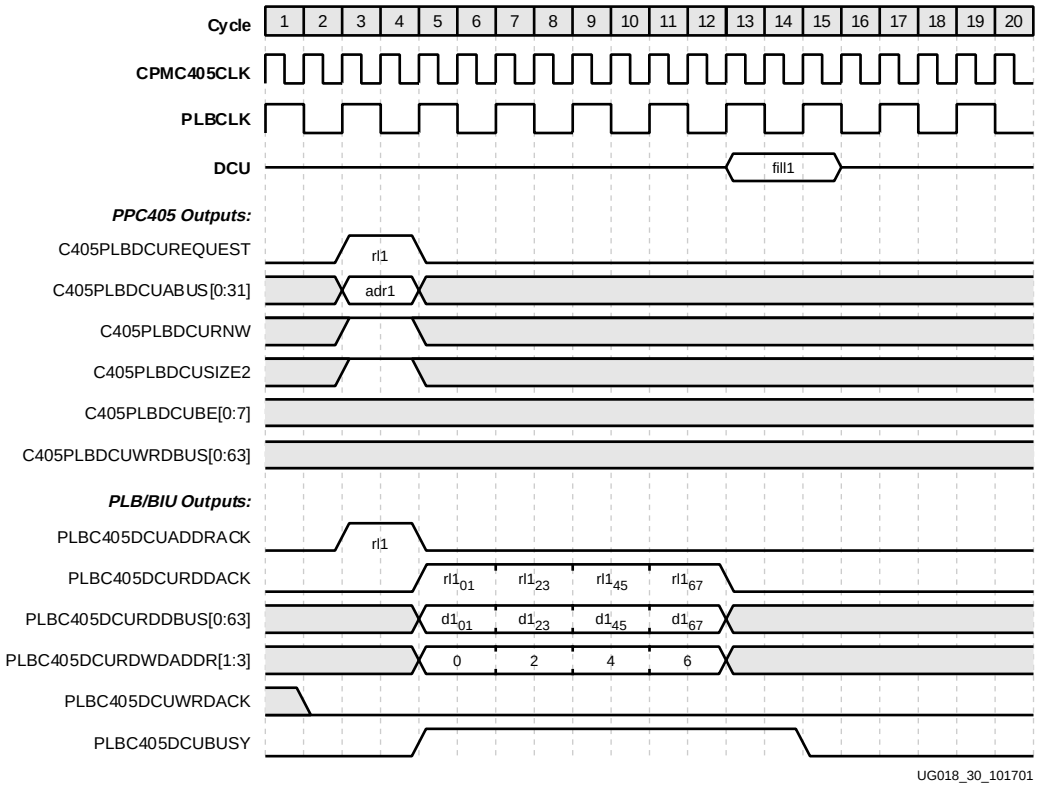

Figure 2-26: **DSPLB 2:1 Core-to-PLB Line Read**

DSPLB 3:1 Core-to-PLB Line Write

The timing diagram in Figure 2-27 shows a line write in a system with a PLB clock that runs at one third the frequency of the PPC405x3 clock.

The line write (wl1) is requested by the DCU in PLB cycle 2, which corresponds to PPC405x3 cycle 4. The BIU responds in the same cycle. The request is made in response to a flush in PPC405x3 cycles 1 and 2 (flush1). Data is sent from the DCU to the BIU in PLB cycles 2 through 5 (PPC405x3 cycles 4 through 15).

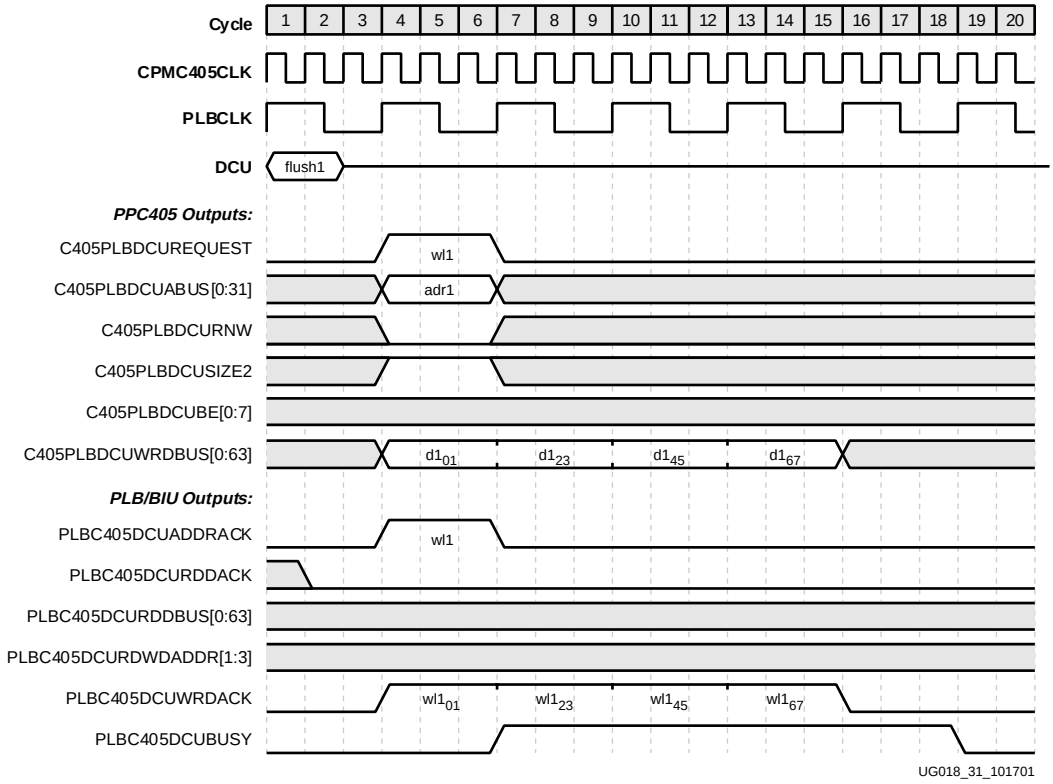

Figure 2-27: **DSPLB 3:1 Core-to-PLB Line Write**

DSPLB Aborted Data-Access Request

The timing diagram in Figure 2-28 shows an aborted data-access request. The request is aborted because of a core reset. The BIU is not reset.

A line write (wl1) is requested by the DCU in cycle 3 in response to a cache flush (represented by the flush1 transaction in cycles 1 through 2). The BIU responds in the same cycle the request is made by the DCU. Data is sent from the DCU to the BIU in cycles 3 through 6.

A line read (rl2) is address pipelined with the previous line write. The rl2 request is made by the DCU in cycle 5 and the BIU responds in the same cycle. However, the processor also aborts the request in cycle 5. Therefore, no data is transferred from the BIU to the DCU in response to this request.

Because the BIU is not reset, it must complete the first line write even though the processor asserts the PLB abort signal during the line write.

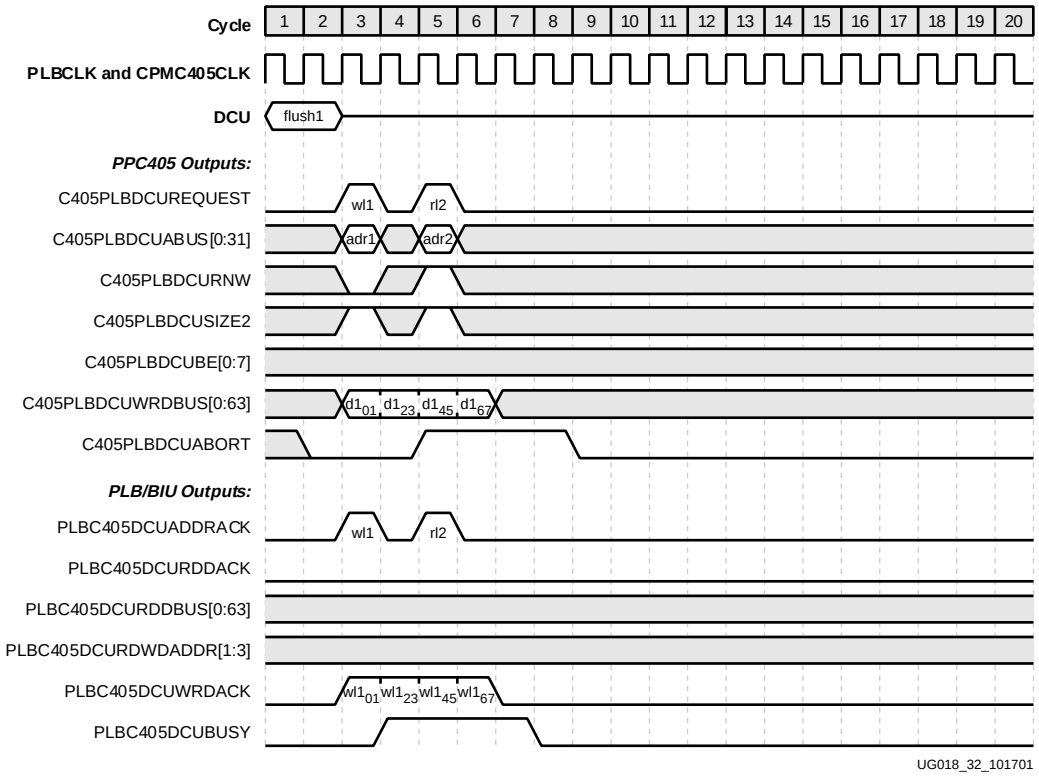

Figure 2-28: **DSPLB Aborted Data-Access Request**

Device-Control Register Interface

The device-control register (DCR) interface provides a mechanism for the processor block to initialize and control peripheral devices that reside on the same FPGA chip. For example, the memory-transfer characteristics and address assignments for a bus-interface unit (BIU) can be configured by software using DCRs. The DCRs are accessed using the PowerPC **mfdcr** and **mtdcr** instructions.

The DCR interface consists of the following:

- A 10-bit address bus.
- Separate 32-bit input data and output data busses.
- Separate read and write control signals.
- A read / write acknowledgement signal.

Because the processor block is the only bus master on the bus, the address bus is driven by the processor block and received by each peripheral containing DCRs. The read and write control signals are also distributed to each DCR peripheral.

The preferred implementation of the DCR data bus is as a distributed, multiplexed chain. Each peripheral in the chain has a DCR input-data bus connected to the DCR output-data bus of the previous peripheral in the chain (the first peripheral is attached to the processor block). Each peripheral multiplexes this bus with the outputs of its DCRs and passes the resulting DCR bus as an output to the next peripheral in the chain. The last peripheral in the chain has its DCR output-data bus attached to the processor block DCR input-data interface. This implementation enables future DCR expansion without requiring changes to I/O devices due to additional loading.

There are two options for connecting the acknowledge signals. The acknowledge signals from the DCRs can be latched and forwarded in the chain with the DCR data bus. Alternatively, combinatorial logic, such as OR gates, can be used to combine and forward the acknowledge signal to the processor block.

Figure 2-29, page 97 shows an example DCR chain implementation in an FPGA chip. The acknowledge signal in this example is formed using combinatorial logic (OR gate).

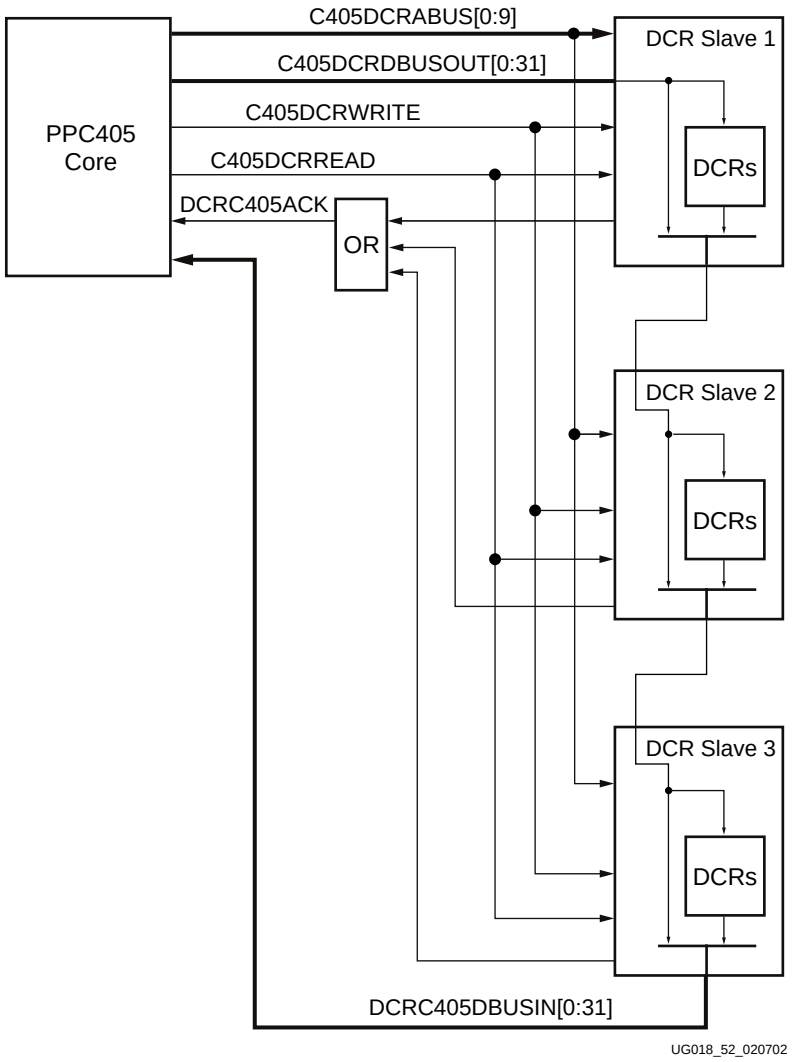

Figure 2-29: **DCR Chain Block Diagram**

The acknowledge signal is interlocked with the read and write control signals. The interlock mechanism enables the DCR interface to communicate with peripheral devices that are clocked at different frequencies than the PPC405x3. A DCR access takes at least three PPC405x3 cycles. If a DCR access is not acknowledged in 64 processor cycles, the access times out. No error occurs when a DCR access times-out. Instead, the processor begins executing the next-sequential instruction.

The interlock mechanism requires that the rising edge of the slower clock (either the PPC405x3 clock or the peripheral clock) correspond to the rising edge of the faster clock. This means that the clocks for the DCR logic and the clocks for the PPC405x3 must be derived from a common source. The common source frequency is multiplied or divided before being sent to the PPC405x3 or DCR logic.

The DCR interface operates in two ways, referred to as *mode 0* and *mode 1* (these are hard wired modes, not programmable modes):

- In mode 0, the PPC405x3 and FPGA can be clocked at different frequencies without affecting the interface handshaking protocol. In this mode, an acknowledgement follows a read or write operation. The acknowledgement cannot be deasserted until the read or write signal is deasserted.
- In mode 1, the core and FPGA must run at the same frequency. In this mode, an acknowledgement follows a read or write operation. However, the acknowledgement can be deasserted one cycle after it is asserted. This enables the fastest back-to-back DCR access (three cycles).

Figure 2-30 illustrates a logical implementation of the DCR bus interface. This implementation enables a DCR slave to run at a different clock speed than the PPC405x3. The acknowledge signal is latched and forwarded with the DCR bus. The bypass multiplexor minimizes data-bus path delays when the DCR is not selected. To ensure reusability across multiple FPGA environments, all DCR slave logic should use the specified implementation.

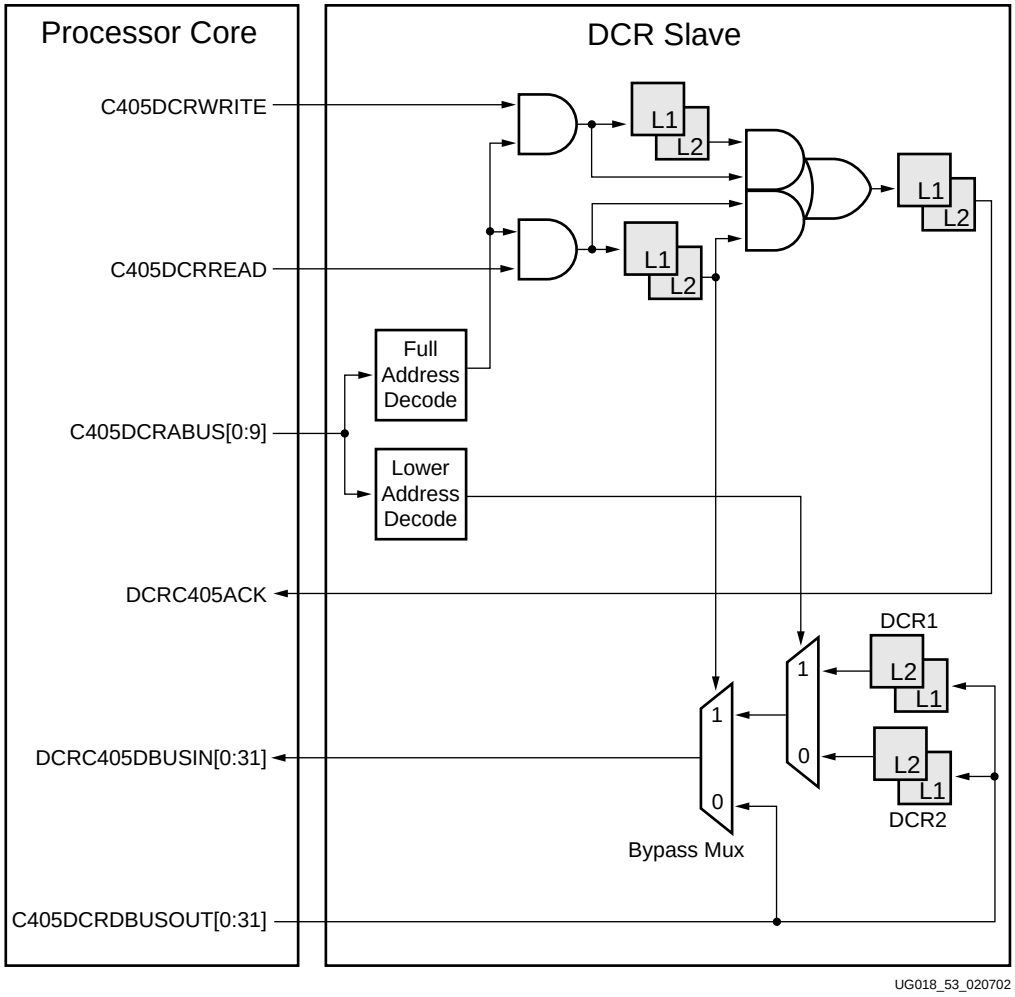

Figure 2-30: **DCR Bus Implementation**

DCR Interface I/O Signal Summary

Figure 2-31 shows the block symbol for the DCR interface. The signals are summarized in Table 2-18.

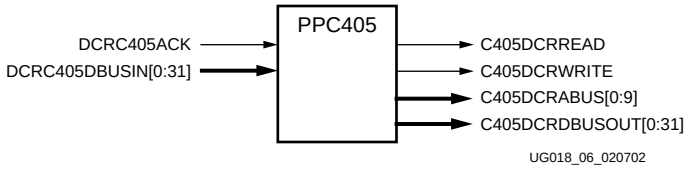

Figure 2-31: **DCR Interface Block Symbol**

Table 2-18: **DCR Interface I/O Signals**

Signal	I/O Type	If Unused	Function
C405DCRREAD	O	No Connect	Indicates a DCR read request occurred.
C405DCRWRITE	O	No Connect	Indicates a DCR write request occurred.
C405DCRABUS[0:9]	O	No Connect	Specifies the address of the DCR access request.
C405DCRDBUSOUT[0:31]	O	No Connect or attach to input bus	The 32-bit DCR write-data bus.
DCRC405ACK	I	0	Indicates a DCR access has been completed by a peripheral.
DCRC405DBUSIN[0:31]	I	0x0000_0000 or attach to output bus	The 32-bit DCR read-data bus.

DCR Interface I/O Signal Descriptions

The following sections describe the operation of the DCR interface I/O signals.

C405DCRREAD (output)

When asserted, this signal indicates the processor block is requesting the contents of a DCR (reading from the DCR) in response to the execution of a *move-from DCR* instruction (**mfdcr**). The contents of the DCR address bus are valid when this request is asserted (the request is asserted one processor cycle after the processor block begins driving the DCR address bus). This signal is deasserted two processor cycles after the DCR acknowledge signal is asserted. When deasserted, no DCR read request exists. DCRs must not drive the DCR bus if this signal is not asserted.

The processor block waits up to 64 cycles for a read request to be acknowledged. If a DCR does not acknowledge the request in this time, the read times out. No error occurs when a DCR read times-out. Instead, the processor begins executing the next-sequential instruction.

DCR read requests are not interrupted by the processor block. If this signal is asserted, only a DCR acknowledgement or read time-out cause the signal to be deasserted.

This signal is deasserted during reset.

C405DCRWRITE (output)

When asserted, this signal indicates the processor block is requesting that the contents of a DCR be updated (writing to the DCR) in response to the execution of a *move-to DCR* instruction (**mtdcr**). The contents of the DCR address bus are valid when this request is

asserted (the request is asserted one processor cycle after the processor block begins driving the DCR address bus). This signal is deasserted two processor cycles after the DCR acknowledge signal is asserted. When deasserted, no DCR write request exists.

The processor block waits up to 64 cycles for a write request to be acknowledged. If a DCR does not acknowledge the request in this time, the write times out. No error occurs when a DCR write times-out. Instead, the processor begins executing the next-sequential instruction.

DCR write requests are not interrupted by the processor block. If this signal is asserted, only a DCR acknowledgement or write time-out cause the signal to be deasserted.

This signal is deasserted during reset.

C405DCRABUS[0:9] (output)

This bus specifies the address of the DCR access request. This bus remains stable during the execution of a **mfdcr** or **mtdcr** instruction. However, the contents of this bus are valid only when either a DCR read request or DCR write request are asserted by the processor. The processor does not begin driving a new DCR address until the DCR acknowledge signal corresponding to the previous DCR access has been deasserted for at least one cycle.

The address driven by this bus corresponds to the DCR number (DCRN) and not the split DCR field (DCRF) encoded in the **mfdcr** or **mtdcr** instruction. For example, if the DCRN is 0x2AA, the DCR address bus is driven with the value 0x2AA. However, the DCRF encoded by the DCR instruction is 0x155. See the *PowerPC Processor Reference Guide* for more information on these instructions.

C405DCRDBUSOUT[0:31] (output)

This write-data bus is driven by the processor block when a **mtdcr** or **mfdcr** instruction is executed. Its contents are valid only when a DCR write-request or DCR read-request is asserted. When a **mtdcr** instruction is executed, this bus contains the data to be written into a DCR. When a **mfdcr** instruction is executed, this bus contains the value 0x0000_0000.

During reset, this bus is driven with the value 0x0000_0000. Peripherals can use this value to initialize the DCRs.

DCRC405ACK (input)

When asserted, this signal indicates a peripheral device acknowledges the processor block request for DCR access. For a read-access request, the peripheral device should assert this signal when the DCR read-data bus is driven with the appropriate DCR contents (the bus contains valid data). For a write-access request, the peripheral device should assert this signal when the DCR write-data bus is latched into the appropriate DCR. Peripheral devices should assert this signal only when all of the following are true:

- They contain the accessed DCR.
- A valid read-access or write-access DCR request exists.
- The peripheral device has driven the DCR bus (read access) or latched the DCR bus (write access).

Deasserting the acknowledge signal after it has been asserted depends on the DCR interface mode (these are hard wired modes, not programmable modes):

- In mode 0, the acknowledgement cannot be deasserted until the read or write signal is deasserted. This enables the PPC405x3 and FPGA to be clocked at different frequencies without affecting the interface handshaking protocol.
- In mode 1, the acknowledge signal should be deasserted in the cycle after it is asserted

(the peripheral device and the PPC405x3 are clocked at the same frequency). It is not necessary to wait for the read-access or write-access signal to be deasserted. This enables the fastest back-to-back DCR access (three cycles).

DCRC405DBUSIN[0:31] (input)

This read-data bus is latched (read) by the processor block when a peripheral device asserts the DCR acknowledge signal in response to a DCR read-access request. A peripheral device must drive this bus only when it contains the accessed DCR and the DCR read-access signal is asserted by the processor block.

Peripheral devices should drive only the bits implemented by the specified DCR. A value of 0x0000_0000 is driven onto the DCR write-data bus by the processor block during a read-access request. This value is passed along the DCR chain until modified by the appropriate peripheral. The end of the DCR chain is attached to the DCR read-data bus input to the processor block. Thus, the processor reads the updated value of all implemented bits, and unimplemented (and unattached) bits retain a value of 0.

DCR Interface Timing Diagrams

The following timing diagrams show typical transfers that can occur on the DCR interface using the two interface modes. Unless otherwise noted, optimal timing relationships are used to improve the readability of the timing diagrams. The assertion of C405DCRREAD / C405DCRWRITE refers to a read or write operation, not both. The processor block cannot perform a simultaneous read and write of the DCR bus.

DCR Interface Mode 0, 1:1 Clocking, Latched Acknowledge

The example in Figure 2-32 assumes the following:

- The PPC405x3 and the peripheral containing the DCR are clocked at the same frequency.
- The acknowledge signal is latched and forwarded with the DCR bus as shown in Figure 2-30, page 98.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted (mode 0 interface operation).

Using these assumptions, the fastest back-to-back DCR access occurs every seven cycles. The implementation of the acknowledge signal causes it to be asserted two cycles after the access request. It is deasserted in the cycle after the access request is deasserted.

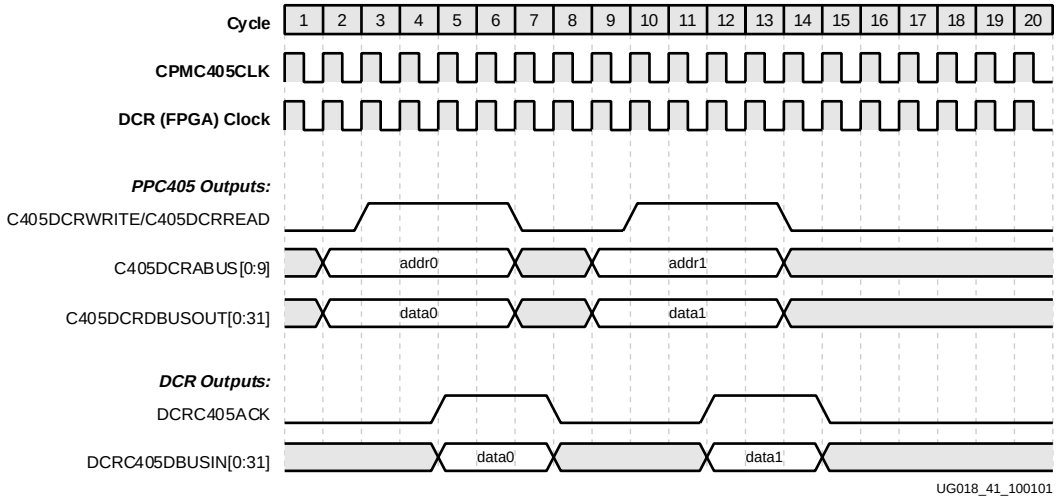

Figure 2-32: **DCR Interface Mode 0, 1:1 Clocking, Latched Acknowledge**

DCR Interface Mode 0, 1:1 Clocking, Combinatorial Acknowledge

The example in Figure 2-33 assumes the following:

- The PPC405x3 and the peripheral containing the DCR are clocked at the same frequency.
- The acknowledge signal is generated by combinatorial logic using the acknowledge signal from each peripheral, as shown in Figure 2-29, page 97.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted (mode 0 interface operation).

Using these assumptions, the fastest back-to-back DCR access occurs every four cycles. The implementation of the acknowledge signal causes it to be asserted in the same cycle as the access request. It is assumed that the selected DCR can latch / drive the DCR bus in the same cycle. The combinatorial logic enables the acknowledge signal to be deasserted in the same cycle that the access request is deasserted.

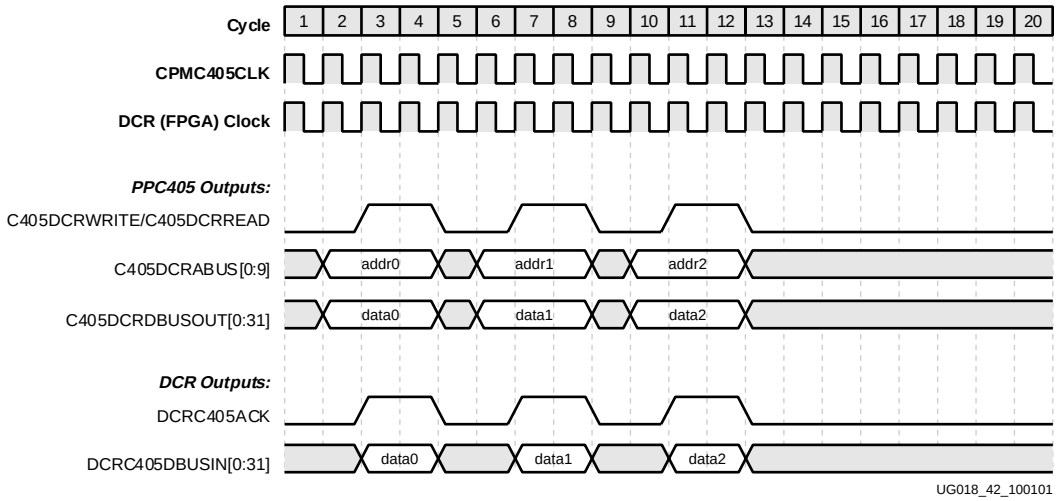

Figure 2-33: **DCR Interface Mode 0, 1:1 Clocking, Combinatorial Acknowledge**

DCR Interface Mode 0, 2:1 Clocking, Latched Acknowledge

The example in Figure 2-34 assumes the following:

- The PPC405x3 is clocked at twice the frequency of the peripheral containing the DCR.
- The acknowledge signal is latched and forwarded with the DCR bus as shown in Figure 2-30, page 98.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted (mode 0 interface operation).

Using these assumptions, the fastest back-to-back DCR access occurs every ten PPC405x3 cycles. The implementation of the acknowledge signal causes it to be asserted two DCR cycles (four PPC405x3 cycles) after the access request. It is deasserted in the DCR cycle after the access request is deasserted.

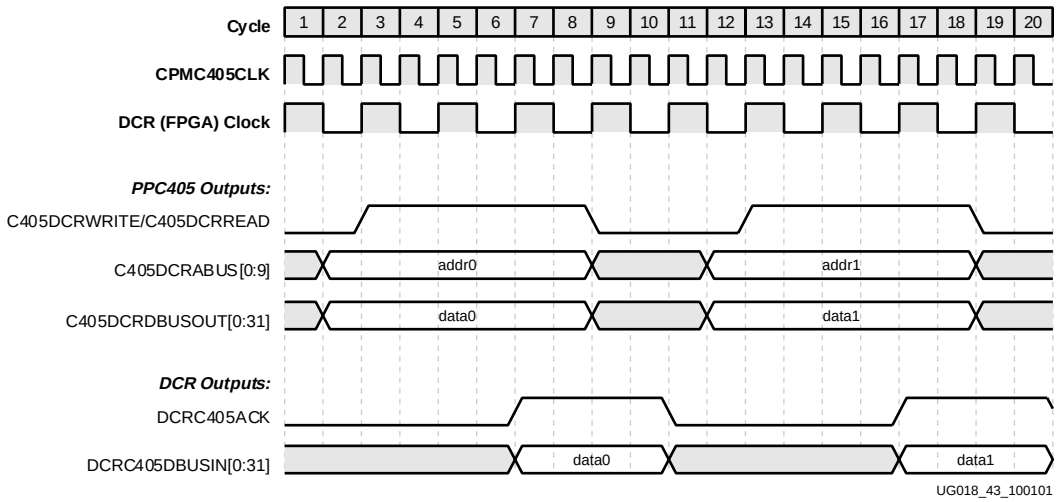

Figure 2-34: **DCR Interface Mode 0, 2:1 Clocking, Latched Acknowledge**

DCR Interface Mode 0, 1:2 Clocking, Latched Acknowledge

The example in Figure 2-35 assumes the following:

- The PPC405x3 is clocked at half the frequency of the peripheral containing the DCR.
- The acknowledge signal is latched and forwarded with the DCR bus as shown in Figure 2-30, page 98.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted (mode 0 interface operation).

Using these assumptions, the fastest back-to-back DCR access occurs every six PPC405x3 (twelve DCR) cycles. The implementation of the acknowledge signal causes it to be asserted two DCR cycles (one PPC405x3 cycles) after the access request. It is deasserted in the DCR cycle after the access request is deasserted.

The processor block does not present a new DCR address until at least one cycle after the acknowledge is deasserted. In this example, the PPC405x3 and DCR clocks are not interlocked (the rising edges do not coincide) when the acknowledge is deasserted. The clocks interlock one-half a PPC405x3 cycle later, but the processor block must wait an entire additional cycle before it can present a new DCR address.

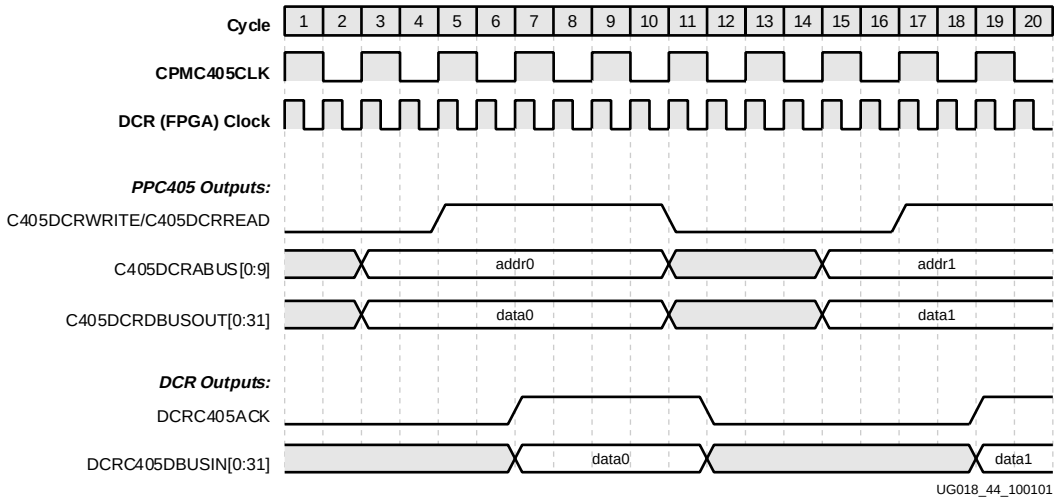

Figure 2-35: **DCR Interface Mode 0, 1:2 Clocking, Latched Acknowledge**

DCR Interface Mode 1, 1:1 Clocking, Combinatorial Acknowledge

The example in Figure 2-36 assumes the following:

- The PPC405x3 and the peripheral containing the DCR are clocked at the same frequency.
- The acknowledge signal is generated by combinatorial logic using the acknowledge signal from each peripheral, as shown in Figure 2-29, page 97.
- The acknowledge signal is deasserted in the cycle after it is asserted (mode 1 interface operation).

Using these assumptions, the fastest back-to-back DCR access occurs every three cycles. This represents the fastest speed at which the processor can present successive DCR requests on the DCR interface.

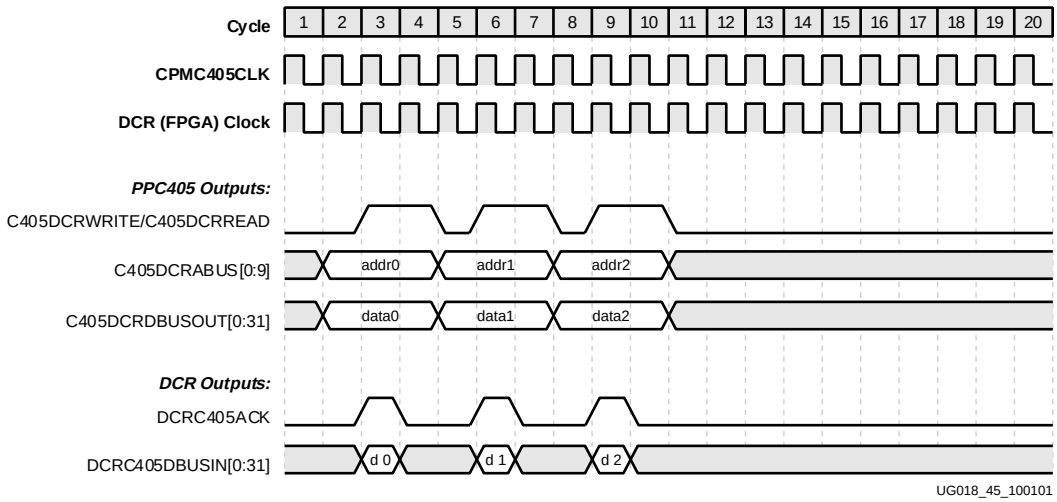

Figure 2-36: **DCR Interface Mode 1, 1:1 Clocking, Combinatorial Acknowledge**

External Interrupt Controller Interface

The PowerPC embedded-environment architecture defines two classes of interrupts: critical and noncritical. The interrupt handler for an external critical interrupt is located at exception-vector offset 0x0100. The interrupt handler for an external noncritical interrupt is located at exception-vector offset 0x0200. Generally, the processor prioritizes critical interrupts ahead of noncritical interrupts when they occur simultaneously (certain debug exceptions are handled at a lower priority). Critical interrupts use a different save/restore register pair (SRR2 and SRR3) than is used by noncritical interrupts (SRR0 and SRR1). This enables a critical interrupt to interrupt a noncritical-interrupt handler. The state saved by the noncritical interrupt is not overwritten by the critical interrupt. See the *PowerPC Processor Reference Guide* for more information on exception and interrupt processing.

Logic external to the processor block can be used to cause critical and noncritical interrupts. External interrupt sources are collected by the external interrupt controller (EIC) and presented to the processor block as either a critical or noncritical interrupt. Once an external interrupt request is asserted, the EIC must keep the signal asserted until software deasserts it. This is typically done by writing to a DCR in the EIC peripheral logic.

Software can enable and disable external interrupts using the following bits in the machine-state register MSR:

- Noncritical interrupts are controlled by MSR[EE]. When set to 1, noncritical interrupts are enabled. When cleared to 0, they are disabled.
- Critical interrupts are controlled by MSR[CE]. When set to 1, critical interrupts are enabled. When cleared to 0, they are disabled.

The states of the EE and CE bits are reflected by output signals on the processor block CPM interface. See **Clock and Power Management Interface**, page 33, for more information.

An external interrupt is considered pending if it occurs while the corresponding class is disabled. The EIC continues to assert the interrupt request. When software later enables the interrupt class, the interrupt occurs and the interrupt handler deasserts the request by writing to a DCR in the EIC.

EIC Interface I/O Signal Summary

Figure 2-37 shows the block symbol for the EIC interface. The signals are summarized in Table 2-19.

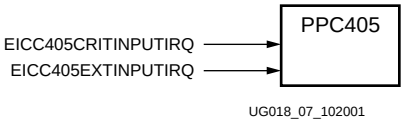

Figure 2-37: **EIC Interface Block Symbol**

Table 2-19: **EIC Interface I/O Signals**

Signal	I/O Type	If Unused	Function
EICC405CRITINPUTIRQ	I	0	Indicates an external critical interrupt occurred.
EICC405EXTINPUTIRQ	I	0	Indicates an external noncritical interrupt occurred.

EIC Interface I/O Signal Descriptions

The following sections describe the operation of the EIC interface I/O signals.

EICC405CRITINPUTIRQ (input)

When asserted, this signal indicates the EIC is requesting that the processor block respond to an external critical interrupt. When deasserted, no request exists. The EIC is responsible for collecting critical interrupt requests from other peripherals and presenting them as a single request to the processor block. Once asserted, this signal remains asserted by the EIC until software deasserts the request (this is typically done by writing to a DCR in the EIC).

EICC405EXTINPUTIRQ (input)

When asserted, this signal indicates the EIC is requesting that the processor block respond to an external noncritical interrupt. When deasserted, no request exists. The EIC is responsible for collecting noncritical interrupt requests from other peripherals and presenting them as a single request to the processor block. Once asserted, this signal remains asserted by the EIC until software deasserts the request (this is typically done by writing to a DCR in the EIC).

Virtex-II Pro PPC405 JTAG Debug Port

The Virtex-II Pro PPC405 core features a JTAG interface to support software debugging. Many debuggers, such as RISCWatch from IBM, SingleStep from Wind River and the GNU Debugger (GDB) in the Xilinx Embedded Development Kit (EDK), use the PPC405 JTAG interface for this purpose.

Like all other signals on the PPC405 core, the user must define the connections from the JTAG interface to the outside world. Since these connections can only be made through programmable interconnect, the Virtex-II Pro must be configured before the PPC405 JTAG interface is available.

The PPC405 JTAG logic may be connected through the Virtex-II Pros native JTAG port (series connection), or directly to programmable I/O (individual connection). The primary consideration in choosing a connection style is knowing which connection your software debugger requires.

JTAG Interface I/O Signal Table

Figure 2-38 shows the block symbol for the JTAG interface.

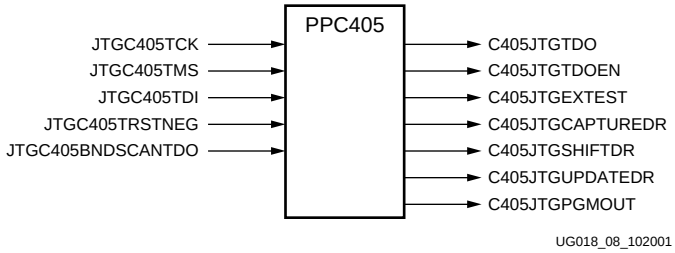

Figure 2-38: **JTAG Interface Block Symbol**

JTAG Interface I/O Signal Descriptions

The following sections describe the operation of the JTAG interface I/O signals.

JTGC405TCK (input)

This input is the JTAG TCK (Test ClocK) signal. The TMS and TDI signals are latched on the rising edge of TCK, while TDO is valid on the falling edge of TCK. The maximum TCK frequency is one-half the CPMC405CLOCK frequency.

JTGC405TMS (input)

This input is the JTAG TMS (Test Mode Select) signal. It is latched by the processor on the rising edge of TCK. The value of the signal is typically changed by external logic on the falling edge of TCK. The TMS signal is used to select the next state in the TAP (JTAG) state machine.

JTGC405TDI (input)

This input is the JTAG TDI signal. It is latched by the processor on the rising edge of TCK.

The value of the signal is typically changed by external logic on the falling edge of TCK.

Data received on this input signal is placed into the Instruction Register or the appropriate Data Register as specified by the TAP state machine.

JTGC405TRSTNEG (input)

This input is the active-low JTAG test reset (TRST) signal. This signal may be either tied high or wired to a user I/O. Note that the Virtex-II Pro device does not implement the TRST signal. If JTC405TRSTNEG is tied high, the PPC405 TAP may be reset synchronously by clocking five 1's on TMS. This signal is automatically used by the processor block during power-on reset to reset the JTAG logic.

JTGC405BNDSCANTDO (input)

This input should not be used; leave it unconnected.

C405JTGTDO (output)

This output is the JTAG TDO (Test Data Out) signal. It is driven by the processor with a new value on the falling edge of the JTAG clock when the PPC405 TAP is in either the Shift-DR or Shift-IR state. The C405JTGTDO output is not valid in other TAP states.

C405JTGTDOEN (output)

This output is asserted (logic High) when the C405JTGTDO signal is valid.

C405JTGEXTEST (output)

This output should not be used; leave it unconnected.

C405JTGCAPTUREDR (output)

This output is asserted (logic High) when the PPC405 TAP is in the Capture-DR state. Most designs do not require this signal and should leave it unconnected.

C405JTGSHIFTDR (output)

This output is asserted (logic High) when the PPC405 TAP is in the Shift-DR state. Most designs do not require this signal and should leave it unconnected.

C405JTGUPDATEDR (output)

This output is asserted (logic High) when the PPC405 TAP is in the Update-DR state. Most designs do not require this signal and should leave it unconnected.

C405JTGPGMOUT (output)

This signal indicates the state of a general purpose program bit in the JTAG debug control register (JDCR), and is used by some software debuggers. Its function and operation are determined by the external application. This signal should be left unconnected in most cases.

Virtex-II Pro JTAG Instruction Register

Virtex-II Pro devices contain zero, one, two, or four PPC405 cores. The Instruction Register length depends upon the number of PPC405 cores the device features, but it does not

matter whether or not those cores are used. Table 2-20 gives the IR length for all Virtex-II Pro devices.

Table 2-20: **Virtex-II Pro IR Lengths**

Device	# PPC405 Cores	IR Length
XC2VP2	0	6
XC2VP4	1	10
XC2VP7	1	10
XC2VP20	2	14
XC2VP30	2	14
XC2VP40	2	14
XC2VP50	2	14
XC2VP70	2	14
XC2VP100	2	14
XC2VP125	4	22

The six least significant bits of the Virtex-II Pro Instruction Register always comprise the FPGA Instruction Register. The remaining bits are ignored unless the PPC405 cores are connected in series with the FPGA JTAG logic, as described in the "Connecting PPC405 JTAG logic in Series with the dedicated Virtex-II Pro JTAG logic" section below. When the PPC405 JTAG logic is connected in this way, its Instruction Register automatically replaces the "dummy" register for the upper IR bits. Figure 2-39 illustrates the default Virtex-II Pro Instruction Register data path, whileFigure 2-40 illustrates the data path for the series PPC405 JTAG connection

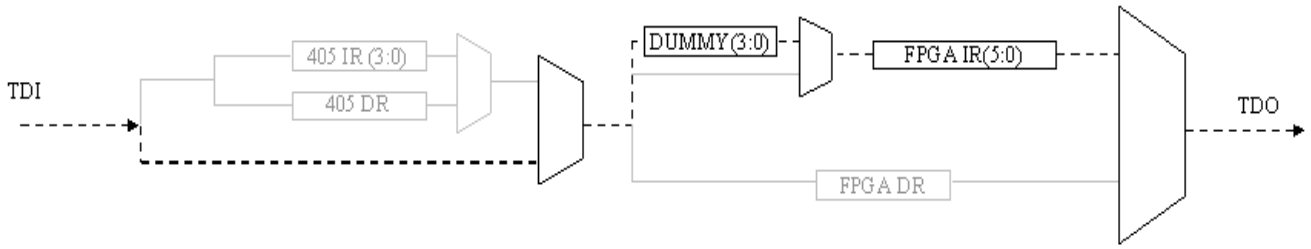

Figure 2-39: **Default Virtex-II Pro Instruction Register data path (XC2VP4, XC2VP7)**

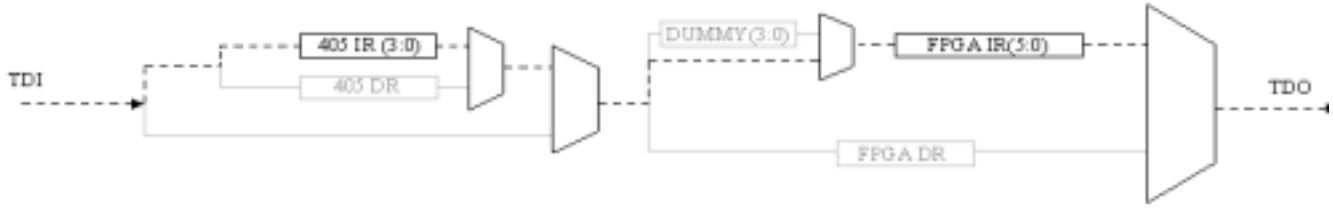

Figure 2-40: **Virtex-II Pro Instruction Register data path for Series PPC405 JTAG connection (XC2VP4, XC2VP7)**

The PPC405 JTAG logic implements eight instructions: PPC_DEBUG_1, PPC_DEBUG_2, ..., PPC_DEBUG_8. If the PPC405 JTAG logic is connected in series with the FPGA JTAG logic, the value "100000" must be loaded into the FPGA Instruction Register. Most users will not be concerned with how these instructions are used because the details of the PPC405 JTAG debugging interface are handled by debugging software. Users who need detailed information about the PPC405 JTAG debugging interface should contact IBM for this specification.

Table 2-21: **PPC405 Instruction Opcodes**

Instruction	Opcode
PPC_BYPASS	1111
PPC_DEBUG_1	0101
PPC_DEBUG_2	0111
PPC_DEBUG_3	1001
PPC_DEBUG_4	1010
PPC_DEBUG_5	1011
PPC_DEBUG_6	1100
PPC_DEBUG_7	1101
PPC_DEBUG_8	1110

The PPC405 cores do not have their own BSDL files; instead, the necessary INSTRUCTION_OPCODES and other information are incorporated in the Virtex-II Pro BSDL file. The PPC405 cores are not available for interconnect tests (i.e. EXTEST, SAMPLE/PRELOAD), as they do not have a boundary scan register. All Virtex-II Pro boundary scan tests are performed through the FPGA boundary scan register.

Connecting PPC405 JTAG logic directly to Programmable I/O

The simplest way to access the PPC405 JTAG logic is to wire the processor core's JTAG signals directly to programmable I/O. For devices with multiple PPC405 cores, users may wire each set of PPC405 JTAG signals directly to programmable I/O (Figure 2-42); chain the processors together with programmable interconnect and wire the combined PPC405 JTAG chain to programmable I/O (Figure 2-43) or multiplex a single set of JTAG pins to multiple cores (Figure 2-44).

Each of these connection styles requires additional I/O and a separate JTAG chain for the PPC405 core(s). The PPC405 cores must not be placed in the same JTAG chain as the dedicated Virtex-II Pro JTAG pins because the chain will be broken by the missing PPC405 JTAG logic prior to FPGA configuration (Figure 2-41).

The /TRST signal, which is not implemented on any Xilinx devices, is available on the IBM PPC405 core. This signal may be wired to user I/O or internally tied high. If wired to user I/O, an external 10K Ohm pullup resistor should be placed on the trace.

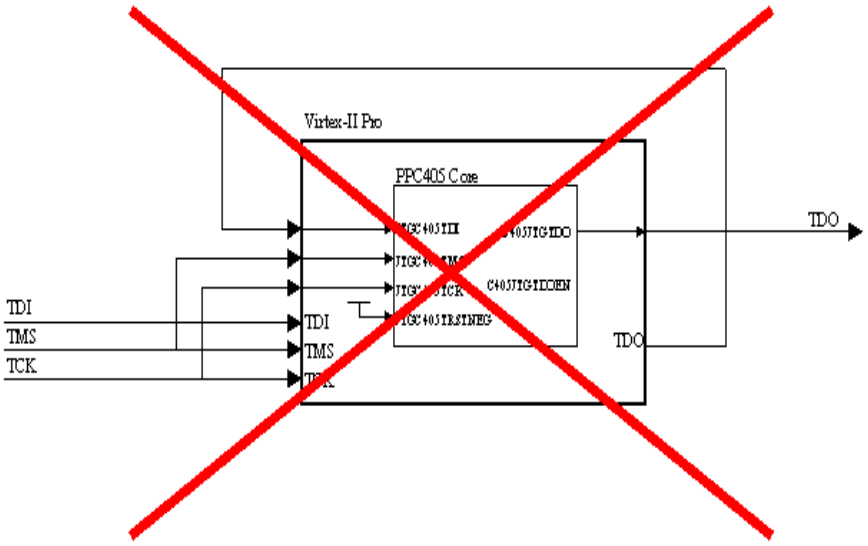

Figure 2-41: **Incorrect wiring of JTAG chain with individual PPC405 connections**

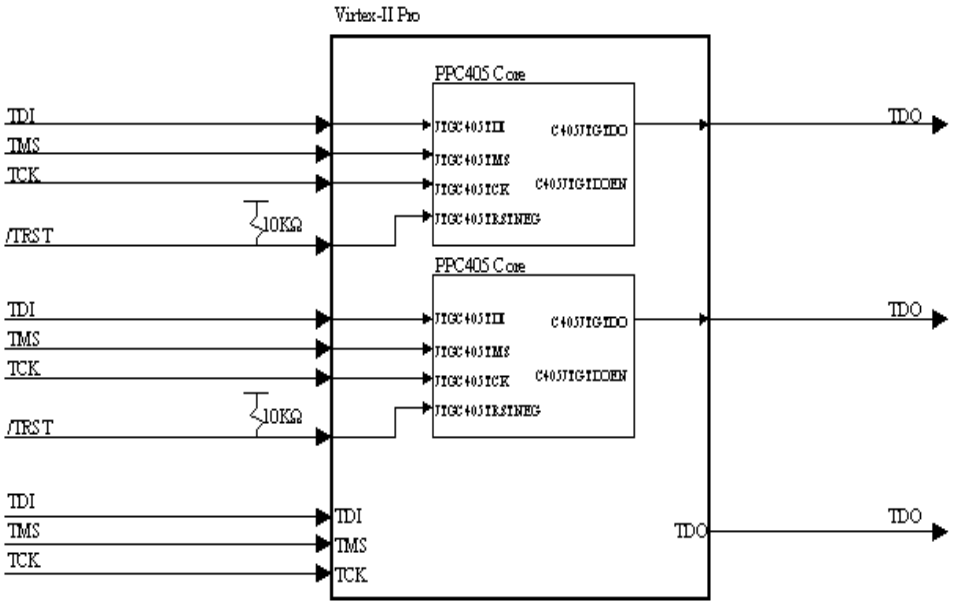

Figure 2-42: **Correct wiring of JTAG chains with individual PPC405 connections (separate JTAG chains)**

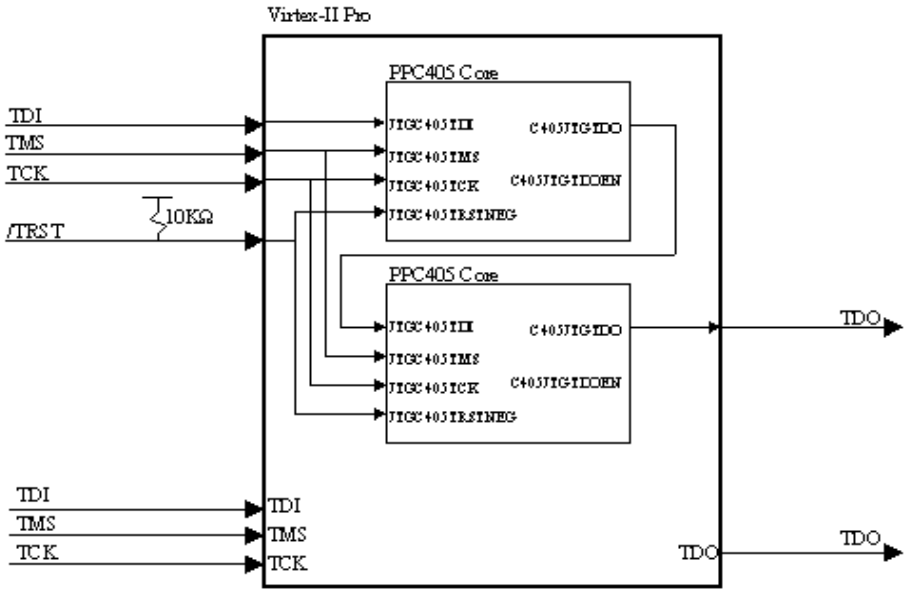

Figure 2-43: **Correct wiring of JTAG chains with individual PPC405 JTAG connections (internally chained PPC405 cores)**

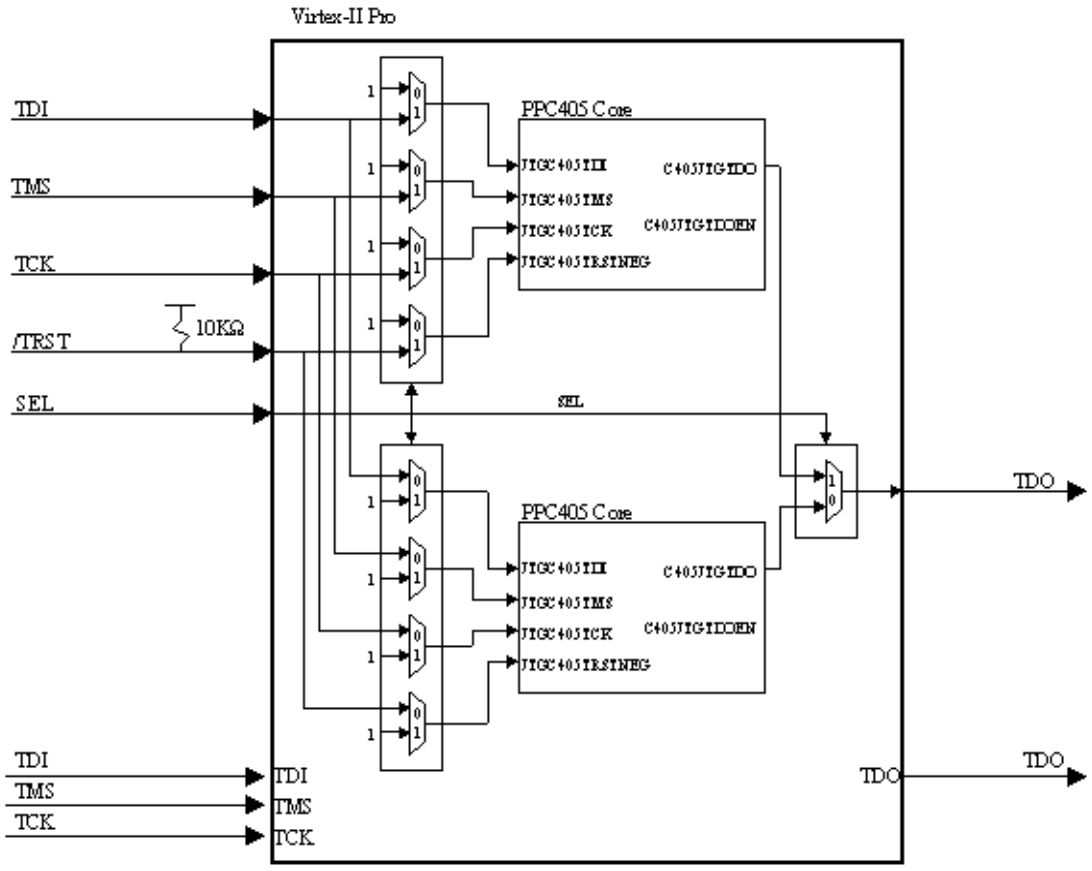

Figure 2-44: **Correct wiring of JTAG chain with multiplexed PPC405 connection**

Connecting PPC405 JTAG logic in Series with the dedicated Virtex-II Pro JTAG logic

An alternative to connecting the PPC405 JTAG logic directly to programmable I/O is to wire it in series with the dedicated Virtex-II Pro JTAG logic. This is done by wiring the JTAG signals on the PPC405 core to a special design element called the JTAGPPC primitive in the user design. As described in the "Virtex-II Pro JTAG Instruction Register" section above, the Virtex-II Pro Instruction Register length remains constant, regardless of how the PPC405 cores are used and regardless of whether or not the device is configured.

Prior to configuration, the most-significant IR bits are placed in a dummy register which is either 4, 8, or 16 bits in length, depending on the number of available PPC405 cores in the Virtex-II Pro device (see Table 2-20). This register is used as a placeholder only. After configuration, if the user connects the PPC405 JTAG logic in series with the dedicated Virtex-II Pro JTAG logic, the most significant IR bits are used by the PPC405 cores. Thus, the overall IR length remains the same for the Virtex-II Pro device at all times.

When the PPC405 JTAG logic is connected in series with the dedicated Virtex-II Pro JTAG logic, the C405JTGTDO signal of each core is connected to the JTGC405TDI of the next. The JTGC405TCK and JTGC405TMS signals are connected to each PPC405 core in parallel. The C405JTGTDOEN output of each PPC405 cores must be OR'ed to the TDO_TS_PPC input of the JTAGPPC primitive (for devices with only one PPC405 core, wire the C405JTGTDOEN output directly to the TDO_TS_INPUT on the JTAGPPC primitive). The /TRST signal, which is not implemented on the Virtex-II Pro device, is implemented on the IBM PPC405 core. When wiring the PPC405 JTAG logic in series with the FPGA JTAG logic, this signal must be pulled High as shown in Figure 2-45.

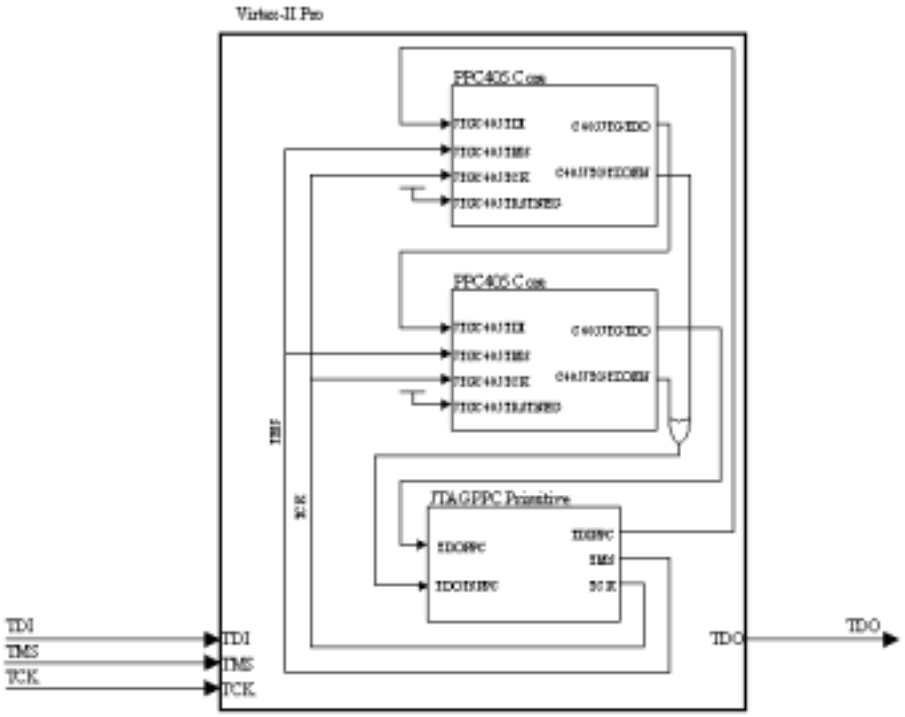

Figure 2-45: **PPC405 core JTAG logic connected in series with FPGA JTAG logic using the JTAGPPC primitive**

When the PPC405 JTAG logic is connected in series with the dedicated Virtex-II Pro JTAG logic, only one JTAG chain is required on the printed circuit board. All JTAG logic in the Virtex-II Pro is accessed through the dedicated JTAG pins on the Virtex-II Pro with this connection style.

For devices with more than one PPC405 core, users must connect the JTAG logic for ALL of the PPC405 cores on the device when using this connection style, even if some are not otherwise used. Only the JTAG signals on unused PPC405 cores need to be connected. The PPC405 core that first sees TDI from the JTAGPPC primitive recognizes the first four most significant bits in the Instruction Register; the next PPC405 core sees the next four most significant bits, and so on.

VHDL and Verilog Instantiation Templates

VHDL and Verilog instantiation templates for some connection styles are provided:

- Single PPC Core: Individual Connection to user I/O (SINGLE_PPC_JTAG_INDIVIDUAL)
- Single PPC Core: Serial Connection through Virtex-II Pro JTAG pins (SINGLE_PPC_JTAG_SERIAL)
- Two PPC Cores: Serial Connection through Virtex-II Pro JTAG pins (TWO_PPC_JTAG_SERIAL)

For clarity, these instantiation templates only describe connections for the JTAG-related I/Os on the PPC405 core. Not all PPC405 I/Os are shown.

```
-- Module: SINGLE_PPC_JTAG_INDIVIDUAL
-- Description: VHDL instantiation template for individual connection
of a single PPC405 core to user I/O
-- Device: 2VP2, 2VP4, 2VP7

library IEEE;
use IEEE.std_logic_1164.all;

entity SINGLE_PPC_JTAG_INDIVIDUAL is
port (
TCK_IN: in std_logic;
TDI_IN: in std_logic;
TMS_IN: in std_logic;
TRSTNEG_IN: in std_logic;
TDO_OUT: out std_logic;
end SINGLE_PPC_JTAG_INDIVIDUAL;

architecture SINGLE_PPC_JTAG_INDIVIDUAL_arch of
SINGLE_PPC_JTAG_INDIVIDUAL is

-- Component Declaration
component PPC405
port(
...
JTGC405TCK : in std_logic;
JTGC405TMS: in std_logic;
JTGC405TDI: in std_logic;
JTGC405TRSTNEG: in std_logic;
C405JTGTDO: out std_logic;
JTGC405BNDSCANTDO: in std_logic;
C405JTGTDOEN: out std_logic;
C405JTGEXTEST: out std_logic;
C405JTGCAPTUREDR: out std_logic;
C405JTGSHIFTDR: out std_logic;
C405JTGUPDATEDR: out std_logic;
C405JTGPGMOUT: out std_logic;
...
);
end component

begin

-- Component Instantiation
U_PPC1 : PPC405
port map (
```

```
                ...
                JTGC405TCK => TCK_IN,
                JTGC405TDI => TDI_IN,
                JTGC405TMS => TMS_IN,
                JTGC405TRSTNEG => TRSTNEG_IN,
C405JTGTDO => TDO_OUT,
JTGC405BNDSCANTDO => open,
C405JTGTDOEN => open,
C405JTGEXTEST => open,
C405JTGCAPTUREDR => open,
C405JTGSHIFTDR => open,
C405JTGUPDATEDR=> open,
C405JTGPGMOUT=> open,
                          ...
                          );

end SINGLE_PPC_JTAG_INDIVIDUAL_arch;



// Module: SINGLE_PPC_JTAG_INDIVIDUAL
// Description: Verilog instantiation template for individual
connection of a single PPC405
// core to user I/O
// Device: 2VP4 and 2VP7

module SINGLE_PPC_JTAG_INDIVIDUAL (
  TCK_IN,
  TDI_IN,
  TMS_IN,
  TRSTNEG_IN
  TDO_OUT
  );

  input TCK_IN;
  input TDI_IN;
  input TMS_IN;
  input TRSTNEG_IN;

  output TDO_OUT;

// Component Instantiation
PPC405 U_PPC1(
           ...
           .JTGC405TCK (TCK_IN),
           .JTGC405TDI (TDI_IN),
           .JTGC405TMS (TMS_IN),
          .JTGC405TRSTNEG (TRSTNEG_IN),
  .C405JTGTDO (TDO_OUT),
  .JTGC405BNDSCANTDO (),
  .C405JTGTDOEN (),
  .C405JTGEXTEST (),
  .C405JTGCAPTUREDR (),
  .C405JTGSHIFTDR (),
  .C405JTGUPDATEDR (),
  .C405JTGPGMOUT (),
            ...
      );
```

```
endmodule;



-- Module: SINGLE_PPC_JTAG_SERIAL
-- Description: VHDL instantiation template for serial connection of a
single PPC405
-- core to Virtex-II Pro JTAG logic
-- Device: 2VP4 and 2VP7

library IEEE;
use IEEE.std_logic_1164.all;

entity SINGLE_PPC_JTAG_SERIAL is
  port (
    );
end SINGLE_PPC_JTAG_SERIAL;

architecture SINGLE_PPC_JTAG_SERIAL_arch of SINGLE_PPC_JTAG_SERIAL is

-- Component Declaration
component PPC405
  port(
    ...
    JTGC405TCK : in std_logic;
    JTGC405TMS: in std_logic;
    JTGC405TDI: in std_logic;
    JTGC405TRSTNEG: in std_logic;
    C405JTGTDO: out std_logic;
    JTGC405BNDSCANTDO: in std_logic;
    C405JTGTDOEN: out std_logic;

    C405JTGEXTEST: out std_logic;
    C405JTGCAPTUREDR: out std_logic;
    C405JTGSHIFTDR: out std_logic;
    C405JTGUPDATEDR: out std_logic;
    C405JTGPGMOUT: out std_logic;
    ...
    );
end component;

component JTAGPPC
  port(
    TDOTSPPC : in std_logic;
    TDOPPC : in std_logic;
    TMS : out std_logic;
    TDIPPC : out std_logic;
    TCK : out std_logic;
    );
end component;

signal TDO_TS_PPC : std_logic;
signal TDO_PPC : std_logic;
signal TMS_PPC : std_logic;
signal TDI_PPC : std_logic;
signal TCK_PPC : std_logic;
```

```
begin

-- Component Instantiation
U_PPC1 : PPC405
  port map (
             ...
                 JTGC405TCK => TCK_PPC,
                JTGC405TDI => TDI_PPC,
                  JTGC405TMS => TMS_PPC,
                JTGC405TRSTNEG => 1,
    C405JTGTDO => TDO_PPC,
    JTGC405BNDSCANTDO => open,
    C405JTGTDOEN => TDO_TS_PPC,
    C405JTGEXTEST => open,
    C405JTGCAPTUREDR => open,
    C405JTGSHIFTDR => open,
    C405JTGUPDATEDR=> open,
    05JTGPGMOUT=> open,
                          ...
                          );

U_JTAG : JTAGPPC
  port map (
    TDOTSPPC => TDO_TS_PPC,
    TDOPPC => TDO_PPC,
    TMS => TMS_PPC,
    TDIPPC => TDI_PPC,
    TCK => TCK_PPC
    );

end SINGLE_PPC_JTAG_SERIAL_arch;
```

```
// Module: SINGLE_PPC_JTAG_SERIAL
// Description: Verilog instantiation template for serial connection of
a single PPC405
// core to Virtex-II Pro JTAG logic
// Device: 2VP4 and 2VP7

module SINGLE_PPC_JTAG_SERIAL ();

  wire TDO_TS_PPC;
  wire TDO_PPC;
  wire TMS_PPC;
  wire TDI_PPC;
  wire TCK_PPC;

// Component Instantiation
PPC405 U_PPC1(
             ...
                 .JTGC405TCK (TCK_PPC),
                .JTGC405TDI (TDI_PPC),
                  .JTGC405TMS (TMS_PPC),
                .JTGC405TRSTNEG (1'b1),
   .C405JTGTDO (TDO_PPC),
   .JTGC405BNDSCANTDO (),
```

```
   .C405JTGTDOEN (TDO_TS_PPC),
   .C405JTGEXTEST (),
   .C405JTGCAPTUREDR (),
   .C405JTGSHIFTDR (),
   .C405JTGUPDATEDR (),
   .C405JTGPGMOUT (),
                        ...
                        );

JTAGPPC U_JTAG(
   TDOTSPPC (TDO_TS_PPC),
   TDOPPC (TDO_PPC),
   TMS (TMS_PPC),
   TDIPPC (TDI_PPC),
   TCK (TCK_PPC)
   );

endmodule;
```

```
-- Module: TWO_PPC_JTAG_SERIAL
-- Description: VHDL instantiation template for serial connection of
two PPC405
-- cores to Virtex-II Pro JTAG logic
-- Devices: 2VP20, 2VP30, 2VP40, 2VP50, 2VP70, 2VP100


library IEEE;
use IEEE.std_logic_1164.all;

entity TWO_PPC_JTAG_SERIAL is
  port (
    );
end TWO_PPC_JTAG_SERIAL

architecture TWO_PPC_JTAG_SERIAL_arch of TWO_PPC_JTAG_SERIAL is

-- Component Declaration
component PPC405
  port(
    ...
    JTGC405TCK : in std_logic;
    JTGC405TMS: in std_logic;
    JTGC405TDI: in std_logic;
    JTGC405TRSTNEG: in std_logic;
    C405JTGTDO: out std_logic;
    JTGC405BNDSCANTDO: in std_logic;
C405JTGTDOEN: out std_logic;
    C405JTGEXTEST: out std_logic;
    C405JTGCAPTUREDR: out std_logic;
    C405JTGSHIFTDR: out std_logic;
    C405JTGUPDATEDR: out std_logic;
    C405JTGPGMOUT: out std_logic;
    ...
    );
end component
```

```
component JTAGPPC
  port(
    TDOTSPPC : in std_logic;
    TDOPPC : in std_logic;
    TMS : out std_logic;
    TDIPPC : out std_logic;
    TCK : out std_logic;
    );
end component;

signal TDO_TS_PPC : std_logic;
signal TMS_PPC : std_logic;
signal TDI_PPC : std_logic;
signal TCK_PPC : std_logic;
signal TDO_OUT1 : std_logic;
signal TDO_OUT2 : std_logic;
signal TDO_TS_OUT1 : std_logic;
signal TDO_TS_OUT2 : std_logic;

begin

TDO_TS_PPC <= TDO_TS_OUT1 OR TDO_TS_OUT2;

-- Component Instantiation
U_PPC1 : PPC405
  port map (
                  ...
     JTGC405TCK => TCK_PPC,
                JTGC405TDI => TDI_PPC
                  JTGC405TMS => TMS_PPC
                JTGC405TRSTNEG => 1,
    C405JTGTDO => TDO_OUT1,
    JTGC405BNDSCANTDO => open,
    C405JTGTDOEN =>TDO_TS_OUT1;
    C405JTGEXTEST => open,
    C405JTGCAPTUREDR => open,
    C405JTGSHIFTDR => open,
    C405JTGUPDATEDR=> open,
    C405JTGPGMOUT=> open,
                          ...
                          );

U_PPC2 : PPC405
  port map (
                  ...
                 JTGC405TCK => TCK_PPC,
                JTGC405TDI => TDO_OUT1,
                  JTGC405TMS => TMS_PPC,
                JTGC405TRSTNEG => 1,
    C405JTGTDO => TDO_OUT2,
    JTGC405BNDSCANTDO => open,
    C405JTGTDOEN => TDO_TS_OUT2,
    C405JTGEXTEST => open,
    C405JTGCAPTUREDR => open,
    C405JTGSHIFTDR => open,
    C405JTGUPDATEDR=> open,
    C405JTGPGMOUT=> open,
                          ...
                          );
```

```
U_JTAG : JTAGPPC
  port map (
    TDOTSPPC => TDO_TS_PPC,
    TDOPPC => TDO_OUT2,
    TMS => TMS_PPC,
    TDIPPC => TDI_PPC,
    TCK => TCK_PPC
    );

end TWO_PPC_JTAG_SERIAL_arch;
```

```
// Module: TWO_PPC_JTAG_SERIAL
// Description: Verilog instantiation template for serial connection of
two PPC405
// cores to Virtex-II Pro JTAG logic
// Devices: 2VP20, 2VP30, 2VP40, 2VP50, 2VP70, 2VP100


module TWO_PPC_JTAG_SERIAL ();

  wire TDO_TS_PPC;
  wire TMS_PPC;
  wire TDI_PPC;
  wire TCK_PPC;
  wire TDO_OUT1;
  wire TDO_OUT2;
  wire TDO_TS_OUT1;
  wire TDO_TS_OUT2;

or o1(TDO_TS_PPC, TDO_TS_OUT1, TDO_TS_OUT2);

// Component Instantiation
PPC405 U_PPC1(
            ...
                .JTGC405TCK (TCK_PPC),
               .JTGC405TDI (TDI_PPC),
                 .JTGC405TMS (TMS_PPC),
               .JTGC405TRSTNEG (1'b1),
   .C405JTGTDO (TDO_OUT1),
   .JTGC405BNDSCANTDO (),
   .C405JTGTDOEN (TDO_TS_OUT1),
   .C405JTGEXTEST (),
   .C405JTGCAPTUREDR (),
   .C405JTGSHIFTDR (),
   .C405JTGUPDATEDR (),
   .C405JTGPGMOUT (),
                        ...
                        );

PPC405 U_PPC2(
            ...
                .JTGC405TCK (TCK_PPC),
               .JTGC405TDI (TDO_OUT1),
                 .JTGC405TMS (TMS_PPC),
```

```
                .JTGC405TRSTNEG (1'b1),
    .C405JTGTDO (TDO_OUT2),
    .JTGC405BNDSCANTDO (),
    .C405JTGTDOEN (TDO_TS_OUT2),
    .C405JTGEXTEST (),
    .C405JTGCAPTUREDR (),
    .C405JTGSHIFTDR (),
    .C405JTGUPDATEDR (),
    .C405JTGPGMOUT (),
                          ...
                          );

JTAGPPC U_JTAG(
    TDOTSPPC (TDO_TS_PPC),
    TDOPPC (TDO_OUT2),
    TMS (TMS_PPC),
    TDIPPC (TDI_PPC),
    TCK (TCK_PPC)
    );

endmodule;
```

Debug Interface

The debug interface enables an external debugging tool (such as RISCWatch) to operate the PPC405x3 debug resources in external-debug mode. External-debug mode can be used to alter normal program execution and it provides the ability to debug system hardware as well as software. The mode supports starting and stopping the processor, single-stepping instruction execution, setting breakpoints, and monitoring processor status. These capabilities are described in the *PowerPC Processor Reference Guide*.

Debug Interface I/O Signal Summary

Figure 2-46 shows the block symbol for the debug interface. The signals are summarized in Table 2-22. See Appendix A, **RISCWatch and RISCTrace Interfaces** for information on attaching a RISCWatch to the debug interface signals.

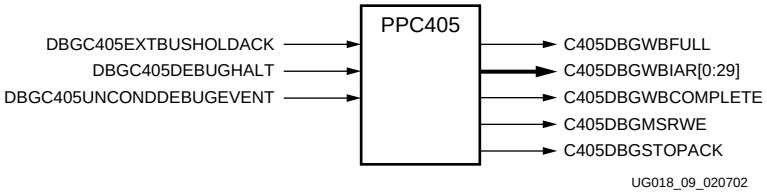

Figure 2-46: **Debug Interface Block Symbol**

Table 2-22: **Debug Interface I/O Signals**

Signal	I/O Type	If Unused	Function
DBGC405EXTBUSHOLDACK	I	0	Indicates the bus controller has given control of the bus to an external master.
DBGC405DEBUGHALT	I	0	Indicates the external debug logic is placing the processor in debug halt mode.
DBGC405UNCONDDEBUGEVENT	I	0	Indicates the external debug logic is causing an unconditional debug event.
C405DBGWBFULL	O	No Connect	Indicates the PPC405x3 writeback pipeline stage is full.
C405DBGWBIAR[0:29]	O	No Connect	The address of the current instruction in the PPC405x3 writeback pipeline stage.
C405DBGWBCOMPLETE	O	No Connect	Indicates the current instruction in the PPC405x3 writeback pipeline stage is completing.
C405DBGMSRWE	O	No Connect	Indicates the value of MSR[WE].
C405DBGSTOPACK	O	No Connect	Indicates the PPC405x3 is in debug halt mode.

Debug Interface I/O Signal Descriptions

The following sections describe the operation of the debug interface I/O signals.

DBGC405EXTBUSHOLDACK (input)

When asserted, this signal indicates that the bus controller (for example, a PLB arbiter) has given control of the bus to an external master. When deasserted, an external master does not have control of the bus. This signal is used by the PPC405x3 debug logic (and the external debugger) as an indication that the processor might not have control of the bus and therefore might not be able to respond immediately to certain debug operations. External FPGA logic generates this signal using output signals from the bus controller.

DBGC405DEBUGHALT (input)

When asserted, this signal stops the processor from fetching and executing instructions so that an external debug tool can operate the processor. From this state, known as *debug halt mode*, an external debugger controls the processor using the JTAG interface and the private JTAG hardware debug instructions. The clocks are not stopped. When this signal is deasserted, the processor operates normally.

This signal enables an external debugger to stop the processor without using the JTAG interface. A stop command issued through the JTAG interface (using a private JTAG instruction) is discarded when the processor is reset. The debug halt signal can be asserted during a reset so that the processor is stopped at the first instruction to be executed when reset is exited.

In systems that deactivate the clocks to manage power, the debug halt signal should be used to restart the clocks (if stopped) to enable an external debugger to operate the processor. After the debugger finishes its operation and deasserts the debug halt signal, the clocks can be stopped to return the processor to sleep mode.

This is a positive active signal. However, the debug halt signal produced by the RISCWatch debugger is negative active. FPGA logic that attaches to a RISCWatch debugger must invert the signal before sending it to the PPC405x3.

DBGC405UNCONDDEBUGEVENT (input)

When asserted, this signal causes an unconditional debug event and sets the UDE bit in the debug-status register (DBSR) to 1. When this signal is deasserted, the processor operates normally. Software can initialize the PPC405x3 debug resources to perform any of the following operations when an unconditional debug event occurs:

- Cause a debug interrupt in internal debug mode.
- Stop the processor in external debug mode.
- Cause a trigger event on the processor block trace interface.

C405DBGWBFULL (output)

When asserted, this signal indicates that the PPC405x3 writeback-pipeline stage is full. It also indicates that writeback instruction-address bus (C405DBGWBIAR[0:29]) contains a valid instruction address. When deasserted, the writeback stage is not full and the contents of the writeback instruction-address bus are not valid.

C405DBGWBIAR[0:29] (output)

When the writeback-full signal (C405DBGWBFULL) is asserted, this bus contains the address of the instruction in the PPC405x3 writeback-pipeline stage. If the writeback-full signal is not asserted, the contents of this bus are invalid.

C405DBGWBCOMPLETE (output)

When asserted, this signal indicates that the instruction in the PPC405x3 writeback-pipeline stage is completing. The address of the completing instruction is contained on the writeback instruction-address bus (C405DBGWBIAR[0:29]). If the writeback-complete signal is not asserted, the instruction on the writeback instruction-address bus is not completing. The writeback-complete signal is valid only when the writeback-full signal (C405DBGWBFULL) is asserted. The signal is not valid if the writeback-full signal is deasserted.

C405DBGMSRWE (output)

This signal indicates the state of the MSR[WE] (wait-state enable) bit. When asserted, wait state is enabled (MSR[WE]=1). When deasserted, wait state is disabled (MSR[WE]=0). When in the wait state, the processor stops fetching and executing instructions, and no longer performs memory accesses. The processor continues to respond to interrupts, and can be restarted through the use of external interrupts or timer interrupts. Wait state can also be exited when an external debug tool clears WE or when a reset occurs.

C405DBGSTOPACK (output)

When asserted, this signal indicates that the PPC405x3 is in debug halt mode. When deasserted, the processor is not in debug halt mode.

Trace Interface

The processor uses the trace interface when operating in real-time trace-debug mode. Real-time trace-debug mode supports real-time tracing of the instruction stream executed by the processor. In this mode, debug events are used to cause external trigger events. An external trace tool (such as RISCTrace) uses the trigger events to control the collection of trace information. The broadcast of trace information on the trace interface occurs independently of external trigger events (trace information is always supplied by the processor). Real-time trace-debug does not affect processor performance.

Real-time trace-debug mode is always enabled. However, the trigger events occur only when both internal-debug mode and external debug mode are disabled (DBCR0[IDM]=0 and DBCR0[EDM]=0). Most trigger events are blocked when either of those two debug modes are enabled. See the *PowerPC Processor Reference Guide* for more information on debug events.

Trace Interface Signal Summary

Figure 2-47 shows the block symbol for the trace interface. The signals are summarized in Table 2-23. See Appendix A, **RISCWatch and RISCTrace Interfaces** for information on attaching a RISCTrace to the trace interface signals.

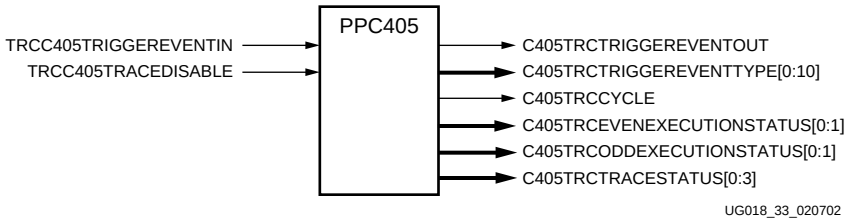

Figure 2-47: **Trace Interface Block Symbol**

Table 2-23: **Trace Interface Signals**

Signal	I/O Type	If Unused	Function
C405TRCTRIGGEREVENTOUT	O	Wrap to Trigger Event In	Indicates a trigger event occurred.
C405TRCTRIGGEREVENTTYPE[0:10]	O	No Connect	Specifies which debug event caused the trigger event.
C405TRCCYCLE	O	No Connect	Specifies the trace cycle.
C405TRCEVENEXECUTIONSTATUS[0:1]	O	No Connect	Specifies the execution status collected during the first of two processor cycles.
C405TRCODDEXECUTIONSTATUS[0:1]	O	No Connect	Specifies the execution status collected during the second of two processor cycles.
C405TRCTRACESTATUS[0:3]	O	No Connect	Specifies the trace status.
TRCC405TRIGGEREVENTIN	I	Wrap to Trigger Event Out	Indicates a trigger event occurred and that trace status is to be generated.
TRCC405TRACEDISABLE	I	0	Disables trace collection and broadcast.

Trace Interface I/O Signal Descriptions

The following sections describe the operation of the trace interface I/O signals.

C405TRCTRIGGEREVENTOUT (output)

When asserted, this signal indicates that a trigger event occurred. The trigger event is caused by any debug event when both internal-debug mode and external debug mode are disabled (DBCR0[IDM]=0 and DBCR0[EDM]=0). If this signal is deasserted, no trigger event occurred.

FPGA logic can combine this signal with the trigger-event type signals to produce a qualified version of the trigger signal. The qualified signal is wrapped to the trigger-event input signal in the same trace cycle. The external trace tool also monitors the trigger-event input signal to synchronize its own trace collection. This capability can be used to implement various trace collection schemes.

C405TRCTRIGGEREVENTTYPE[0:10] (output)

These signals are used to identify which debug event caused the trigger event. Table 2-24 shows which debug event corresponds to each bit in the trigger event-type bus. The specified debug event occurred when its corresponding signal is asserted. The debug event did not occur if its corresponding signal is deasserted.

Table 2-24: **Purpose of C405TRCTRIGGEREVENTTYPE[0:10] Signals**

Bit	Debug Event
0	Instruction Address Compare 1 (IAC1)
1	Instruction Address Compare 2 (IAC2)
2	Instruction Address Compare 3 (IAC3)

Table 2-24: **Purpose of C405TRCTRIGGEREVENTTYPE[0:10] Signals *(Continued)***

Bit	Debug Event
3	Instruction Address Compare 4 (IAC4)
4	Data Address Compare 1 (DAC1)—Read
5	Data Address Compare 1 (DAC1)—Write
6	Data Address Compare 2 (DAC2)—Read
7	Data Address Compare 2 (DAC2)—Write
8	Trap Instruction (TDE)
9	Exception Taken (EDE)
10	Unconditional (UDE)

FPGA logic can combine these signals with the trigger-event output signal to produce a qualified version of the trigger signal. The qualified signal is wrapped to the trigger-event input signal in the same trace cycle. The external trace tool also monitors the trigger-event input signal to synchronize its own trace collection. This capability can be used to implement various trace collection schemes.

C405TRCCYCLE (output)

This signal defines the cycle that execution status and trace status are broadcast on the trace interface (this is referred to as the trace cycle). Although the PPC405x3 collects execution status and trace status every processor cycle, the information is made available to the trace interface once every two cycles. The information collected during those two cycles is broadcast over the trace interface in a single trace cycle. For this reason, the trace cycle is produced by the processor once every two processor clocks. Operating the trace interface in this manner helps reduce the amount of I/O switching during trace collection.

C405TRCEVENEXECUTIONSTATUS[0:1] (output)

These signals are used to specify the execution status collected during the first of two processor cycles. The PPC405x3 collects execution status and trace status every processor cycle, but the information is made available to the trace interface once every two cycles. The information collected during those two cycles is broadcast over the trace interface in a single trace cycle.

C405TRCODDEXECUTIONSTATUS[0:1] (output)

These signals are used to specify the execution status collected during the second of two processor cycles. The PPC405x3 collects execution status and trace status every processor cycle, but the information is made available to the trace interface once every two cycles. The information collected during those two cycles is broadcast over the trace interface in a single trace cycle.

C405TRCTRACESTATUS[0:3] (output)

These signals provide additional information required by a trace tool when reconstructing an instruction execution sequence. This information is collected every processor cycle, but it is made available to the trace interface once every two cycles. The information collected during those two cycles is broadcast over the trace interface in a single trace cycle.

TRCC405TRIGGEREVENTIN (input)

When asserted, this signal indicates that a trigger event occurred. The PPC405x3 uses this signal to generate additional information that is output on the trace-status bus. This information corresponds to the execution status produced on the even and odd execution-status busses. When deasserted, the information is not generated.

This signal can be produced by FPGA logic using the trigger event output signal. The output signal can be combined with the trigger event-type signals before it is returned as the input signal. This capability can be used to implement various trace collection schemes. The external trace tool should monitor the trigger-event input signal to synchronize its own trace collection.

TRCC405TRACEDISABLE (input)

When asserted, this signal disables the collection and broadcast of trace information. Trace information already collected by the processor when this signal is asserted is broadcast on the trace interface before tracing is disabled. When deasserted, trace collection and broadcast proceed normally.

Additional FPGA Specific Signals

Figure 2-48 shows the block symbol for the additional FPGA signals used by the processor block. The signals are summarized in Table 2-25.

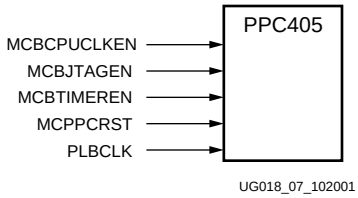

Figure 2-48: **FPGA-Specific Interface Block Symbol**

Table 2-25: **Additional FPGA I/O Signals**

Signal	I/O Type	If Unused	Function
MCBCPUCLKEN	I	1	Indicates the PPC405x3 clock enable should follow GWE during a partial reconfiguration.
MCBJTAGEN	I	1	Indicates the JTAG clock enable should follow GWE during a partial reconfiguration.
MCBTIMEREN	I	1	Indicates the timer clock enable should follow GWE during a partial reconfiguration.
MCPPCRST	I	1	Indicates the processor block should be reset when GSR is asserted during a partial reconfiguration.
PLBCLK	I	Required	PLB clock.

Additional FPGA I/O Signal Descriptions

The following sections describe the operation of the FPGA I/O signals.

MCBCPUCLKEN (input)

When asserted, this signal indicates that the enable for the core clock zone (CPMC405CPUCLKEN) should follow (match the value of) the global write enable (GWE) during the FPGA startup sequence. When deasserted, the enable for the core clock zone ignores (is independent of) the value of GWE.

MCBJTAGEN (input)

When asserted, this signal indicates that the enable for the JTAG clock zone (CPMC405JTAGCLKEN) should follow (match the value of) the global write enable (GWE) during the FPGA startup sequence. When deasserted, the enable for the JTAG clock zone ignores (is independent of) the value of GWE.

MCBTIMEREN (input)

When asserted, this signal indicates that the enable for the timer clock zone (CPMC405TIMERCLKEN) should follow (match the value of) the global write enable (GWE) during the FPGA startup sequence. When deasserted, the enable for the timer clock zone ignores (is independent of) the value of GWE.

MCPPCRST (input)

When asserted, this signal indicates that the processor block should be reset (the core reset signal, RSTC405RESETCORE, is asserted) when the global set reset (GSR) signal is deasserted during the FPGA startup sequence. When MPPCRST is deasserted, the core reset signal ignores (is independent of) the value of GSR.

PLBCLK (input)

This signal is the source clock for all PLB logic.

XILINX®

Chapter 3

PowerPC® 405 OCM Controller

Introduction

The On-Chip Memory (OCM) controller serves as a dedicated interface between the block RAMs in the FPGA and OCM signals available on the embedded PPC405 core. The OCM signals on the processor block are designed to provide a quick access to a fixed amount of Instruction and Data memory space. The OCM controller, an integral component of the PPC405 core architecture, provides an interface to both the 64-bit Instruction-side Block RAM (ISBRAM) and the 32-bit Data-Side Block RAM (DSBRAM). The FPGA designer can choose to implement only ISBRAM, only DSBRAM, both ISBRAM and DSBRAM, or no ISBRAM and no DSBRAM. The OCM controller is capable of addressing up to 16MB of DSBRAM and 16MB of ISBRAM. The number of block RAMs in the device may limit the maximum amount of OCM supported.

Typical applications for Data-Side OCM (DSOCM) include scratch pad memory, as well as use of the dual-port feature of block RAM to enable a bidirectional data transfer between processor and FPGA. Typical applications for Instruction-Side OCM (ISOCM) include storage of interrupt service routines. One of the primary advantages of OCM comes from the fact that it guarantees a fixed latency of execution as there is no bus arbitration required for the OCM interface. Also, it reduces cache pollution and thrashing, since the cache remains available for caching code from other memory resources.

Functional Features

Common Features

- Separate Instruction and Data memory interface between the processor block and BRAMs in FPGA. Eliminates processor local bus (PLB) arbitration between instruction and data-side interfaces to external memory.
- Dedicated interface to Device Control Register (DCR) bus for ISOCM and DSOCM controllers. Dedicated DCR bus loop, inside the processor block, for the OCM controllers.
- Multi-cycle mode option for I-side and D-side interfaces. Multi-cycle operation uses an **N:1** processor-to-BRAM clock ratio, where N is an integer from 1 through 4.
- FPGA configurable DCR register addresses within DSOCM and ISOCM controllers.
- Independent 16MB logical memory space available within PPC405 memory map for each of the DSOCM and ISOCM controllers.

Data-Side OCM (DSOCM)

- 32-bit Data Read bus and 32-bit Data Write bus

- Byte write access to DSBRAM support
- Second port of dual port DSBRAM is available to read / write from an FPGA interface
- 22-bit address to DSBRAM port
- DCR Registers: DSCNTL, DSARC
- Three alternatives to write into DSBRAM: BRAM initialization, CPU, FPGA hardware using second port. See the **Application Example** section on page 155.

Instruction-Side OCM (ISOCM)

The ISOCM interface contains a 64-bit read only port, for instruction fetches, and a 32-bit write only port, to initialize or test the ISBRAM. When implementing the read only port, the user must deassert the write port inputs.

- 64-bit Data Read Only bus (two BRAM clock cycles)
- 32-bit Data Write Only bus (through DCR)
- Separate 21-bit Read Only and Write Only addresses to ISBRAM
- DCR registers: ISCNTL, ISARC, ISINIT, ISFILL
- Two alternatives to setup ISBRAM contents:
  - Use DCR to access the 32-bit Data Write Only Bus
  - Initialize ISBRAM during FPGA configuration. This is the preferred method.

OCM Controller Operation

The OCM controller is of a *distributed* style in that it is split into two blocks, one for the ISOCM interface and the other for DSOCM interface. This arrangement provides the following advantages:

- The overall efficiency of the core is improved by eliminating the need for arbitration between two sets of operations on each side, i.e., Load / Store on D-side and I-Fetch on I-side.
- Controller performance is improved because there is no need to share a common address and data bus between the I-side and D-side interfaces to the block RAM.
- By keeping the two interfaces separate, it is relatively easy to pick and choose one or the other interface as needed by hardware / software designers.
- The programmer's model is simplified, as there is no requirement to deal with a singular memory space between I-side and D-side interfaces.

Operational Summary

DSOCM Controller

The DSOCM controller accepts an address and associated control signals from the processor during a "load" instruction, and passes the valid address to the block RAM interface. For "store" instructions, a valid address from the processor is accompanied by store data in addition to associated control signals. It is important to note here that load instructions have a priority over store instructions at the DSOCM interface.

ISOCM Controller

The ISOCM controller accepts an address and associated control signals from the processor during an "instruction fetch" cycle, and passes the valid address to the block RAM

interface. The instructions in ISBRAM can be stored either by loading the block RAM during FPGA configuration or by using the processor DCR bus.

The DSOCM and ISOCM interfaces are designed to operate independently.

OCM Registers

There are two registers (DSARC andDSCNTL) in DSOCM and four registers (ISARC, ISCNTL, ISINIT and ISFILL) in ISOCM, which can be utilized by system software to set various attributes on each controller.

DCR Interface

The DCR interface serves two purposes:

- Allows the processor to set various attributes on each controller by reading from and writing into DSARC, DSCNTL, ISARC, and ISCNTL
- Allows the processor to write instructions into the ISOCM memory array during system initialization, using ISINIT and ISFILL

A separate DCR chain is used for ISOCM and DSOCM, which is multiplexed inside the processor block with the DCR chain external to the processor block.

DSOCM Ports

Refer to Figure 3-1 for the block diagram of DSOCM. All signals are in big endian format. Figure 3-2, page 139, shows an example of a DSOCM-to-BRAM interface.

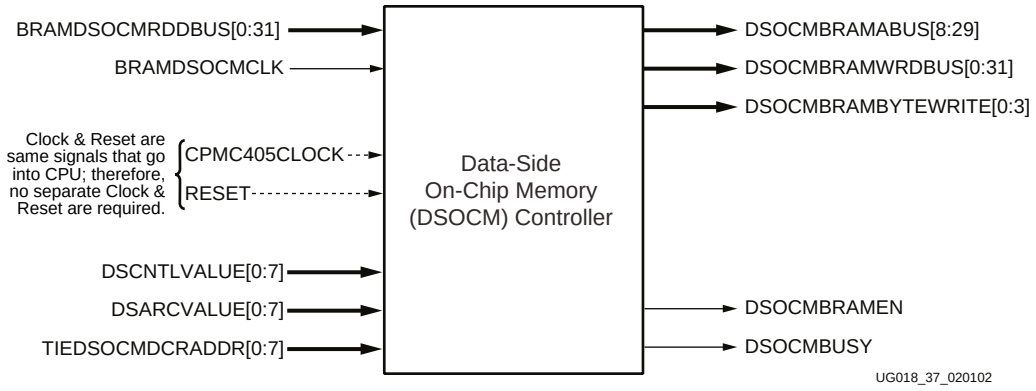

Figure 3-1: **DSOCM Block**

Input Ports

BRAMDSOCMCLK

This signal clocks the DSOCM controller. When in multi-cycle mode, BRAMDSOCMCLK is a **1:N** ratio of the processor clock. The Digital Clock Manager (DCM) should be used to generate the processor clock and the DSOCM clock. The BRAMDSOCMCLK must be an integer multiple of the processor block clock CPMC405CLOCK.

BRAMDSOCMRDDBUS[0:31]

32-bit Read data from block RAMs to DSOCM.

Output Ports

DSOCMBRAMABUS[8:29]

Read or Write address from DSOCM to DSBRAM. A write address is accompanied by a write enable signal for each byte lane of data. Corresponds to CPU address bits [8:29].

DSOCMBRAMWRDBUS[0:31]

32-bit Write data from DSOCM to block RAMs.

DSOCMBRAMBYTEWRITE[0:3]

There are four Write Enable signals to allow independent byte-wide data writes into block RAMs. DSOCMBRAMBYTEWRITE[0] qualifies writes to DSOCMBRAMWRDBUS[0:7], DSOCMBRAMBYTEWRITE[1] qualifies writes to DSOCMBRAMWRDBUS[8:15] and son on.

DSOCMBRAMEN

The block RAM Enable signal is asserted for both reads and writes to the DSBRAM.

DSOCMBUSY

This control signal reflects the value of the DSCNTL[2] bit out to the FPGA fabric. This signal can be used for applications that require a mechanism allowing system software to provide a particular control status to FPGA hardware. It is an optional signal and need not be used.

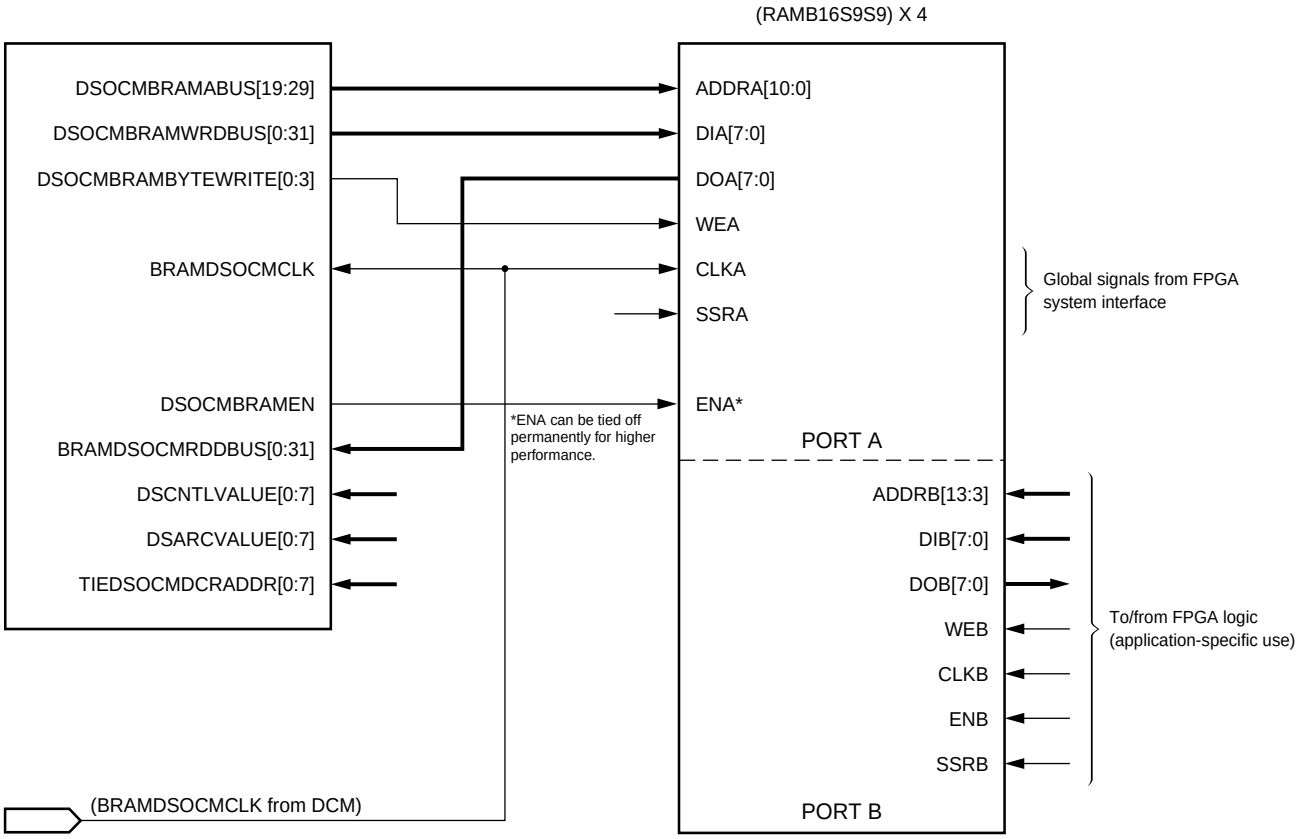

Figure 3-2: **DSOCM to BRAM Interface: 8KBytes Example**

DSOCM Attributes

Attributes are inputs to the OCM from the FPGA that must be connected to initialize registers at FPGA power up, or following a reset.

DSCNTLVALUE[0:7]

Default value that needs to be loaded into DSCNTL register at FPGA power up. See Figure 3-13, page 150, for register bit definitions.

DSARCVALUE[0:7]

Default value that needs to be loaded into DSARC register at FPGA power up. See Figure 3-13, page 150, for register bit definitions.

TIEDSOCMDCRADDR[0:7]

Top 8 bits of DCR address space for DSOCM DCR registers. The DCR address space is 10 bits wide. The two least significant bits are predefined in DSOCM controller.

For example, if TIEDSOCMDCRADDR = `00 0001 11`

then, address of DSARC = `00 0001 1110` = 0x01E
address of DSCNTL = `00 0001 1111` = 0x01F

ISOCM Ports

Refer to Figure 3-3 for the block diagram of ISOCM. All signals are in big endian format. Figure 3-4, page 141, shows an example of an ISOCM-to-BRAM interface.

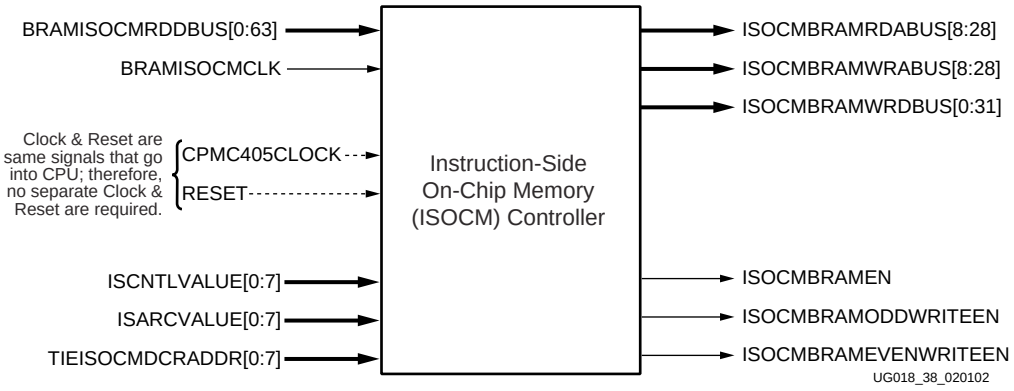

Figure 3-3: **ISOCM Block**

Input Ports

BRAMISOCMCLK

This signal clocks the ISOCM controller. When in multi-cycle mode, BRAMISOCMCLK is a **1:N** ratio of the processor clock. The Digital Clock Manager (DCM) should be used to generate the processor clock and the ISOCM clock. The BRAMISOCMCLK must be an integer multiple of the processor block clock CPMC405CLOCK.

BRAMISOCMRDDBUS[0:63]

64-bit read data from block RAMs to ISOCM.

Output Ports

ISOCMBRAMRDABUS[8:28]

Read address from ISOCM to block RAM. Corresponds to CPU address bits [8:28].

ISOCMBRAMWRABUS[8:28]

NOTE: Optional. Used in dual-port BRAM interface designs only.

Write address from ISOCM to block RAMs. Initially set to value in ISINIT register.

ISOCMBRAMWRDBUS[0:31]

NOTE: Optional. Used in dual-port BRAM interface designs only.

32-bit Write data from ISOCM to block RAMs. Connect to both the even and odd write only ISBRAM data input ports. Initially set to value in ISFILL register.

ISOCMBRAMODDWRITEEN

NOTE: Optional. Used in dual-port BRAM interface designs only.

Write enable to qualify a valid write into a block RAM. This signal enables a write into block RAMs that carry odd instruction words, BRAMISOCMRDDBUS[32:63]. Connect this signal to both the Enable (EN) and Write (WR) inputs of a dual-port ISBRAM port for power savings. For single-port ISBRAM implementations, this signal can be left unconnected.

ISOCMBRAMEVENWRITEEN

NOTE: Optional. Used in dual-port BRAM interface designs only.

Write enable to qualify a valid write into a block RAM. This signal enables a write into block RAMs that carry even instruction words, BRAMISOCMRDDBUS[0:31]. Connect this signal to both the Enable (EN) and Write (WR) inputs of a dual-port ISBRAM port for power savings. For single-port ISBRAM implementations, this signal can be left unconnected.

ISOCMBRAMEN

Block RAM read enable from ISOCM to block RAMs. This signal is asserted for valid ISBRAM read cycles. For highest performance, the BRAM enable input (EN) can be locally tied to a logic 1 level. Power consumption can be reduced by connecting the BRAM enable input (EN) to this signal. Timing analysis is required to verify that the design meets frequency of operation requirements if the enable is not tied to a logic 1 level.

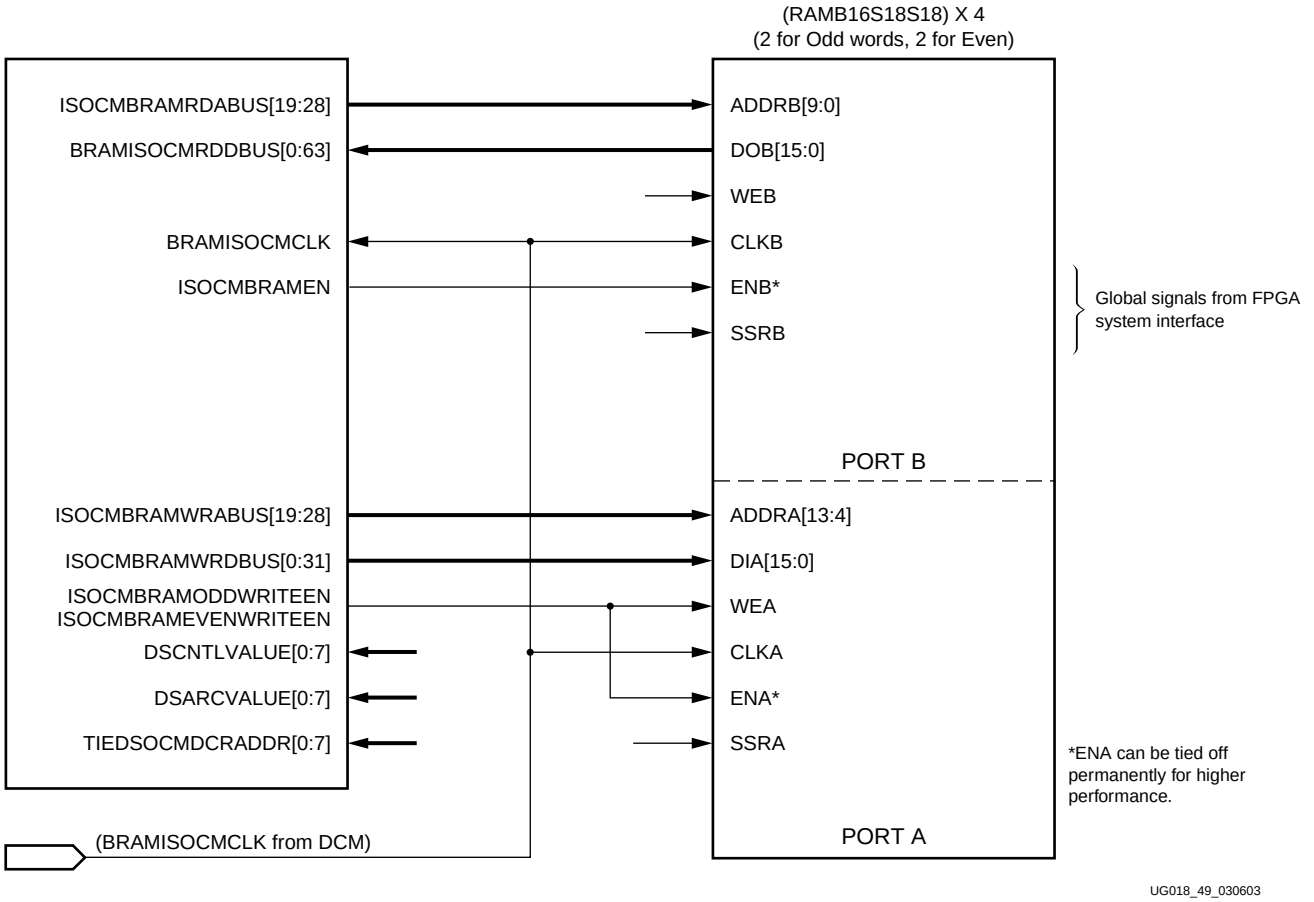

Figure 3-4: **ISOCM to BRAM Interface: 8KBytes Example**

ISOCM Attributes

Attributes are inputs to the OCM, from the FPGA, that must be connected to initialize control registers at FPGA power up, or following a reset. The ISINIT and ISFILL registers cannot be initialized in this manner. These registers are initialized only through “move to DCR” (mtdcr) instructions.

ISCNTLVALUE[0:7]

Default value that needs to be loaded into ISCNTL register, at FPGA power up. See Figure 3-14, page 151, for register bit definitions.

ISARCVALUE[0:7]

Default value that needs to be loaded into ISARC register, at FPGA power up. See Figure 3-14, page 151, for register bit definitions.

TIEISOCMDCRADDR[0:7]

Top 8 bits of DCR address space for ISOCM DCR registers. The DCR address space is 10 bits wide. The two least significant bits are predefined in ISOCM controller.

For example, if TIEISOCMDCRADDR = 00 0010 11

then, address of ISINIT = 00 0010 1100 = 0x02C
address of ISFILL = 00 0010 1101 = 0x02D
address of ISARC = 00 0010 1110 = 0x02E
address of ISCNTL = 00 0010 1111 = 0x02F

Timing Specification

The single-cycle and multi-cycle operation modes are designed to guarantee a certain performance level by the OCM controllers, assuming a certain processor frequency and amount of BRAMs. As additional BRAMs are added to a design, the processor clock frequency must be reduced or wait states must be added in the processor block to insure that the OCM interface operates correctly. When the processor and OCM controller clocks operate at integer multiples of each other, wait cycles are automatically added inside the processor block. The processor core and OCM controllers are aligned on rising edges of the respective clocks.

The speed of the OCM to BRAM interface is determined by running the design through the Xilinx design implementation tools and performing timing analysis on the interface. The interface timing is dependent upon the BRAM memory organization, signal routing delays, signal loading, BRAM memory access time, clock to output times, and setup and hold times of the BRAM and processor block. Users may expect to go through multiple iterations of evaluating OCM BRAM size versus OCM clock frequency in order to achievethe optimum performance.

The OCM clock cycle modes are selected through the DSOCMMCM and ISOCMMCM register bits in the DSCNTL and ISCNTL registers, or the DSCNTLVALUE[0:7] and ISCNTLVALUE[0:7] inputs to the processor block. These control register values and processor block inputs define the clock ratio between the processor core and OCM controllers.

Single-Cycle Mode

In single cycle mode, the processor core, OCM controllers, and BRAMs all run at the same clock speed. Typically, the processor runs at a slower speed than its maximum specified operating frequency, in order to match the speed of the OCM to BRAM interface. The processor frequency must always be reduced when operating in single cycle mode, even when using the smallest supported configuration of DSBRAMs or ISBRAMs.

Multi-Cycle Mode

Multi-cycle mode permits the processor to run at its maximum specified operating frequency. Based upon application specific timing analysis, the clock frequency for the OCM controllers and attached BRAMs is reduced to an integer multiple of the processor clock. Wait states are inserted between each instruction fetch, data load, or data store transaction, internal to the processor block. The transactions start and terminate on rising clock edges of the processor clock and the OCM clock. The Digital Clock Manager (DCM) should be used to generate the clocks for the processor block, OCM controllers, DSBRAMs, and ISBRAMs. Additionally, an identical clock must be applied to an OCM controller (DS or IS) and its corresponding BRAMs for any mode described above. Each controller (DS or IS) can be clocked at a frequency independent of each other.

ISOCM Instruction Fetching

The figures below show two back to back instruction fetches for (a) single cycle mode, and (b) multi-cycle mode with CPMC405CLOCK:BRAMISOCMCLK ratio of 2:1. Note that for both single cycle and multi-cycle mode, the maximum sustainable instruction fetch rate is one instruction per BRAMISOCMCLK period. For designs that utilize other integer clock ratios the important point to note from the timing diagram, is that the rising edge of the BRAMISOCMCLK defines the bus cycle.

In single cycle mode the very first instruction fetch requires four processor clock cycles to complete. The processor core can launch a new address, called back to back operation, as soon as the first address is latched into the OCM controller interface, which is internal to the processor block. The initial access consists of the following sequences:

- The CPU launches the instruction fetch address.
- The OCM controller translates the CPU order and routes the address and control signals onto the ISOCM bus.
- One wait state is introduced to permit the synchronous BRAM to access the data.
- The CPU stores the data.

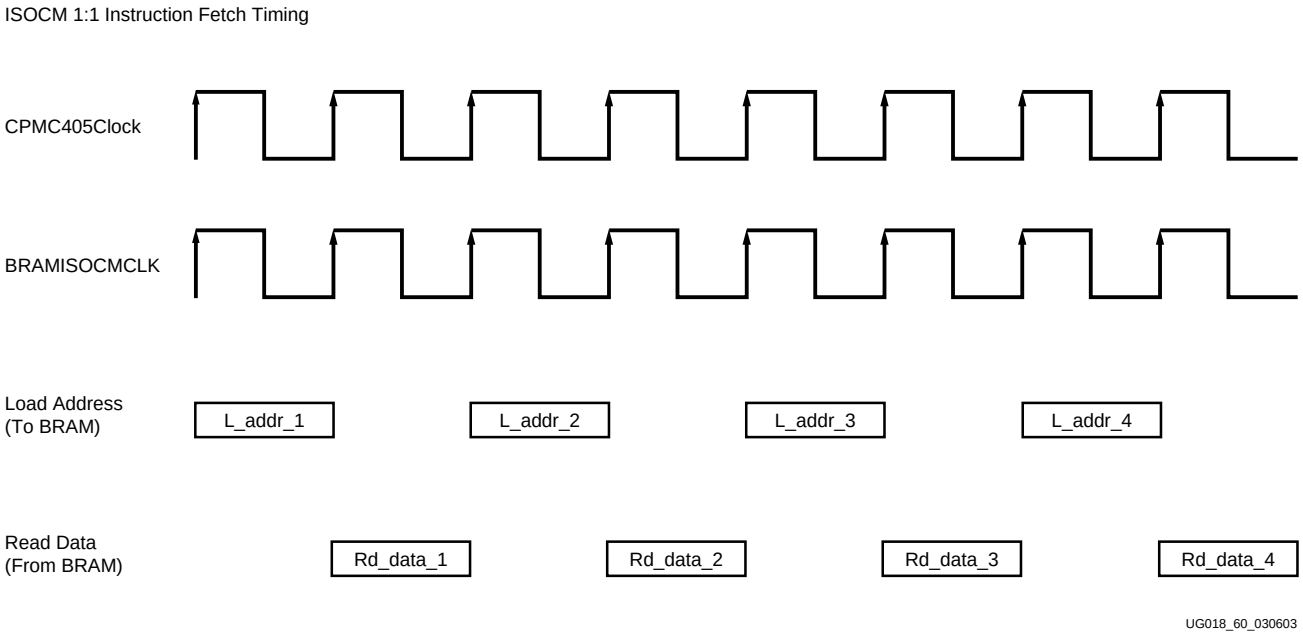

Figure 3-5: **Single Cycle Mode (1:1) Instruction Fetch Timing**

In multi-cycle mode, initial wait cycles are inserted until the CPMC405CLOCK and BRAMISOCMCLK rising edges are aligned.

After the initial startup latency, two instructions (64 bits) can be fetched every two BRAM clock cycles. If a branch instruction is taken, the instruction pipeline must be flushed, and the startup latency will again be encountered beginning with a new instruction address. So, in order to estimate the theoretical maximum number of instruction fetches per second on the OCM interface, counting the period of BRAM clock cycle should be used to determine the maximum throughput.

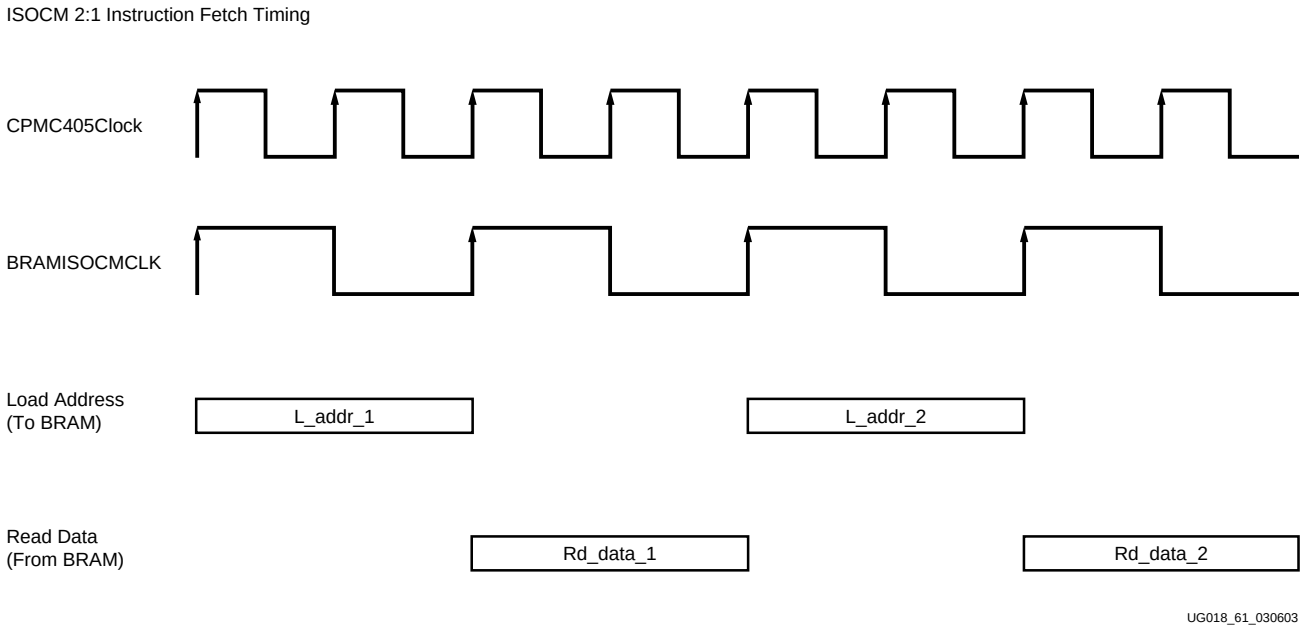

Figure 3-6: **Multi Cycle Mode (2:1) Instruction Fetch Timing**

In the figures above, L_addr_n refers to the OCM controller address outputs ISOCMBRAMRDADDR and Rd_data_n refers to the OCM controller instruction data bus inputs BRAMISOCMRDDBUS from the ISBRAM

DSOCM Data Load

Figure 3-7 and Figure 3-8 show two back to back loads for (a) single cycle mode, and (b) multi-cycle mode with CPMC405CLOCK:BRAMDSOCMCLK ratio of 2:1. Note that for both single cycle and multi-cycle mode, the maximum sustainable load completion is one load per two BRAMDSOCMCLK periods.

In single cycle mode, the very first load requires four processor clock cycles to complete. The processor core can launch a new address, called back to back operation, as soon as the first address is latched into the OCM controller interface, which is internal to the processor block. The initial access consists of the following sequences:

- The CPU launches the load address.
- The OCM controller translates the CPU order and routes the address and control signals onto the DSOCM bus.
- One wait state is introduced to permit the synchronous BRAM to access the data.

- The CPU stores the data into a general-purpose register.

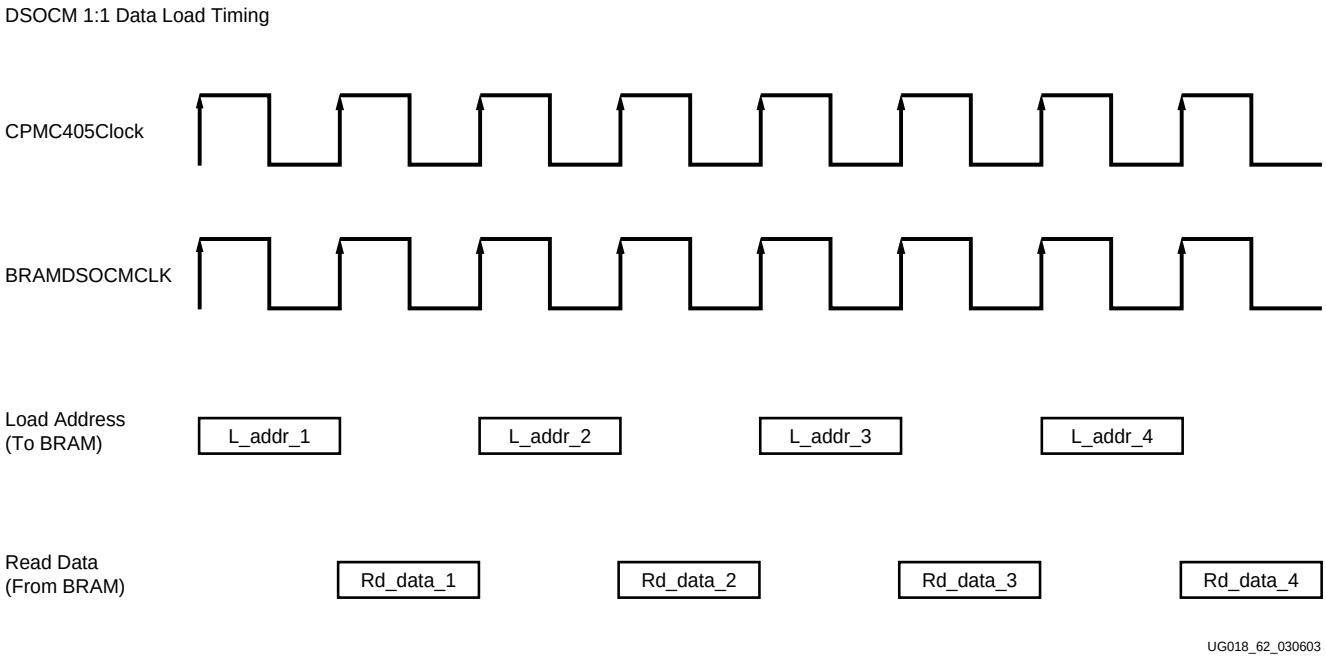

Figure 3-7: **Single Cycle Mode (1:1) Data Load Timing**

In multi-cycle mode, initial wait cycles are inserted until the CPMC405CLOCK and BRAMDSOCMCLK rising edges are aligned. After the initial startup latency, one load (32 bits) can be completed every two BRAMDSOCMCLK clock cycles. So, in order to estimate the theoretical maximum number of loads per second on the OCM interface, the period of BRAM clock should be used and establish throughput. Note that this is only an estimate for load performance..

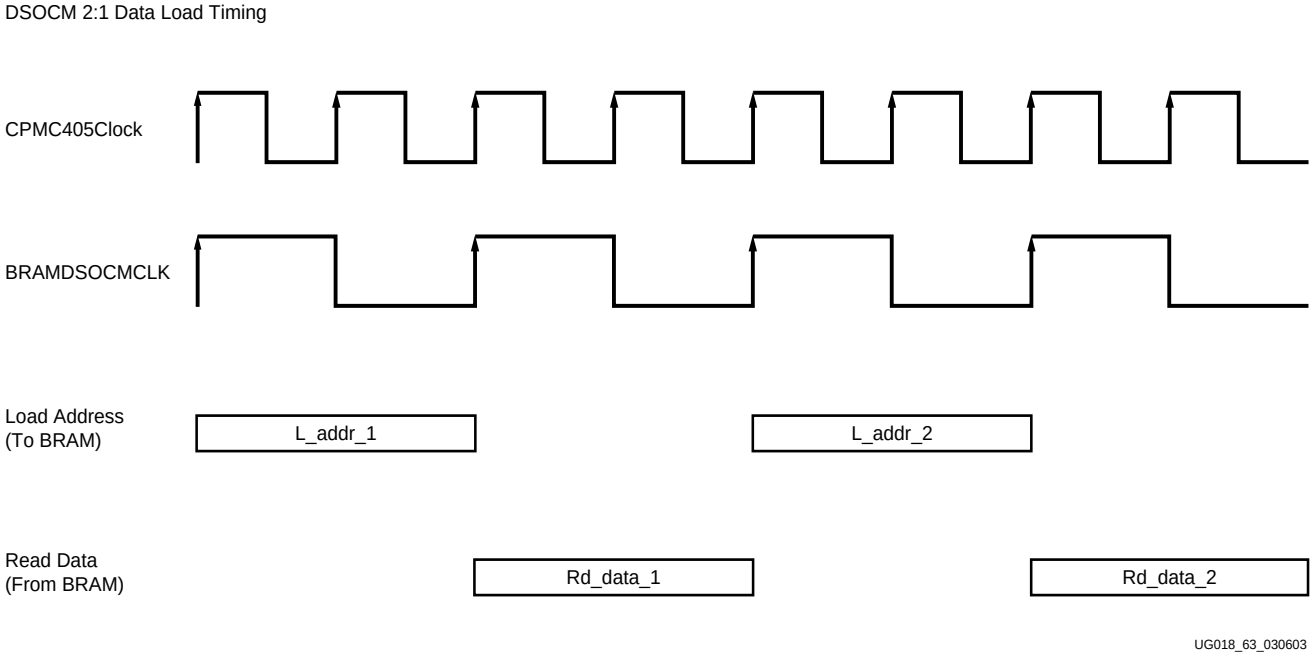

Figure 3-8: **Multi Cycle Mode (2:1) Data Load Timing**

In the figures above, L_addr_n refers to the OCM controller address outputs DSOCMBRAMRDADDR and Rd_data_n refers to the OCM controller data bus inputs BRAMDSOCMRDDBUS from the DSBRAMs

DSOCM Store

Figure 3-9 and Figure 3-10 below shows two back to back stores for (a) single cycle mode, and (b) multi-cycle mode with CPMC405CLOCK:BRAMDSOCMCLK ratio of 2:1. Note that for both single cycle and multi-cycle mode, the maximum sustainable store completion is one store per two BRAMDSOCMCLK periods.

In single cycle mode the very first store requires three processor clock cycles to complete. The processor core can launch a new address, called back to back operation, as soon as the first address is latched into the OCM controller interface, which is internal to the processor block. The initial access consists of the following sequences:

- The CPU launches the store address.
- The OCM controller translates the CPU order and routes the address, data, and control signals onto the DSOCM bus.
- The BRAM stores the data.

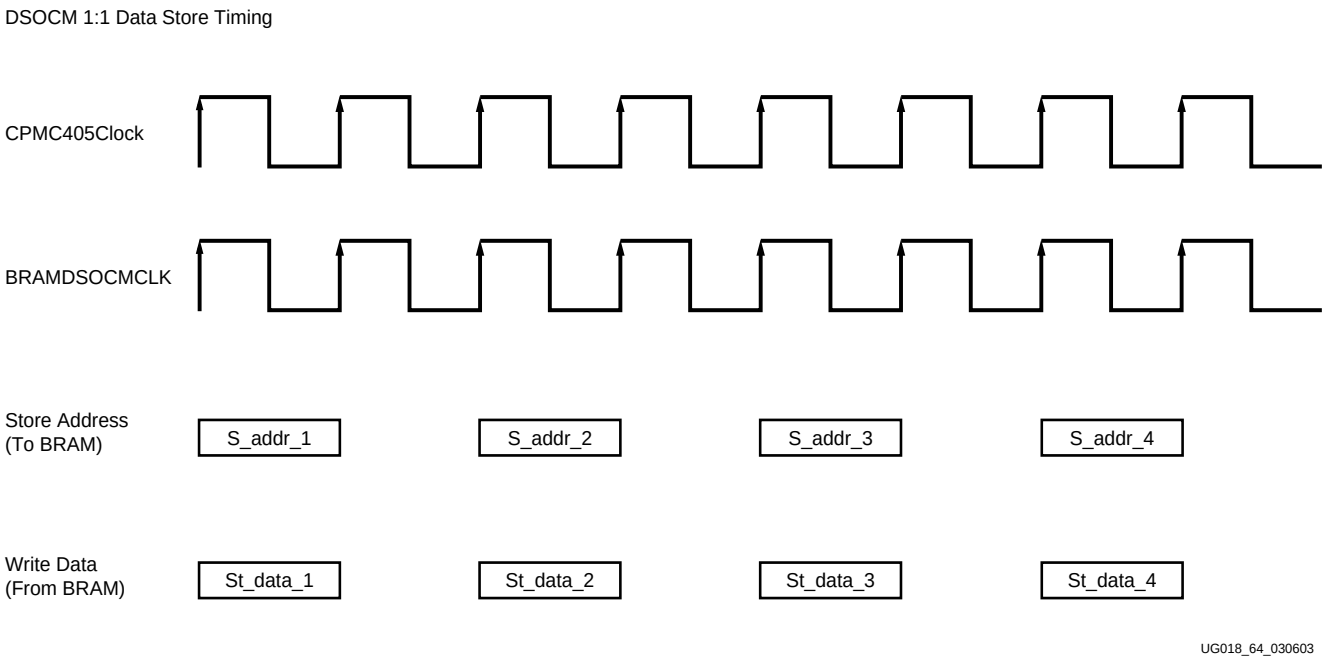

Figure 3-9: **Single Cycle Mode (1:1) Data Store Timing**

In multi-cycle mode, initial wait cycles are inserted until the CPMC405CLOCK and BRAMDSOCMCLK rising edges are aligned.

After the initial startup latency, one store (32 bits) can be completed every 2 BRAM clock cycles, or one store per two BRAMDSOCMCLK clock cycles. In order to estimate the absolute maximum number of stores per second on the OCM interface, the BRAM clock

period should be used. Note that this is only an estimate of store performance on the interface.

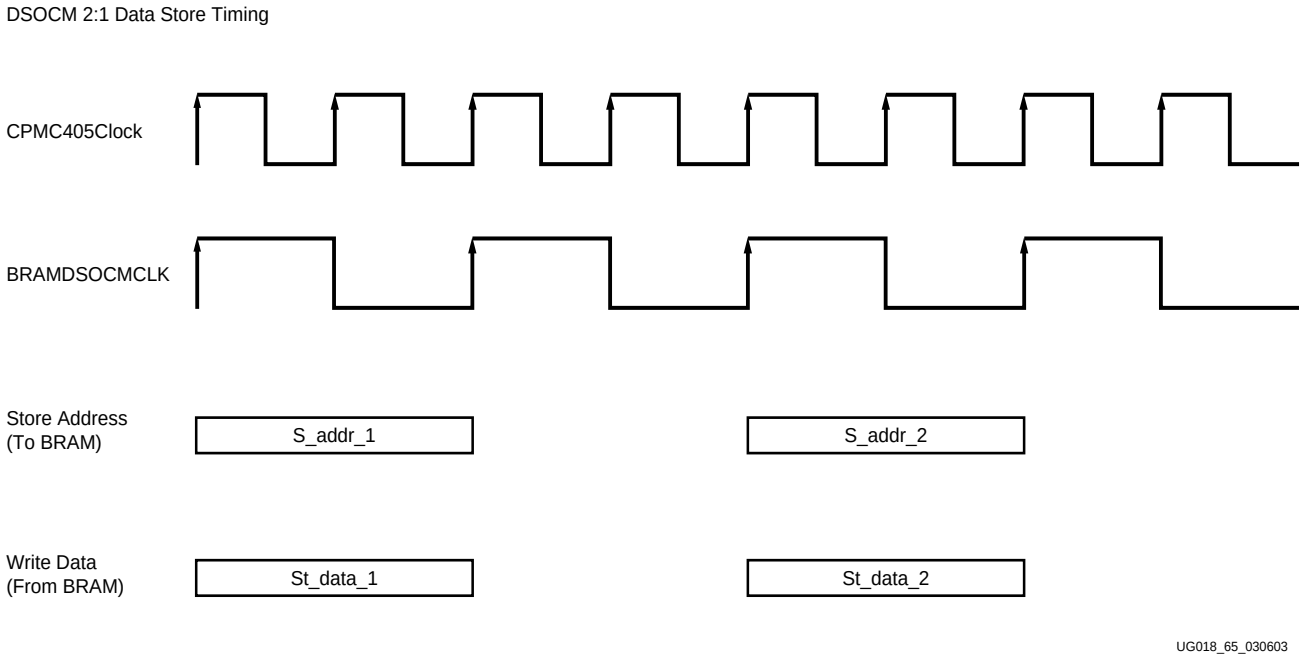

Figure 3-10: **Multi Cycle Mode (2:1) Data Store Timing**

In the figures above, S_addr_n refers to the OCM controller address outputs DSOCMBRAMWRADDR and St_data_n refers to the OCM controller data bus outputs BRAMDSOCMWRDBUS to the DSBRAMs.

Writing to ISBRAM

There are two ways to write into the ISBRAMs. Typically, ISBRAM can be initialized through the configuration bit-stream, during FPGA configuration. The "data2bram" software utility in the design flow tools is used to load BRAM with instructions as well as data. If the application permits, this eliminates the need for using DCR based writes through ISOCM controller.

Write accesses to ISOCM memory can be performed using the Device Control Register (DCR) bus. The OCM controllers are connected to a DCR bus which is internal to the processor block. After the DCR register "ISINIT" is initialized with the start address, every DCR write to the "ISFILL" register results in a write into BRAM. The least significant bit of the ISINIT register is used to control the initial state of the odd and even write enable outputs of the ISOCM. Every write to the ISFILL register causes the ISOCMBRAMEVENWRITEEN and ISOCMBRAMODDWRITEEN processor block outputs to toggle.

The BRAMISOCMCLK clock is the same for both "read" and "write" operations. All of the read and write interface signals must be included in determining the maximum frequency of operation for the OCM interface. These signals include write address, write data, read address, read data and write enable interface signals.

The figures below show the timing diagram for a write to ISBRAM in the case of (a) Single Cycle Mode and (b) Multi Cycle Mode. The timing interface between the OCM controller and the ISBRAM is always with respect to the BRAMISOCMCLK.

ISOCM 1:1 Write Timing

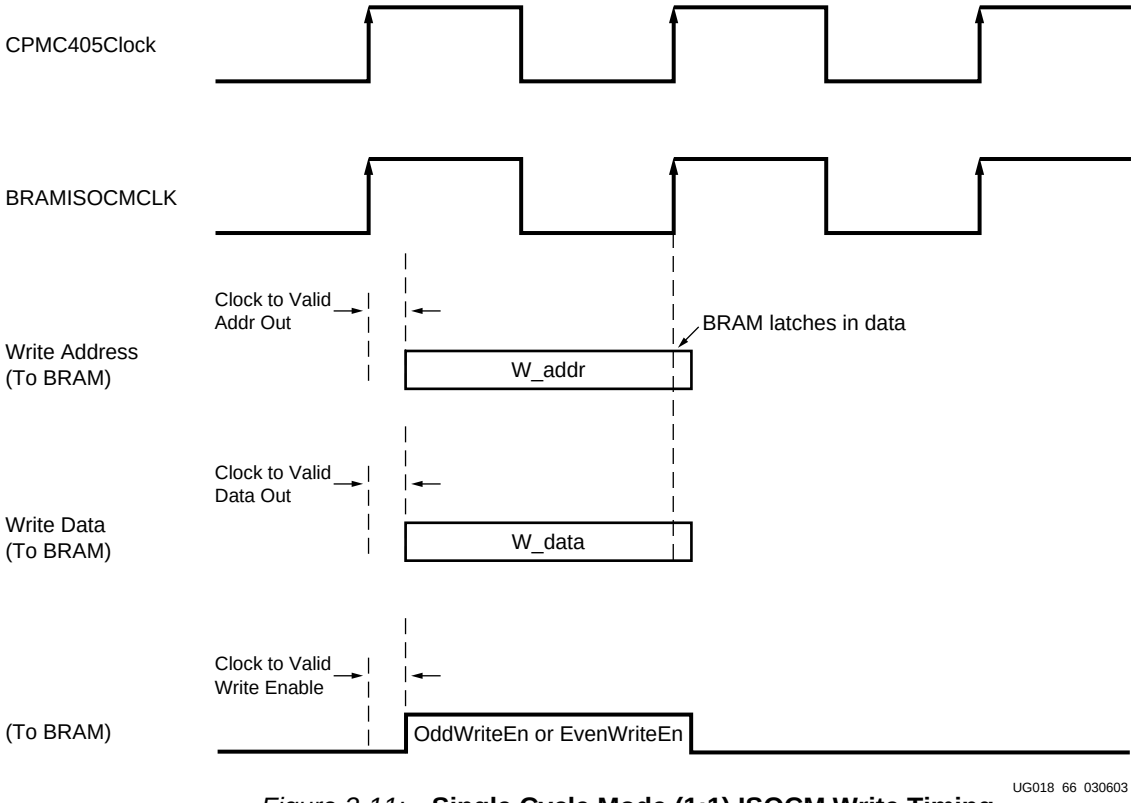

Figure 3-11: **Single Cycle Mode (1:1) ISOCM Write Timing**

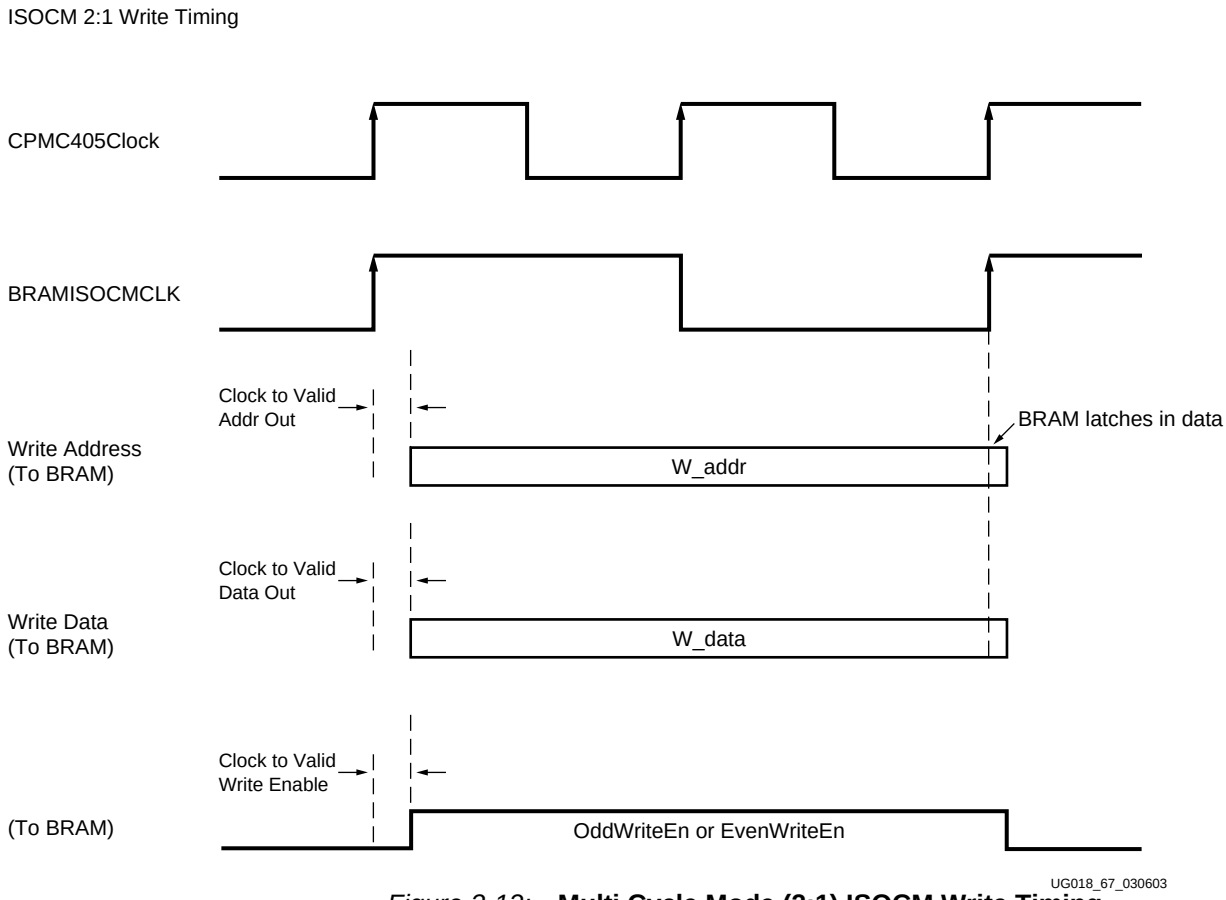

Figure 3-12: **Multi Cycle Mode (2:1) ISOCM Write Timing**

Programmer's Model

DCR Registers

From a system software perspective, the programmer has visibility and access to DCR registers within each interface. Typically, **mtdcr** and **mfdcr** instructions can be used to write and read from these registers, respectively. All registers are read / write.

Figure 3-13 lists all the DCR registers and the bit definitions of registers on the D-side OCM interface. Figure 3-14 lists all the DCR registers and the bit definitions of registers on the I-side OCM interface.

The I-OCM and D-OCM interfaces provide DCR registers (DSARC & ISARC) which set the top 8 (base) address bits of the I-OCM and D-OCM (CPU address bits 0:7). These addresses can be used to independently place the I-OCM and D-OCM in any 16MB address range. The I-OCM and D-OCM hardware outputs a maximum of 22 address bits (data-side address bits [8:29] and instruction-side address bits [8:28]) to address block RAM.

The OCM clock cycle modes are selected through the MULTICYCLEMODE control bits (DSOCMMCM and ISOCMMCM) in the DSCNTL and ISCNTL registers. The OCM

memory space decoders are enabled when the DSOCMEN and ISOCMEN bits are asserted in the DSCNTL and ISCNTL registers.

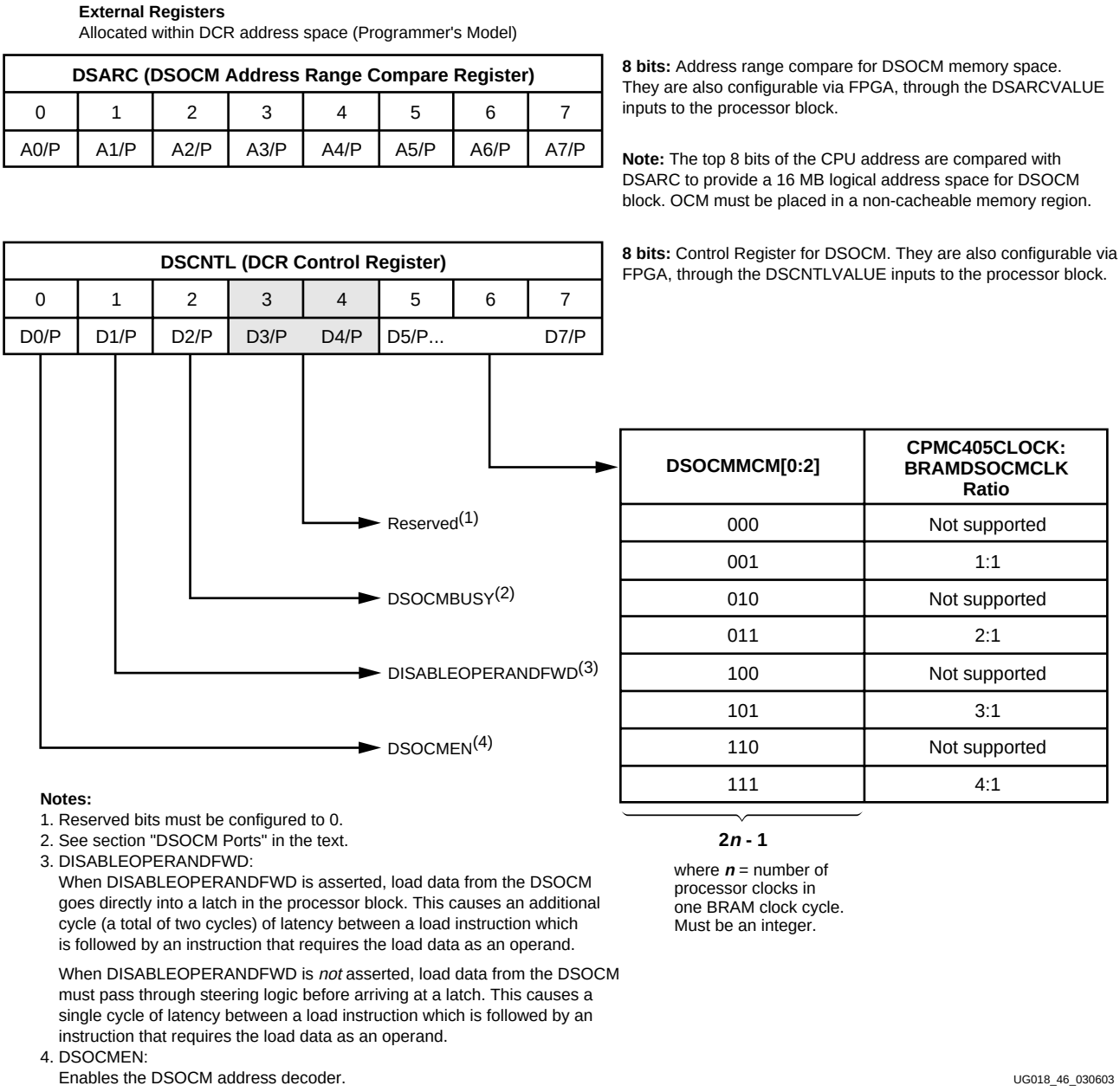

Figure 3-13: **DSOCM DCR Registers**

External Registers
Allocated within DCR address space (Programmer's Model)

ISARC (ISOCM Address Range Compare Register)							
0	1	2	3	4	5	6	7
A0/P	A1/P	A2/P	A3/P	A4/P	A5/P	A6/P	A7/P

8 bits: Address range compare for ISOCM memory space. They are also configurable via FPGA, through the ISARCVALUE inputs to the processor block.

Note: The top 8 bits of the CPU address are compared with ISARC to provide a 16 MB logical address space for ISOCM block. OCM must be placed in a non-cacheable memory region.

ISCNTL (ISOCM Control Register)							
0	1	2	3	4	5	6	7
D0/P	D1/P...			D4/P	D5/P...		D7/P

8 bits: Control Register for ISOCM. They are also configurable via FPGA, through the ISCNTLVALUE inputs to the processor block.

Reserved(1)

ISOCMEN(2)

ISOCMMCM[0:2]	CPMC405CLOCK: BRAMISOCMCLK Ratio
000	Not supported
001	1:1
010	Not supported
011	2:1
100	Not supported
101	3:1
110	Not supported
111	4:1

2n - 1

where ***n*** = number of processor clocks in one OCM clock cycle. Must be an integer.

Notes:
1. Reserved bits must be configured to 0.
2. ISOCMEN:
   Enables the DSOCM address decoder.

UG018_47_030603

Figure 3-14: **ISOCM DCR Registers**

ISINIT is a 22-bit register (A8-A29) and it is mapped on DCR write data bus bits (D8-D29). The write address on BRAM interface is (A8-A28) and address bit A29 is used to control ISOCMBRAMODDWRITEEN and ISOCMBRAMEVENWRITEEN signals. Each time that

the register ISFILL is written, there is one 32-bit instruction written into the BRAMs (odd or even, depending on value of address bit A29)..

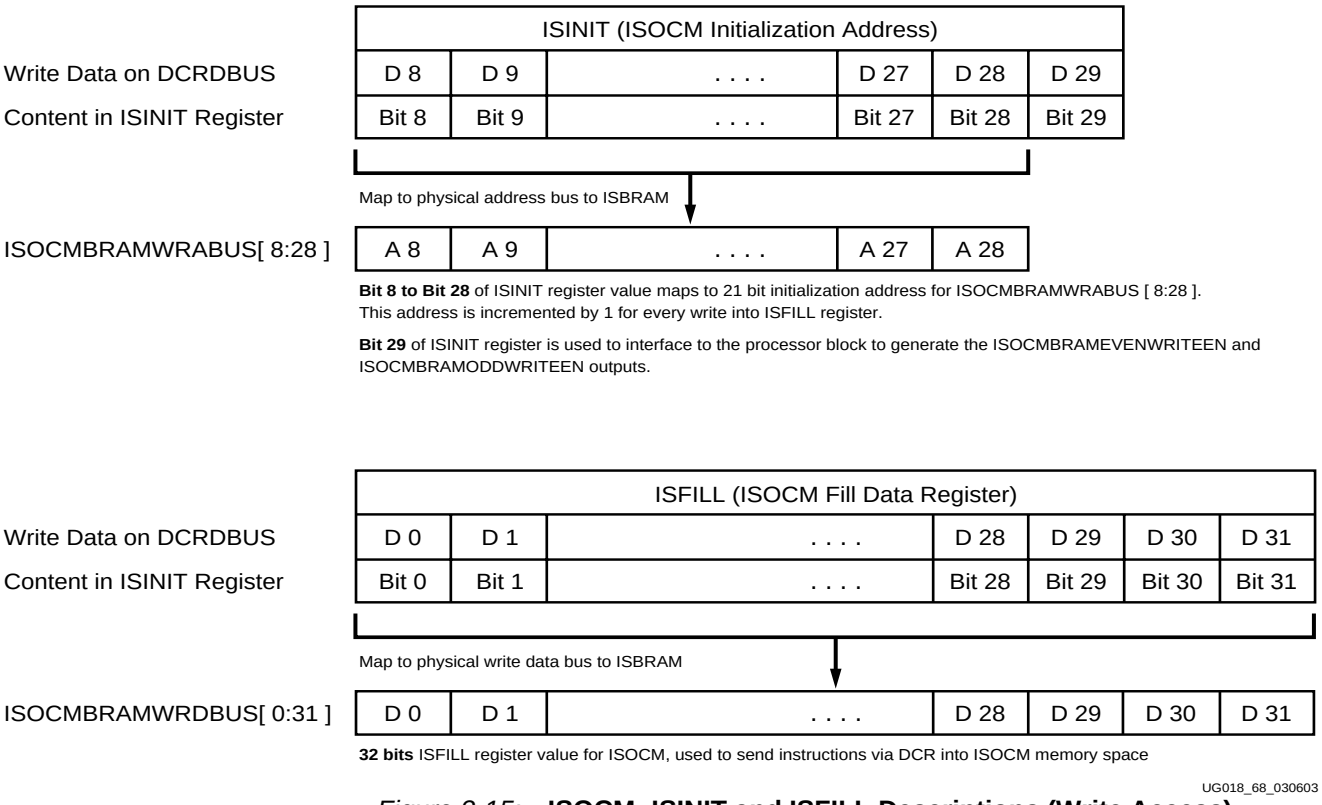

Figure 3-15: **ISOCM: ISINIT and ISFILL Descriptions (Write Access)**

If the ISINIT register is read back on DCR, then bits (A8-A29) are mapped onto DCR read bus bits (D0-D21) as shown below. Please note that the mapping for read access is different from write access. Figure 3-16 shows the read access descriptions of reading back from ISINIT register.

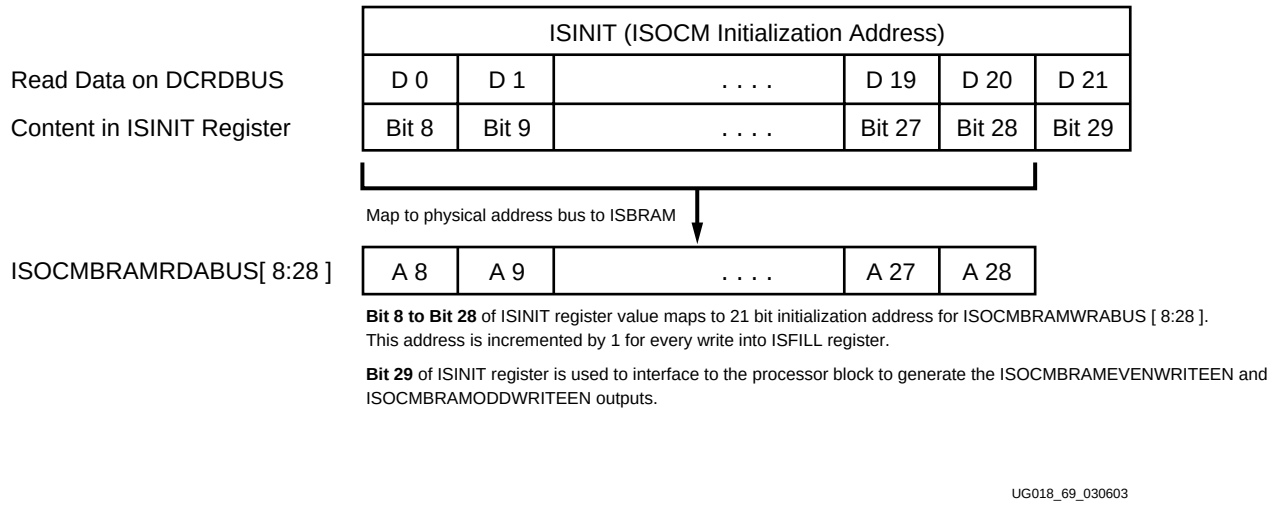

Figure 3-16: **ISOCM: ISINIT and ISFILL Descriptions (Read Access)**

Note that BRAMs can also be initialized through the configuration bit-stream, during FPGA configuration. The "data2bram" software utility in the design flow tools can be used to load BRAM with instructions and data.

References

1. *Virtex-II Pro Platform FPGA Handbook*
2. PowerPC Processor Reference Guide.

Application Notes

Interfacing to Block RAM

Refer to the *Virtex-II Pro Platform FPGA Handbook* for a detailed description of the block RAM resources. The block RAMs are synchronous and provide a dual-port access to 16K bits of data, not including parity. A useful feature is the different configurations each block RAM can have. This is shown in Table 3-1.

Table 3-1: **Block RAM Configurations in a Virtex-II FPGA**

Word Width	Depth	Address	Data
1	16,384	A<13:0>	D<0>
2	8192	A<13:1>	D<1:0>
4	4096	A<13:2>	D<3:0>
9	2048	A<13:3>	D<7:0> + P<0>
18	1024	A<13:4>	D<15:0> + P<1:0>
36	512	A<13:5>	D<31:0> + P<3:0>

Block RAMs can be used as the source of data and instruction storage space for OCM. Since the block RAMs are not dedicated to the OCM, any size of block RAM desired can be used.

The **Application Example** section shows a reference OCM design with both ISBRAM and DSBRAM.

ISOCM

The instruction side data bus is 64-bits wide. A majority of applications will benefit from a higher processor frequency with ISBRAM in multi-cycle mode versus ISBRAM operation in single cycle mode. In general, designers will make performance tradeoffs for single cycle mode, with reduced processor frequency, versus multi-cycle mode of operation where processor is running at highest possible operating frequency. For example, to achieve the highest instruction fetch performance in multi-cycle mode:

- Use two 512x36-bit BRAM devices and a 2:1 or 3:1 clock ratio (depending on the timing achieved after place and route). This is a minimum BRAM configuration on ISOCM.
- Avoid decoding on the on the address bus by gradually reducing the data bus width and increasing the address depth, per BRAM, to obtain a higher amount of memory. Use multiple BRAM elements to create the 64-bit data bus.

- Use the configuration bitstream to initialize the ISBRAM. Implementing the write only port increases routing congestion around the processor block and attached memory. See Table 3-2 for supported configurations.
- For memory requirements higher than 128KB, implement address decoding to select between two or more banks of ISBRAM. Multiplex read data between two or more banks of BRAM. This method will typically result in the slowest BRAMISOCMCLK speed.

DSOCM

The data side data bus is 32-bits wide. A majority of applications will benefit from a higher processor frequency with DSBRAM in multi-cycle mode versus DSBRAM operation in single cycle mode. In general, designers will make performance tradeoffs for single cycle mode, with reduced processor frequency, versus multi-cycle mode of operation where processor is running at highest possible operating frequency. For example, to achieve the highest data load / store performance in multi-cycle mode:

- Use four 2Kx9-bit BRAM devices and a 2:1 or 3:1 clock ratio (depending on the timing achieved after place and route). This is the minimum BRAM configuration on DSOCM.
- Avoid decoding on the address bus by gradually reducing the data bus width and increasing the address depth, per BRAM, to obtain a higher amount of memory. Use multiple BRAM elements to create the 32-bit data bus. See Table 3-2 for supported configurations.
- For memory requirements higher than 64KB, implement address decoding to select between two or more banks of DSBRAM. Multiplex read data between two or more banks of BRAM. This method will typically result in the slowest BRAMDSOCMCLK speed.

Size vs. Performance

For a given processor clock frequency, the larger the size of block RAM, the greater the associated performance penalty. Past a certain threshold of BRAM, the OCM may not function correctly if the block RAM access time is greater than the microprocessor clock period. For designs with large amounts of block RAM, the designer may need to use multi-cycle mode and / or reduce the microprocessor clock frequency.

There may be applications associated with OCM that necessitate a dual-port implementation of block RAMs rather than the default single-port implementation. Table 3-2 shows a sample block RAM implementation for the I-side and D-side OCM interfaces, illustrating the recommended method to connect ISBRAM and DSBRAM. The data and control signals will be routed differently depending upon the total amount of memory required for the application.

The instruction-side and data-side clock mode control bits (ISOCMMCM and DSOCMMCM) will change depending upon processor clock frequency, BRAM access time, signal loading, and signal routing delays. As the total amount of OCM increases, the performance decreases. The minimum DSBRAM is 8KB and the minimum ISBRAM is 4KB.

When 14 LSB bits of Processor Block Address outputs are used, 64KB / 128KB is addressable from DSOCM and ISOCM interface using the fourteen least-significant bits of address outputs from processor block. Maximum of 16MB is addressable using all addresses from processor block with increased access time for BRAMs.

Table 3-2: **Typical ISBRAM and DSBRAM Configuration Range Using the 14 Least-Significant Processor Block Address Outputs**

Total Memory Used	32-Bit Data OCM		64-Bit Instruction OCM	
	BRAM Quantity	BRAM Port Size	BRAM Quantity	BRAM Port Size
128KB	N / A	N / A	64	16K x 1
64KB	32	16K x 1	32	8K x 2
32KB	16	8K x 2	16	4K x 4
16KB	8	4K x 4	8	2K x 9
8KB	4	2K x 9	4	1K x 16
4KB	Note (1)	1K x 16	2	512 x 36
2KB	Note (1)	512 x 36	Note (2)	Note(3)

Notes:

1. The processor byte write function is not supported for this configuration.
2. The ISBRAM must be 64 bits wide.
3. For larger ISBRAM and DSBRAM configurations, additional address outputs can be used with increased access times.

Application Example

This section contains a Verilog reference design containing an OCM interface to a 4KB ISBRAM and a dual port 8KB DSBRAM.

```
//***************************************************************
// File: ocm_example.v
//
// PowerPC 405 On Chip Memory Controller Application Example
// Features:
//   1. Digital Clock Manager for PPC/OCM/BRAM clock generation.
//   2. REFCLK input generates 2X and 4X clocks.
//   3. On Chip Memory Controller will use CLK2X.
//   4. DSBRAM and ISBRAM also use CLK2X.
//   5. PowerPC 405 will use CLK4X.
//   6. 4KB ISBRAM dual ported design for DCR write access.
//   7. 8KB DSBRAM dual ported design for FPGA R/W access.
//
// Application Notes:
//   1. User connects this module up to the Processor Block.
//   2. Processor block inputs not shown in this example:
//      A. DSARCVALUE  = 8'hF4
//      B. DSCNTLVALUE = 8'h83 (2:1 PPC/OCM clock ratio)
//      C. ISARCVALUE  = 8'hFF
//      D. ISCNTLVALUE = 8'h83 (2:1 PPC/OCM clock ratio)
//   3. Processor Block OCM outputs not used: DSOCMBUSY
//***************************************************************


module OCM ( REFCLK, BRAMDSOCMCLK, BRAMISOCMCLK, CPMC405CLOCK, RST,
```

```
          DSOCMBRAMABUS, DSOCMBRAMEN, DSOCMBRAMWRDBUS, BRAMDSOCMRDDBUS,
            DSOCMBRAMBYTEWRITE, ADDRB, ENB, CLKB, DINB, DOUTB, WEB,
            ISOCMBRAMRDABUS, ISOCMBRAMWRABUS, ISOCMBRAMEN,
            ISOCMBRAMWRDBUS, ISOCMBRAMODDWRITEEN, ISOCMBRAMEVENWRITEEN,
            BRAMISOCMRDDBUS );

// Digital Clock Manager I/O:
 input           REFCLK;
 output          CPMC405CLOCK;  //PowerPC405 clock
 output          RST;

//*****************************************************************
// Data Side On Chip Memory I/O:
// NOTE: OCM interface uses big endian bit format.
//*****************************************************************

 input  [8:29] DSOCMBRAMABUS;      // A29 is LSB
 input         DSOCMBRAMEN;        // BRAM enable
 output        BRAMDSOCMCLK;       // DSOCM and DSBRAM clock
 input  [0:31] DSOCMBRAMWRDBUS;    // Bit 31 is the LSB!
 output [0:31] BRAMDSOCMRDDBUS;    // Bit 31 is the LSB!
 input   [0:3] DSOCMBRAMBYTEWRITE; // Bit 0 controls the MSB byte

//*****************************************************************
// FPGA logic interface to "B" side of Dual Port BRAM:
// NOTE: FPGA interface uses little endian bit format.
//*****************************************************************

 input   [10:0]  ADDRB; // FPGA address, A0 is LSB
 input           ENB;   // FPGA enable
 input           CLKB;  // FPGA clock
 input   [31:0]  DINB;  // FPGA side data input
 output  [31:0]  DOUTB; // FPGA side data output
 input   [ 3:0]  WEB;   // FPGA write enables

//*****************************************************************
// Instruction Side On Chip Memory I/O:
// NOTE: OCM interface uses big endian bit format.
//*****************************************************************

 input  [8:28]  ISOCMBRAMWRABUS;       // BRAM Write Address
 input  [8:28]  ISOCMBRAMRDABUS;       // BRAM Read Address
 input          ISOCMBRAMEN;           // BRAM enable
 output         BRAMISOCMCLK;          // ISOCM and ISBRAM clock
 input  [0:31]  ISOCMBRAMWRDBUS;       // Store data bus to BRAM
 input          ISOCMBRAMODDWRITEEN;   // Odd word write enable
 input          ISOCMBRAMEVENWRITEEN;  // Even word write enable
 output [0:63]  BRAMISOCMRDDBUS;      // Load data bus from BRAM


// Instantiate OCM Application Example lower level modules here:

wire      OCMCLK;
wire      CPMC405CLOCK;

assign BRAMDSOCMCLK = OCMCLK;
assign BRAMISOCMCLK = OCMCLK;

ocmclk     OCM1
```

```
     (
     .REFCLK            ( REFCLK ),
     .CLK               ( ),
     .CLK2X             ( OCMCLK ),
     .CLKMULT           ( CPMC405CLOCK ),
     .RST               ( RST )
     );

DS_bram_wrap  OCM2
     (
     .DSOCMBRAMABUS       ( DSOCMBRAMABUS[19:29] ),
     .DSOCMBRAMEN         ( DSOCMBRAMEN ),
     .BRAMDSOCMCLK        ( OCMCLK ),
     .DSOCMBRAMWRDBUS     ( DSOCMBRAMWRDBUS ),
     .BRAMDSOCMRDDBUS     ( BRAMDSOCMRDDBUS ),
     .DSOCMBRAMBYTEWRITE  ( DSOCMBRAMBYTEWRITE ),
     .ADDRB               ( ADDRB ),
     .ENB                 ( ENB ),
     .CLKB                ( CLKB ),
     .DINB                ( DINB ),
     .DOUTB               ( DOUTB ),
     .WEB                 ( WEB )
     );

IS_bram_wrap  OCM3
     (
     .ISOCMBRAMRDABUS         ( ISOCMBRAMRDABUS[20:28] ),
     .ISOCMBRAMWRABUS         ( ISOCMBRAMWRABUS[20:28] ),
     .ISOCMBRAMEN             ( ISOCMBRAMEN ),
     .BRAMISOCMCLK            ( OCMCLK ),
     .ISOCMBRAMWRDBUS         ( ISOCMBRAMWRDBUS ),
     .ISOCMBRAMODDWRITEEN     ( ISOCMBRAMODDWRITEEN ),
     .ISOCMBRAMEVENWRITEEN    ( ISOCMBRAMEVENWRITEEN ),
     .BRAMISOCMRDDBUS         ( BRAMISOCMRDDBUS[0:63] )
     );


endmodule // OCM



module ocmclk (REFCLK, CLK, CLK2X, CLKMULT, RST );

   input      REFCLK;      // input clock
   output     CLK;         // buffered 1X clock
   output     CLK2X;       // buffered 2X clock
   output     CLKMULT;     // buffered 4X clock
   output     RST;         // logic RST

   wire          dcm_locked, refclk_in;
   wire          clk_i,     clk2x_i,     clkmult_i;
   reg           RST;
   reg  [2:0]    startup_counter;

// Clock Generation, REFCLK = CLK, Multiply Clock Up to CPU Freq
   IBUFG  buf0 (.I(REFCLK), .O(refclk_in));

// Set to 4X REFCLK for CPU clock multiplier
   defparam dcm1.CLKFX_MULTIPLY = 12'h004;
```

```
DCM dcm1 (.CLKFB     ( CLK ),
          .CLKIN     ( refclk_in ) ,
          .DSSEN     ( 1'b0 ),
          .PSCLK     ( 1'b0 ),
          .PSEN      ( 1'b0 ),
          .PSINCDEC  ( 1'b0 ),
          .RST       ( 1'b0 ),
          .CLK0      ( clk_i ),
          .CLK90     ( ),
          .CLK180    ( ),
          .CLK270    ( ),
          .CLK2X     ( clk2x_i ),
          .CLK2X180  ( ),
          .CLKDV     ( ),
          .CLKFX     ( clkmult_i ),
          .CLKFX180  ( ),
          .LOCKED    ( dcm_locked ),
          .PSDONE    ( ),
          .STATUS    ( ) );

BUFG   buf1 (.I ( clk_i ),         .O ( CLK ));
BUFG   buf2 (.I ( clk2x_i ),       .O ( CLK2X ));
BUFG   buf3 (.I ( clkmult_i ),     .O ( CLKMULT ));


// Startup Reset Signal
always @ ( posedge CLK )
  if ( !dcm_locked )
       startup_counter <= 3'b0;
  else if ( startup_counter != 3'b111 )
       startup_counter <= startup_counter + 1;

always @ ( posedge CLK or negedge dcm_locked )
  if ( !dcm_locked )
       RST <= 1'b1;
  else
       RST <= ( startup_counter != 3'b111 );

endmodule  //ocmclk



//*****************************************************
//  Data Side BlockSelect RAM interface module.
//  DSBRAMS are enabled once DSOCMEN is asserted.
//
//*****************************************************

module DS_bram_wrap (DSOCMBRAMABUS,
                     DSOCMBRAMEN,
                     BRAMDSOCMCLK,
                     DSOCMBRAMWRDBUS,
                     BRAMDSOCMRDDBUS,
                     DSOCMBRAMBYTEWRITE,
                     ADDRB,
                     ENB,
                     CLKB,
                     DINB,
```

```
                    DOUTB,
                    WEB
                   );

// DSOCM controller interface to "A" side of Dual Port BRAM.
// Processor side supports byte writes to data buffer.

input  [10:0] DSOCMBRAMABUS;      // A00 is the least significant
input         DSOCMBRAMEN;         // BRAM chip select
input        BRAMDSOCMCLK;       // DSOCM clock with 180 deg phase shift
input  [31:0] DSOCMBRAMWRDBUS;     // Din0 is the LSB
output [31:0] BRAMDSOCMRDDBUS;     // Dout0 is the LSB
input   [3:0] DSOCMBRAMBYTEWRITE;  // Bit 3 controls the MSB byte

// FPGA logic interface to "B" side of Dual Port BRAM:
input   [10:0]  ADDRB;     // FPGA address
input           ENB;       // FPGA enable
input           CLKB;      // FPGA clock
input   [31:0]  DINB;      // FPGA side data input
output  [31:0]  DOUTB;     // FPGA side data output
input   [ 3:0]  WEB;       // FPGA write enables


  RAMB16_S9_S9  u3 ( // PPC405 Byte 0 (MSB)
                     .WEA     ( DSOCMBRAMBYTEWRITE[3] ),
                     .ENA     ( DSOCMBRAMEN ),
                     .SSRA    ( 1'b0 ),
                     .CLKA    ( BRAMDSOCMCLK ),
                     .ADDRA   ( DSOCMBRAMABUS[10:0] ),
                     .DIPA    ( 1'b0 ),
                     .DIA     ( DSOCMBRAMWRDBUS[31:24] ),
                     .DOA     ( BRAMDSOCMRDDBUS[31:24] ),
                     .DOPA    ( ),

                     // FPGA I/F Byte 3 (MSB)
                     .WEB     ( WEB[3] ),
                     .ENB     ( ENB ),
                     .SSRB    ( 1'b0 ),
                     .CLKB    ( CLKB ),
                     .ADDRB   ( ADDRB[10:0] ),
                     .DIPB    ( 1'b0 ),
                     .DIB     ( DINB[31:24] ),
                     .DOB     ( DOUTB[31:24] ),
                     .DOPB    ( )
                     );

  RAMB16_S9_S9  u2 (// PPC405 Byte 1
                     .WEA     ( DSOCMBRAMBYTEWRITE[2] ),
                     .ENA     ( DSOCMBRAMEN ),
                     .SSRA    ( 1'b0 ),
                     .CLKA    ( BRAMDSOCMCLK ),
                     .ADDRA   ( DSOCMBRAMABUS[10:0] ),
                     .DIPA    ( 1'b0 ),
                     .DIA     ( DSOCMBRAMWRDBUS[23:16]),
                     .DOA     ( BRAMDSOCMRDDBUS[23:16]),
                     .DOPA    ( ),

                     // FPGA I/F Byte 2
                     .WEB     ( WEB[2] ),
```

```
                        .ENB     ( ENB ),
                        .SSRB    ( 1'b0 ),
                        .CLKB    ( CLKB ),
                        .ADDRB   ( ADDRB[10:0] ),
                        .DIPB    ( 1'b0 ),
                        .DIB     ( DINB[23:16] ),
                        .DOB     ( DOUTB[23:16] ),
                        .DOPB    ( )
                        );

    RAMB16_S9_S9  u1 (// PPC405 Byte 2
                        .WEA     ( DSOCMBRAMBYTEWRITE[1] ),
                        .ENA     ( DSOCMBRAMEN ),
                        .SSRA    ( 1'b0 ),
                        .CLKA    ( BRAMDSOCMCLK ),
                        .ADDRA   ( DSOCMBRAMABUS[10:0] ),
                        .DIPA    ( 1'b0 ),
                        .DIA     ( DSOCMBRAMWRDBUS[15:8] ),
                        .DOA     ( BRAMDSOCMRDDBUS[15:8] ),
                        .DOPA    ( ),

                        // FPGA I/F Byte 1
                        .WEB     ( WEB[1] ),
                        .ENB     ( ENB ),
                        .SSRB    ( 1'b0 ),
                        .CLKB    ( CLKB ),
                        .ADDRB   ( ADDRB[10:0] ),
                        .DIPB    ( 1'b0 ),
                        .DIB     ( DINB[15:8] ),
                        .DOB     ( DOUTB[15:8] ),
                        .DOPB    ( )
                        );

    RAMB16_S9_S9  u0 (// PPC405 Byte 3 (LSB)
                        .WEA     ( DSOCMBRAMBYTEWRITE[0] ),
                        .ENA     ( DSOCMBRAMEN ),
                        .SSRA    ( 1'b0 ),
                        .CLKA    ( BRAMDSOCMCLK ),
                        .ADDRA   ( DSOCMBRAMABUS[10:0] ),
                        .DIPA    ( 1'b0 ),
                        .DIA     ( DSOCMBRAMWRDBUS[7:0] ),
                        .DOA     ( BRAMDSOCMRDDBUS[7:0] ),
                        .DOPA    ( ),

                        // FPGA I/F Byte 0 (LSB)
                        .WEB     ( WEB[0] ),
                        .ENB     ( ENB ),
                        .SSRB    ( 1'b0 ),
                        .CLKB    ( CLKB ),
                        .ADDRB   ( ADDRB[10:0] ),
                        .DIPB    ( 1'b0 ),
                        .DIB     ( DINB[7:0] ),
                        .DOB     ( DOUTB[7:0] ),
                        .DOPB    ( )
                        );

endmodule  //DS_bram_wrap
```

```
//***********************************************************
//  Instruction Side BRAM interface module.
//  ISBRAMS are enabled only when ISOCMBRAMEN is asserted.
//***********************************************************

module IS_bram_wrap
     (
     ISOCMBRAMRDABUS,
     ISOCMBRAMWRABUS,
     ISOCMBRAMEN,
     BRAMISOCMCLK,
     ISOCMBRAMWRDBUS,
     ISOCMBRAMODDWRITEEN,
     ISOCMBRAMEVENWRITEEN,
     BRAMISOCMRDDBUS
     );

 input  [8:0]   ISOCMBRAMWRABUS;       // BRAM Write Address
 input  [8:0]   ISOCMBRAMRDABUS;       // BRAM Read Address
 input          ISOCMBRAMEN;           // BRAM enable
 input          BRAMISOCMCLK;          // BRAM clock
 input  [31:0]  ISOCMBRAMWRDBUS;       // Store data bus to BRAM
 input          ISOCMBRAMODDWRITEEN;   // BRAM Odd word write enable
 input          ISOCMBRAMEVENWRITEEN;  // BRAM Even word write enable
 output [0:63]  BRAMISOCMRDDBUS;       // Load data bus from BRAM


  RAMB16_S36_S36  u2 (.DOA     ( BRAMISOCMRDDBUS[32:63] ),
                      .DOB     ( ),
                      .DOPA    ( ),
                      .DOPB    ( ),
                      .ADDRA   ( ISOCMBRAMRDABUS[8:0] ),
                      .CLKA    ( BRAMISOCMCLK ),
                      .DIA     ( ISOCMBRAMWRDBUS[31:0] ),
                      .DIPA    ( 4'b0000 ),
                      .ENA     ( ISOCMBRAMEN ),
                      .SSRA    ( 1'b0 ),
                      .WEA     ( 1'b0 ),
                      .ADDRB   ( ISOCMBRAMWRABUS[8:0] ),
                      .CLKB    ( BRAMISOCMCLK ),
                      .DIB     ( ISOCMBRAMWRDBUS[31:0] ),
                      .DIPB    ( 4'b0000 ),
                      .ENB     ( ISOCMBRAMEN ),
                      .SSRB    ( 1'b0 ),
                      .WEB     ( ISOCMBRAMODDWRITEEN )
                     );

  RAMB16_S36_S36  u1 (.DOA     ( BRAMISOCMRDDBUS[0:31] ),
                      .DOB     ( ),
                      .DOPA    ( ),
                      .DOPB    ( ),
                      .ADDRA   ( ISOCMBRAMRDABUS[8:0] ),
                      .CLKA    ( BRAMISOCMCLK ),
                      .DIA     ( ISOCMBRAMWRDBUS[31:0] ),
                      .DIPA    ( 4'b0000 ),
                      .ENA     ( ISOCMBRAMEN ),
                      .SSRA    ( 1'b0 ),
                      .WEA     ( 1'b0 ),
```

```
                                .ADDRB   ( ISOCMBRAMWRABUS[8:0] ),
                                .CLKB    ( BRAMISOCMCLK ),
                                .DIB     ( ISOCMBRAMWRDBUS[31:0] ),
                                .DIPB    ( 4'b0000 ),
                                .ENB     ( ISOCMBRAMEN ),
                                .SSRB    ( 1'b0 ),
                                .WEB     ( ISOCMBRAMEVENWRITEEN )
                               );

endmodule  //IS_bram_wrap
```

Appendix A

RISCWatch and RISCTrace Interfaces

This appendix summarizes the interface requirements between the PPC405x3 and the RISCWatch and RISCTrace tools.

The requirement for separate JTAG and trace connectors is being replaced with a single Mictor connector to improve the electrical and mechanical characteristics of the interface. Pin assignments for the Mictor connector are included in the signal-mapping tables.

RISCWatch Interface

The RISCWatch tool communicates with the PPC405x3 using the JTAG and debug interfaces. It requires a 16-pin, male 2x8 header connector located on the target development board. The layout of the connector is shown in Figure A-1 and the signals are described in Table A-1. A mapping of PPC405x3 to RISCWatch signals is provided in Table A-2. At the board level, the connector should be placed as close as possible to the processor chip to ensure signal integrity. Position 14 is used as a connection key and does not contain a pin.

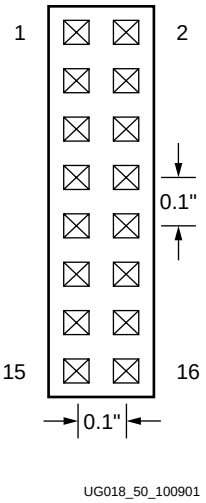

Figure A-1: **JTAG-Connector Physical Layout**

Table A-1: **JTAG Connector Signals for RISCWatch**

Pin	RISCWatch		Description
	I/O	Signal Name	
1	Input	TDO	JTAG test-data out.
2	No Connect	Reserved	
3	Output	TDI[1]	JTAG test-data in.
4	Output	$\overline{\text{TRST}}$	JTAG test reset.
5	No Connect	Reserved	
6	Output	+Power[2]	Processor power OK
7	Output	TCK[3]	JTAG test clock.
8	No Connect	Reserved	
9	Output	TMS	JTAG test-mode select.
10	No Connect	Reserved	
11	Output	$\overline{\text{HALT}}$	Processor debug halt mode.
12	No Connect	Reserved	
13	No Connect	Reserved	
14	KEY	No pin should be placed at this position.	
15	No Connect	Reserved	
16		GND	Ground

Notes:
1. A 10KΩ pull-up resistor should be connected to this signal to reduce chip-power consumption. The pull-up resistor is not required.
2. The +POWER signal, is provided by the board, and indicates whether the processor is operating. This signal does not supply *power* to the debug tools or to the processor. A series resistor (1KΩ or less) should be used to provide short-circuit current-limiting protection.
3. A 10KΩ pull-up resistor must be connected to these signals to ensure proper chip operation when these inputs are not used.

Table A-2: **PPC405x3 to RISCWatch Signal Mapping**

PPC405x3		RISCWatch		JTAG Connector Pin	Mictor Connector Pin
Signal	I/O	Signal	I/O		
C405JTGTDO[1]	Output	TDO	Input	1	11
JTGC405TDI	Input	TDI	Output	3	19
JTGC405TRSTNEG	Input	$\overline{\text{TRST}}$	Output	4	21
JTGC405TCK	Input	TCK	Output	7	15
JTGC405TMS	Input	TMS	Output	9	17
DBGC405DEBUGHALT[2]	Input	$\overline{\text{HALT}}$	Output	11	7

Notes:
1. This signal must be driven by a tri-state device using C405JTGTDOEN as the enable signal.
1. This signal must be inverted between the PPC405x3 and the RISCWatch.

RISCTrace Interface

The RISCTrace tool communicates with the PPC405x3 using the trace interface. It requires a 20-pin, male 2x10 header connector (3M 3592-6002 or equivalent) located on the target development board. The layout of the connector is shown in Figure A-2 and the signals are described in Table A-3. A mapping of PPC405x3 to RISCTrace signals is provided in Table A-4. At the board level, the connector should be placed as close as possible to the processor chip to ensure signal integrity. An index at pin one and a key notch on the same side of the connector as the index are required.

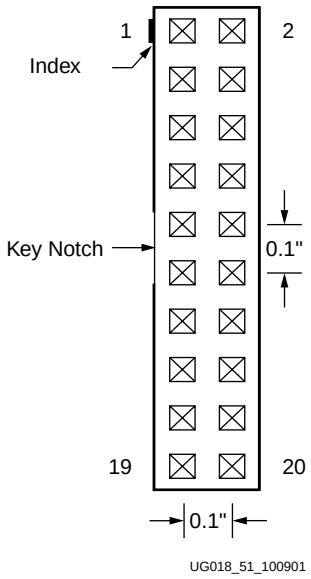

Figure A-2: **Trace-Connector Physical Layout**

Table A-3: **Trace Connector Signals for RISCTrace**

Pin	RISCTrace		Description
	I/O	Signal Name	
1	No Connect	Reserved	
2	No Connect	Reserved	
3	Output	TrcClk	Trace cycle.
4	No Connect	Reserved	
5	No Connect	Reserved	
6	No Connect	Reserved	
7	No Connect	Reserved	
8	No Connect	Reserved	
9	No Connect	Reserved	
10	No Connect	Reserved	
11	No Connect	Reserved	
12	Output	TS1O	Execution status.

Table A-3: **Trace Connector Signals for RISCTrace *(Continued)***

Pin	RISCTrace		Description
	I/O	Signal Name	
13	Output	TS2O	Execution status.
14	Output	TS1E	Execution status.
15	Output	TS2E	Execution status.
16	Output	TS3	Trace status.
17	Output	TS4	Trace status.
18	Output	TS5	Trace status.
19	Output	TS6	Trace status.
20		GND	Ground

Table A-4: **PPC405x3 to RISCTrace Signal Mapping**

PPC405x3		RISCTrace		Trace Connector Pin	Mictor Connector Pin
Signal	I/O	Signal	I/O		
C405TRCCYCLE	Output	TrcClk	Input	3	6
C405TRCODDEXECUTIONSTATUS[0]	Output	TS1O	Input	12	24
C405TRCODDEXECUTIONSTATUS[1]	Output	TS2O	Input	13	26
C405TRCEVENEXECUTIONSTATUS[0]	Output	TS1E	Input	14	28
C405TRCEVENEXECUTIONSTATUS[1]	Output	TS2E	Input	15	30
C405TRCTRACESTATUS[0]	Output	TS3	Input	16	32
C405TRCTRACESTATUS[1]	Output	TS4	Input	17	34
C405TRCTRACESTATUS[2]	Output	TS5	Input	18	36
C405TRCTRACESTATUS[3]	Output	TS6	Input	19	38

XILINX®

Appendix B

Signal Summary

Interface Signals

Table B-1 lists the PPC405x3 interface signals in alphabetical order. A cross reference is provided to each signal description. The signal naming conventions used are described on page 32.

Table B-1: **PPC405x3 Interface Signals in Alphabetical Order**

Signal	I/O Type	Interface	If Unused	Function
C405CPMCORESLEEPREQ	O	CPM, page 36	No Connect	Indicates the core is requesting to be put into sleep mode.
C405CPMMSRCE	O	CPM, page 35	No Connect	Indicates the value of MSR[CE].
C405CPMMSREE	O	CPM, page 35	No Connect	Indicates the value of MSR[EE].
C405CPMTIMERIRQ	O	CPM, page 35	No Connect	Indicates a timer-interrupt request occurred.
C405CPMTIMERRESETREQ	O	CPM, page 36	No Connect	Indicates a watchdog-timer reset request occurred.
C405DBGMSRWE	O	Debug, page 127	No Connect	Indicates the value of MSR[WE].
C405DBGSTOPACK	O	Debug, page 127	No Connect	Indicates the PPC405x3 is in debug halt mode.
C405DBGWBCOMPLETE	O	Debug, page 127	No Connect	Indicates the current instruction in the PPC405x3 writeback pipeline stage is completing.
C405DBGWBFULL	O	Debug, page 126	No Connect	Indicates the PPC405x3 writeback pipeline stage is full.
C405DBGWBIAR[0:29]	O	Debug, page 126	No Connect	The address of the current instruction in the PPC405x3 writeback pipeline stage.
C405DCRABUS[0:9]	O	DCR, page 100	No Connect	Specifies the address of the DCR access request.
C405DCRDBUSOUT[0:31]	O	DCR, page 100	No Connect or attach to input bus	The 32-bit DCR write-data bus.
C405DCRREAD	O	DCR, page 99	No Connect	Indicates a DCR read request occurred.
C405DCRWRITE	O	DCR, page 99	No Connect	Indicates a DCR write request occurred.
C405JTGCAPTUREDR	O	JTAG, page 110	No Connect	Indicates the TAP controller is in the capture-DR state.
C405JTGEXTEST	O	JTAG, page 110	No Connect	Indicates the JTAG EXTEST instruction is selected.
C405JTGPGMOUT	O	JTAG, page 110	No Connect	Indicates the state of a general purpose program bit in the JTAG debug control register (JDCR).

Table B-1: **PPC405x3 Interface Signals in Alphabetical Order *(Continued)***

Signal	I/O Type	Interface	If Unused	Function
C405JTGSHIFTDR	O	JTAG, page 110	No Connect	Indicates the TAP controller is in the shift-DR state.
C405JTGTDO	O	JTAG, page 110	No Connect	JTAG TDO (test-data out).
C405JTGTDOEN	O	JTAG, page 110	No Connect	Indicates the JTAG TDO signal is enabled.
C405JTGUPDATEDR	O	JTAG, page 110	No Connect	Indicates the TAP controller is in the update-DR state.
C405PLBDCUABORT	O	DSPLB, page 75	No Connect	Indicates the DCU is aborting an unacknowledged data-access request.
C405PLBDCUABUS[0:31]	O	DSPLB, page 71	No Connect	Specifies the memory address of the data-access request.
C405PLBDCUBE[0:7]	O	DSPLB, page 73	No Connect	Specifies which bytes are transferred during single-word transfers.
C405PLBDCUCACHEABLE	O	DSPLB, page 72	No Connect	Indicates the value of the cacheability storage attribute for the target address.
C405PLBDCUGUARDED	O	DSPLB, page 73	No Connect	Indicates the value of the guarded storage attribute for the target address.
C405PLBDCUPRIORITY[0:1]	O	DSPLB, page 75	No Connect	Indicates the priority of the data-access request.
C405PLBDCUREQUEST	O	DSPLB, page 70	No Connect	Indicates the DCU is making a data-access request.
C405PLBDCURNW	O	DSPLB, page 71	No Connect	Specifies whether the data-access request is a read or a write.
C405PLBDCUSIZE2	O	DSPLB, page 71	No Connect	Specifies a single word or eight-word transfer size.
C405PLBDCUU0ATTR	O	DSPLB, page 73	No Connect	Indicates the value of the user-defined storage attribute for the target address.
C405PLBDCUWRDBUS[0:63]	O	DSPLB, page 76	No Connect	The DCU write-data bus used to transfer data from the DCU to the PLB slave.
C405PLBDCUWRITETHRU	O	DSPLB, page 72	No Connect	Indicates the value of the write-through storage attribute for the target address.
C405PLBICUABORT	O	ISPLB, page 49	No Connect	Indicates the ICU is aborting an unacknowledged fetch request.
C405PLBICUABUS[0:29]	O	ISPLB, page 48	No Connect	Specifies the memory address of the instruction-fetch request. Bits 30:31 of the 32-bit address are assumed to be zero.
C405PLBICUCACHEABLE	O	ISPLB, page 48	No Connect	Indicates the value of the cacheability storage attribute for the target address.
C405PLBICUPRIORITY[0:1]	O	ISPLB, page 49	No Connect	Indicates the priority of the ICU fetch request.
C405PLBICUREQUEST	O	ISPLB, page 47	No Connect	Indicates the ICU is making an instruction-fetch request.
C405PLBICUSIZE[2:3]	O	ISPLB, page 48	No Connect	Specifies a four word or eight word line-transfer size.
C405PLBICUU0ATTR	O	ISPLB, page 49	No Connect	Indicates the value of the user-defined storage attribute for the target address.
C405RSTCHIPRESETREQ	O	Reset, page 41	Required	Indicates a chip-reset request occurred.
C405RSTCORERESETREQ	O	Reset, page 41	Required	Indicates a core-reset request occurred.

Table B-1: **PPC405x3 Interface Signals in Alphabetical Order *(Continued)***

Signal	I/O Type	Interface	If Unused	Function
C405RSTSYSRESETREQ	O	Reset, page 41	Required	Indicates a system-reset request occurred.
C405TRCCYCLE	O	Trace, page 130	No Connect	Specifies the trace cycle.
C405TRCEVENEXECUTIONSTATUS[0:1]	O	Trace, page 130	No Connect	Specifies the execution status collected during the first of two processor cycles.
C405TRCODDEXECUTIONSTATUS[0:1]	O	Trace, page 130	No Connect	Specifies the execution status collected during the second of two processor cycles.
C405TRCTRACESTATUS[0:3]	O	Trace, page 130	No Connect	Specifies the trace status.
C405TRCTRIGGEREVENTOUT	O	Trace, page 129	Wrap to Trigger Event In	Indicates a trigger event occurred.
C405TRCTRIGGEREVENTTYPE[0:10]	O	Trace, page 129	No Connect	Specifies which debug event caused the trigger event.
C405XXXMACHINECHECK	O	Control, page 38	No Connect	Indicates a machine-check error has been detected by the PPC405x3.
CPMC405CLOCK	I	CPM, page 34	Required	PPC405x3 clock input (for all non-JTAG logic, including timers).
CPMC405CORECLKINACTIVE	I	CPM, page 35	0	Indicates the CPM logic disabled the clocks to the core.
CPMC405CPUCLKEN	I	CPM, page 34	1	Enables the core clock zone.
CPMC405JTAGCLKEN	I	CPM, page 35	1	Enables the JTAG clock zone.
CPMC405TIMERCLKEN	I	CPM, page 34	1	Enables the timer clock zone.
CPMC405TIMERTICK	I	CPM, page 35	1	Increments or decrements the PPC405x3 timers every time it is active with the CPMC405CLOCK.
DBGC405DEBUGHALT	I	Debug, page 126	0	Indicates the external debug logic is placing the processor in debug halt mode.
DBGC405EXTBUSHOLDACK	I	Debug, page 126	0	Indicates the bus controller has given control of the bus to an external master.
DBGC405UNCONDDEBUGEVENT	I	Debug, page 126	0	Indicates the external debug logic is causing an unconditional debug event.
DCRC405ACK	I	DCR, page 100	0	Indicates a DCR access has been completed by a peripheral.
DCRC405DBUSIN[0:31]	I	DCR, page 101	0x0000_0000 or attach to output bus	The 32-bit DCR read-data bus.
EICC405CRITINPUTIRQ	I	EIC, page 108	0	Indicates an external critical interrupt occurred.
EICC405EXTINPUTIRQ	I	EIC, page 108	0	Indicates an external noncritical interrupt occurred.
JTGC405BNDSCANTDO	I	JTAG, page 110	0	JTAG boundary scan input from the previous boundary scan element TDO output.
JTGC405TCK	I	JTAG, page 109	See IEEE 1149.1	JTAG TCK (test clock).
JTGC405TDI	I	JTAG, page 109	1	JTAG TDI (test-data in).
JTGC405TMS	I	JTAG, page 109	1	JTAG TMS (test-mode select).
JTGC405TRSTNEG	I	Reset, page 42	Required	Performs a JTAG test reset (TRST).

Table B-1: **PPC405x3 Interface Signals in Alphabetical Order *(Continued)***

Signal	I/O Type	Interface	If Unused	Function
JTGC405TRSTNEG	I	JTAG, page 110	Required	JTAG $\overline{\text{TRST}}$ (test reset).
MCBCPUCLKEN	I	FPGA, page 132	1	Indicates the PPC405x3 clock enable should follow GWE during a partial reconfiguration.
MCBJTAGEN	I	FPGA, page 132	1	Indicates the JTAG clock enable should follow GWE during a partial reconfiguration.
MCBTIMEREN	I	FPGA, page 132	1	Indicates the timer clock enable should follow GWE during a partial reconfiguration.
MCPPCRST	I	FPGA, page 133	1	Indicates the PPC405x3 should be reset when GSR is asserted during a partial reconfiguration.
PLBC405DCUADDRACK	I	DSPLB, page 77	0	Indicates a PLB slave acknowledges the current data-access request.
PLBC405DCUBUSY	I	DSPLB, page 81	0	Indicates the PLB slave is busy performing an operation requested by the DCU.
PLBC405DCUERR	I	DSPLB, page 81	0	Indicates an error was detected by the PLB slave during the transfer of data to or from the DCU.
PLBC405DCURDDACK	I	DSPLB, page 79	0	Indicates the DCU read-data bus contains valid data for transfer to the DCU.
PLBC405DCURDDBUS[0:63]	I	DSPLB, page 79	0x0000_0000 _0000_0000	The DCU read-data bus used to transfer data from the PLB slave to the DCU.
PLBC405DCURDWDADDR[1:3]	I	DSPLB, page 79	0b000	Indicates which word or doubleword of an eight-word line transfer is present on the DCU read-data bus.
PLBC405DCUSSIZE1	I	DSPLB, page 78	0	Specifies the bus width (size) of the PLB slave that accepted the request.
PLBC405DCUWRDACK	I	DSPLB, page 80	0	Indicates the data on the DCU write-data bus is being accepted by the PLB slave.
PLBC405ICUADDRACK	I	ISPLB, page 50	0	Indicates a PLB slave acknowledges the current ICU fetch request.
PLBC405ICUBUSY	I	ISPLB, page 53	0	Indicates the PLB slave is busy performing an operation requested by the ICU.
PLBC405ICUERR	I	ISPLB, page 54	0	Indicates an error was detected by the PLB slave during the transfer of instructions to the ICU.
PLBC405ICURDDACK	I	ISPLB, page 51	0	Indicates the ICU read-data bus contains valid instructions for transfer to the ICU.
PLBC405ICURDDBUS[0:63]	I	ISPLB, page 51	0x0000_0000 _0000_0000	The ICU read-data bus used to transfer instructions from the PLB slave to the ICU.
PLBC405ICURDWDADDR[1:3]	I	ISPLB, page 52	0b000	Indicates which word or doubleword of a four-word or eight-word line transfer is present on the ICU read-data bus.
PLBC405ICUSSIZE1	I	ISPLB, page 50	0	Specifies the bus width (size) of the PLB slave that accepted the request.
PLBCLK	I	FPGA, page 133	Required	PLB clock.
RSTC405RESETCHIP	I	Reset, page 41	Required	Indicates a chip-reset occurred.

Table B-1: **PPC405x3 Interface Signals in Alphabetical Order *(Continued)***

Signal	I/O Type	Interface	If Unused	Function
RSTC405RESETCORE	I	Reset, page 41	Required	Resets the PPC405x3 core logic, data cache, instruction cache, and the on-chip memory controller (OCM).
RSTC405RESETSYS	I	Reset, page 42	Required	Indicates a system-reset occurred. Resets the logic in the PPC405x3 JTAG unit.
TIEC405DETERMINISTICMULT	I	Control, page 37	Required	Specifies whether all multiply operations complete in a fixed number of cycles or have an early-out capability.
TIEC405DISOPERANDFWD	I	Control, page 38	Required	Disables operand forwarding for load instructions.
TIEC405MMUEN	I	Control, page 37	Required	Enables the memory-management unit (MMU)
TRCC405TRACEDISABLE	I	Trace, page 131	0	Disables trace collection and broadcast.
TRCC405TRIGGEREVENTIN	I	Trace, page 131	Wrap to Trigger Event Out	Indicates a trigger event occurred and that trace status is to be generated.

Appendix C

Processor Block Timing Model

Introduction

This section explains all of the timing parameters associated with the IBM PPC405 Processor Block. It is intended to be used in conjunction with Module 3 of the Virtex II Pro Data Sheet and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values, refer to the data sheet.

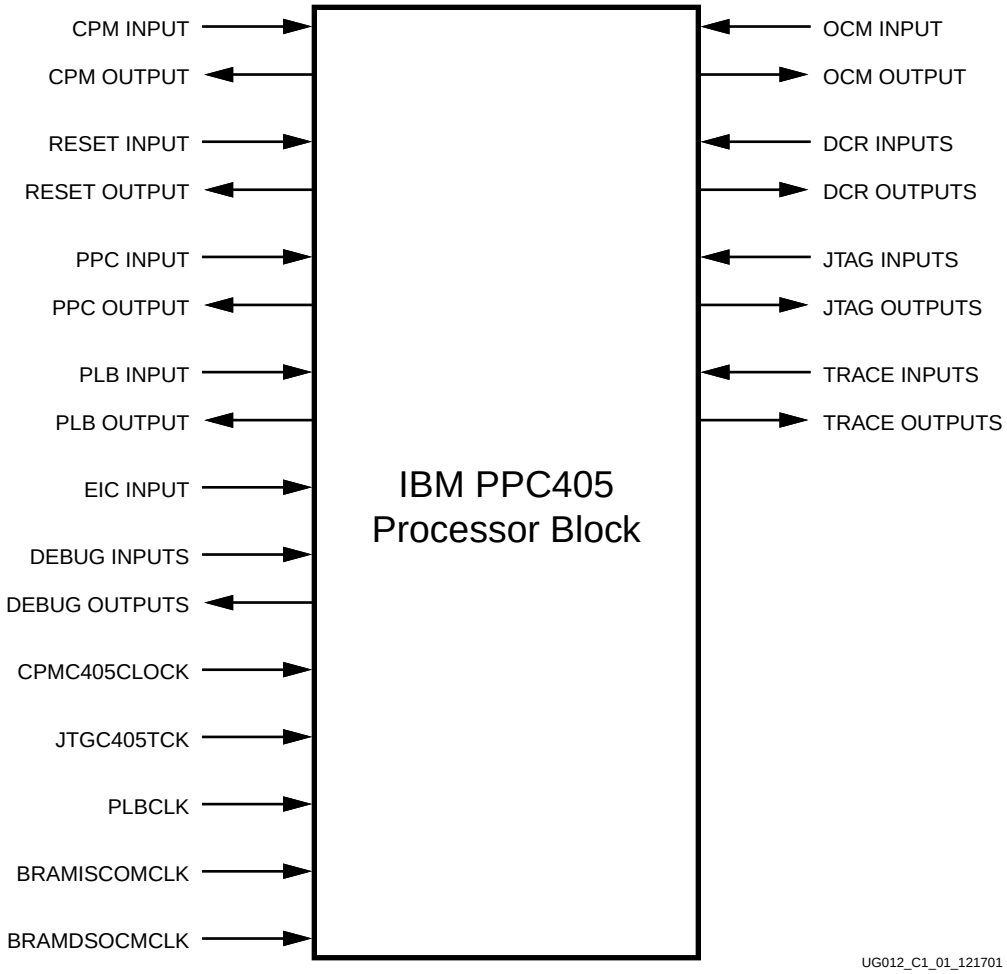

Figure C-1: **PowerPC 405 Processor Block (Simplified)**

There are hundreds of signals entering and exiting the processor block. The model presented in this section treats the processor block as a "black box." Propagation delays internal to the processor block and core logic are ignored. Signals are characterized with setup and hold times for inputs and clock to valid output times for outputs. Signals are grouped by which interface block they originate from: Processor Local Bus (PLB) , Device Control Register (DCR), External Interrupt Controller (EIC), Reset (RST), Clock and Power Management (CPM), Debug (DBG), PowerPC miscellaneous (PPC), Trace Port (TRC), JTAG, Instruction-Side On-Chip Memory (ISOCM), and Data-Side On-Chip Memory (DSOCM).

Table C-1 associates five clocks with their corresponding interface blocks. All signal parameters discussed in this section are characterized at a rising clock edge. Exceptions to this rule, such as for the JTAG signals, are pointed out where applicable.

Table C-1: **Clocks and Corresponding Processor Interface Blocks**

CLOCK SIGNAL	DESCRIPTION	INTERFACE
CPMC405CLOCK	Main processor core clock	DCR EIC RST CPM DBG PPC TRC
PLBCLK	Processor Local Bus clock	PLB
JTAGC405TCK	Clock for JTAG logic within the processor core	JTAG
BRAMISOCMCLK	Clock for the ISOCM Controller	ISOCM
BRAMDSOCMCLK	Clock for the DSOCM Controller	DSOCM

Timing Parameters

Parameter designations are constructed to reflect the functions they perform, as well as the interface blocks and clocks they correspond to. The following three sections explain the meaning of each of the basic timing parameter designations used in the tables:

Setup/Hold Times of Inputs Relative to Clock

Basic Format:

ParameterName_BLOCK

where

ParameterName=T with subscript string defining the timing relationship

BLOCK=name of applicable PPC405 processor interface block
(refer to Table C-1)

ParameterName Format:

T_{PxCK}=Setup time before clock edge
T_{PCKx}=Hold time after clock edge

where

x=C(Control inputs)
D(Data inputs)

Setup/Hold Time (Examples):

T_{PCCK}_PLB / T_{PCKC}_PLBSetup / hold times of PLB Control inputs relative to rising edge of PLB clock

T_{PDCK}_ISOCM / T_{PCKD}_ISOCMSetup / hold times of BRAMISOCM Data inputs relative to rising edge of BRAMISOCM clock

Clock to Output Delays

Basic Format:

ParameterName_BLOCK

where

ParameterName=T with subscript string defining the timing relationship

BLOCK=name of applicable PPC405 processor interface block (refer to Table C-1)

ParameterName Format:

T_{PCKx}=Delay time from clock edge to output

where

x=AO(Address outputs)
CO(Control outputs)
DO(Data outputs)

Output Delay Time (Examples):

T_{PCKAO}_ISOCMRising edge of BRAMISOCM clock to BRAMISOCM Address outputs
T_{PCKCO}_DCRRising edge of Core clock to DCR Control outputs
T_{PCKDO}_PLBRising edge of PLB clock to PLB Data outputs

Clock Pulse Width

ParameterName Format:

T_{xPWH}=Minimum pulse width, High state
T_{xPWL}=Minimum pulse width, Low state

where
x=C(Core)
P(PLB)
J(JTAG)
I(ISOCM)
D(DSOCM)

Pulse Width (Examples):

T_{CPWH}Minimum pulse width, core clock, High state
T_{PPWL}Minimum pulse width, PLB clock, Low state

Timing Parameter Tables and Diagram

The following five tables list the timing parameters as reported by the implementation tools relative to the clocks given in Table C-1, along with the signals from the processor

block that correspond to each parameter. A timing diagram (Figure C-2) illustrates the timing relationships.

Table C-2: **Parameters Relative to the Core Clock (CPMC405CLOCK)**

Parameter	Function	Signals
Setup / Hold:		
$T_{PCCK_}$DCR / $T_{PCKC_}$DCR	Control inputs	DCRC405ACK
$T_{PDCK_}$DCR / $T_{PCKD_}$DCR	Data inputs	DCRC405DBUSIN[0:31]
$T_{PCCK_}$CPM / $T_{PCKC_}$CPM	Control inputs	CPMC405TIMERTICK CPMC405CPUCLKEN CPMC405TIMERCLKEN CPMC405JTAGCLKEN
$T_{PCCK_}$RST / $T_{PCKC_}$RST	Control inputs	RSTC405RESETCHIP RSTC405RESETCORE RSTC405RESETSYS
$T_{PCCK_}$DBG / $T_{PCKC_}$DBG	Control inputs	DBGC405DEBUGHALT DBGC405UNCONDDEBUGEVENT
$T_{PCCK_}$TRC / $T_{PCKC_}$TRC	Control inputs	TRCC405TRACEDISABLE TRCC405TRIGGEREVENTIN
$T_{PCCK_}$EIC / $T_{PCKC_}$EIC	Control inputs	EICC405CRITINPUTIRQ EICC405EXTINPUTIRQ
Clock to Out:		
$T_{PCKCO_}$DCR	Control outputs	C405DCRREAD C405DCRWRITE
$T_{PCKAO_}$DCR	Address outputs	C405DCRABUS[0:9]
$T_{PCKDO_}$DCR	Data outputs	C405DCRDBUSOUT[0:31]
$T_{PCKCO_}$CPM	Control outputs	C405CPMMSREE C405CPMMSRCE C405CPMTIMERIRQ C405CPMTIMERRESETREQ C405CPMCORESLEEPREQ
$T_{PCKCO_}$RST	Control outputs	C405RSTCHIPRESETREQ C405RSTCORERESETREQ C405RSTSYSRESETREQ

Table C-2: **Parameters Relative to the Core Clock (CPMC405CLOCK) *(Continued)***

Parameter	Function	Signals
$T_{PCKCO_}$DBG	Control outputs	C405DBGMSRWE C405DBGSTOPACK C405DBGWBCOMPLETE C405DBGWBFULL C405DBGWBIAR[0:29]
$T_{PCKCO_}$PPC	Control outputs	C405XXXMACHINECHECK
$T_{PCKCO_}$TRC	Control outputs	C405TRCCYCLE C405TRCEVENEXECUTIONSTATUS[0:1] C405TRCODDEXECUTIONSTATUS[0:1] C405TRCTRACESTATUS[0:3] C405TRCTRIGGEREVENTOUT C405TRCTRIGGEREVENTTYPE[0:10]
Clock:		
T_{CPWH}	Clock pulse width, High state	CPMC405CLOCK
T_{CPWL}	Clock pulse width, Low state	CPMC405CLOCK

Table C-3: **Parameters Relative to the PLB Clock (PLBCLK)**

Parameter	Function	Signals
Setup / Hold:		
$T_{PCCK_}$PLB / $T_{PCKC_}$PLB	Contol inputs	PLBC405DCUADDRACK PLBC405DCUBUSY PLBC405DCUERR PLBC405DCURDDACK PLBC405DCUSSIZE1 PLBC405DCUWRDACK PLBC405ICURDWDADDR[1:3] PLBC405DCURDWDADDR[1:3] PLBC405ICUADDRACK PLBC405ICUBUSY PLBC405ICUERR PLBC405ICURDDACK PLBC405ICUSSIZE1
$T_{PDCK_}$PLB / $T_{PCKD_}$PLB	Data inputs	PLBC405ICURDDBUS[0:63] PLBC405DCURDDBUS[0:63]
Clock to Out:		

Table C-3: **Parameters Relative to the PLB Clock (PLBCLK) *(Continued)***

Parameter	Function	Signals
$T_{PCKCO_}$PLB	Control outputs	C405PLBDCUABORT C405PLBDCUBE[0:7] C405PLBDCUCACHEABLE C405PLBDCUGUARDED C405PLBDCUPRIORITY[0:1] C405PLBDCUREQUEST C405PLBDCURNW C405PLBDCUSIZE2 C405PLBDCUU0ATTR C405PLBDCUWRITETHRU C405PLBICUABORT C405PLBICUCACHEABLE C405PLBICUPRIORITY[0:1] C405PLBICUREQUEST C405PLBICUSIZE[2:3] C405PLBICUU0ATTR
$T_{PCKDO_}$PLB	Data outputs	C405PLBDCUWRDBUS[0:63]
$T_{PCKAO_}$PLB	Address outputs	C405PLBDCUABUS[0:31] C405PLBICUABUS[0:29]
Clock:		
T_{PPWH}	Clock pulse width, High state	PLBCLK
T_{PPWL}	Clock pulse width, Low state	PLBCLK

Table C-4: **Parameters Relative to the JTAG Clock (JTAGC405TCK)**

Parameter	Function	Signals
Setup / Hold:		
$T_{PCCK_}$JTAG / $T_{PCKC_}$JTAG	Control inputs	JTGC405BNDSCANTDO JTGC405TDI JTGC405TMS JTGC405TRSTNEG CPMC405CORECLKINACTIVE DBGC405EXTBUSHOLDACK
Clock to Out:		
$T_{PCKCO_}$JTAG	Control outputs	C405JTGCAPTUREDR C405JTGEXTEST(1) C405JTGPGMOUT(2) C405JTGSHIFTDR C405JTGTDO(1) C405JTGTDOEN(1) C405JTGUPDATEDR
Clock:		

Table C-4: **Parameters Relative to the JTAG Clock (JTAGC405TCK) *(Continued)***

Parameter	Function	Signals
T_{JPWH}	Clock pulse width, High state	JTGC405TCK
T_{JPWL}	Clock pulse width, Low state	JTGC405TCK

Notes:
1. Synchronous to the negative edge of JTGC405TCK
2. Synchronous to CPMC405CLOCK

Table C-5: **Parameters Relative to the ISOCM Clock (BRAMISOCMCLK)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PDCK_}$ISOCM / $T_{PCKD_}$ISOCM	Data inputs	BRAMISOCMRDDBUS[0:63]
Clock to Out:		
$T_{PCKCO_}$ISOCM	Control outputs	ISOCMBRAMEN ISOCMBRAMODDWRITEEN ISOCMBRAMEVENWRITEEN
$T_{PCKAO_}$ISOCM	Address outputs	ISOCMBRAMRDABUS[8:28] ISOCMBRAMWRABUS[8:28]
$T_{PCKDO_}$ISOCM	Data outputs	ISOCMBRAMWRDBUS[0:31]
Clock:		
T_{IPWH}	Clock pulse width, High state	BRAMISOCMCLK
T_{IPWL}	Clock pulse width, Low state	BRAMISOCMCLK

Table C-6: **Parameters Relative to the DSOCM Clock (BRAMDSOCMCLK)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PDCK_}$DSOCM / $T_{PCKD_}$DSOCM	Data inputs	BRAMDSOCMRDDBUS[0:31]
Clock to Out:		
$T_{PCKCO_}$DSOCM	Control outputs	DSOCMBRAMEN DSOCMBRAMBYTEWRITE[0:3] DSOCMBUSY
$T_{PCKDO_}$DSOCM	Data outputs	DSOCMBRAMWRDBUS[0:31]
$T_{PCKAO_}$DSOCM	Address outputs	DSOCMBRAMABUS[8:29]

Table C-6: **Parameters Relative to the DSOCM Clock (BRAMDSOCMCLK) *(Continued)***

Parameter	Function	Signals
Clock:		
T_{DPWH}	Clock, Pulse width high	BRAMDSOCMCLK
T_{DPWL}	Clock, Pulse width low	BRAMDSOCMCLK

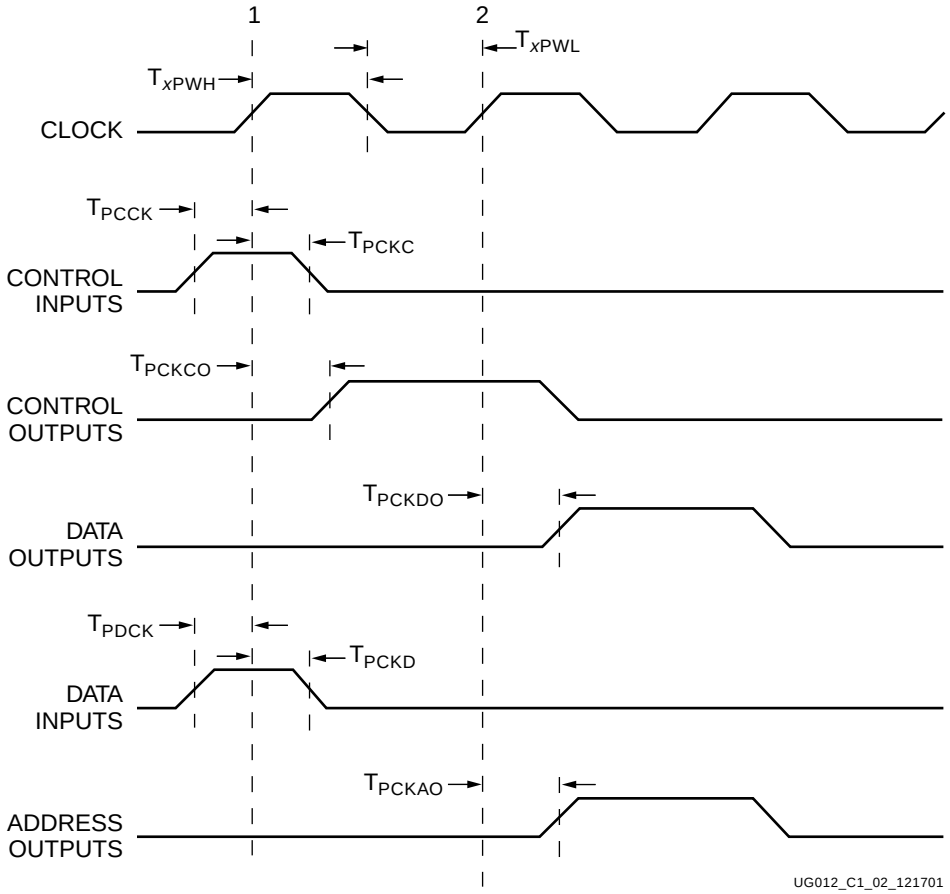

Figure C-2: **Processor Block Timing Relative to Clock Edge**

Index

S

T

U

V

W