PowerPC 405 Processor Block Reference Guide

Embedded Development Kit

UG018 (v2.4) January 11, 2010

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/16/02	1.0	Initial Embedded Development Kit (EDK) release.
09/02/03	1.1	Updated for EDK 6.1 release
08/20/04	2.0	Updated to include Virtex-4 functionality.
07/20/05	2.1	• Updated Table 2-24, JTAG Instruction Register length for Virtex-4 devices: FX12, FX40, FX60. • Added (Virtex-4 only) DCREMACENABLER output port to Figure 2-29 and Table B-1. Added (Virtex-4 only) parity error detection sections for instruction cache, data cache and unified translation lookaside buffer. Introduced new privileged registers (Virtex-4 only) Core Configuration Register 1 (CCR1) and Machine Check Syndrome Register (MCSR). • Completely revised “PowerPC 405 APU Controller” in Chapter 4 including adding descriptions to figures and an entirely new section: “Flushed FCM Instructions.”

Date	Version	Revision
06/05/07	2.2	• Added Virtex-4 FX fractional clock width support in "PLB" in Chapter 2. • Removed figure titled "Incorrect Wiring of JTAG Chain with Individual PPC405 Connections" in "Connecting PPC405 JTAG Logic Directly to Programmable I/O" in Chapter 2. • Clarified "Execution Re-ordering" in Chapter 3. • On Table 3-5, page 150, clarified DSOCMBRAMEN for Virtex-4 designs.
05/16/08	2.3	• Updated "Preface," page 9. • Consistent use of the save/restore register acronym (SSR) revised in Table 4-4, page 193.
01/11/10	2.4	• See XCN10007, *Removing Support for PowerPC405 Parity Checking in all Virtex-4 FX FPGAs, v1.0*, for detailed product revisions: • Removed parity detection and discussion of the Core Configuration Register for Virtex-4 FPGAs from "PowerPC 405 Software Features" in Chapter 1. • Removed "Memory Management Unit Operation (Virtex-4 FPGAs only)" section from "Memory Management Unit" in Chapter 1. • Removed "Instruction Cache Unit Parity Operation (Virtex-4 FPGAs only)" and "Data Cache Unit Parity Operation (Virtex-4 FPGAs only)" sections from "Instruction and Data Caches" in Chapter 1.

Table of Contents

Chapter 3: PowerPC 405 OCM Controller

Chapter 4: PowerPC 405 APU Controller

About This Guide

This guide serves as a technical reference describing the hardware interface to the PowerPC® 405 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.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Introduction to the PowerPC 405 Processor," provides an overview of the PowerPC embedded-environment architecture and the features supported by the PowerPC 405 processor block.
- Chapter 2, "Input/Output Interfaces," describes the interface signals into and out of the PowerPC 405 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 PowerPC 405 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 PowerPC 405 core.
- Chapter 4, "PowerPC 405 APU Controller," describes the Auxiliary Processor Unit controller, which allows the designer to extend the native PowerPC 405 instruction set with custom instructions that are executed by an FPGA Fabric Co-processor Module (FCM). The APU controller is available only in the Virtex®-4 FX family.
- Appendix A, "RISCWatch and RISCTrace Interfaces," describes the interface requirements between the PowerPC 405 processor block and the RISCWatch and RISCTrace tools.
- Appendix B, "Signal Summary," lists all PowerPC 405 interface signals in alphabetical order.
- Appendix C, "Processor Block Timing Model," explains all of the timing parameters associated with the IBM PPC405 Processor Block.

Additional Resources

To search the database of silicon and software questions and answers, or to create a technical support case in WebCase, see the Xilinx website at: http://www.xilinx.com/support.

Additional information of potential interest to readers of this manual is contained in UG011: *PowerPC Processor Reference Guide*

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *`design_name`*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** `[`*`option_name`*`]` *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**`{`**`on`**`|`**`off`**`}`
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**`{`**`on`**`|`**`off`**`}`

Convention	Meaning or Use	Example

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *`block_name loc1 loc2 ... locn;`*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Reference to a location in another document	See Figure 2-5 in the *Virtex-4 User Guide.*
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

General Conventions

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

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

Convention	Definition
mnemonic	Instruction mnemonics are shown in lower-case bold.
variable	Variable items are shown in italic.
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
$OBJECT_{b:b}$	A range of bits in any object (a register, an instruction, an address, or a field)
$OBJECT_{b,b,...}$	A list of bits in any object (a register, an instruction, an address, or a field)

Table 1-1: **General Notational Conventions *(Cont'd)***

Convention	Definition
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 1-2 lists the PowerPC 405 registers used in this document and their descriptive names.

Table 1-2: **PowerPC 405 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 PowerPC 405 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 PowerPC 405 processor clock is prevented from toggling. The execution state of the PowerPC 405 processor 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 PowerPC 405 processor out of the sleep state. The PowerPC 405 processor clock begins toggling and the execution state of the PowerPC 405 processor advances from that of the sleep state.
word	Four bytes, or 32 bits.

XILINX®

Chapter 1

Introduction to the PowerPC 405 Processor

The PowerPC 405 processor is a 32-bit implementation of the *PowerPC embedded-environment architecture* that is derived from the PowerPC architecture. Specifically, the PowerPC 405 processor is an embedded PowerPC 405D5 (for Virtex®-II Pro) or 405F6 (for Virtex-4) processor core. The term *processor block* is used throughout this document to refer to the combination of a PPC405D5 or PPC405F6 core, on-chip memory logic (OCM), an APU controller (Virtex-4 only), 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 PowerPC 405 core. The following topics are included:

- "PowerPC Architecture"
- "PowerPC 405 Software Features"
- "PowerPC 405 Hardware Organization"
- "PowerPC 405 Performance"

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 PowerPC 405 processor are compatible with other PowerPC implementations. Privileged software generally is not compatible. The migration of privileged software from the PowerPC architecture to the PowerPC 405 processor is in many cases straightforward because of the simplifications made by the PowerPC embedded-environment architecture. Refer to the *PowerPC Processor Reference Guide* for more information on programming the PowerPC 405 processor.

PowerPC Embedded-Environment Architecture

The PowerPC 405 processor 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

Table 1-2: **OEA Features of the PowerPC Embedded-Environment Architecture *(Cont'd)***

Operating Environment	Features
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

PowerPC 405 Software Features

The PowerPC 405 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 PowerPC 405 VEA and OEA support is also available in implementations of the PowerPC Book-E architecture. Key software features of the PowerPC 405 processor 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 PowerPC 405 processor. Refer to the *PowerPC Processor Reference Guide* for more information on using these resources.

Privilege Modes

Software running on the PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor 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

PowerPC 405 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 PowerPC 405 processor 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 PowerPC 405 processor. Descriptions of the registers are in the following sections.

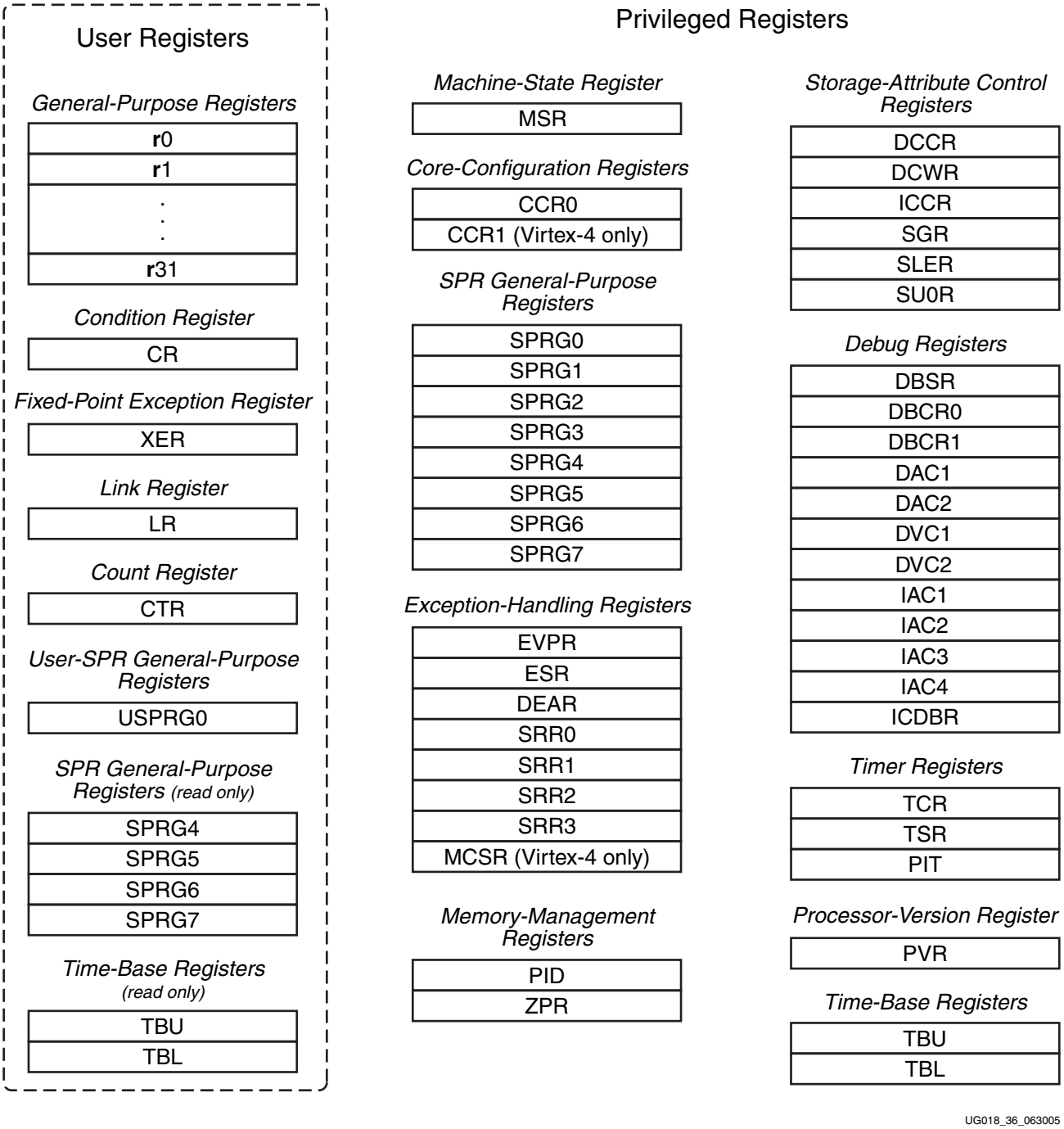


Figure 1-1: **PowerPC 405 Registers**

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 PowerPC 405 processor. The OCM controllers are examples of devices that contain DCRs. Although the DCRs are not part of the PowerPC 405 implementation, they are accessed using the **mtdcr** and **mfdcr** instructions. The DCRs can be accessed only by privileged software.

PowerPC 405 Registers (Virtex-4 devices only)

The Machine Check Syndrome Register (MCSR) contains status information to determine the source of instruction or data PLB errors for Virtex-4 designs. Register information can be found in the *PowerPC Processor Reference Guide*.

PowerPC 405 Hardware Organization

As shown in Figure 1-2, the PowerPC 405 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.

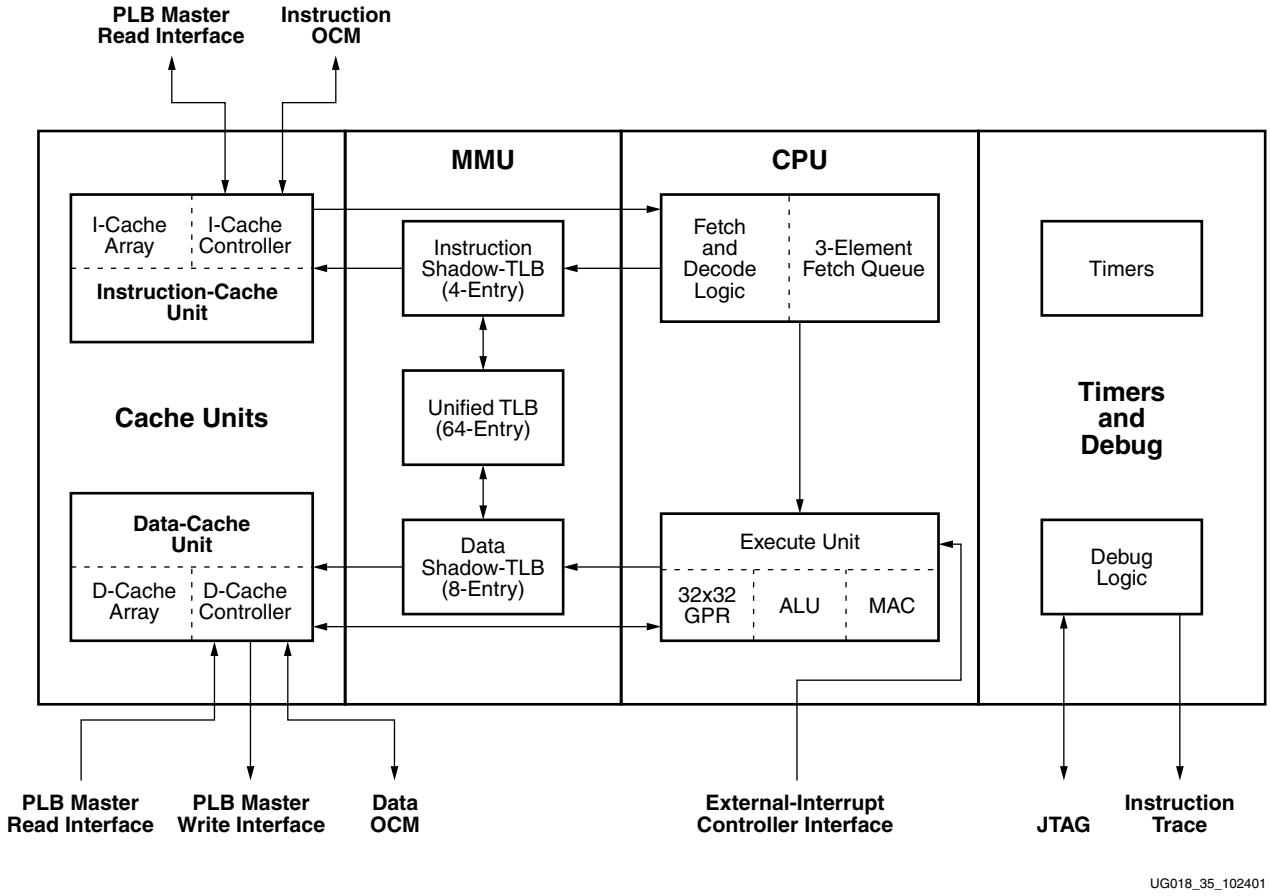


Figure 1-2: **PowerPC 405 Organization[a]**

a. Figure 1-2 is specific to PPC405D5.

Central-Processing Unit

The PowerPC 405 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 PowerPC 405 processor 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 PowerPC 405 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 PowerPC 405 processor to handle critical interrupts independently of noncritical interrupts.

Memory Management Unit

The PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor contains a 64-bit time base and three timers. The time base increments synchronously using the CPU clock or an external clock source. The three timers increment synchronously with the time base. The three timers supported by the PowerPC 405 processor 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 PowerPC 405 processor 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.

PowerPC 405 Interfaces

The PowerPC 405 processor 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.

PowerPC 405 Performance

The PowerPC 405 processor 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 PowerPC 405 processor.

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: **PowerPC 405 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

Table 1-3: **PowerPC 405 Cycles per Instruction *(Cont'd)***

Instruction Class	Execution Cycles
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

This chapter describes all PowerPC 405 input/output signals associated with the following processor block interfaces:

- "Clock and Power Management Interface"
- "CPU Control Interface"
- "Reset Interface"
- "Instruction-Side Processor Local Bus Interface"
- "Data-Side Processor Local Bus Interface"
- "Device-Control Register Interfaces"
- "Internal Device Control Register (DCR) Interface"
- "External DCR Bus Interface"
- "External Interrupt Controller Interface"
- "PPC405 JTAG Debug Port"
- "Debug Interface"
- "Trace Interface"
- "Processor Version Register (PVR) Interface (Virtex-4 FX Only)"
- "Additional FPGA Specific Signals"

The sections within this chapter provide 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."

The Fabric Co-Processor Module (FCM) interface associated with the Virtex-4 FX family PowerPC 405 APU controller, is described separately in Chapter 4, "PowerPC 405 APU Controller."

Appendix B, "Signal Summary," alphabetically lists the signals described in this chapter. The I/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[SIGNAME2][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.
- SIGNAME2 is an uppercase name identifying the secondary 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 "Location" column in the table identifies whether the functional unit resides inside 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
APU	Auxiliary Processor Unit Controller	Inside
FCM	Fabric Co-Processor Module	Outside

Table 2-1: **Signal Name Prefix Definitions *(Cont'd)***

Prefix1 or Prefix2	Definition	Location
BRAM	Block RAM	Outside
XXX	Unspecified FPGA unit	Outside

Notes:

1. Not to be confused with the OCM *controllers*, located inside 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 core. In addition to this, the Virtex-4 FX family PowerPC 405 processors also use separate clocks for the APU and DCR controller. Two types of processor clock controls 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 PowerPC 405 CPM 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 PowerPC 405 core and controllers. 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 PowerPC 405 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 PowerPC 405 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 PowerPC 405 clock, CPMC405CLOCK. Instead of using the global-local enables to prevent the clock signal from propagating through a zone, CPM logic can stop the PowerPC 405 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 PowerPC 405 processor from sleep mode when any of the following occurs:

- A timer interrupt or timer reset is asserted by the PowerPC 405 processor.
- A chip-reset or system-reset request is asserted (this request comes from a source other than the PowerPC 405 processor).
- 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 PowerPC 405 processor. See "DBGC405DEBUGHALT (Input)" for more information.

CPM Interface I/O Signal Summary

Figure 2-1 shows the block symbol for the CPM interface. The block RAM clocks associated with the data-side and instruction-side OCM are described in chapter Chapter 3, "PowerPC 405 OCM Controller." The signals are summarized in Table 2-2.

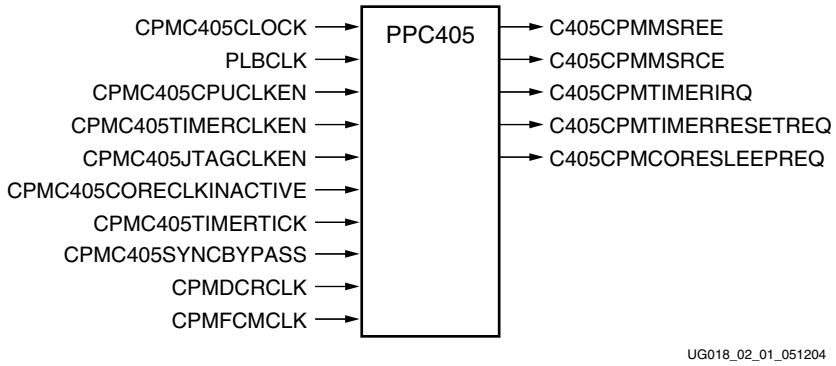


Figure 2-1: **CPM Interface Block Symbol**

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

Signal	I/O Type	If Unused	Function
CPMC405CLOCK	I	Required	PowerPC 405 clock input (for all non-JTAG logic, including timers).
PLBCLK	I	Required	PLB clock interface clock (lacks CPM prefix due to legacy naming).
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 PowerPC 405 timers every time it is active with the CPMC405CLOCK.
CPMC405SYNCBYPASS	I	1	Virtex-4 FX only. Bypass PLB re-synchronization inside the PowerPC 405 core for Virtex-II Pro compatibility.
CPMDCRCLK	I	0	Virtex-4 FX only. DCR bus interface clock for PPC405 synchronization.

Table 2-2: **CPM Interface I/O Signals *(Cont'd)***

Signal	I/O Type	If Unused	Function
CPMFCMCLK	I	0	Virtex-4 FX only. FCM interface clock for the APU Controller.
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 PowerPC 405 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).

PLBCLK (Input)

This signal is the source clock for all PLB logic.

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 PowerPC 405 register (JDSR). An external debug tool (such as RISCWatch) can read this register and determine that the PowerPC 405 processor is in sleep mode. This signal should be asserted by the CPM when it places the PowerPC 405 processor 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 PowerPC 405 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 PowerPC 405 clock frequency if CPMC405TIMERTICK is held active.

CPMC405SYNCBYPASS (Input, Virtex-4 FX Only)

Allows the user to bypass the PLB synchronization module inside the PowerPC core and instead use a Virtex-II Pro compatible synchronizer in the processor block. When this signal is enabled, integer clock ratios between 1:1 and 16:1 are possible. If disabled, the user can use fractional clock ratios of $N/2$ and $N/3$ for any integer N, but must also ensure that PLB and CPU clocks are rising-edge aligned, and accept additional latency for the synchronization.

CPMDCRCLK (Input, Virtex-4 FX Only)

This is the DCR interface clock used by the PPC to synchronize communication between the PowerPC's internal clock domain (CPMC405CLOCK) and the DCR bus transactions performed using the DCR slave clocks. The PowerPC core to DCR interface clock ratio can be any integer between 1:1 and 16:1. Clocks must be rising-edge aligned.

CPMFCMCLK (Input, Virtex-4 FX Only)

This is the re-synchronization clock for transactions between the APU controller and an FCM. Allows the APU controller internally to run at the CPMC405CLOCK speed, independently of the FCM interface transaction speed. CPMFCMCLK would typically be the same clock that clocks the FCM internally. PowerPC core to FCM interface clock ratio can be any integer between 1:1 and 16:1. Clocks must be rising-edge aligned.

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. Consequently, 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 PowerPC 405 processor 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. Consequently, 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 PowerPC 405 processor 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. Consequently, the CPM should latch the C405CPMMSREE, C405CPMMSRCE, and C405CPMTIMERIRQ signals before using them to control the processor clocks.

System Design Considerations for Clock Domains

The high-level view of an embedded system with the PowerPC 405 processor and CoreConnect bus architecture includes:

- PowerPC 405 Processor.
- Processor Local Bus (PLB) peripherals.
- Instruction-side and Data-side On-Chip Memory Controller (OCM).
- Device Control Register (DCR) peripherals.
- Fabric Co-Processor Module (FCM): Virtex-4 designs only.

These clocks communicate to the processor block the specific clock ratio between the processor block clock and the other system clocks in the design.

- CPMC405CLOCK, main Processor Block clock.
- PLBCLK, primary PLB I/O Bus clock.
- BRAMISOCMCLK, reference clock for the I-Side OCM controller.
- BRAMDSOCMCLK, reference clock for the D-Side OCM controller.
- CPMFCMCLK, reference clock for the APU controller (Virtex-4 designs only).
- CPMDCRCLK, reference clock for the external DCR bus (Virtex-4 designs only).

The PowerPC405 processor block supports multiple clock domains. Using several DCM and BUFG components are recommended to create and drive the clock domains. The clock domains include the PLB, FCM, DCR, and OCM clocks.

PLB

The PLB is used as an interface between the processor block and the higher performance peripherals. The processor block has some internal logic to generate the appropriate enabling signals for controlling the PLB. The PLB clock must be rising-edge aligned to the processor block. All communication between the processor block and the PLB are based upon the rising edge of the CPMC405CLOCK. The PLB is synchronous with the processor block. The allowed supported integer clock frequency ratios between the processor block and the PLB are 1:1, 2:1, 3:1 . . . up to 16:1. As an example, the processor block can be run at 300 MHz while the PLB bus is run at 100 MHz, in a 3:1 ratio.

In Virtex-4 FX devices only, fractional clock ratios of N/2 and N/3 for any integer N, are supported by setting the CPMC405SYNCBYPASS input to 0. If fractional clock ratios are used, the user must also ensure that PLB and CPU clocks are rising-edge aligned, and accept additional latency for the synchronization.

DCR

The processor block clock and the DCR clock must come from the same source and be in phase with each other. The DCR clock covers both of the processor block DCR and the memory mapped DCR. The clock ratio between the DCR clock domain and the processor block can run at any integer clock ratio from 1:1 to 16:1 as long as the bus transaction completes in 64 processor block cycles. If the bus transaction does not complete in 64 processor block clock cycles, the processor block will time out and move on to the next instruction.

Virtex-II Pro Specific

For Virtex-II Pro devices, there is no CPMDCRCLK input to the processor block. Users can either set appropriate timing constraints (multi-cycle path, false path, etc.), or simply include DCR re-synchronization logic to simply the steps to analyze the timing related to DCR interface.

Virtex-4 Specific

For Virtex-4 FX parts there is a dedicated DCR clock input and re-synchronization registers handling the clock boundary.

FCM (Virtex-4 FX only)

An FCM is used for highest performance integration of custom functionality defined in the FPGA fabric with the execution pipeline of the PowerPC. The FCM clock would typically be the same clock that clocks the FCM internally. PowerPC core to FCM interface clock ratios can range from 1:1 to 16:1. The clocks must be rising-edge aligned.

OCM

For high speed access, the OCM clock domain covers the interface between the processor block and the block RAM surrounding the processor block. There are two independent clocks for the OCM controllers in the processor block: BRAMDSOCMCLK (data side controller) and BRAMISOCMCLK (instruction side controllers).

The data side controller and the instruction side controllers can run at different frequencies, based upon the access time of the block RAM. When the processor block, OCM controller, and block RAMs run at the same clock frequency, the processor is in single-cycle mode. Multi-cycle mode occurs when the processor is running at a higher frequency than the block RAMs. In the single-cycle mode and multi-cycle mode, the BRAMISOCMCLK and BRAMDSOCMCLK signals are provided to the OCM controller as inputs.

Through timing analysis, the clock ratio between the processor block clock and the block RAM clocks is determined by the worst case access time between the OCM controller interface and the block RAM interface. Based upon the timing analysis, most designs use multi-cycle mode.

The processor block clock and the BRAMDSOCMCLK must be integer multiples. The same is true for the BRAMISOCMCLK with respect to the processor block clock. They need not share the same integer values nor integer clock ratio with respect to the PLB clock. Because the clock ratio between the processor block and the OCM clocks is unknown, the processor block has control registers in the OCM controllers. The control registers are ISCNTL[0:7] and DSCNTL[0:7] for the instruction side and data side, respectively. Refer to Chapter 3, “PowerPC 405 OCM Controller” for more details.

CPU Control Interface

The CPU control interface is used primarily to provide CPU setup information to the PowerPC 405 processor. It is also used to report the detection of a machine check condition within the PowerPC 405 processor.

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.

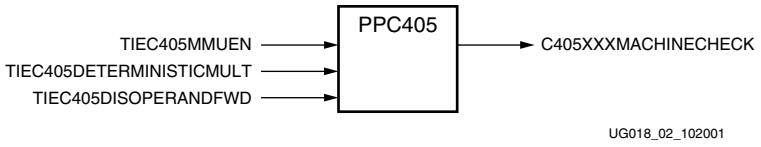


Figure 2-2: **CPU Control Interface Block Symbol**

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	0	***Important:*** *This signal should always be driven low.* 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 PowerPC 405 processor.

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 PowerPC 405 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*.

TIEC405DETERMINISTICMULT (Input)

Note: This signal should always be driven low. Setting it high may produce erroneous results.

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

Note: In Table 2-4, above, 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 the 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 may improve the PowerPC 405 performance (clock frequency).

C405XXXMACHINECHECK (Output)

When asserted, this signal indicates the PowerPC 405 processor has 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 PowerPC 405 processor 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 only the processor block, including PowerPC 405 execution units, cache units, the device control register controller (DCR), and the on-chip memory controller (OCM). On Virtex-4 FX devices, it also resets the auxiliary processor unit controller (APU). 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 occurs only 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 a combination of the two. Reset input signals must meet the following minimum requirements:

- The reset input signals must be synchronized with the PowerPC 405 clock.

- The reset input signals must be asserted for at least eight (CPMC405CLOCK) 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 sixteen 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 core reset, a debugger could be used to locate the external error source that 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.

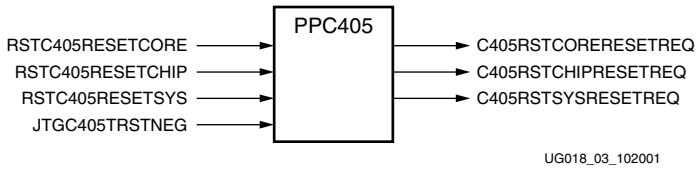


Figure 2-3: **Reset Interface Block Symbol**

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 PowerPC 405 core logic, data cache, instruction cache, and interface controllers.
RSTC405RESETCHIP	I	Required	Indicates a chip-reset occurred.
RSTC405RESETSYS	I	Required	Indicates a system-reset occurred. Resets the logic in the PowerPC 405 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 asserted, this signal 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. Unlike GSR, this output has no associated reset connectivity in the FPGA.

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. Unlike GSR, this output has no associated reset connectivity in the FPGA.

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 PowerPC 405 core logic, data cache, instruction cache, and the interface controllers. The PowerPC 405 processor 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 44 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 PowerPC 405 core logic, data cache, instruction cache, and the interface controllers). The signal does not reset logic in the processor block. The PowerPC 405 processor 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 PowerPC 405 processor does not record a chip reset type in DBSR[MRR] when this signal is deasserted.

Table 2-5, page 44 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 PowerPC 405 core logic, data cache, instruction cache, and the interface controllers). This signal resets the logic in the PowerPC 405 JTAG unit, but it does not reset any other processor block logic. The PowerPC 405 processor 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 PowerPC 405 processor 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 initialize the JTAG unit properly.

Table 2-5, page 44 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 PowerPC 405 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 PowerPC 405 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 when the accessed memory location is non-cacheable. A fetch request contains the following information:

- A fetch request is indicated by C405PLBICUREQUEST. See "C405PLBICUREQUEST (Output)."
- The target address of the instruction to be fetched is specified by the address bus, C405PLBICUABUS[0:29]. See "C405PLBICUABUS[0:29] (Output)." 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]. See "C405PLBICUSIZE[2:3] (Output)." 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. See "C405PLBICUCACHEABLE (Output)." Cacheable transfers are always performed with an eight-word transfer size.
- The user-defined storage attribute is indicated by C405PLBICUU0ATTR. See "C405PLBICUU0ATTR (Output)."
- The request priority is indicated by C405PLBICUPRIORITY[0:1]. See "C405PLBICUPRIORITY[0:1] (Output)." The PLB arbiter uses this information to prioritize simultaneous requests from multiple PLB masters.

The processor can abort a PLB fetch request using C405PLBICUABORT. See "C405PLBICUABORT (Output)." 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. See "PLBC405ICUADDRACK (Input)."
- Instructions sent from the PLB slave to the ICU during a line transfer are indicated as valid using PLBC405ICURDDACK. See "PLBC405ICURDDACK (Input)."
- The PLB-slave bus width, or size (32-bit or 64-bit), is specified by PLBC405ICUSSIZE1. See "PLBC405ICUSSIZE1 (Input)." 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]. See "PLBC405ICURDDBUS[0:63] (Input)." 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]. See "PLBC405ICURDWDADDR[1:3] (Input)."

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.
- 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.

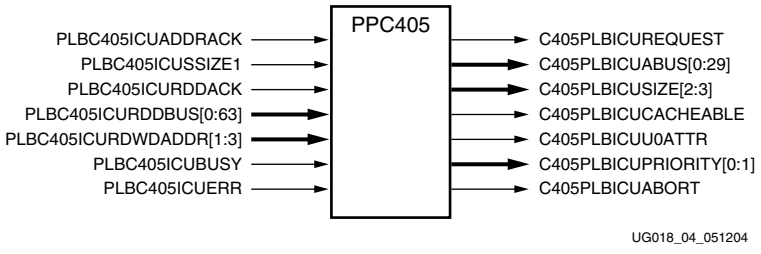


Figure 2-4: **Instruction-Side PLB Interface Block Symbol**

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.
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 "PLBCLK (Input)" 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 PowerPC 405 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 during which 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 in which they the ICU presents them. 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 in the cycles during which 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). 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 58, 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.

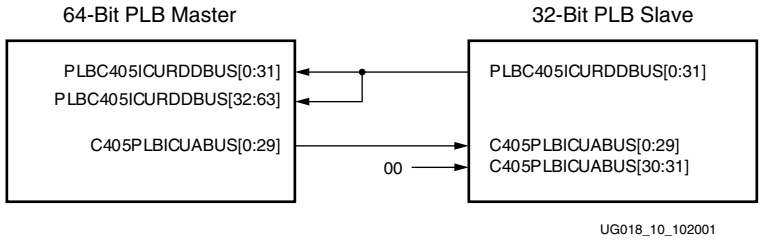


Figure 2-5: **Attachment of ISPLB Between 32-Bit Slave and 64-Bit Master**

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, above.

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]	ICU Read-Data Bus [32:63]
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. An "x" indicates a don't-care value in PLBC405ICURDWDADDR[1:3].

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 PowerPC 405 processor 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 abbreviations that appear in the timing diagrams are defined in Table 2-11.

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

Abbreviation(1)	Description	Where Used
rl#	Fetch-request identifier	Request (C405PLBICUREQUEST) Request acknowledge (PLBC405ICUADDRACK) Read-data acknowledge (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 PowerPC 405 instruction-fetch unit from the fill buffer as they become available (bypass)	ICU
prefetch#	The ICU speculatively prefetches instructions from the BIU	ICU

Table 2-11: **ISPLB Timing Diagram Abbreviations *(Cont'd)***

Abbreviation(1)	Description	Where Used	
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. The # symbol 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).

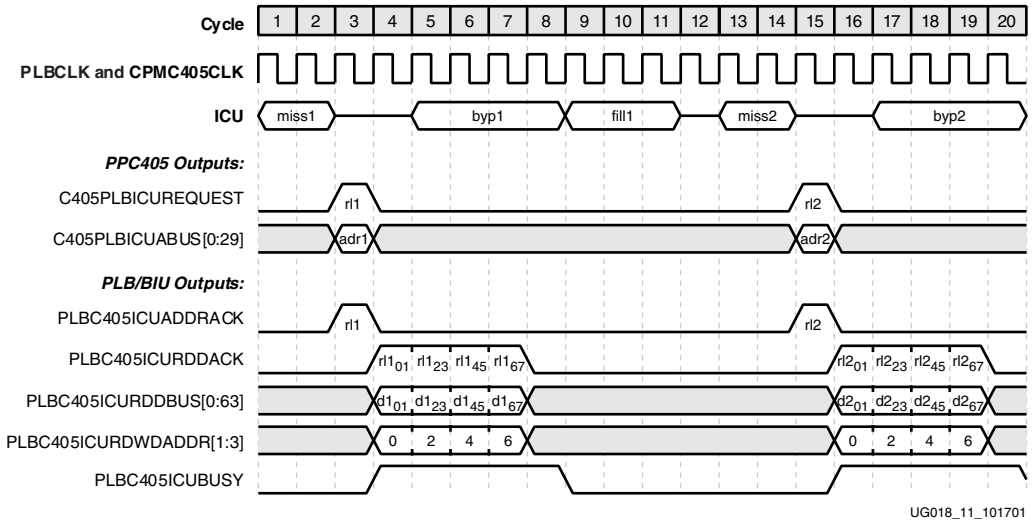


Figure 2-6: **ISPLB Non-Pipelined Cacheable Sequential Fetch (Case 1)**

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).

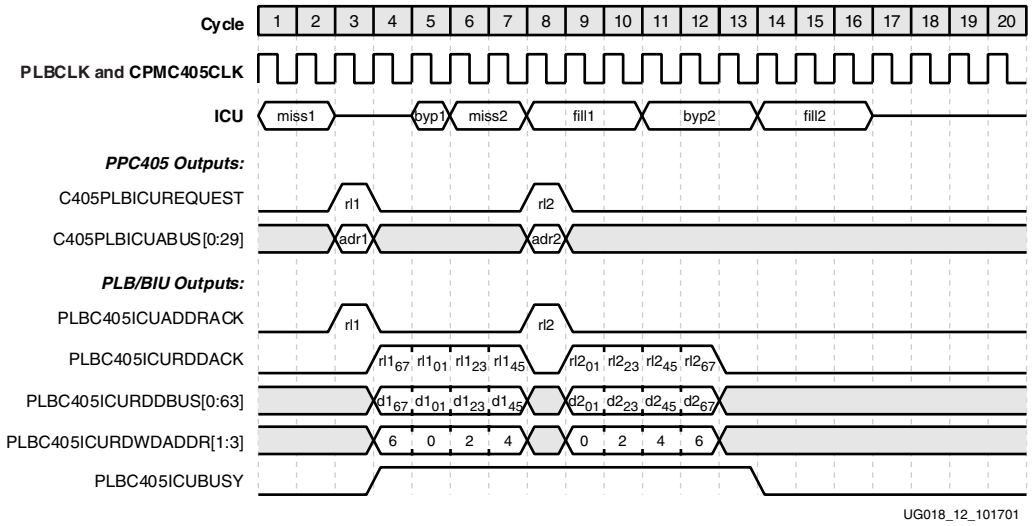


Figure 2-7: **ISPLB Non-Pipelined Cacheable Sequential Fetch (Case 2)**

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.

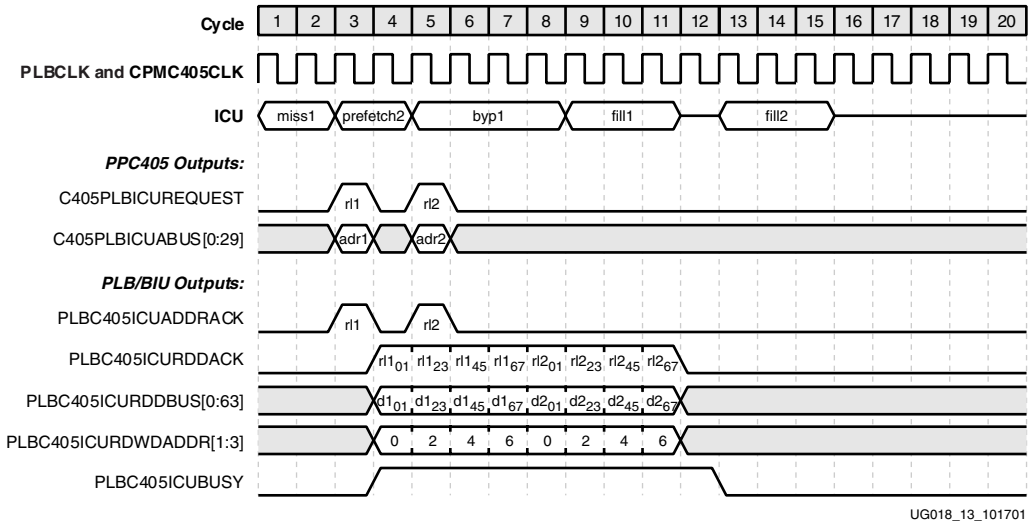


Figure 2-8: **ISPLB Pipelined Cacheable Sequential Fetch (Case 1)**

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).

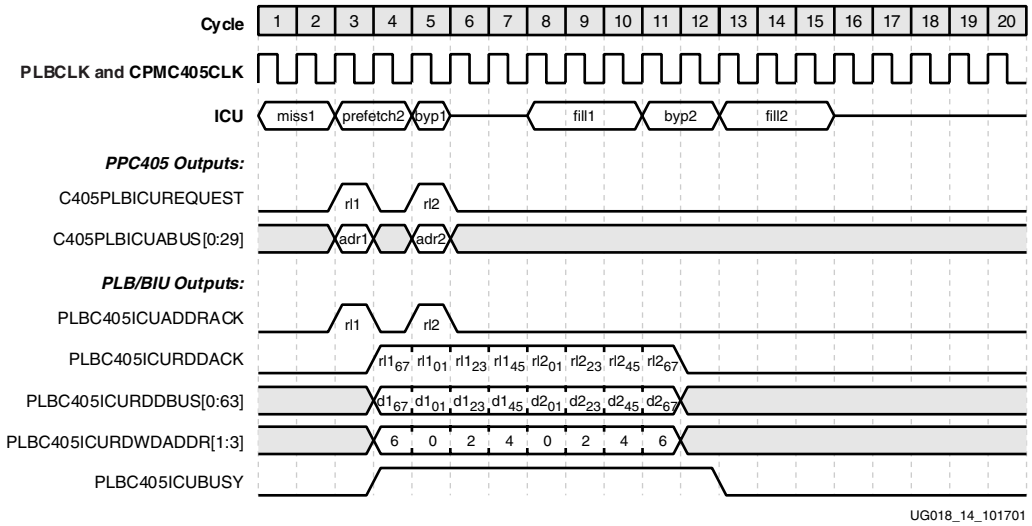


Figure 2-9: **ISPLB Pipelined Cacheable Sequential Fetch (Case 2)**

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.

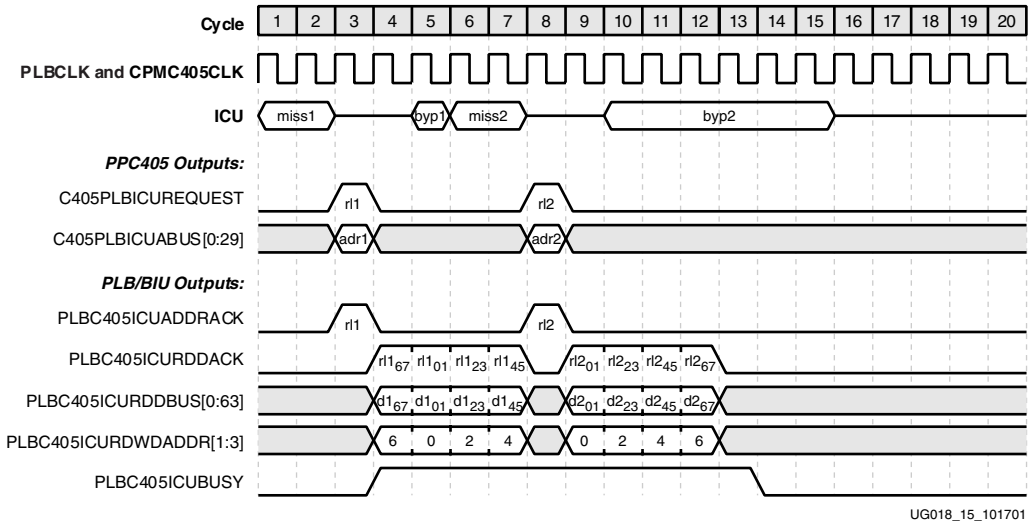


Figure 2-10: **ISPLB Non-Pipelined Non-Cacheable Sequential Fetch**

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.

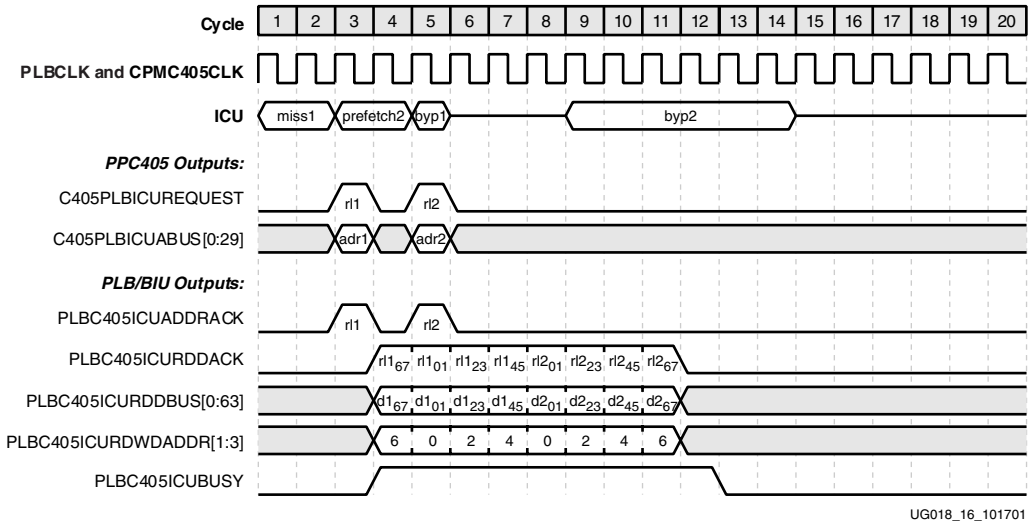


Figure 2-11: **ISPLB Pipelined Non-Cacheable Sequential Fetch**

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 PowerPC 405 clock.

The line read (rl1) is requested by the ICU in PLB cycle 2, which corresponds to PowerPC 405 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 (PowerPC 405 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).

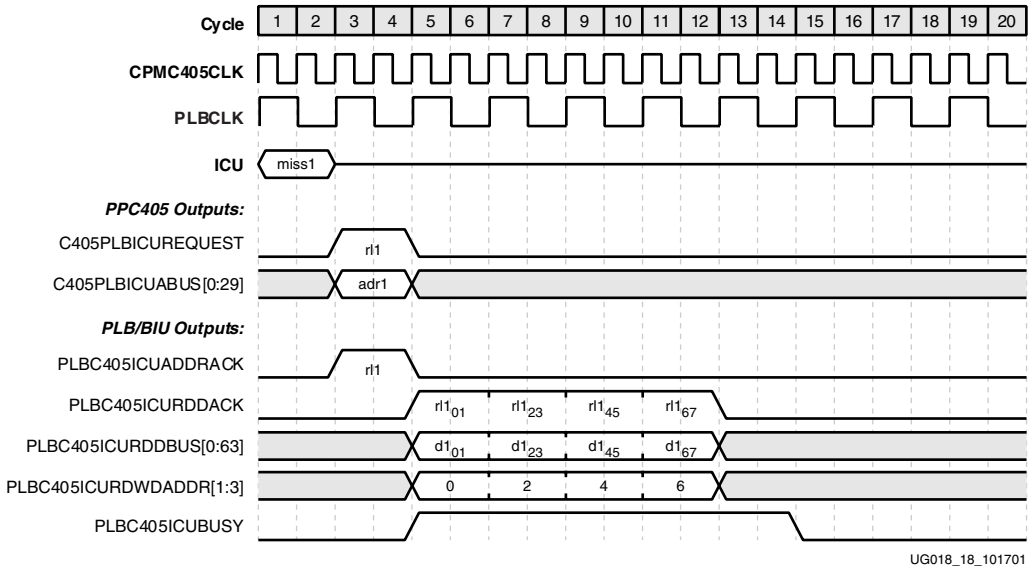


Figure 2-12: **ISPLB 2:1 Core-to-PLB Line Fetch**

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 PowerPC 405 clock.

The line read (rl1) is requested by the ICU in PLB cycle 2, which corresponds to PowerPC 405 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 (PowerPC 405 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).

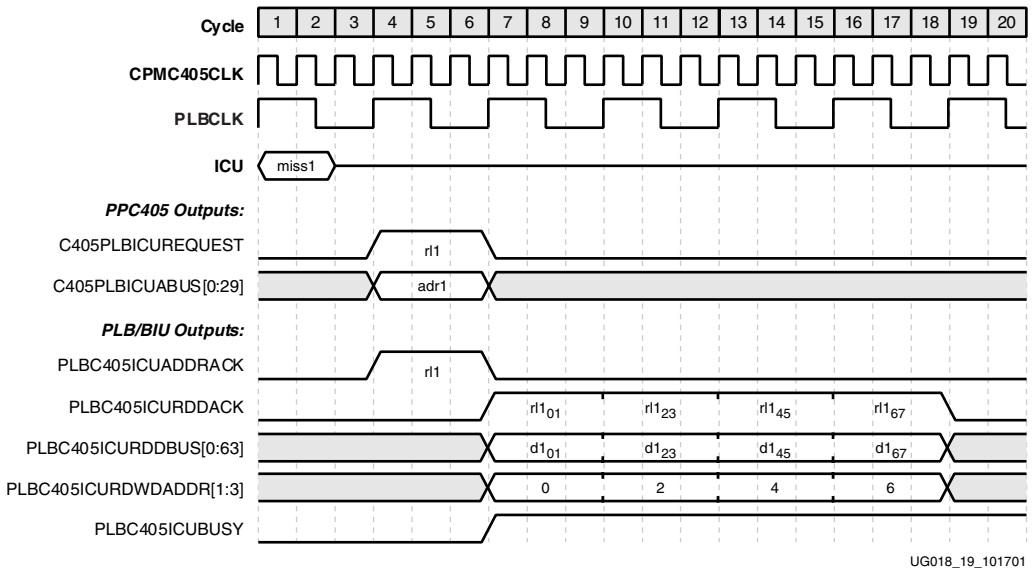


Figure 2-13: **ISPLB 3:1 Core-to-PLB Line Fetch**

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.

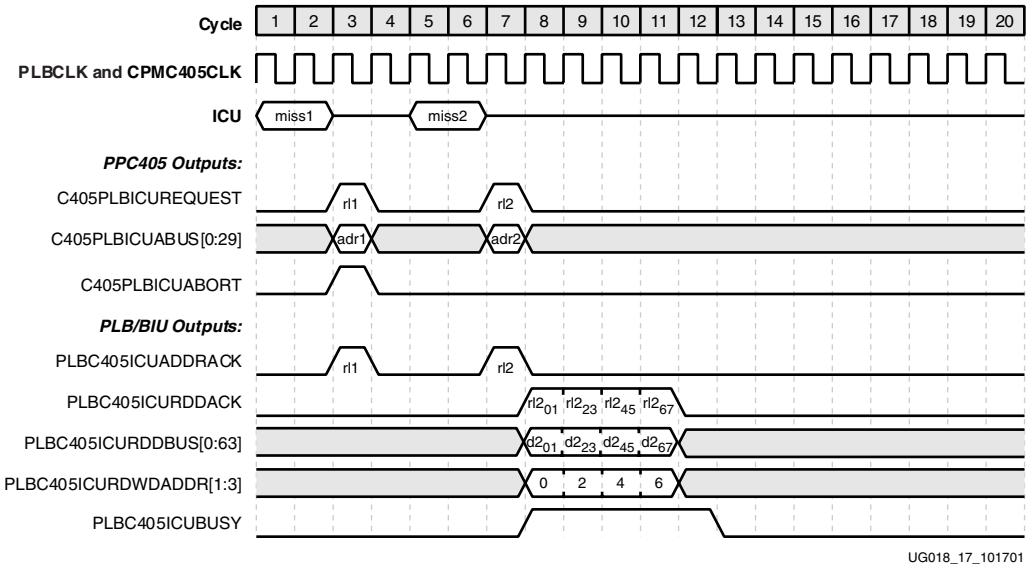


Figure 2-14: **ISPLB Aborted Fetch Request**

Data-Side Processor Local Bus Interface

The data-side processor local bus (DSPLB) interface enables the PowerPC 405 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 the *PowerPC Processor Reference Guide* for more information on the operation of the PowerPC 405 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. See "C405PLBDCUREQUEST (Output)."
- The type of request (read or write) is indicated by C405PLBDCURNW. See "C405PLBDCURNW (Output)."
- The target address of the data to be accessed is specified by the address bus, C405PLBDCUABUS[0:31]. See "C405PLBDCUABUS[0:31] (Output)."

- The transfer size is specified as a single word or as eight words (cache line) using C405PLBDCUSIZE2. See "C405PLBDCUSIZE2 (Output)." 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]. See "C405PLBDCUBE[0:7] (Output)." 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. See "C405PLBDCUCACHEABLE (Output)." Cacheable transfers are performed using word or line transfer sizes.
- The write-through storage attribute is indicated by C405PLBDCUWRITETHRU. See "C405PLBDCUWRITETHRU (Output)."
- The guarded storage attribute is indicated by C405PLBDCUGUARDED. See "C405PLBDCUGUARDED (Output)."
- The user-defined storage attribute is indicated by C405PLBDCUU0ATTR. See "C405PLBDCUU0ATTR (Output)."
- The request priority is indicated by C405PLBDCUPRIORITY[0:1]. See "C405PLBDCUPRIORITY[0:1] (Output)." 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. See "C405PLBDCUABORT (Output)." 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. See "PLBC405DCUADDRACK (Input)."
- Data sent during a read transfer from the PLB slave to the DCU over the read-data bus are indicated as valid using PLBC405DCURDDACK. See "PLBC405DCURDDACK (Input)." Data sent during a write transfer from the DCU to the PLB slave over the write-data bus are indicated as valid using PLBC405DCUWRDACK. See "PLBC405DCUWRDACK (Input)."
- The PLB-slave bus width, or size (32-bit or 64-bit), is specified by PLBC405DCUSSIZE1. See "PLBC405DCUSSIZE1 (Input)." 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]. See "PLBC405DCURDDBUS[0:63] (Input)." Data writes are transferred from the DCU to the PLB slave over the DCU write-data bus, C405PLBDCUWRDBUS[0:63]. See "C405PLBDCUWRDBUS[0:63] (Output)." 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]. See "C405PLBDCUBE[0:7] (Output)." 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]. See "C405PLBDCUABUS[0:31]

(Output)." 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]. See "PLBC405DCURDWDADDR[1:3] (Input)." 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.

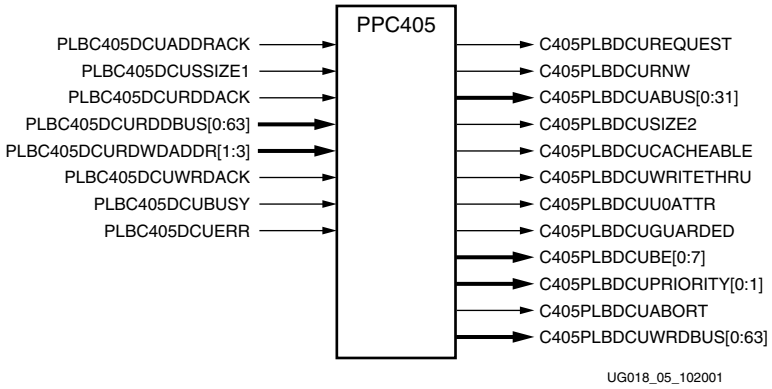


Figure 2-15: **Data-Side PLB Interface Block Symbol**

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.
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.

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

Signal	I/O Type	If Unused	Function
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 "PLBCLK (Input)" 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 PowerPC 405 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] specifies 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.

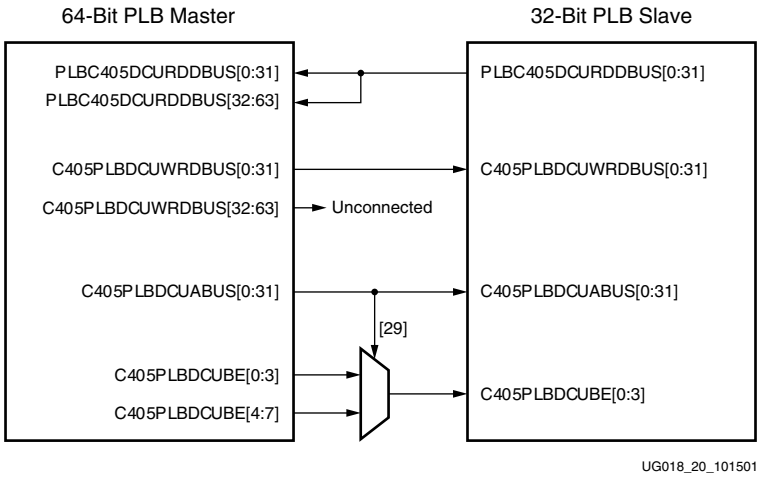


Figure 2-16: **Attachment of DSPLB Between 32-Bit Slave and 64-Bit Master**

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 encoding 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, above.

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 *(Cont'd)***

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 2-14 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.

C405PLBDCUWRDBUS[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 78. 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 58, 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 81, 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 58, 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 81, 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 78). 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 78 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 2-16 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 2-16 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 78 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]	DCU Read-Data Bus [32:63]
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. An "x" indicates a don't-care value in PLBC405DCURDWDADDR[1:3].

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 PowerPC 405 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: **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]) (C405PLBDCUWRDBUS[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]) (C405PLBDCUWRDBUS[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) (C405PLBDCUWRDBUS[0:63])
#	Used to identify the order doublewords are sent to the DCU	Transfer order	(PLBC405DCURDWDADDR[1:3])

Notes:

1. The "#" symbol 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.

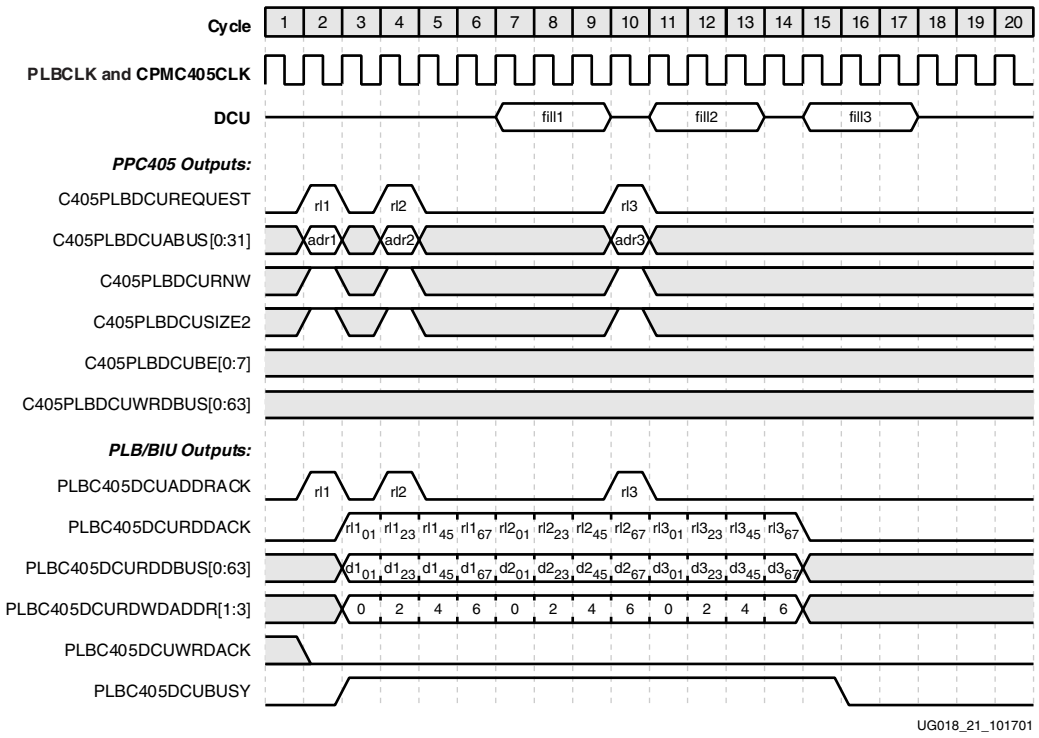


Figure 2-17: **DSPLB Three Consecutive Line Reads**

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.

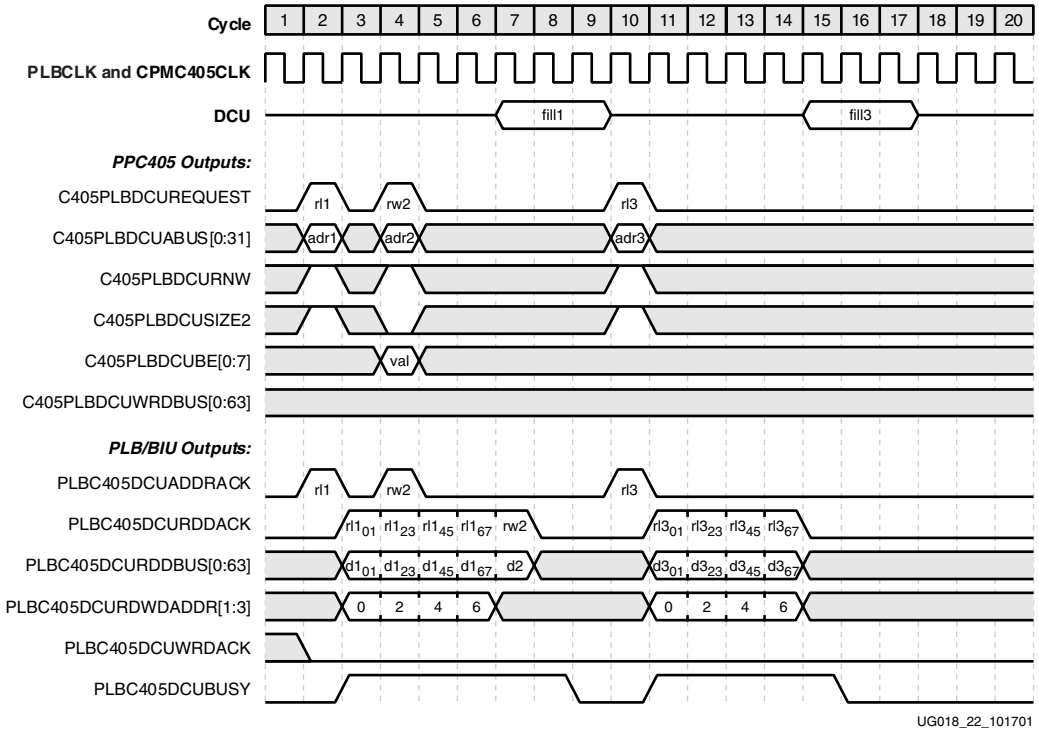


Figure 2-18: **DSPLB Line Read/Word Read/Line Read**

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 PowerPC 405 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.

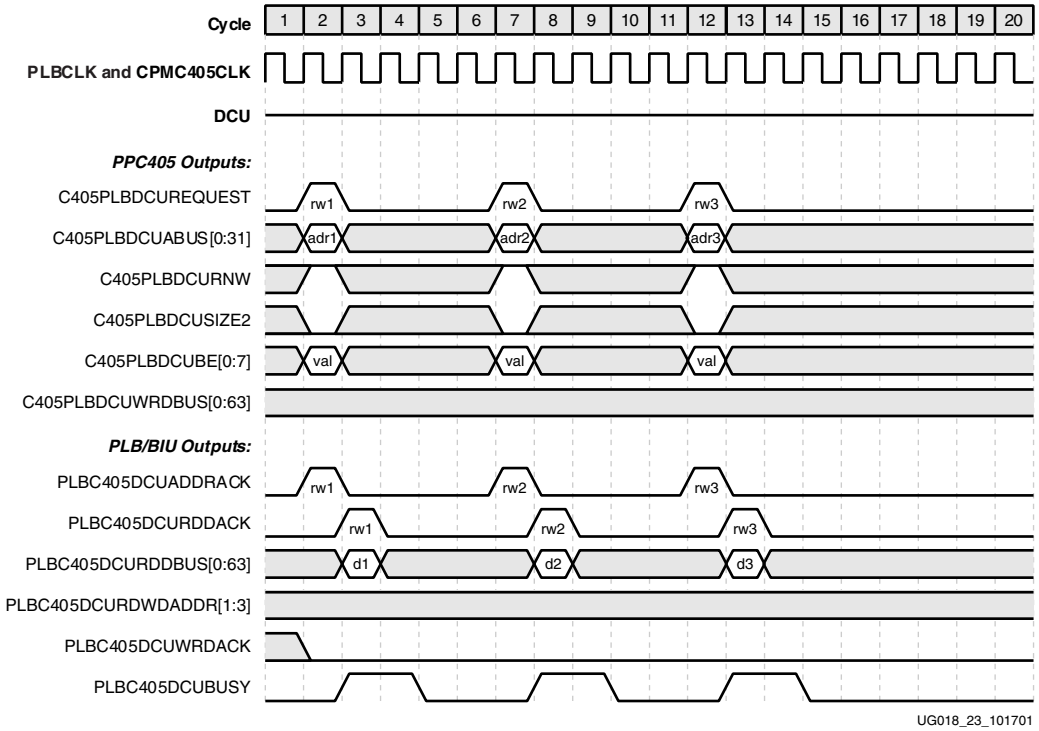


Figure 2-19: **DSPLB Three Consecutive Word Reads**

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.

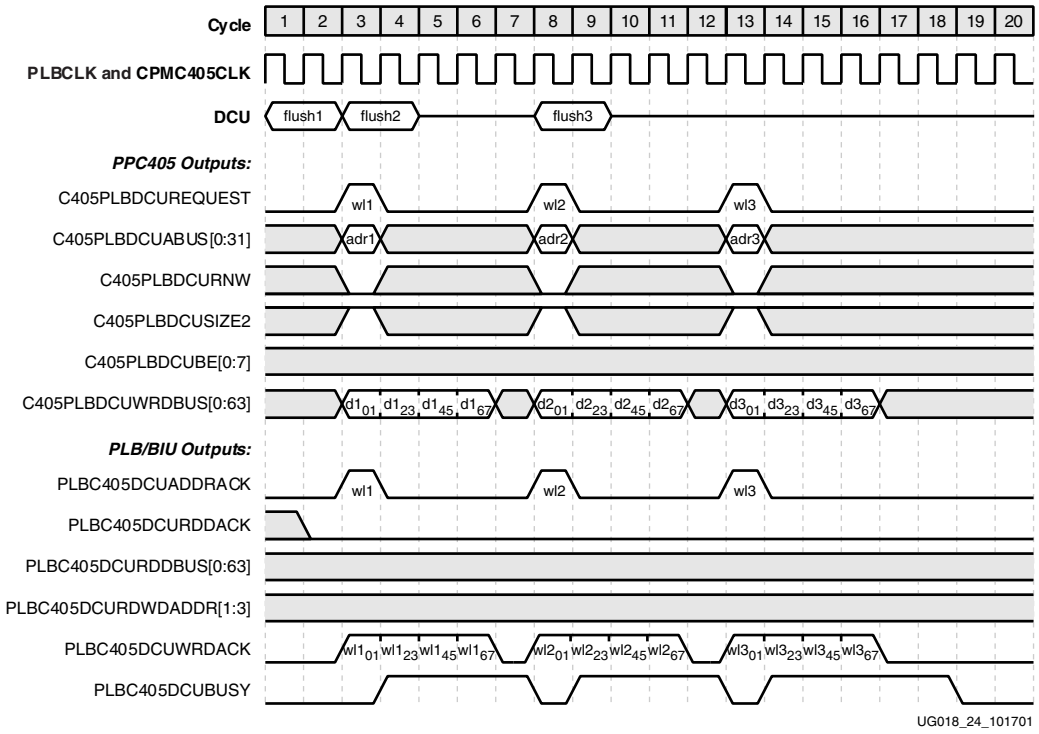


Figure 2-20: **DSPLB Three Consecutive Line Writes**

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.

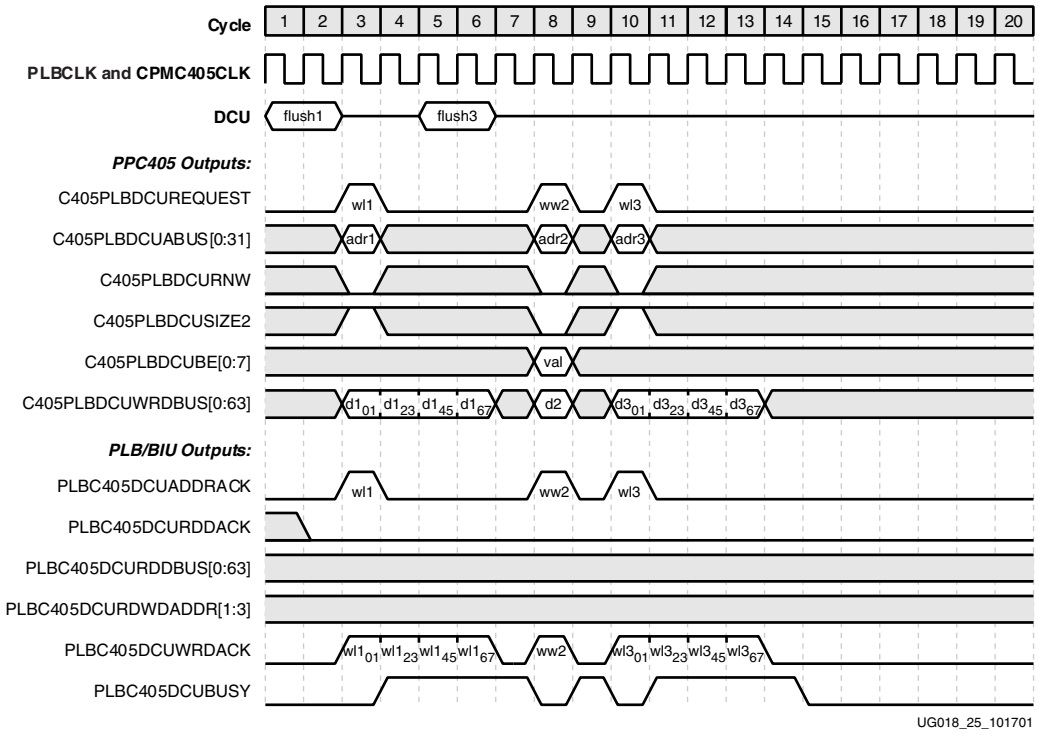


Figure 2-21: **DSPLB Line Write/Word Write/Line Write**

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.

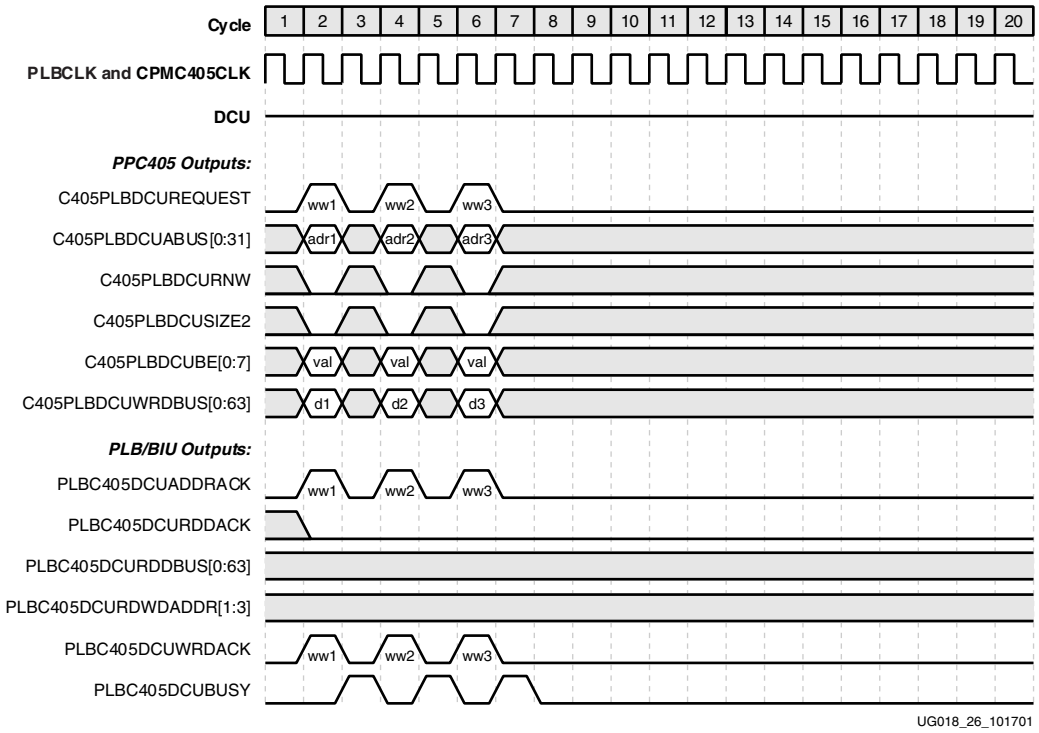


Figure 2-22: **DSPLB Three Consecutive Word Writes**

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).

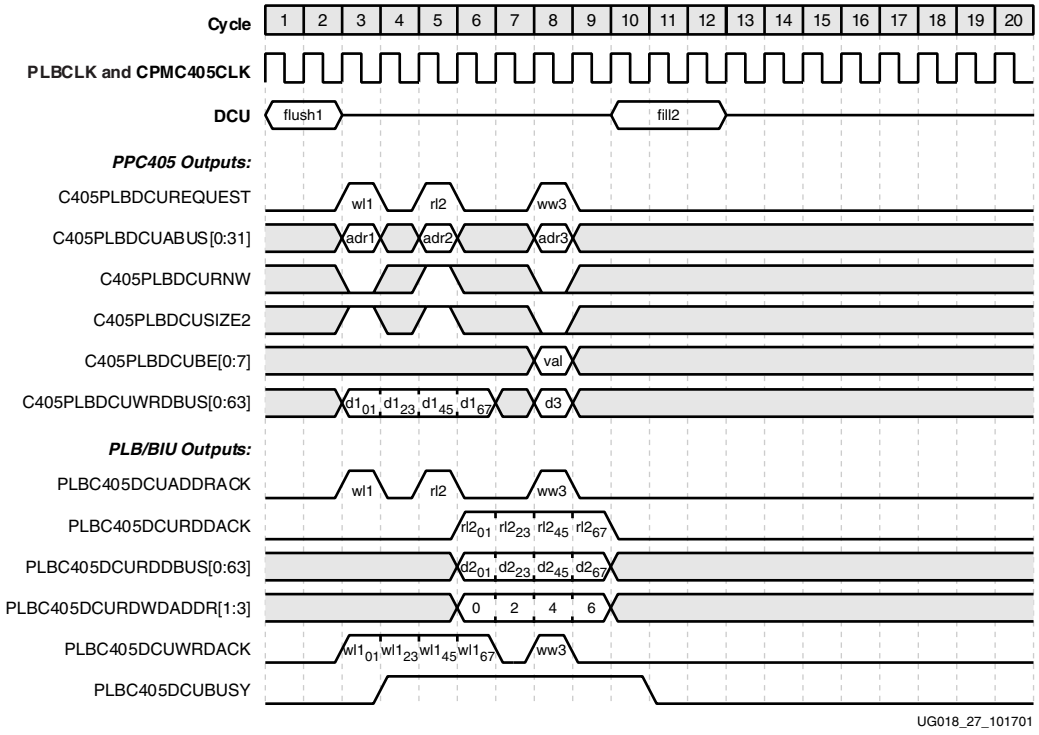


Figure 2-23: **DSPLB Line Write/Line Read/Word Write**

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.

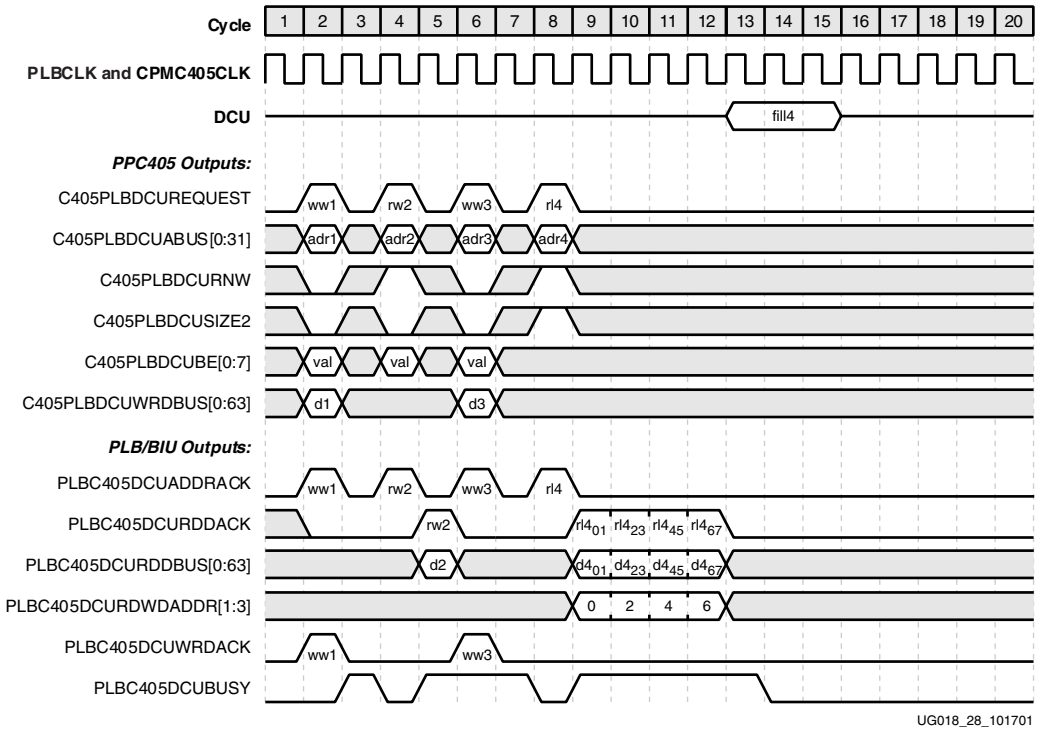


Figure 2-24: **DSPLB Word Write/Word Read/Word Write/Line Read**

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).

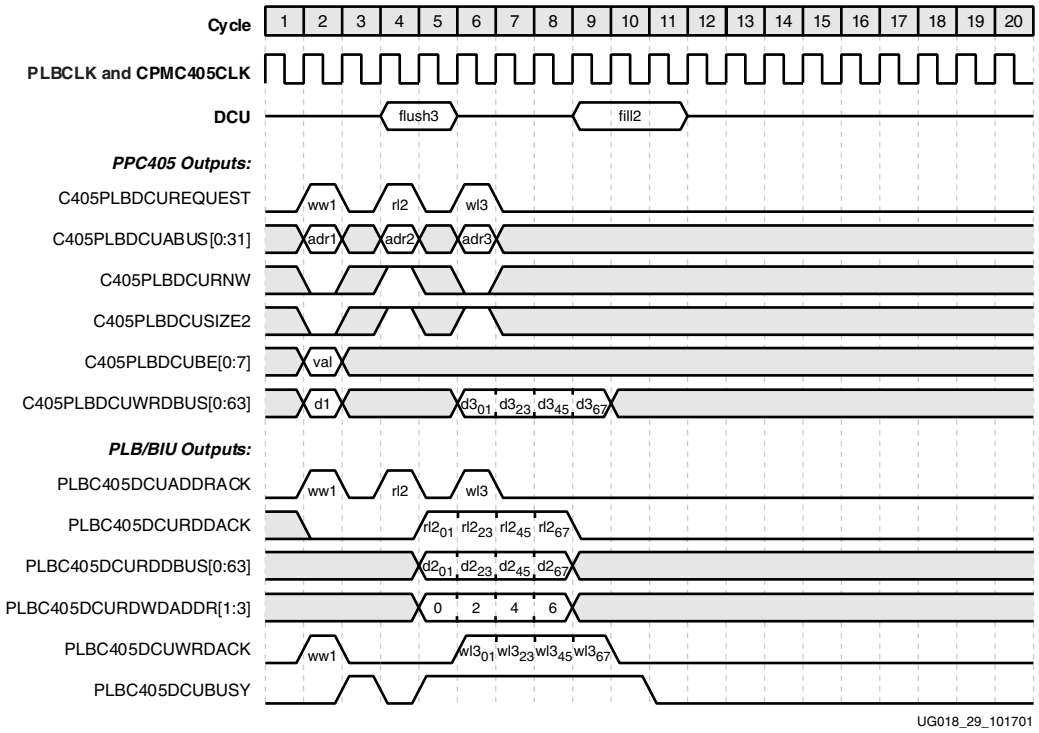


Figure 2-25: **DSPLB Word Write/Line Read/Line Write**

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 PowerPC 405 clock.

The line read (rl1) is requested by the DCU in PLB cycle 2, which corresponds to PowerPC 405 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 (PowerPC 405 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 PowerPC 405 cycles 13 through 15.

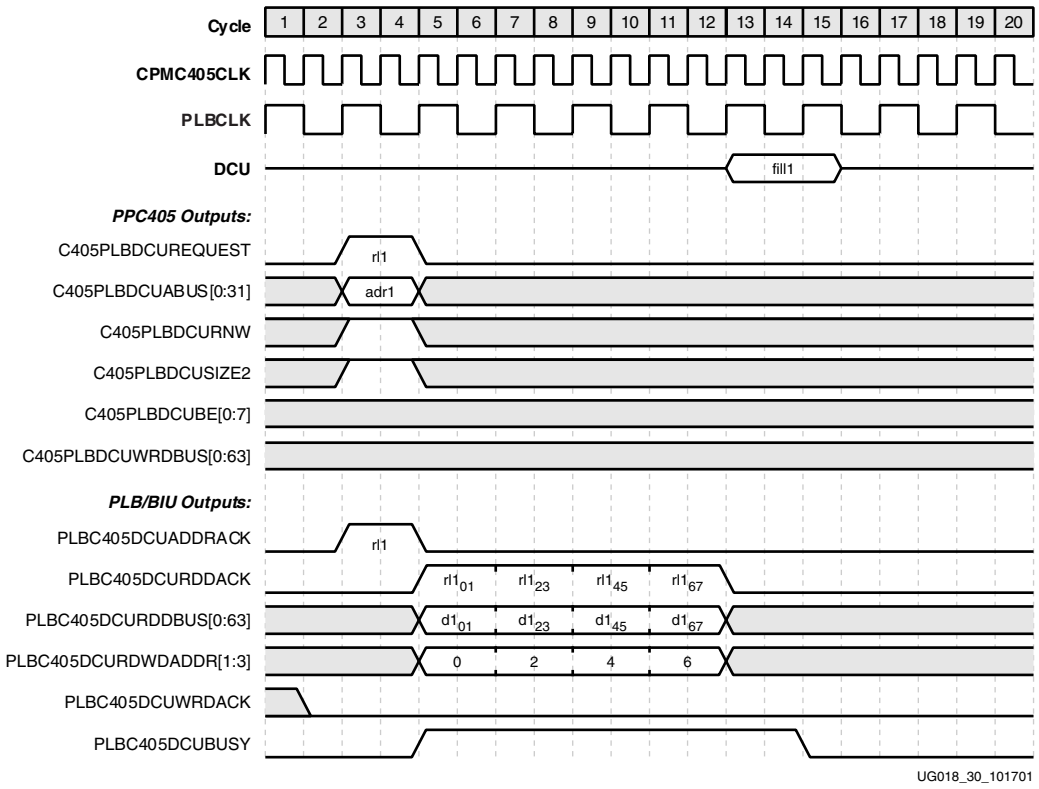


Figure 2-26: **DSPLB 2:1 Core-to-PLB Line Read**

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 PowerPC 405 clock.

The line write (wl1) is requested by the DCU in PLB cycle 2, which corresponds to PowerPC 405 cycle 4. The BIU responds in the same cycle. The request is made in response to a flush in PowerPC 405 cycles 1 and 2 (flush1). Data is sent from the DCU to the BIU in PLB cycles 2 through 5 (PowerPC 405 cycles 4 through 15).

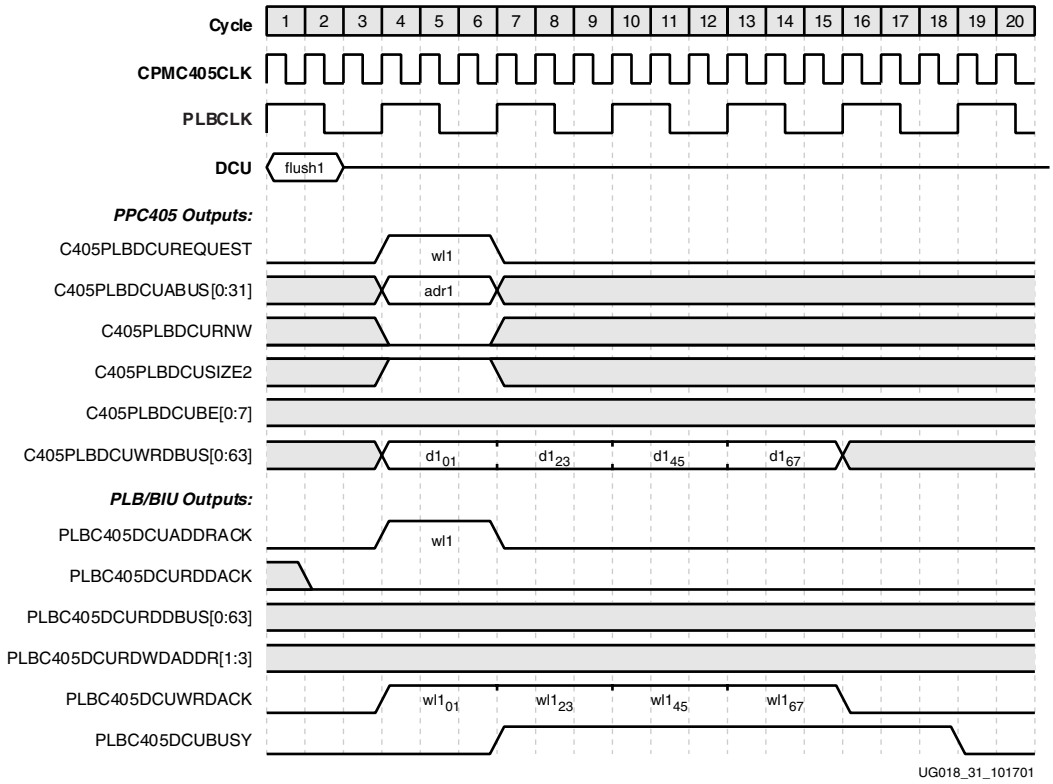


Figure 2-27: **DSPLB 3:1 Core-to-PLB Line Write**

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.

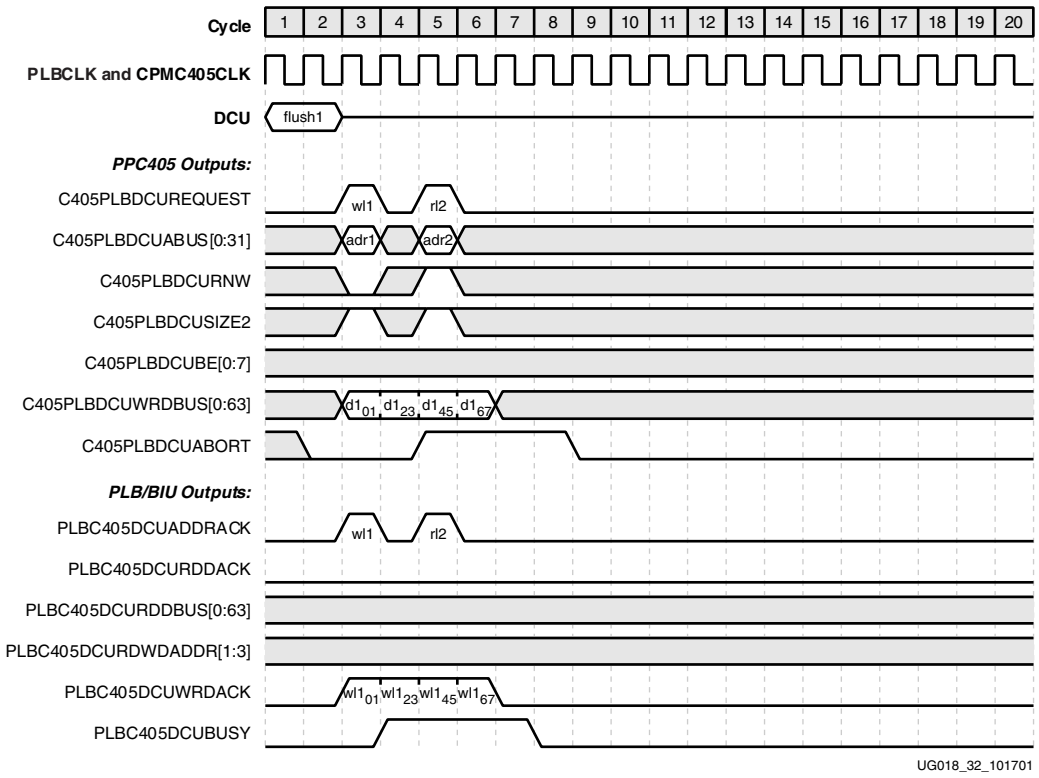


Figure 2-28: **DSPLB Aborted Data-Access Request**

Device-Control Register Interfaces

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 addressing used by these instructions is not memory mapped and thus does not interfere with OCM/PLB memory addressing. All device control registers are defined in a 10-bit, word-aligned range.

The following types of device-control register (DCR) interfaces exist:

- PowerPC block internal device-control register interface.
- General purpose DCR bus interface.
- Dedicated EMAC DCR bus interface (Virtex-4 FX designs only).

The subsequent sections will describe these interfaces and highlight differences between the Virtex-II Pro and Virtex-4 FX DCR functionary.

Internal Device Control Register (DCR) Interface

The PowerPC 405 processor block contains several internal device-control registers, which can be used to control, configure, and hold status for various functional units in the Processor block. These registers are accessed on internal DCR busses, which share their address range with the device-control registers accessed on the external DCR bus. This means that the address locations assigned for internal PowerPC DCR registers must not be populated by registers accessed over the external DCR bus.

Virtex-II Pro Processor Blocks

In Virtex-II Pro processor blocks, there are two functional units that contain device-control registers:

1. The data-side OCM (DSOCM) controller, which contains the DSCNTL and DSARC registers.
2. The instruction-side OCM (ISOCM) controller, which contains the ISCNTL, ISARC, ISINIT, and ISFILL registers.

See Chapter 3, "PowerPC 405 OCM Controller" for address mapping for these registers and for details on how Virtex-II Pro address mapping differs from Virtex-4 address mapping.

The registers contained by the DSOCM and ISOCM controllers are located in two address blocks, which are independently located in the 10-bit DCR address space The locations are defined by the input ports TIEDSOCMDCRADDR[0:7] and TIEISOCMDCRADDR[0:7]. They define the eight most significant address bits for the DSOCM and ISOCM register block addresses respectively. The individual register offset in each block is defined by the tables below:

Table 2-18: **Virtex-II Pro DSOCM DCR Address Offset**

Device Control Register	Offset
DSCNTL	3
DSARC	2
reserved	1
reserved	0

Table 2-19: **Virtex-II Pro ISOCM DCR Address Offset**

Device Control Register	Offset
ISCNTL	3
ISARC	2
ISFILL	1
ISINIT	0

For more information, please refer to the "OCM Controller Operation" section of Chapter 3, "PowerPC 405 OCM Controller."

Note: Virtex-II Pro address mapping differs from the mapping in Virtex-4 FX address mapping. To simplify porting of a design from a Virtex-II Pro device to a Virtex-4 FX device, the user must ensure that the most significant six bits of the two TIE signals are identical and that TIEISOCMDCRADDR[6:7]=00 and TIEDSOCMDCRADDR[6:7]=01.

In Virtex-II Pro devices, a DCR access addressing the internal DCR logic could be visible on the external DCR bus interface as an access.

Virtex-4 FX Processor Blocks

In Virtex-4 FX processor blocks, there are four functional units that contain device-control registers:

1. The data-side OCM (DSOCM) controller, which contains the DSCNTL and DSARC registers.
2. The instruction-side OCM (ISOCM) controller, which contains the ISCNTL, ISARC, ISINIT, and ISFILL registers.
3. The APU Controller, which contains the APUCFG and UDICFG registers.
4. The Ethernet MAC DCR Bus Interface (with a fixed connection to the hard EMAC controller), which contains the RDYstatus, cntlReg, dataRegLSW, and dataRegMSW registers.

These registers are located in a single address block in the 10-bit DCR address space using the input port TIEDCRADDR[0:5]. This input port defines the six most significant address bits of the register block address. The individual register offset in each block is defined in Table 2-20.

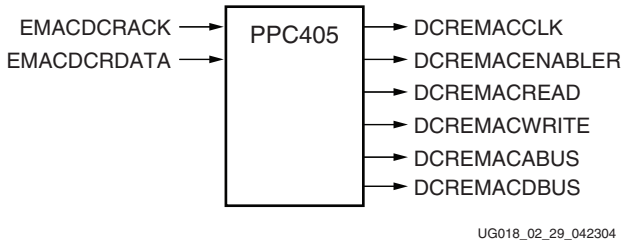
Table 2-20: **Virtex-4 FX Internal DCR Address Offset**

Block	Device Control Register	Offset
EMAC	RDYstatus	15
	cntlReg	14
	dataRegLSW	13
	dataRegMSW	12
Reserved	-	8:11
DSOCM	DSCNT	7
	DSARC	6
APU	APUCFG	5
	UDICFG	4
ISOCM	ISCNT	3
	ISARC	2
	ISFILL	1
	ISINIT	0

For more information on DCR functionality in the OCM controller, refer to the "OCM Controller Operation" section of Chapter 3, "PowerPC 405 OCM Controller."

For more information on DCR functionality in the APU controller, refer to Chapter 4, "PowerPC 405 APU Controller."

The Ethernet MAC DCR Bus interface looks like a complete DCR bus interface on the processor block symbol, however, this interface is hard wired to the pair of Ethernet MAC blocks that are associated with each PowerPC. Thus, this interface is not available to the user for connection to the FPGA fabric. Figure 2-29 shows the block symbol for the dedicated EMAC DCR interface.



Note: This block symbol is provided for completeness. Though not available to the user, the user will be able to see these signals when modeling the hardware.

Figure 2-29: **Dedicated EMAC DCR Bus Interface Block Symbol**

The output DCREMACENABLER is always asserted. By connecting this port to the DCREMACENABLE input of the embedded tri-mode Ethernet MAC, the DCR interface of the EMAC is enabled. Alternatively, the DCRENACENABLE input of the EMAC can be tied High to enable the DCR bus interface.

For more information on DCR functionality in the Ethernet MAC controller, refer to the separate *Virtex-4 Ethernet Media Access Controller User Guide*.

In Virtex-4 FX devices, a DCR access addressing the internal DCR logic will not be visible on the external DCR bus interface as an access.

External DCR Bus Interface

The DCR interface of CoreConnect DCR bus peripherals consists of the following:

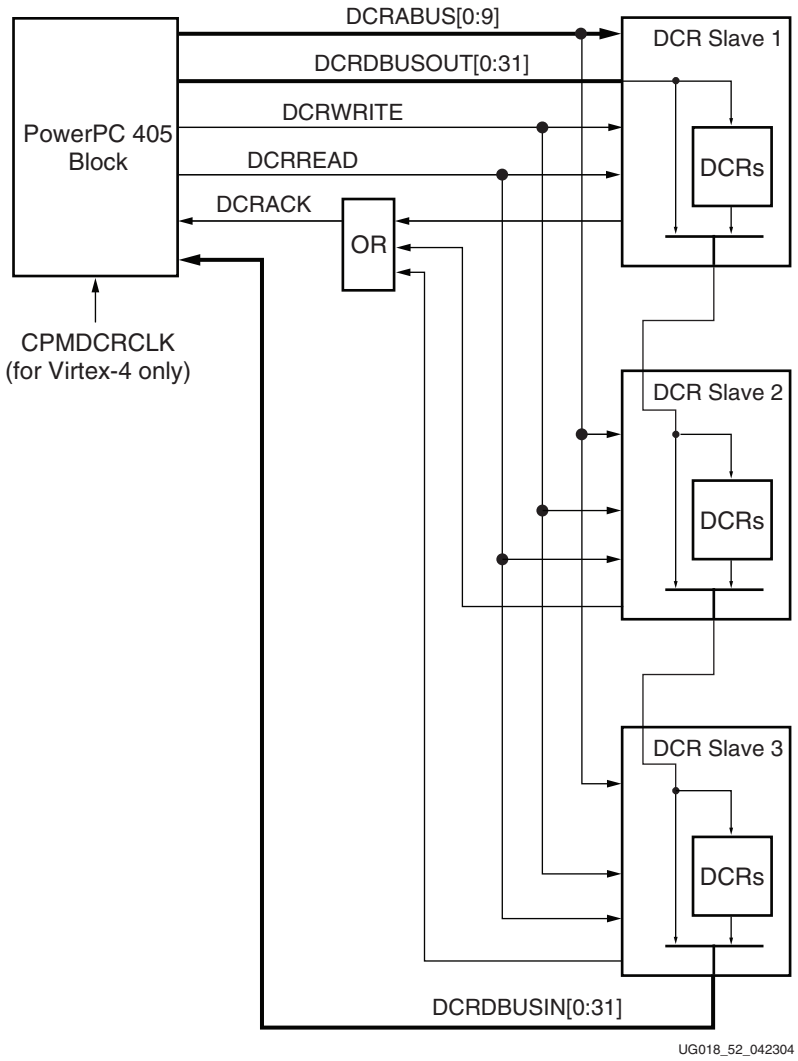
- A 10-bit address bus.
- Separate 32-bit input and output data busses.
- Separate read and write control signals.
- A read/write acknowledgement signal.

On Virtex-4 FX devices there is also a clock associated with the interface: CPMDCRCLK (see the "Clock and Power Management Interface" section of this chapter).

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 2-30 shows an example DCR chain implementation in an FPGA chip. The acknowledge signal in this example is formed using combinatorial logic (OR gate).



Note: Abbreviated signal names are used.

Figure 2-30: **DCR Chain Block Diagram**

The PowerPC external DCR interface in Virtex-II Pro devices is clocked by the processor core clock (CPMC405CLOCK), but the Virtex-4 FX external interface is clocked by an input to the processor block (CPMDCRCLK).

DCR slaves can use clock frequencies that are different (faster or slower) from the one the PowerPC 405 external DCR interface is using. The only requirement is that every rising edge of the slower clock align with a rising edge of the faster clock. This means that the clocks for the external DCR slaves and the clock for the PowerPC 405 interface must be derived from a common source. The reason different frequencies are possible is that the access protocol of the bus implements full handshaking, meaning that the Acknowledge signal sent on a Read/Write access is only deasserted after the Read/Write signal has been deasserted. If a DCR access is not acknowledged within 64 processor core cycles

(CPMC405CLOCK), the access times out. No error is flagged on time-out. The processor just continues to execute the next instruction.

Figure 2-31 illustrates a logical implementation of the DCR bus interface. This implementation enables a DCR slave to run at a different clock speed than the PowerPC 405 processor. 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.

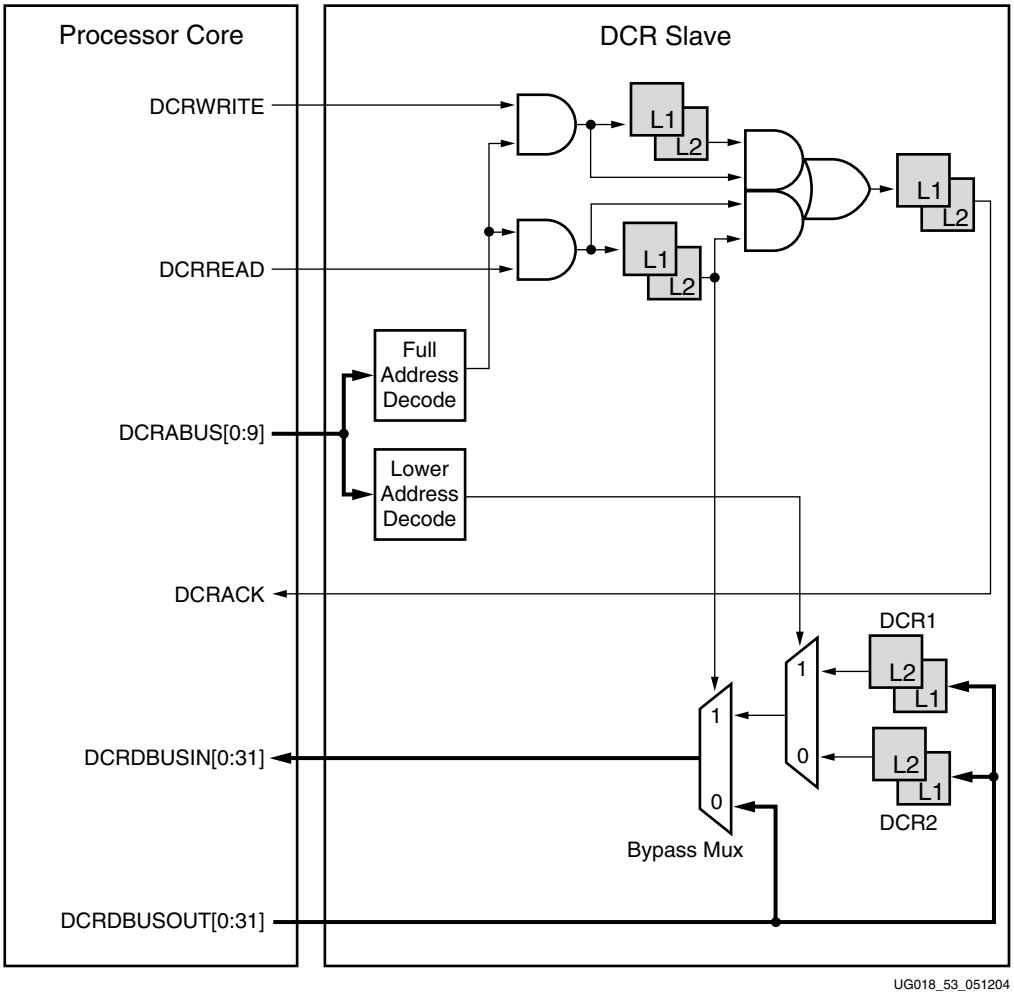


Figure 2-31: **DCR Bus Implementation**

External DCR Bus Interface I/O Signal Summary

Virtex-II Pro DCR Interface

Figure 2-32 shows the block symbol for the DCR interface. The signals are summarized in Table 2-21.

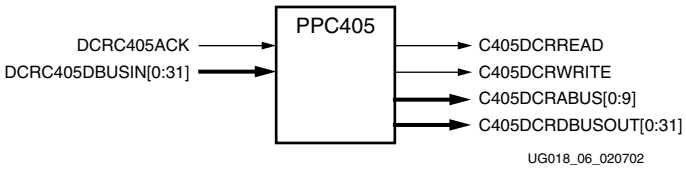


Figure 2-32: **Virtex-II Pro DCR Interface Block Symbol**

Table 2-21: **Virtex-II Pro 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.

Virtex-4 FX DCR Interface

The external general purpose Virtex-4 FX DCR interface is identical to its predecessors with the following exceptions:

- Dedicated, re-synchronization registers implemented in the PowerPC block.
- Interface signals are renamed

The re-synchronization registers allow decoupling of the internal PowerPC clock frequency from the DCR bus transactions by re-synchronizing the interface to a dedicated DCR clock (CPMDCRCLK, see "Clock and Power Management Interface"). This ensures that the internal PowerPC clock frequency can be kept high regardless of DCR transaction speed.

The Table 2-22 describes the name mapping between the Virtex-4 FX DCR interface signals relative to Virtex-II Pro signals.

Table 2-22: **Virtex-4 FX DCR Interface Name Correlation with Virtex-II Pro Names**

Virtex-4 FX Name	Virtex-II Pro Names
EXTDCRREAD	C405DCRREAD
EXTDCRWRITE	C405DCRWRITE
EXTDCRABUS[0:9]	C405DCRABUS[0:9]
EXTDCRDBUSOUT[0:31]	C405DCRDBUSOUT[0:31]
EXTDCRACK	DCRC405ACK
EXTDCRDBUSIN[0:31]	DCRC405DBUSIN[0:31]

External DCR Bus Interface I/O Signal Descriptions

The following sections describe the operation of the DCR interface I/O signals. Signals are presented with both Virtex-II Pro and Virtex-4 FX names.

C405DCRREAD/EXTDCRREAD (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.

In Virtex-II Pro designs the request is asserted one CPMC405CLOCK cycle after the processor block begins driving the DCR address bus and it is deasserted two cycles after the DCR acknowledge signal is asserted. In Virtex-4 FX designs the request is asserted in the same CPMDCRCLK cycle as, or one cycle after, the processor block begins driving the DCR address bus and it is deasserted at least one cycle after the DCR acknowledge signal is asserted. DCR read requests are not interrupted by the processor block. If this signal is asserted, only a DCR acknowledgement or read time-out will deassert it. For details see signal "DCRC405ACK/EXTDCRACK (Input)."

This signal is deasserted during reset.

C405DCRWRITE/EXTDCRWRITE (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).

In Virtex-II Pro designs the request is asserted one CPMC405CLOCK cycle after the processor block begins driving the DCR address and write-data bus. It is deasserted two cycles after the DCR acknowledge signal is asserted. In Virtex-4 FX the request is asserted in the same CPMDCRCLK cycle as, or one cycle after, the processor block begins driving the DCR address and write-data bus. It is deasserted at least one cycle after the DCR acknowledge signal is asserted. DCR write requests are not interrupted by the processor block. If this signal is asserted, only a DCR acknowledgement or write time-out will deassert it. For details see signal "DCRC405ACK/EXTDCRACK (Input)."

This signal is deasserted during reset.

C405DCRABUS[0:9]/EXTDCRABUS[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.

C405DCRDBUSOUT[0:31]/EXTDCRDBUSOUT[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/EXTDCRACK (Input)

When asserted, this signal indicates a peripheral device acknowledges the processor block request for DCR access. A peripheral device should assert this signal only when all of the following are true:

- The peripheral device contains the addressed DCR.
- A DCR read or write request exists.
- The peripheral device is driving the DCR data bus (read access).
- The peripheral device latched the DCR data bus (write access).

The acknowledgement should not be deasserted until the read/write signal is deasserted. This allows the PowerPC 405 processor and a peripheral device to be clocked at different frequencies without affecting the interface handshaking protocol.

The processor block waits up to 64 processor core clock (CPMC405CLOCK) cycles for a read/write request to be acknowledged. If a DCR does not acknowledge the request in this time, the access times out. No error occurs when a DCR access is timed-out, the processor simply goes on to execute the next instruction.

DCRC405DBUSIN[0:31]/EXTDCRDBUSIN[0:31] (Input)

This read-data bus is latched 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.

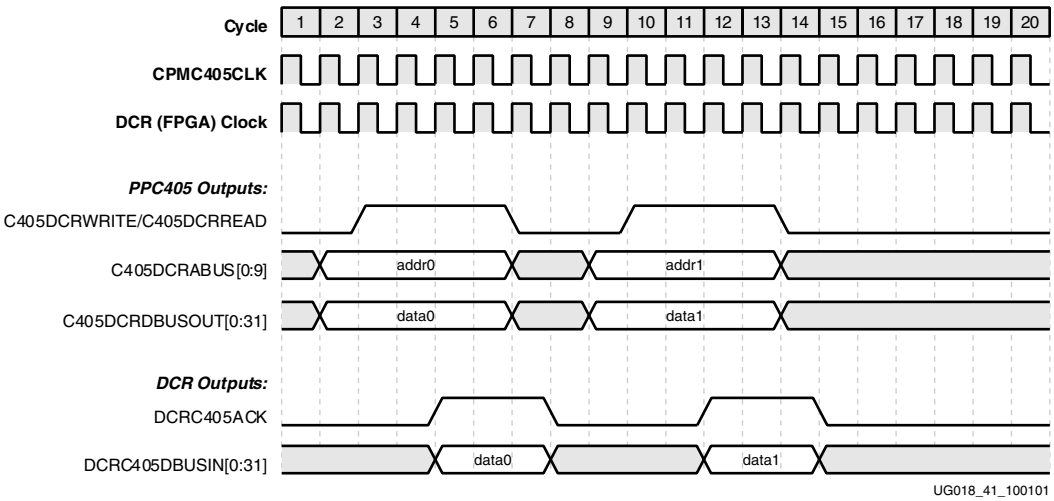
External DCR Bus Interface Timing Diagrams

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 DCRREAD/DCRWRITE 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 1:1 Clocking, Latched Acknowledge

The example in Figure 2-33 assumes the following:

- The PowerPC 405 processor 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 2-31, page 105.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted.



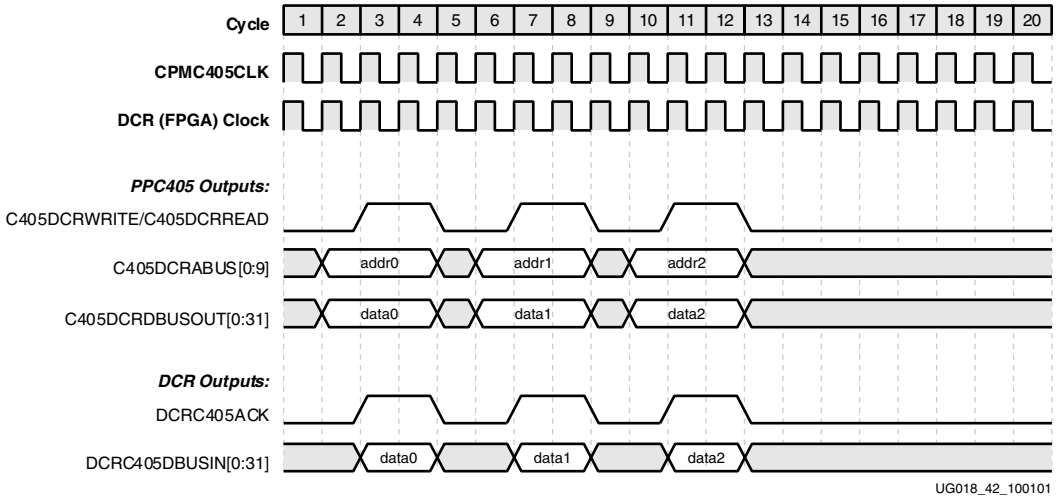
Note: Abbreviated signal names are used.

Figure 2-33: **DCR Interface 1:1 Clocking, Latched Acknowledge**

DCR Interface 1:1 Clocking, Combinatorial Acknowledge

The example in Figure 2-34 assumes the following:

- The PowerPC 405 processor and the peripheral containing the DCR are clocked at the same frequency.
- The acknowledge signal is generated by combinatorial logic from the DCR read/write signal.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted.



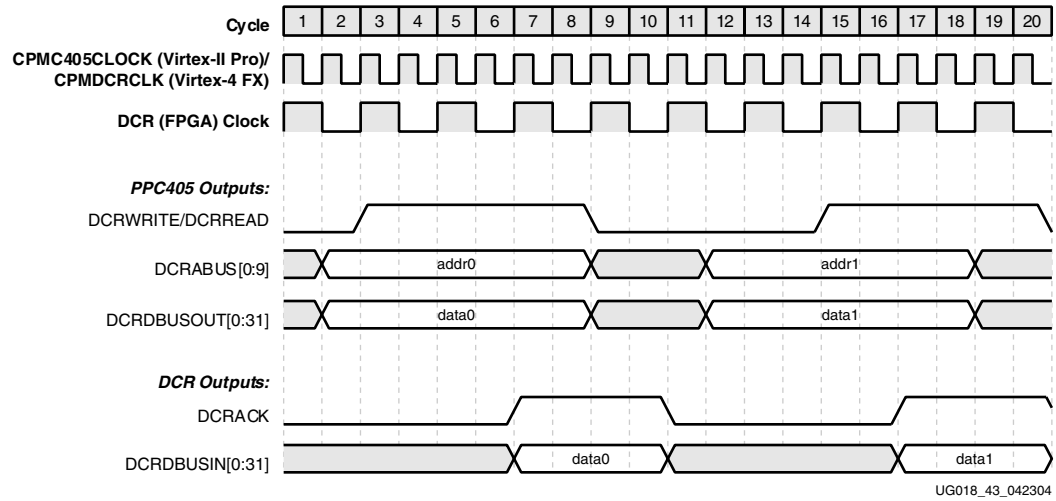
Note: Abbreviated signal names are used.

Figure 2-34: **DCR Interface 1:1 Clocking, Combinatorial Acknowledge**

DCR Interface 2:1 Clocking, Latched Acknowledge

The example in Figure 2-35 assumes the following:

- The PowerPC 405 DCR interface is clocked at twice the frequency of the peripheral containing the addressed DCR.
- The acknowledge signal is latched and forwarded with the DCR bus as shown in Figure 2-31, page 105.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted.



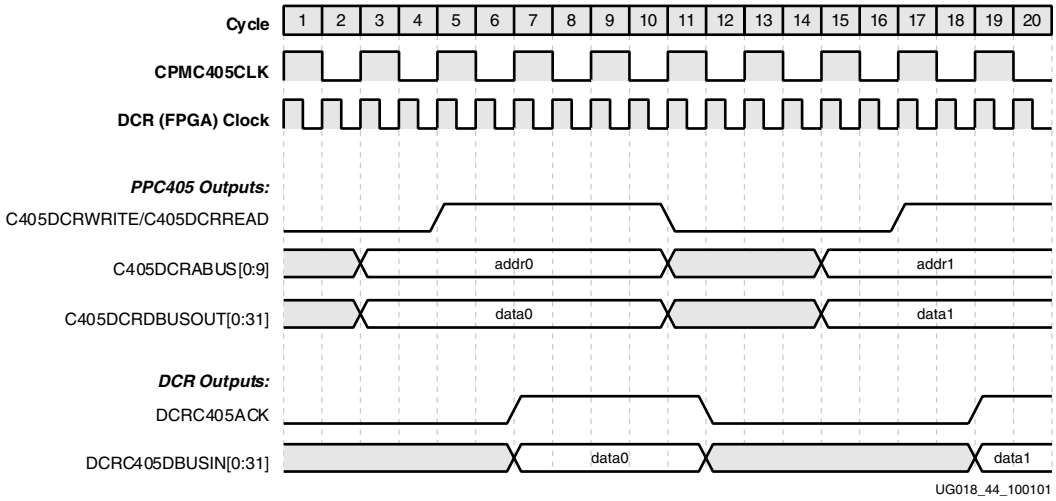
Note: Abbreviated signal names are used.

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

DCR Interface 1:2 Clocking, Latched Acknowledge

The example in Figure 2-36 assumes the following:

- The PowerPC 405 DCR interface is clocked at half the frequency of the peripheral containing the addressed DCR.
- The acknowledge signal is latched and forwarded with the DCR bus as shown in Figure 2-31, page 105.
- After the acknowledge signal is asserted, it is not deasserted until the appropriate read-access or write-access request signal is deasserted.



Note: Abbreviated signal names are used.

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

External DCR Timing Consideration (Virtex-II Pro)

Users need to be aware that there is no DCR clock input to the processor block of the Virtex-II Pro FPGAs. When dealing with signals that cross CPU clock domain and DCR clock domain, users may want to add re-synchronization flip-flops to simply timing constraints, or set up appropriate multi-cycle/false path constraints in the UCF file.

An example for the re-synchronization of DCR interface can be found in Xilinx Embedded Development Kit (EDK). Please refer to the Virtex-II Pro PowerPC 405 wrapper IP datasheet (DS304) for details.

The Virtex-4 FX family does have a DCR clock input and does not have the synchronization issues mentioned here.

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 35, 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 2-23.

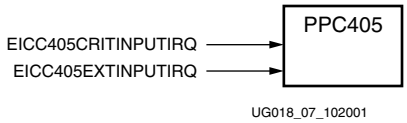


Figure 2-37: **EIC Interface Block Symbol**

Table 2-23: **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).

PPC405 JTAG Debug Port

The 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 FPGA must be configured before the PPC405 JTAG interface is available.

The PPC405 JTAG logic may be connected through the native JTAG port (series connection) of the FPGA, 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 Signals

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

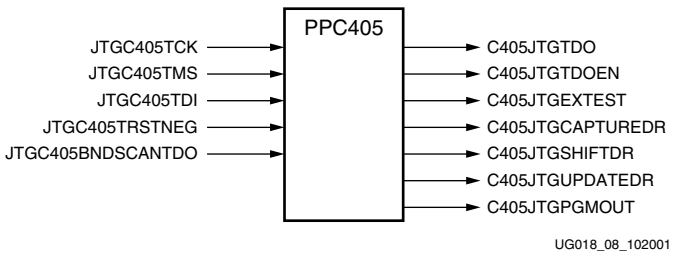


Figure 2-38: **JTAG Interface Block Symbol**

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 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.

JTAG Instruction Register

Virtex-II Pro and Virtex-4 FX devices contain zero, one, or two PowerPC405 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 2-24 gives the IR length for all Virtex-II Pro and Virtex-4 FX devices.

Table 2-24: **Virtex-II Pro and Virtex-4 FX IR Lengths**

Device	# PPC405 Cores	IR Length
XC2VP2	0	6
XC2VP4	1	10
XC2VP7	1	10
XC2VP20	2	14
XC2VPX20	1	10
XC2VP30	2	14
XC2VP40	2	14
XC2VP50	2	14
XC2VP70	2	14
XC2VPX70	2	14
XC2VP100	2	14
XC4VFX12	1	10
XC4VFX20	1	10
XC4VFX40	2	14
XC4VFX60	2	14
XC4VFX100	2	14
XC4VFX140	2	14

The six least significant bits of the parts 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 Device 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 Instruction Register data path, and Figure 2-40 illustrates the data path for the series PPC405 JTAG connection.

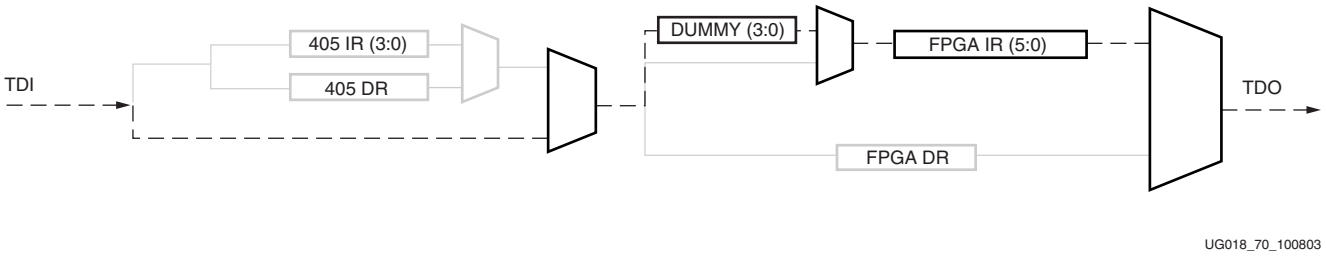


Figure 2-39: **Default Instruction Register Data Path in Virtex Devices with a Single PPC405 core**

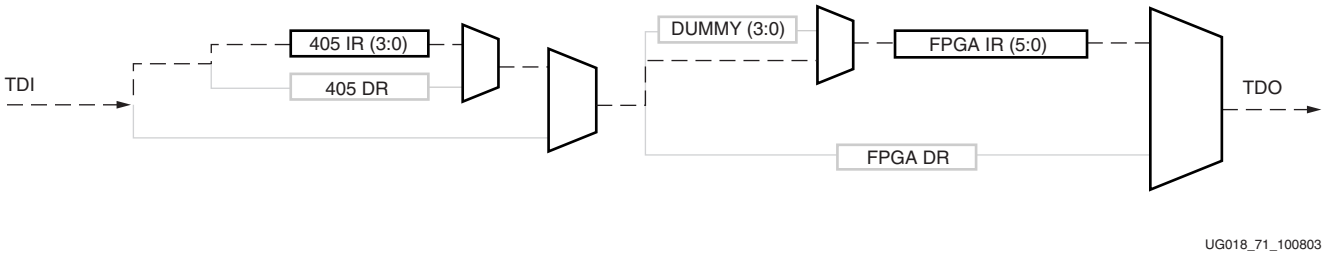


Figure 2-40: **Instruction Register Data Path for Series PPC405 JTAG Connection**

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.

Table 2-25: **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 device 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 device 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 2-41); chain the processors together with programmable interconnect and wire the combined PPC405 JTAG chain to programmable I/O (Figure 2-42) or multiplex a single set of JTAG pins to multiple cores (Figure 2-43).

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 device JTAG pins because the chain will be broken by the missing PPC405 JTAG logic prior to FPGA configuration.

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 10 KΩ pullup resistor should be placed on the trace.

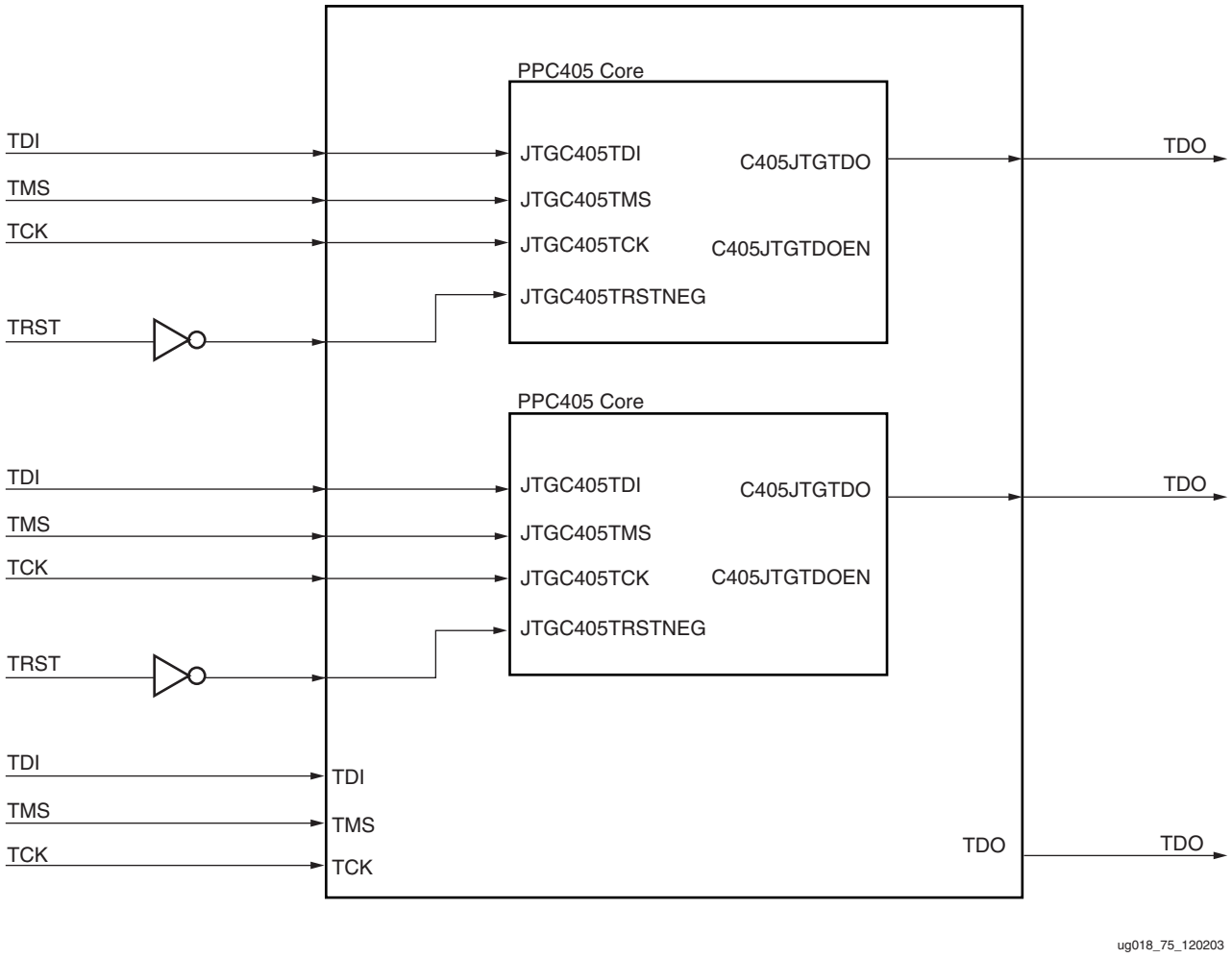


Figure 2-41: **Correct Wiring of JTAG Chains with Individual PPC405 Connections (Separate JTAG Chains)**

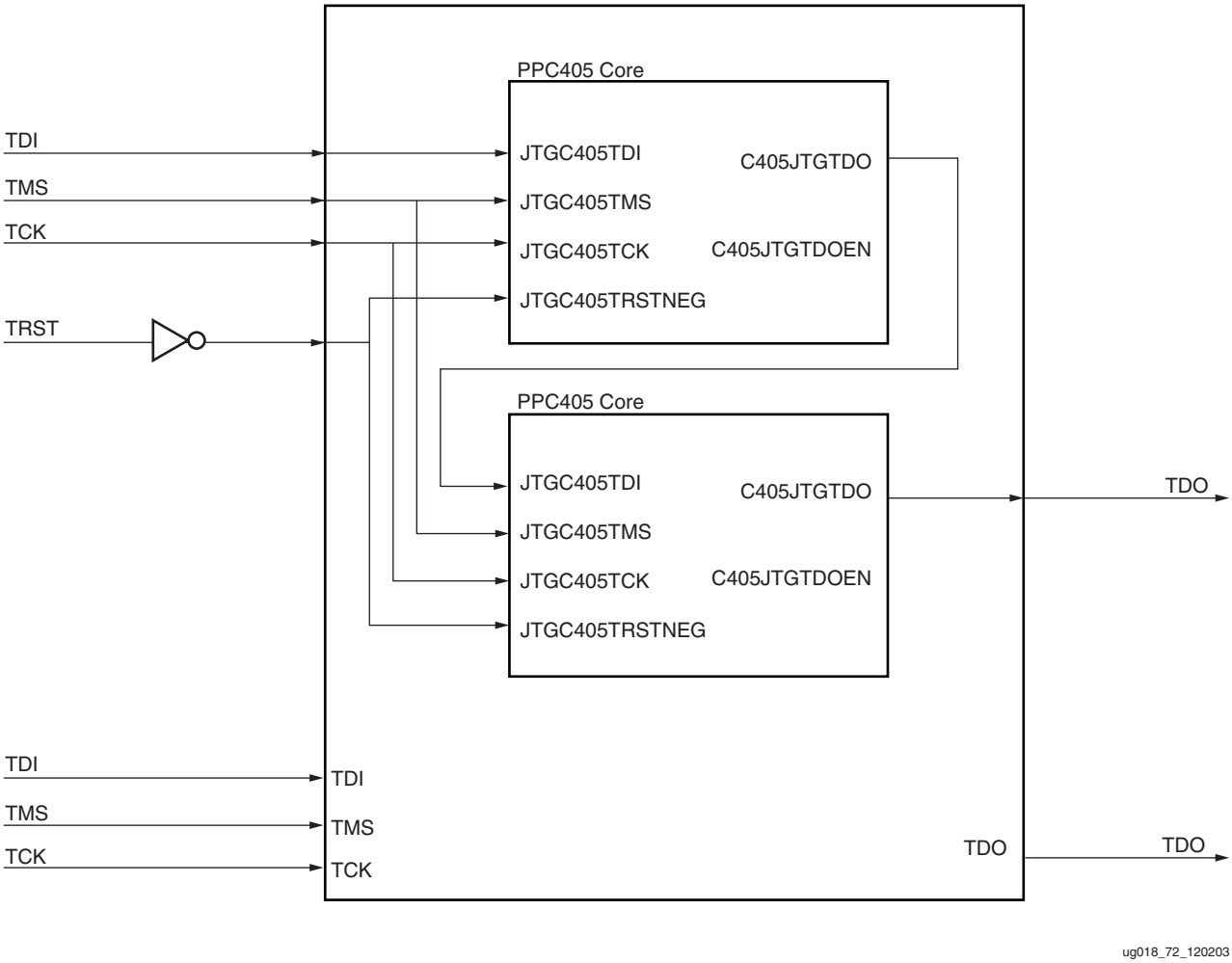


Figure 2-42: **Correct Wiring of JTAG Chains with Individual PPC405 JTAG Connections (Internally Chained PPC405 Cores)**

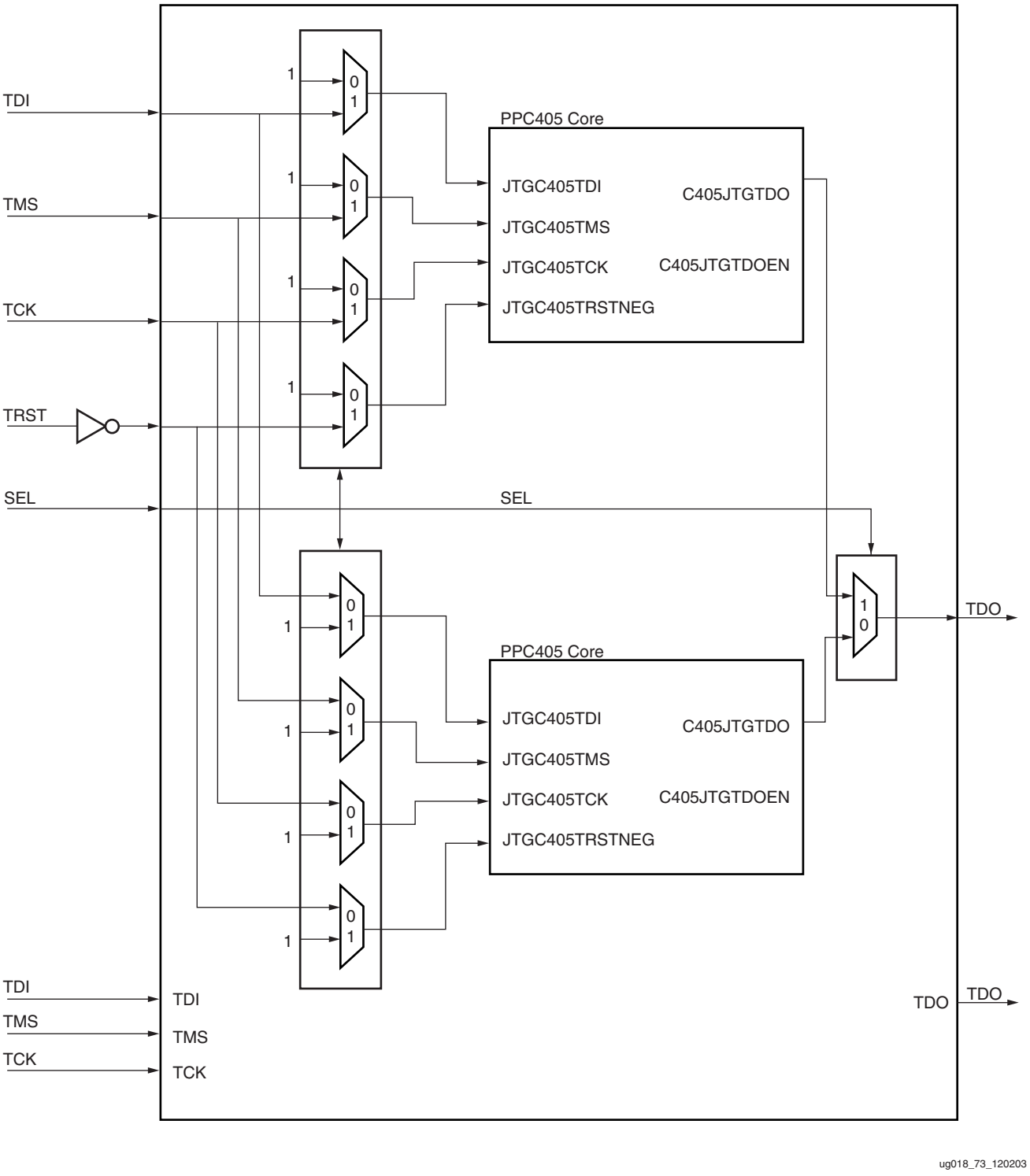


Figure 2-43: **Correct Wiring of JTAG Chain with Multiplexed PPC405 Connection**

Connecting PPC405 JTAG Logic in Series with the Dedicated Device JTAG Logic

An alternative to connecting the PPC405 JTAG logic directly to programmable I/O is to wire it in series with the dedicated device 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 "JTAG Instruction Register" section above, the 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 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 device JTAG logic, the most significant IR bits are used by the PPC405 cores. Thus, the overall IR length remains the same for the device at all times.

When the PPC405 JTAG logic is connected in series with the dedicated JJTAG 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 ORed 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 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 2-44.

For more information, see the Virtex-II Pro or Virtex-4 user guides.

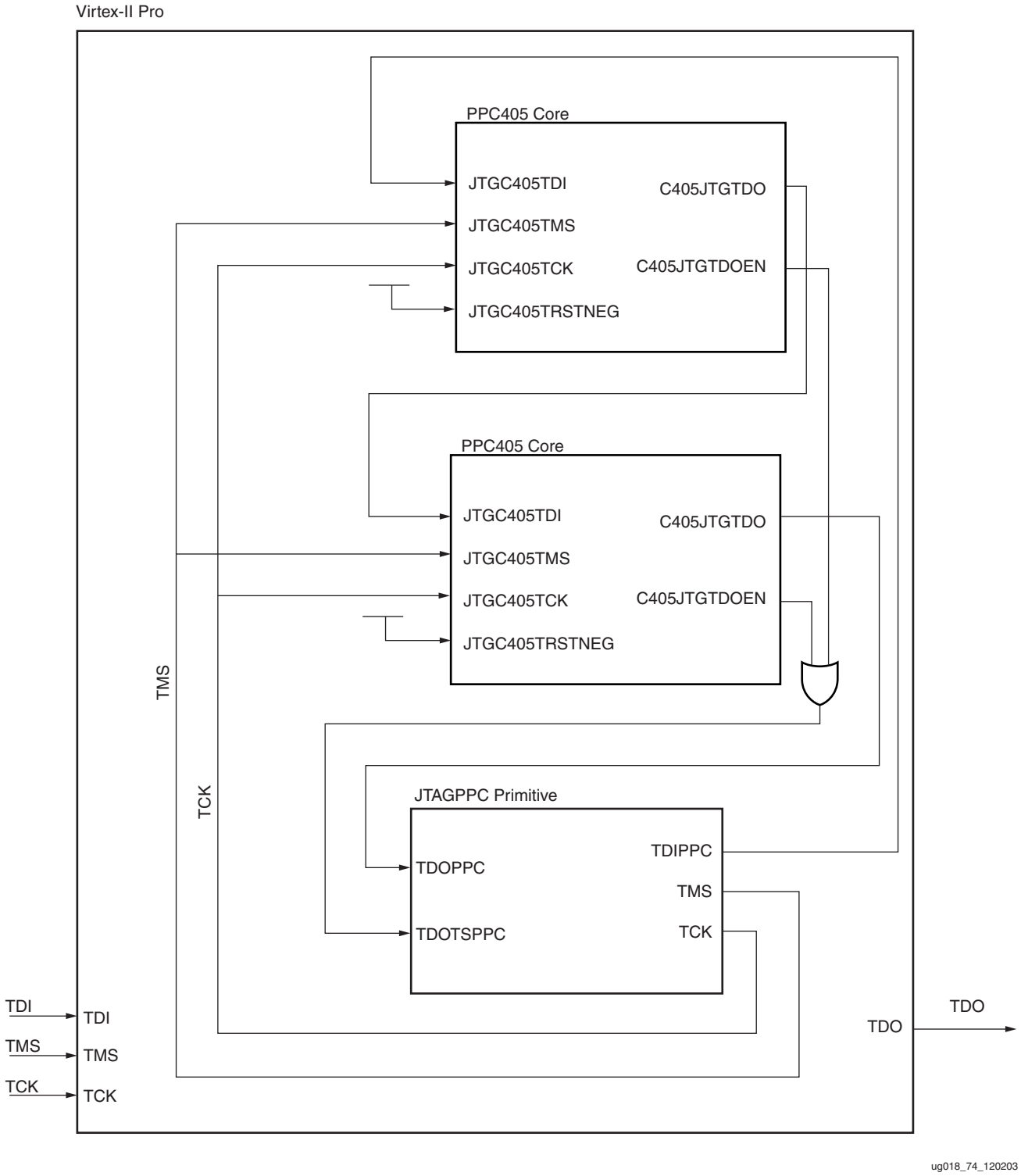


Figure 2-44: **PPC405 Core JTAG Logic Connected in Series with FPGA JTAG Logic Using the JTAGPPC Primitive**

When the PPC405 JTAG logic is connected in series with the dedicated device JTAG logic, only one JTAG chain is required on the printed circuit board. All JTAG logic is accessed through the dedicated JTAG pins 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. The JTAG signals are the *only* 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 dedicated JTAG pins (SINGLE_PPC_JTAG_SERIAL)
- Two PPC Cores: Serial Connection through dedicated 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

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

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 dedicated JTAG logic

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 dedicated JTAG logic

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 dedicated JTAG logic


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 dedicated JTAG logic


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 PowerPC 405 processor 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 2-45 shows the block symbol for the debug interface. The signals are summarized in Table 2-26. See Appendix A, "RISCWatch and RISCTrace Interfaces" for information on attaching a RISCWatch to the debug interface signals.

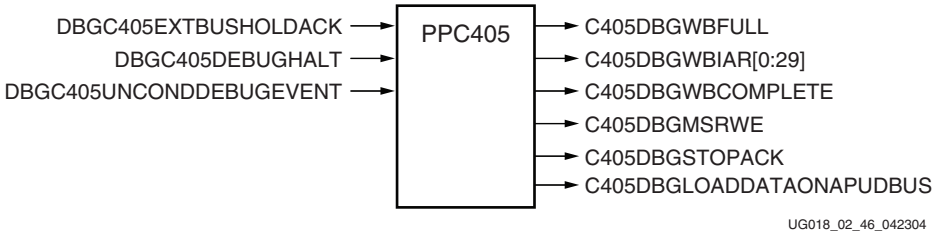


Figure 2-45: **Debug Interface Block Symbol**

Table 2-26: **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 PowerPC 405 writeback pipeline stage is full.
C405DBGWBIAR[0:29]	O	No Connect	The address of the current instruction in the PowerPC 405 writeback pipeline stage.
C405DBGWBCOMPLETE	O	No Connect	Indicates the current instruction in the PowerPC 405 writeback pipeline stage is completing.
C405DBGMSRWE	O	No Connect	Indicates the value of MSR[WE].
C405DBGSTOPACK	O	No Connect	Indicates the PowerPC 405 is in debug halt mode.
C405DBGLOADDATAONAPUDBUS	O	No Connect	Virtex-4 FX only. Valid load data transferred between the APU controller and PowerPC 405 core.

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 PowerPC 405 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 PowerPC 405 processor.

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 PowerPC 405 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 PowerPC 405 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 PowerPC 405 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 PowerPC 405 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 PowerPC 405 processor is in debug halt mode. When deasserted, the processor is not in debug halt mode.

C405DBGLOADDATAONAPUDBUS (Output, Virtex-4 FX only)

This signal is asserted when there is a valid load data being transferred between the APU controller logic and the PowerPC 405 core.

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 2-46 shows the block symbol for the trace interface. The signals are summarized in Table 2-27. See Appendix A, "RISCWatch and RISCTrace Interfaces" for information on attaching a RISCTrace to the trace interface signals.

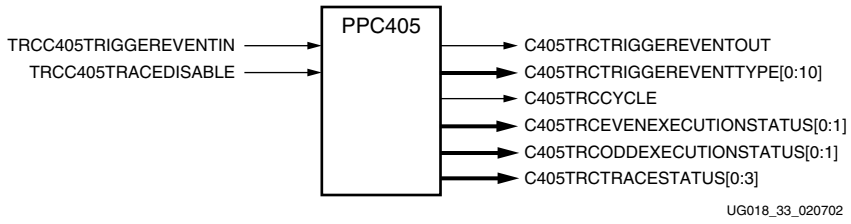


Figure 2-46: **Trace Interface Block Symbol**

Table 2-27: **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 2-28 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 2-28: **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)
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 PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 processor 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.

Processor Version Register (PVR) Interface (Virtex-4 FX Only)

The Virtex-4 PowerPC block provides user access to eight bits in the Processor Version Register (PVR) in the processor. One possible use for these tie signals is to identify different processors in a multi processor system or to encode some processor environment description allowing generic code to adapt its execution on that basis.

PVR Interface I/O Signal Summary

The PVR provides software access to a five field 32-bit value. The fields are: Owner Identifier, Processor Core Family, Cache Array size, Processor core version, and FPGA identifier. The least significant nibbles of the Owner and FPGA identifier are available on the PowerPC interface as tie-offs.

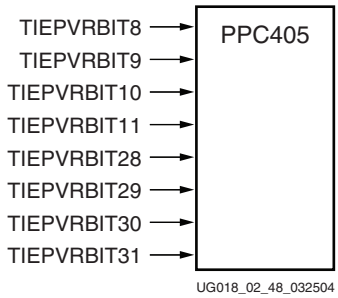


Figure 2-47: **PVR Interface Block Symbol**

Table 2-29: **PVR Interface I/O Signals**

Signal	I/O Type	If Unused	Function
TIEPVRBIT8	I	No Connect	Set bit 8 in Processor Version Register (OWN field)
TIEPVRBIT9	I	No Connect	Set bit 9 in Processor Version Register (OWN field)
TIEPVRBIT10	I	No Connect	Set bit 10 in Processor Version Register (OWN field)
TIEPVRBIT11	I	No Connect	Set bit 11 in Processor Version Register (OWN field)
TIEPVRBIT28	I	No Connect	Set bit 28 in Processor Version Register (AID field)
TIEPVRBIT29	I	No Connect	Set bit 29 in Processor Version Register (AID field)
TIEPVRBIT30	I	No Connect	Set bit 30 in Processor Version Register (AID field)
TIEPVRBIT31	I	No Connect	Set bit 31 in Processor Version Register (AID field)

PVR Interface I/O Signal Descriptions

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

TIEPVRBIT8 (Input)

When tied high sets Processor Version Register bit 8 to 1.

TIEPVRBIT9 (Input)

When tied high sets Processor Version Register bit 9 to 1.

TIEPVRBIT10 (Input)

When tied high sets Processor Version Register bit 10 to 1.

TIEPVRBIT11 (Input)

When tied high sets Processor Version Register bit 11 to 1.

TIEPVRBIT28 (Input)

When tied high sets Processor Version Register bit 28 to 1.

TIEPVRBIT29 (Input)

When tied high sets Processor Version Register bit 29 to 1.

TIEPVRBIT30 (Input)

When tied high sets Processor Version Register bit 30 to 1.

TIEPVRBIT31 (Input)

When tied high sets Processor Version Register bit 31 to 1.

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 2-30.

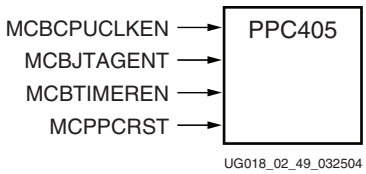


Figure 2-48: **FPGA Specific Interface Block Symbol**

Table 2-30: **Additional FPGA I/O Signals**

Signal	I/O Type	If Unused	Function
MCBCPUCLKEN	I	1	Indicates the PowerPC 405 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.

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.

XILINX®

Chapter 3

PowerPC 405 OCM Controller

Introduction

The On-Chip Memory (OCM) controller serves as a dedicated interface between the FPGA block RAMs and the OCM signals contained within the embedded PowerPC 405 core. The OCM controller provides non-cacheable access to instruction-side and data-side memory spaces.

The data-side interface supports a 32-bit, bi-directional memory interface, and the instruction-side interface supports a 64-bit unidirectional memory interface. Unlike the Processor Local Bus (PLB) interface, the OCM controller does not require bus arbitration to access the FPGA fabric resources. Each OCM controller is capable of addressing up to 16 MB of memory, however, the amount of block RAM in the device may limit the maximum size of OCM supported. Typical applications of data-side OCM (DSOCM) for the Virtex-II Pro and Virtex-4 product families can utilize the block RAMs dual-port feature of to enable both read and write data transfer between processor and FPGA. One possible application for instruction-side OCM (ISOCM) is the storage of interrupt service routines. In addition, its non-cacheable feature eliminates cache pollution and thrashing.

In the Virtex-II Pro family, the DSOCM and ISOCM controllers are designed to interface specifically to block RAMs with fixed latencies.

In the Virtex-4 family, the DSOCM controller has an enhanced feature to support memory-mapped peripherals via additional control signals. This extended feature enables the DSOCM controller to interface to multiple block RAMs with different latencies, as well as to slave peripherals with variable latencies. In addition, the Virtex-4 ISOCM controller has an improved interface for software debugging.

The enhanced features that exist only within the Virtex-4 family will be clearly labeled "Virtex-4 Only." Otherwise, the description applies to both Virtex-II Pro and Virtex-4 families.

The following topics are covered in this chapter:

Comparison of Virtex-II Pro and Virtex-4 OCM Controllers

The Virtex-4 OCM controller is completely backward compatible with the Virtex-II Pro OCM controller. Table 3-1 highlights the new features available only on the Virtex-4 OCM controller. Detailed discussion of these features will be provided later in this chapter.

Table 3-1: **Features Introduced in the Virtex-4 OCM Controller**

Feature	Primary Advantage	ISOCM	DSOCM
Variable latency for read and write access to DSOCM	Wide range of new applications utilizing memory-mapped I/O	N/A	Yes
DCR-based read access to ISOCM.	Support software debugging for ISOCM.	Yes	N/A
Auto clock ratio detection and enhanced clocking support.	Eliminate the need to load wait state register using software. Up to 8:1 clock ratio supported.	Yes	Yes

Functional Features

Common Features for DSOCM and ISOCM

- Separate instruction and data memory interface between the processor block and the block RAMs in the FPGA. Eliminates processor local bus (PLB) arbitration between instruction- and data-side interfaces to external memory.
- Dedicated interface to the Device Control Register (DCR) bus for the ISOCM and DSOCM controllers. Dedicated DCR bus loop inside the processor block for the OCM controllers.
- FPGA-configurable DCR register addresses within the DSOCM and ISOCM controllers.
- Independent 16 MB logical memory space available within PowerPC 405 memory map for each of the DSOCM and ISOCM controllers.
- Multi-cycle mode option for instruction-side and data-side interfaces. Multi-cycle operation uses an *N:1* processor-to-block RAM clock ratio.
  - For Virtex-II Pro designs, N is an integer from 1 through 4.
  - For Virtex-4 designs, N is an integer from 1 through 8.
- *Virtex-4 devices only:* Optional auto clock ratio detection to eliminate the need for programming the control registers of the CPU-to-block RAM clock ratio. This feature simplifies the programming model to use DSOCM and ISOCM.

Features for 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.

- *Virtex-4 devices only:* Optional support for variable latency for read or write data transfer.

Features for Instruction-Side OCM (ISOCM)

The ISOCM interface contains a 64-bit read only port for instruction fetches and a 32-bit read and write port to initialize or test the ISBRAM.

- 64-bit Data Read Only bus (two block RAM clock cycles)
- For Virtex-II Pro devices, 32-bit Data Write Only bus through DCR instruction. For Virtex-4 devices, 32-bit Data Read and Write bus through DCR instruction.
- 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 bus.
  - Initialize ISBRAM during FPGA configuration.

Table 3-2 summarizes the features of the DSOCM and ISOCM controllers. Only the Virtex-4 features are identified with a separate entry in the table.

Table 3-2: **DSOCM and ISOCM Features**

Feature	Data-Side OCM Interface	Instruction-Side OCM Interface
Non-cacheable memory space.	16 MB	16 MB
Data bus width (load/store/fetch).	32-bit bi-directional (load/store)	64-bit unidirectional (Instruction fetch)
Data bus width (DCR read/write) for instruction side memory interface and software debugger.	Not applicable	32-bit(1)
Byte write support.	Yes	Not applicable
Maximum performance.	One load/store for every two BRAMDSOCMCLK cycles	Two instruction fetches for every two BRAMISOCMCLK cycles
Address bus.	22 bits	21 bits
DCR control registers.	DSARC and DSCNTL	ISARC, ISCNTL, ISINIT, and ISFILL
OCM DCR control register base address selection.	Virtex-II Pro: TIEDSOCMDCRADDR Virtex-4: TIEDCRADDR+offset(1)	Virtex-II Pro: TIEISOCMDCRADDR Virtex-4: TIEDCRADDR+offset(2)
Default settings applied at power up through dedicated processor inputs (see "DSOCM Ports" and "ISOCM Ports").	DSARCVALUE and DSCNTLVALUE	ISARCVALUE and ISCNTLVALUE
OCM Clock.	BRAMDSOCMCLK	BRAMISOCMCLK

Table 3-2: **DSOCM and ISOCM Features *(Cont'd)***

Feature	Data-Side OCM Interface	Instruction-Side OCM Interface
Clock Ratio (PowerPC 405:OCM) Virtex-II Pro devices Virtex-4 devices	 Integer: 1:1 through 4:1 1:1 through 8:1	 Integer: 1:1 through 4:1 1:1 through 8:1
Clock ratio automatic detection.	Virtex-4 only	Virtex-4 only
Variable Latency Read/Write	Virtex-4 only	Not applicable
Initialize block RAM during FPGA device configuration.	Yes	Yes
Processor access to initialize memory in fabric.	Load and store instructions	DCR read and write instructions

Notes:

1. 32-bit write only port for Virtex-II Pro designs. 32-bit read/write port for Virtex-4 designs.
2. Refer to the section "Device-Control Register Interfaces" in Chapter 2 for more information.

OCM Controller Operation

The OCM controller is distributed into two blocks, one for the ISOCM interface and the other for the DSOCM interface, as shown in Figure 3-1.

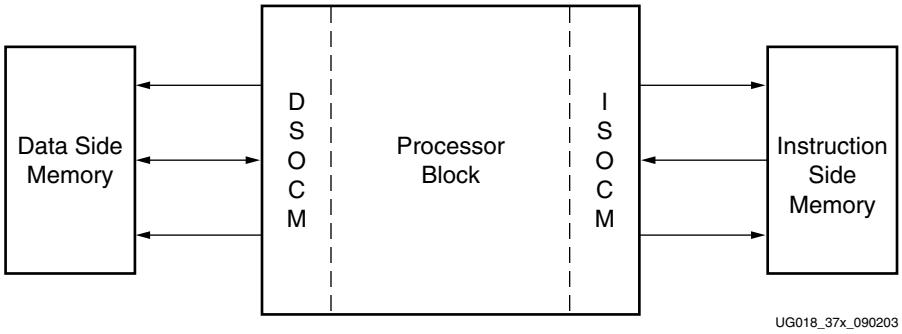


Figure 3-1: **OCM Controller Interfaces**

The DSOCM and ISOCM interfaces are designed to operate independently of each other. This provides the following advantages:

- The overall efficiency of the core is improved by eliminating the need for OCM arbitration between two sets of operations, that is, loads and stores on the data-side interface and instruction fetches on the instruction-side interface.
- Overall controller performance is improved because there is no need to share a common address and data bus between the instruction-side and data-side interfaces to the block RAM.
- Having two separate interfaces allows selection of either one or both interfaces as required by the specific application.
- The two control registers: DSARC and ISARC, define the base addresses for the OCM instruction-side and data-side memory spaces. The registers are initialized on power

up with the value on the input ports: DSARCVALUE[0:7] and ISARCVALUE[0:7] respectively. The two registers can also be loaded using DCR write assembly instructions (mtdcr).

The value of DSARC and ISARC defines the most significant eight address bits for the two 16 MB memory spaces (instruction and data) available on the OCM, assuming OCM address decoding is enabled in bit 0 of the ISCNTL/DSCNTL registers.

Notice that the instruction-side and data-side OCM interfaces can reside in the same 16 MB space or dedicate two 16 MB spaces, i.e., DSARCVALUE[0:7] and ISARCVALUE[0:7] can be the same value, or they can be different values. However, once the 16 MB space(s) is defined for instruction-side and data-side OCMs, PLB/OPB memory spaces cannot overlap with the OCM space(s). For more details, refer to the "Programmer's Model" section later in this chapter.

OCM DCR-Based Control Registers (Accessed Via DCR Instructions)

There are two registers (DSARC and DSCNTL) in the DSOCM and four registers (ISARC, ISCNTL, ISINIT and ISFILL) in the ISOCM.

The DSARC/ISARC, DSCNTL/ISCNTL control registers, must be initialized *before* using DSOCM/ISOCM interfaces, which also means load and store data via DSOCM and fetching instructions to the instruction side interface. There are two ways to initialize these registers:

1. Use DCR assembly instructions (mtdcr, mfdcr) to access all six OCM control registers. The DCR address for these registers are summarized under the heading "Device-Control Register Interfaces" in Chapter 2.
2. Specify the associated input ports of the processor block. The values that tie to the 8-bit input ports DSARCVALUE[0:7], DSCNTLVALUE[0:7] will be the initial value of DSARC and DSCNTL registers after power on. Similarly, the values that tie to the 8bit input ports ISARCVALUE[0:7], ISCNTLVALUE[0:7] will be the initial value of ISARC and ISCNTL registers after power on. Notice that if the processor system will be boot from the ISOCM memory, the ISARC and ISCNTL registers must be initialized using this method.

The ISINIT and ISFILL registers are used for content initialization of the instruction side of OCM memory and for software debugging purposes.

- In Virtex-II Pro designs: allows the processor to *write* instructions into the ISOCM memory array during system initialization, using the ISINIT and the ISFILL registers.
- In Virtex-4 designs: allows the processor to *write* instructions and *read* instructions from the ISOCM memory array using the ISINIT and the ISFILL registers.

More information regarding the functionality of these OCM control registers will be described in the "Programmer's Model" section of this chapter.

DSOCM Controller Load/Store Operation

The DSOCM controller accepts an address and associated control signals from the processor during a load instruction, and passes a valid address to the DSOCM's FPGA fabric or block RAM interface. For store instructions, a valid address from the processor is accompanied by store data and by the associated control signals. The DSOCM controller performs an address decode on the eight most significant processor address bits to determine if the load/store instruction is for the data-side OCM interface. The DSARC

register defines the 16 MB memory region that is valid for the DSOCM. Load instructions have a priority over store instructions at the DSOCM interface

Non-Memory Peripherals for DSOCM

The OCM interface is designed to connect to memory. To correctly implement non-memory peripherals that attach to DSOCM, designers must be aware of two OCM specific behaviors: execution re-ordering and store-data bypass.

Execution Re-ordering

Under certain conditions, the OCM controller will change the order in which DSOCM Load and Store instructions are executed. A Store access may be executed after a Load, even though the Store is fetched before the Load by the processor. If maintained execution order is necessary in the peripheral, the designer is responsible for enforcement. This can be done in driver routines by issuing a dummy Store between the operations. A hardware solution is to add a semaphore that flags the completion of the Store operation.

Store-data Bypass

A Store followed immediately by a Load from the same address may be handled as an internal operand forward in the OCM controller. This means that the data returned to the processor as the result of the access isn't taken from the data returned by the peripheral, but rather from an internal OCM buffer. To ensure that the Load data is read from the peripheral, the same techniques can be used as for execution reordering. Execution re-ordering of accesses to the same address will only occur in combination with store-data bypass, thus ensuring memory consistency.

ISOCM Controller Instruction Fetch Operation

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 ISOCM interface. Instructions stored in a block RAM can be loaded into it during FPGA device configuration. Alternatively, the processor can load the ISOCM space using the ISINIT and ISFILL registers on the DCR bus.

There are two datapaths from the processor block to access the instruction-side memory:

- The main 64-bit, read only port for instruction fetch. Since this port is 64-bits wide, two instructions will be fetched at once.
- The secondary 32-bit port for memory initialization and software debug. For Virtex-II Pro designs, this port is write only, so it has limited software debug capability. For Virtex-4 designs, this port supports both reads and writes and therefore has improved software debug capabilities.

DSOCM Ports

Figure 3-2 and Figure 3-3 are the block diagrams of the Virtex-4 and Virtex-II Pro DSOCM. All signals are in big endian format.

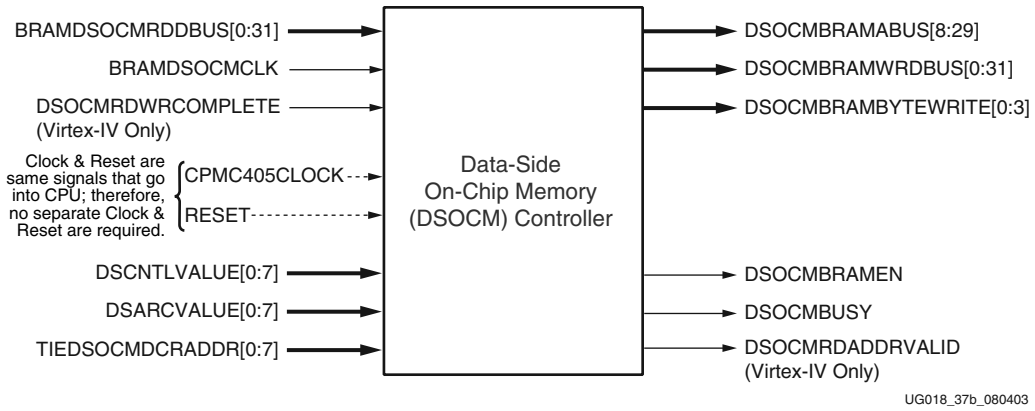


Figure 3-2: **Virtex-4 DSOCM Interface**

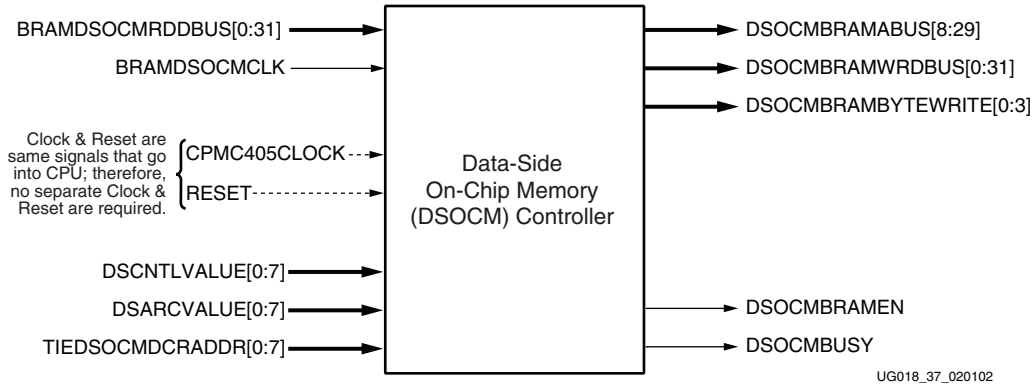


Figure 3-3: **Virtex-II Pro DSOCM Interface**

DSOCM Input Ports

Table 3-3 describes the Data Side OCM (DSOCM) input ports.

Table 3-3: **DSOCM Input Ports**

Port	Direction	Description
BRAMDSOCMCLK	Input	This signal clocks the DSOCM controller and the data side interface logic (Virtex-4 only) or memory located in the FPGA fabric. When in multi-cycle mode, the processor clock is in an *N*:1 ratio with BRAMDSOCMCLK. The frequency of BRAMDSOCMCLK must be an integer multiple of the processor block clock input, CPMC405CLOCK (CPU Clock). The rising edge of BRAMDSOCMCLK must align with the rising edge of CPMC405CLOCK. • For Virtex-4, *N* is an integer from 1 to 8. • For Virtex-II Pro, *N* is an integer from 1 to 4. ***Note:*** To generate clocks with integer ratios, a Digital Clock Manager (DCM) feature in the Virtex-II Pro and Virtex-4 fabric can be included in the application system.
BRAMDSOCMRDDBUS[0:31]	Input	32-bit read data bus from the FPGA fabric to the DSOCM controller. For Virtex-II Pro applications, this bus originates from the read data port of the block RAM. For Virtex-4 applications, the bus can originate from the block RAM and/or other memory-mapped peripherals located in the fabric.
DSOCMRWCOMPLETE (Virtex-4 only)	Input	Virtex-4 devices support variable latencies for the module interface with the DSOCM controller. Virtex-4 designs differ from Virtex-II Pro designs in that a Virtex-4 load or store operation can take an integer *multiple number* of block RAM clock cycles. DSOCMRWCOMPLETE indicates that a read access or a write access is complete. The signal should be asserted *for one and only one* BRAMDSOCMCLK cycle. For read accesses, the DSOCMRWCOMPLETE signal should be accompanied by read data in the same clock cycle. For both read and write operations, this signal informs the DSOCM controller in the processor block that the current bus transaction is complete. The DSOCM can issue the next read or write access, if required. Unlike the CoreConnect bus architecture (PLB, OPB and DCR) there are no complex bus protocols to handle a bus error, an abortion, or bus timeout scenarios in this DSOCM interface. Users need to design bus timeout logic to guarantee a fabric response to a valid DSOCM bus cycle. If this signal is not asserted, the processor will operate unpredictably. ***Note:*** If you do not wish to use the variable latency feature of the Virtex-4 DSOCM and are migrating a Virtex-II Pro block RAM design, or the module that interfaces with DSOCM controller has a fixed latency of one, this signal should be tied to logic “1”.

DSOCM Input Ports: Attributes

Attributes are inputs to the OCM controller from the FPGA fabric that *must be connected* to initialize registers at FPGA power up, or following a processor reset. These inputs are used to:

- Define the DSOCM control register DCR addresses in the DCR memory space.
- Define the 16MB memory locations for the DSOCM controller.
- Enable the DSOCM address decoder.
- Define the operating characteristics for the bus interface circuitry.

Table 3-4 describes the DSOCM attributes.

Table 3-4: **DSOCM Attributes**

Attribute	Direction	Description
DSCNTLVALUE[0:7]	Input	This input bus is loaded into the DSCNTL register at FPGA power-up. The value is used to define the basic operational characteristics of the DSOCM controller. Application software can modify the default value by writing to the DSCNTL register. See Figure 3-11, page 164, and Figure 3-12, page 165, for register bit definitions.
DSARCVALUE[0:7]	Input	This input bus is loaded into the DSARC register at FPGA power up. It defines the 16 MB memory space location for the data-side memory interface. See Figure 3-11, page 164, and Figure 3-12, page 165, for register bit definitions.
TIEDSOCMDCRADDR[0:7] (Virtex-II Pro only)(1)	Input	This input bus defines the eight most significant bits of the ten-bit DCR address space for the DSOCM DCR control and status registers. The two least significant bits are predefined within the DSOCM controller. For example, if `TIEDSOCMDCRADDR = 00_0001_11` then: • DCR address of DSARC = `00_0001_1110` = `0x01E` • DCR address of DSCNTL = `00_0001_1111` = `0x01F`
TIEDCRADDR[0:5] (Virtex-4 only)(1)	Input	This input bus defines the six most significant bits of the ten-bit DCR address space for the DCR Control and Status registers associated with the OCM, APU(2), AND Ethernet MAC(3) submodules. For example, if `TIEDCRADDR = 00_0001` then: • DCR address of DSARC = `00_0001_0110 = 0x016` • DCR address of DSCNTL = `00_0001_0111 = 0x017`

Notes:

1. For more information, refer to the "Device-Control Register Interfaces" section in Chapter 2.
2. For more information, refer to Chapter 4, "PowerPC 405 APU Controller."
3. For more information, refer to the *Virtex-4 Ethernet Media Access Controller User Guide*.

DSOCM Output Ports

Table 3-5 describes the data-side OCM (DSOCM) output ports.

Table 3-5: **DSOCM Output Ports**

Port	Direction	Description
DSOCMBRAMEN	Output	This is the block RAM enable signal that is asserted for both reads and writes to the data-side memory interface. In the Virtex-4 FX devices, this signal is asserted for one and only one BRAMDSOCMCLK cycle. DSOCMBRAMABUS[8:29] contains the address and DSOCMBRAMWRDBUS[0:31] contains the data (for write).
DSOCMBRAMABUS[8:29]	Output	Read or write address from the DSOCM controller to the data-side FPGA fabric or memory interface. These 22 address bits correspond to internal PowerPC 405 address bits [8:29]. PowerPC 405 address bits [0:7] are compared against the DSARC register contents, and if a match is decoded, further steps for load/store operation are initiated. For write accesses in both Virtex-II Pro and Virtex-4 devices, the write address is accompanied and qualified by a write enable signal for each byte lane of data. For read accesses, when the DSOCM controller is connected only to the block RAM, DSOCMBRAMEN is asserted and must be used as a valid address qualifier. When the DSOCM controller is connected to a memory-mapped slave peripheral with variable latency (Virtex-4 extended feature), DSOCMBRAMABUS[8:29] will be qualified by the new DSOCMRDADDRVALID signal to indicate a valid read access.
DSOCMBRAMWRDBUS[0:31]	Output	This bus provides 32-bit write data from the DSOCM to the data-side memory interface. If the block RAM is connected to the interface, this port is connected directly to the data input port of the memory. For Virtex-4 applications, this is the write data input to the memory-mapped slave peripheral. The write data bus is further qualified with DSOCMBRAMBYTEWRITE, and will be asserted for one and only one BRAMDSOCMCLK cycle.
DSOCMBRAMBYTEWRITE[0:3]	Output	This signal indicates a write access and qualifies the DSOCMBRAMWRDBUS. Four write enable signals support independent byte-wide data writes into the data-side memory or peripheral. DSOCMBRAMBYTEWRITE[0] qualifies writes to DSOCMBRAMWRDBUS[0:7], DSOCMBRAMBYTEWRITE[1] qualifies writes to DSOCMBRAMWRDBUS[8:15], and so on. If the DSOCM controller is connected to memory-mapped slave peripherals with variable latency (Virtex-4 extended feature), DSOCMBRAMBYTEWRITE must be used as the qualification signal for the write data bus. The signal will be asserted for one and only one BRAMDSOCMCLK cycle. A memory-mapped slave design should register this signal, as well as the write address and write data (DSOCMBRAMABUS[8:29], DSOCMBRAMWRDBUS[0:31]), if the write operation cannot be completed in a single BRAMDSOCMCLK cycle.

Table 3-5: **DSOCM Output Ports *(Cont'd)***

Port	Direction	Description
DSOCMRDADDRVALID (Virtex-4 only)	Output	This signal is used when the DSOCM controller is connected to the logic in the FPGA fabric (e.g. memory-mapped peripheral) with a variable latency. The signal indicates a read access and indicates the read address is valid on the DSOCMBRAMABUS[8:29]. This signal will be asserted for one BRAMDSOCMCLK cycle only. A memory-mapped slave design should register this signal, as well as the read address (DSOCMBRAMABUS[8:29]), if the read operation cannot be completed in the next cycle.
DSOCMWRADDRVALID (Virtex-4 only)	Output	This signal is used when the DSOCM controller is connected to the logic in the FPGA fabric (e.g., memory-mapped peripheral) with a variable latency. The signal indicates a write access and indicates the write address is valid on the DSOCMBRAMABUS[8:29]. This signal is asserted for one BRAMDSOCMCLK cycle only. A memory-mapped slave design should register this signal, as well as the read address (DSOCMBRAMABUS[8:29]) if the read operation cannot be completed in the next cycle.
DSOCMBUSY	Output	This control signal reflects the value of the DSOCM DCR control register DSCNTL[2] bit output to the FPGA fabric. This signal can be used for applications that require a software control mechanism to toggle a control bit to FPGA hardware. It is an optional signal and need not be used.

DSOCM-to-BRAM Interfaces

Figure 3-4 provides an example of a basic Virtex-II Pro DSOCM-to-BRAM interface. Virtex-II Pro devices support only fixed latency connections such as the one shown.

Figure 3-5 shows an example of a basic Virtex-4 DSOCM-to-BRAM interface. Notice that in fixed latency mode, the output DSOCMRDADDRVALID and DSOCMWRADDRVALID can be left unconnected.

Note: Individual byte enables in a Virtex-II Pro device require a minimum of four block RAMs for DSOCM (each BRAM port has a single write enable which is used as byte enable). In a Virtex-4 device, a single block RAM is sufficient, since it can be configured to have individual (that is, four) byte enables in its 32-bit data configuration.

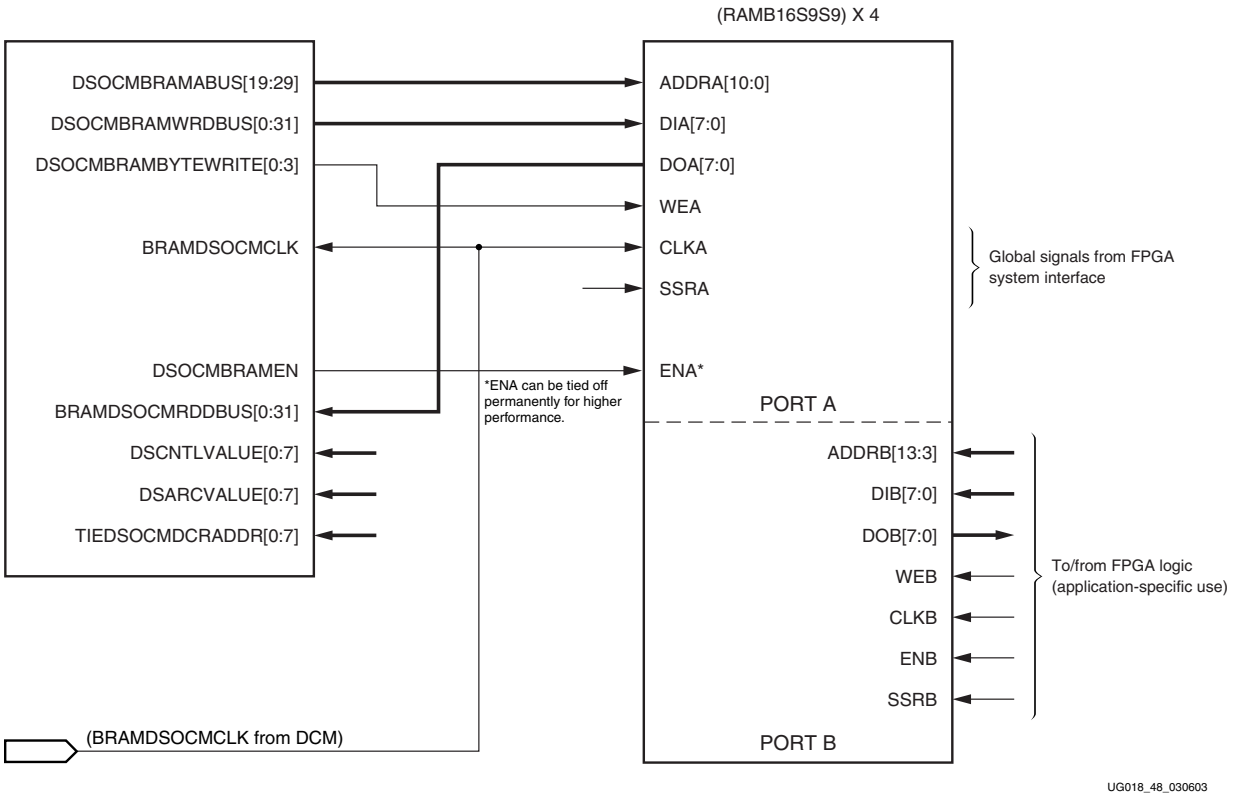
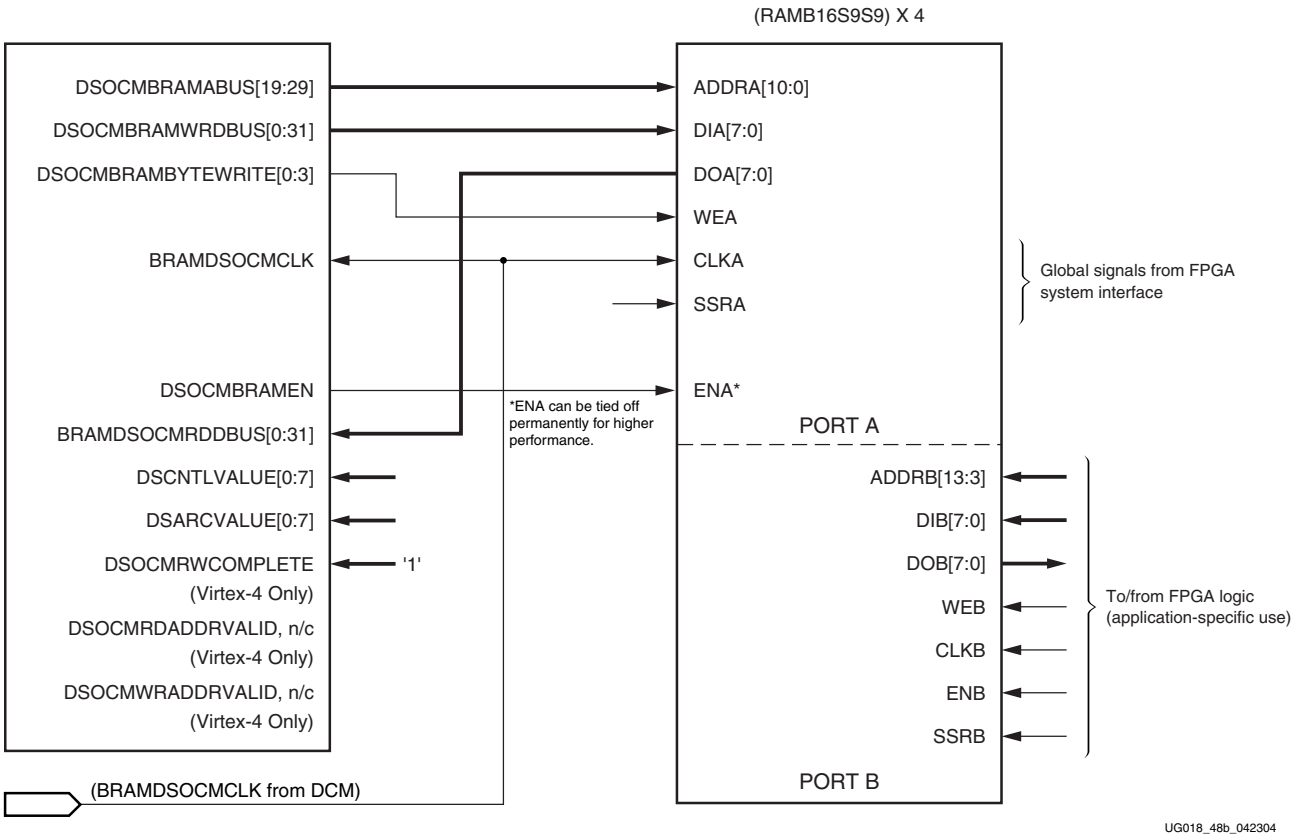


Figure 3-4: **Virtex-II Pro DSOCM to Block RAM Interface: 8-KByte Example**



Note: n/c = no connect

Figure 3-5: **Virtex-4 DSOCM to Block RAM Interface: 8-KByte Example**

Note: For backward compatibility with Virtex-II Pro designs, when connecting DSOCM to block RAM (as shown in Figure 3-5), set DSOCMRWCOMPLETE to logic 1 and leave the DSOCMRDADDRVALID and DSOCMWRADDRVALID signals unconnected.

Figure 3-6 shows the extended feature in Virtex-4 devices for DSOCM-to-Memory-Mapped-Slave-Peripheral interface.

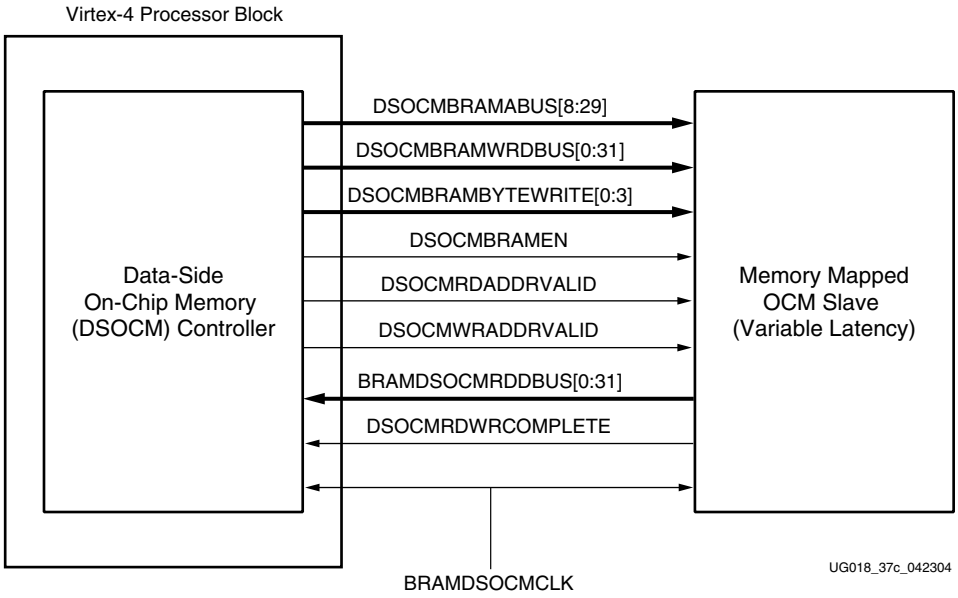


Figure 3-6: **DSOCM to Memory-Mapped Slave Peripheral (Virtex-4 Extended Feature)**

ISOCM Ports

Figure 3-7 and Figure 3-8 are block diagrams of the Virtex-II Pro and Virtex-4 ISOCM. All signals are in big endian format.

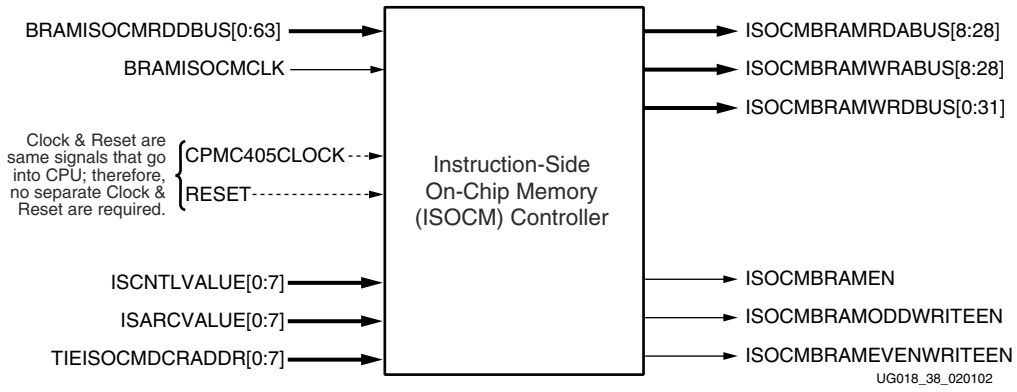


Figure 3-7: **Virtex-II Pro ISOCM Interface**

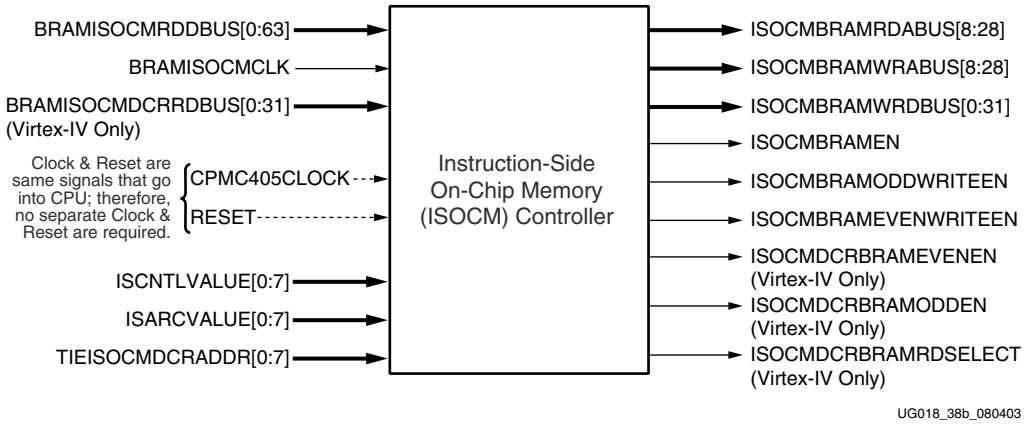


Figure 3-8: **Virtex-4 ISOCM Interface**

ISOCM Input Ports

Table 3-6 describes the Instruction Side OCM (ISOCM) input ports.

Table 3-6: **ISOCM Input Ports**

Port	Direction	Description
BRAMISOCMCLK	Input	This signal clocks the ISOCM controller and the instruction side memory located in the FPGA fabric. When in multi-cycle mode, BRAMISOCMCLK is in a 1:*N* ratio to the processor clock. The Digital Clock Manager (DCM) should be used to generate the processor clock and the ISOCM clock. BRAMISOCMCLK must be an integer multiple of the processor block clock CPMC405CLOCK. • For Virtex-4 devices, *N* is an integer from 1 to 8. • For Virtex-II Pro devices, *N* is an integer from 1 to 4.
BRAMISOCMRDDBUS[0:63]	Input	64-bit read data from the block RAM to the ISOCM controller. The read data bus is the path for instruction fetch of CPU operations.
BRAMISOCMDCRRDDBUS[0:31] (Virtex-4 only)	Input	***Note:*** Optional. Used in dual-port block RAM interface designs only. 32-bit read data from the block RAM to the ISOCM controller using a DCR-based access from the PowerPC 405. This read data bus enables the software debugger to access the software program instructions in the ISOCM memory. In order to insert software breakpoints into the instruction side memory, the debugger must be able to both read and write the code stored in the block RAM.

ISOCM Input Ports, Attributes

Attributes are inputs to the OCM controller, from the FPGA fabric, that must be connected to initialize control registers at FPGA power-up, or following a PowerPC 405 reset. The ISINIT and ISFILL registers cannot be initialized in this manner. These registers are initialized only through "move to DCR" (`mtdcr`) instructions. Application software can also modify the contents of the ISARC and ISCNTL registers using `mtdcr` and `mfdcr` instructions.

Table 3-7 describes the ISOCM attributes.

Table 3-7: **ISOCM Attributes**

Attribute	Direction	Description
ISCNTLVALUE[0:7]	Input	This input bus is loaded into the ISCNTL register at FPGA power-up. The value is used to configure the operational characteristics of the ISOCM controller. See Figure 3-13, page 166, and Figure 3-14, page 167, for register bit definitions.
ISARCVALUE[0:7]	Input	This input bus is loaded into the ISARC register at FPGA power up. It defines the 16 MB memory space location for the instruction-side memory interface. See Figure 3-13, page 166, and Figure 3-14, page 167, for register bit definitions.
TIEISOCMDCRADDR[0:7] Virtex-II Pro Only	Input	This input bus defines the eight most significant bits of the ten-bit DCR address bus for the ISOCM DCR control registers. The two least significant bits are predefined in the ISOCM controller. For example, if TIEISOCMDCRADDR[0:7] = `00_0010_11`, then: • The DCR address of ISINIT register = `00_0010_1100 = 0x02C` • The DCR address of ISFILL = `00_0010_1101 = 0x02D` • The DCR address of ISARC = `00_0010_1110 = 0x02E` • The DCR address of ISCNTL = `00_0010_1111 = 0x02F`
TIEDCRADDR[0:5] Virtex-4 Only	Input	This input bus defines the six most significant bits of the 10-bit DCR address space for DCR control and status registers(1) for the OCM, APU(2), and Ethernet MAC(3) sub modules. For example, if TIEDCRADDR = `00_0001` then: • The DCR address of the ISINIT register = `00_0001_0000 = 0x010` • The DCR address of the ISFILL register = `00_0001_0001 = 0x011` • The DCR address of the ISARC register = `00_0001_0010 = 0x012` • The DCR address of the ISCNTL register = `00_0001_0011 = 0x013`

Notes:

1. For more information, refer to the "Device-Control Register Interfaces" section in Chapter 2.
2. For more information, refer to Chapter 4, "PowerPC 405 APU Controller."
3. For more information, refer to the *Virtex-4 Ethernet Media Access Controller User Guide*.

ISOCM Output Ports

Table 3-8 describes the instruction-side OCM (ISOCM) output ports.

Table 3-8: **ISOCM Output Ports**

Port	Direction	Description
ISOCMBRAMEN	Output	This is a block RAM read enable from the ISOCM controller. This signal is asserted only for valid ISOCM instruction fetch cycles. For the fastest memory access applications, the block RAM enable input (EN) can be locally tied to a logic 1 level. Block RAM power consumption can be reduced by connecting the block RAM enable input (EN) to the ISOCMBRAMEN signal. If the enable is not tied to a logic 1 level, a timing analysis must be run to verify that the design meets frequency of operation requirements.
ISOCMBRAMRDABUS[8:28]	Output	Read address from ISOCM to block RAM. These 21 outputs correspond to PowerPC 405 address bits [8:28]. The read address bus is the path for instruction fetch operations. These 21 address bits corresponds to internal PowerPC 405 address bits [8:28]. PowerPC 405 address bits [0:7] are compared against the ISARC register contents, and if a match is decoded, further steps for instruction fetch are initiated
ISOCMBRAMWRABUS[8:28]	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. In Virtex-II Pro, this bus provides the write address from the ISOCM to block RAM via a DCR-based access. The bus value is initially set to the value stored in the ISINIT register. In Virtex-4, this bus provides both a read and write address via DCR-based access. The bus value is initially set to the value stored in the ISINIT register.
ISOCMBRAMWRDBUS[0:31]	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. This bus provides 32-bit write data from the ISOCM to block RAM via a DCR-based access. It is connected to both the even and odd banks of ISBRAM. It is initially set to the value stored in the ISFILL register.
ISOCMBRAMODDWRITEEN	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. Write enable to qualify a valid write into a block RAM via a DCR-based access. This signal enables a write into a memory bank that contains odd instruction words, that are read back on BRAMISOCMRDDBUS[32:63]. For Virtex-II Pro, connect this signal to both the Enable (EN) and Write Enable (WE) inputs of a dual-port ISBRAM port for power savings. For Virtex-4, connect ISOCMBRAMODDWRITEEN to the Write Enable (WE) input of a dual-port block RAM port and ISOCMDCRBRAMODDEN to the Enable (EN) input of the dual port ISBRAM. For single-port ISBRAM implementations, this signal can be left unconnected.

Table 3-8: **ISOCM Output Ports *(Cont'd)***

Port	Direction	Description
ISOCMBRAMEVENWRITEEN	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. Write enable to qualify a valid write into a block RAM via a DCR-based access. This signal enables a write into the 32-bit memory that contains even instruction words BRAMISOCMRDDBUS[0:31]. For Virtex-II Pro, connect this signal to both the Enable (EN) and Write (WE) inputs of a dual-port ISBRAM port for power savings. For Virtex-4, connect this signal to Write (WE) inputs of a dual-port ISBRAM port and ISOCMDCRBRAMEVENEN to the Enable (EN) input of the dual-port ISBRAM port. For single-port ISBRAM implementations, this signal can be left unconnected.
ISOCMDCRBRAMODDEN (Virtex-4 only)	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. Block RAM enable (odd bank) to qualify a valid read or write from a block RAM via a DCR-based access, in order to access odd instruction words. For Virtex-4, connect this signal to the Enable (EN) input of the dual-port ISBRAM port.
ISOCMDCRBRAMEVENEN (Virtex-4 only)	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. Block RAM enable (even bank) to qualify a valid read or write from block RAM via a DCR-based access, in order to access even instruction words. For Virtex-4, connect this signal to the Enable (EN) input of the dual-port ISBRAM port.
ISOCMDCRBRAMRDSELECT (Virtex-4 only)	Output	***Note:*** Optional. Used in dual-port block RAM interface designs only. Since the DCR bus can only access 32-bit data and the ISOCM has a 64-bit data bus, this output signal, driven by the ISOCM controller, must be used to select between even and odd instruction words using a multiplexer in the FPGA fabric. At logic 1, it selects the odd instruction word; at logic 0, it selects the even instruction word.

Figure 3-9 shows an example of an ISOCM-to-BRAM interface in Virtex-II Pro.

Figure 3-10 shows an example of an ISOCM-to-BRAM interface in Virtex-4.

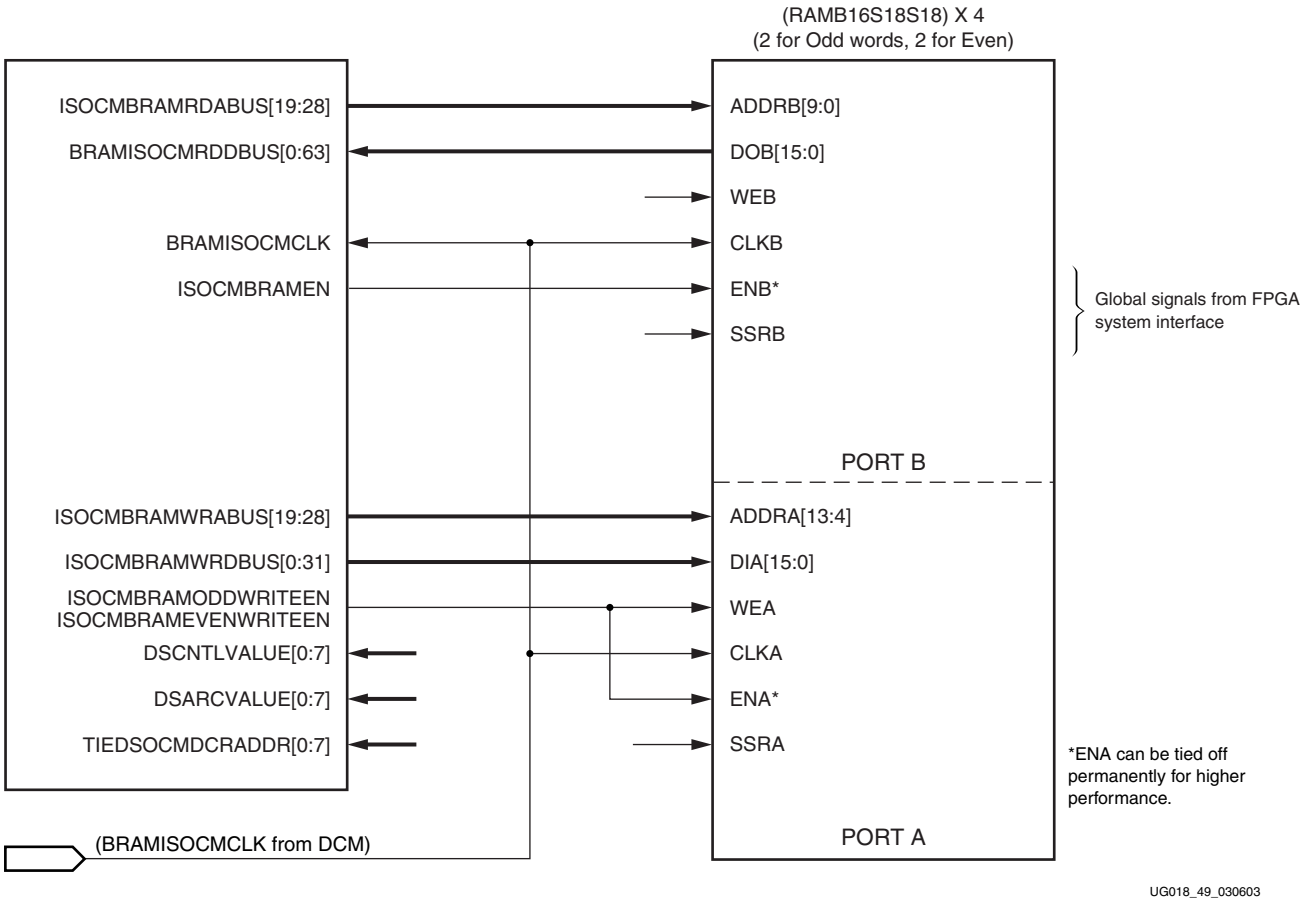


Figure 3-9: **ISOCM to Block RAM Interface: 8 KByte Example in Virtex-II Pro**

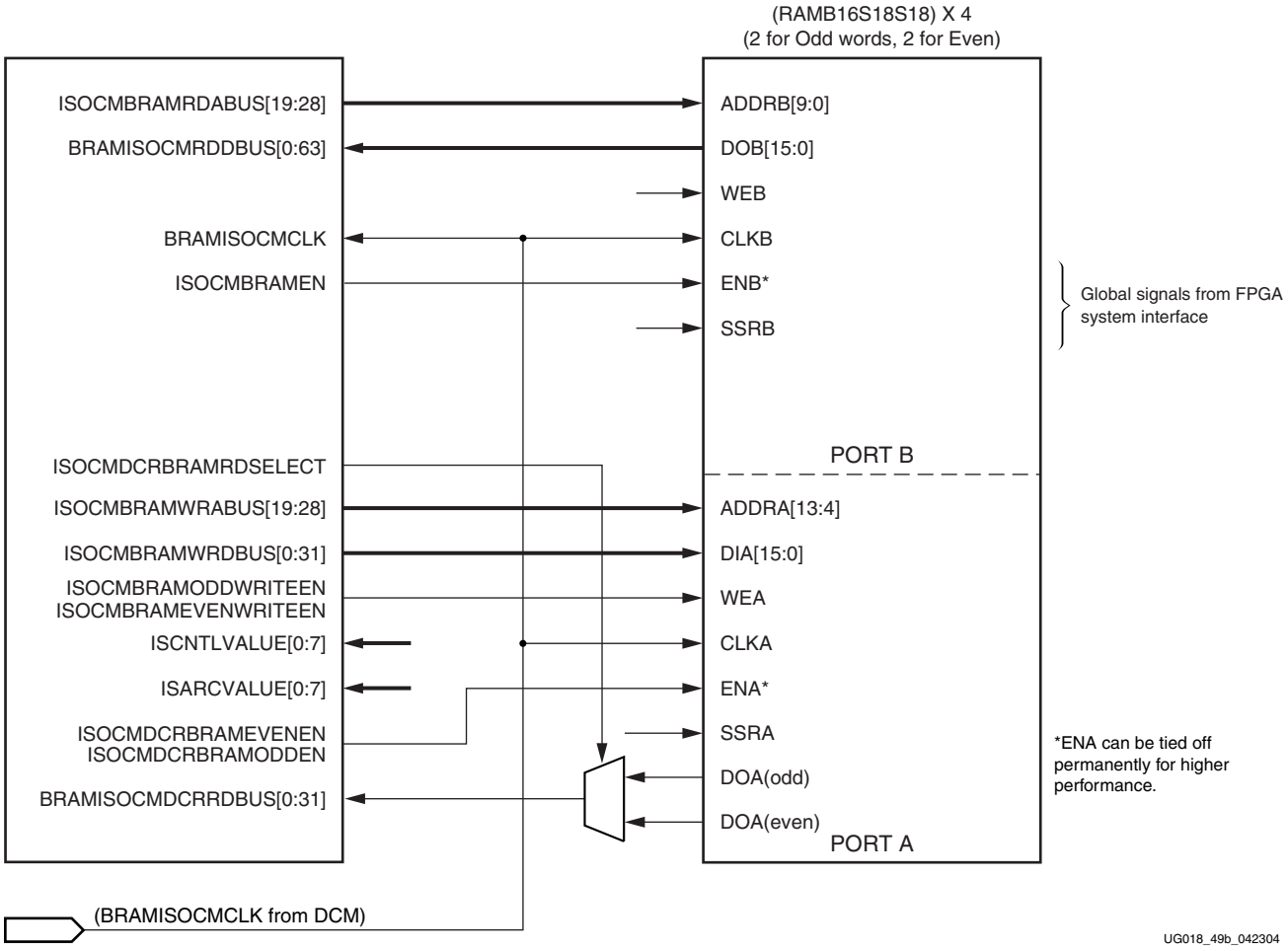


Figure 3-10: **ISOCM to Block RAM Interface: 8 KByte Example in Virtex-4**

Note: See Table 3-8 for descriptions of the signals shown in Table 3-10, above.

Programmer's Model

DCR Registers

Application software has read and write access to the DCR control registers within the OCM controllers. Typically, `mtdcr` and `mfdcr` assembly language instructions are used to write and read respectively from these registers.

Figure 3-11, page 164 and Figure 3-12, page 165 list the DCR control registers and the bit definitions for the DSOCM interface for Virtex-II Pro and Virtex-4. Figure 3-13, page 166 and Figure 3-14, page 167 list the DCR control registers and the bit definitions for the ISOCM interface for Virtex-II Pro and Virtex-4.

DSARC/ ISARC Registers

The ISOCM and DSOCM interfaces provide DCR registers (DSARC & ISARC) which define the eight most significant (base) address bits of the ISOCM and DSOCM memory

locations. These bits are decoded against PowerPC 405 address bits 0:7. These eight most significant address bits permit the OCM controllers to reside independently in any 16 MB, non-cacheable, memory range within the PowerPC 405 32bit address (4 GB) memory space

The ISOCM and DSOCM hardware outputs a maximum of 22 address bits (data-side address bits [8:29] and instruction-side address bits [8:28]) to address memory contained in the FPGA fabric.

DSCNTL Registers

Table 3-9 and Table 3-10 describe the DSCNTL registers in Virtex-II Pro and Virtex-4 devices. For additional information, refer to Figure 3-11, page 164 (Virtex-II Pro) and Figure 3-12, page 165 (Virtex-4).

Table 3-9: **DSCNTL Register for Virtex-II Pro**

Bit 0	DSOCM Enable	If set to 1, address decoding based on the value of DSARC will be enabled. If set to 0, the content in DSARC will be ignored.
Bit 1	DISABLEOPERANDFWD	If set to 1, load data from the DSOCM goes directly into a latch in the processor block. This causes an additional cycle (a total of two cycles) of latency between a load instructions which is followed by an instruction that requires the load data as an operand. If set to 0, load data from the DSOCM/ must pass through steering logic before arriving at a latch. This causes a single cycle of latency between a load instruction which is followed by an instruction that requires the load data as an operand.
Bit 2	DSOCMBUSY	This status bit can be used as a flag indicator to the FPGA fabric. This is an optional signal.
Bit 3	Reserved.	This bit must be configured to 0.
Bit 4	Reserved.	This bit must be configured to 0.
Bit 5:7	DSOCMMCM	CPU Clock and DSOCM Clock ratio. For Virtex-II Pro users, users must setup the ratio in this field with valid clock ratios used in the application system. Then the processor gasket will issue appropriate transaction based on this ratio.

Table 3-10: **DSCNTL Register for Virtex-4**

Bit 0	DSOCM Enable	If set to 1, address decoding based on the value of DSARC will be enabled. If set to 0, the content in DSARC will be ignored.
Bit 1	DISABLEOPERANDFWD	If set to 1, load data from the DSOCM goes directly into a latch in the processor block. This causes an additional cycle (a total of two cycles) of latency between a load instructions which is followed by an instruction that requires the load data as an operand. If set to 0, load data from the DSOCM/ must pass through steering logic before arriving at a latch. This causes a single cycle of latency between a load instruction which is followed by an instruction that requires the load data as an operand.
Bit 2	DSOCMBUSY	This status bit can be used as a flag indicator to the FPGA fabric. This is an optional signal.
Bit 3	Enable Auto Clock Ratio Detection.	If set to 1, automatic clock ratio detection circuits will be enabled and users do not need to setup the CPU Clock / DSOCM Clock ratio in DSCNTL[4:7]. Additionally, when DSOCMMCM is read back, the value of the auto-detected clock ratio is reflected in terms of the wait state value. If set to 0, automatic clock ration detection will be disabled and users need to setup CPU Clock/DSOCM Clock ratio in DSCNTL[4:7]. This is an enhanced feature in Virtex-4 devices and we recommend setting this bit to 1.
Bit 4:7	DSOCMMCM	CPU Clock and OCM Clock ratio. For Virtex-4 devices, if Auto Clock Ratio Detection is enabled users need not setup the ratio in this field. Users can also read back this field to determine the clock ratio detected by the circuits. If Auto Clock Ratio Detection is disabled, users need to setup the ratio in this field. Reading back from this field will return the content set by users previously.

ISCNTL Registers

Table 3-11 and Table 3-12 describe the ISCNTL registers in Virtex-II Pro and Virtex-4 devices. For additional information, refer to Figure 3-13, page 166 (Virtex-II Pro) and Figure 3-14, page 167 (Virtex-4).

Table 3-11: **ISCNTL Register for Virtex-II Pro**

Bit 0	ISOCM Enable	If set to 1, address decoding based on the value of ISARC will be enabled. If set to 0, the content in ISARC will be ignored.
Bit 1:4	Reserved.	This bit must be configured to 0.
Bit 5:7	ISOCMMCM	CPU Clock and ISOCM Clock ratio. For Virtex-II Pro users, users must setup the ratio in this field with valid clock ratios used in the application system. Then the processor gasket will issue appropriate transactions based on this ratio.

Table 3-12: **ISCNTL Register for Virtex-4**

Bit 0	ISOCM Enable	If set to 1, address decoding based on the value of ISARC will be enabled. If set to 0, the content in ISARC will be ignored.
Bit 1	Reserved.	This bit must be configured to 0.
Bit 2	Enable DCR Based Read Back	If this bit is set to 1, reading from ISFILL register using an mfdcr instruction will return the memory content addressed by ISINIT register. If this bit is set to 0, reading from ISFILL register using a "mfdcr" instruction will return the previous content of ISFILL register set by user. This is an enhanced feature in Virtex-4 devices.
Bit 3	Enable Auto Clock Ratio Detection	If set to 1, automatic clock ratio detection circuits will be enabled and users do not need to setup the CPU Clock/ISOCM Clock ratio in ISCNTL[4:7]. Additionally, when ISOCMMCM is read back, the value of the auto-detected clock ratio is reflected in terms of the wait state value. If set to 0, automatic clock ratio detection will be disabled and users need to setup CPU Clock /ISOCM Clock ratio in ISCNTL[4:7]. This is an enhanced feature in Virtex-4 devices, and we recommend setting this bit to 1.
Bit 4:7	ISOCMMCM	CPU Clock and OCM Clock ratio. For Virtex-4 devices, if Auto Clock Ratio Detection is enabled users need not setup the ratio in this field. Users can also read back this field to determine the clock ratio detected by the circuits. If Auto Clock Ratio Detection is disabled, users need to setup the ratio in this field. Reading back from this field will return the content set by users previously.

Virtex-4 Features Compared with Virtex-II Pro Features

In Virtex-4 FPGAs an optional auto clock ratio detection feature was implemented on both the DSOCM and ISOCM. If bit 3 (Enable Auto Clock Ratio Detection) of the DSCNTL/ISCNTL register(s) is 1, then auto clock ratio detection will take place. This is the recommended operation model for Virtex-4 FPGAs. Additionally, when DSOCMMCM/ISOCMMCM is read back, the value of the auto-detected clock ratio is reflected in terms of the wait state value. In Virtex-II Pro FPGAs, the OCM clock cycle modes are selected through the MULTICYCLEMODE control bits (DSOCMMCM and ISOCMMCM) in the DSCNTL and ISCNTL registers.

Virtex-4 FPGAs support a maximum clock ratio of 8:1, and Virtex-II Pro FPGAs support a maximum clock ratio of 4:1. Therefore, Virtex-4 devices have one more control bit in both the ISOCMMCM and the DSOCMMCM registers.

Another extended feature in Virtex-4 FPGAs is the DCR-based read access to the ISOCM to support software debugging. To enable this feature, bit 2 of the ISCNTL register must be enabled.

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

DSARC (DSOCM 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

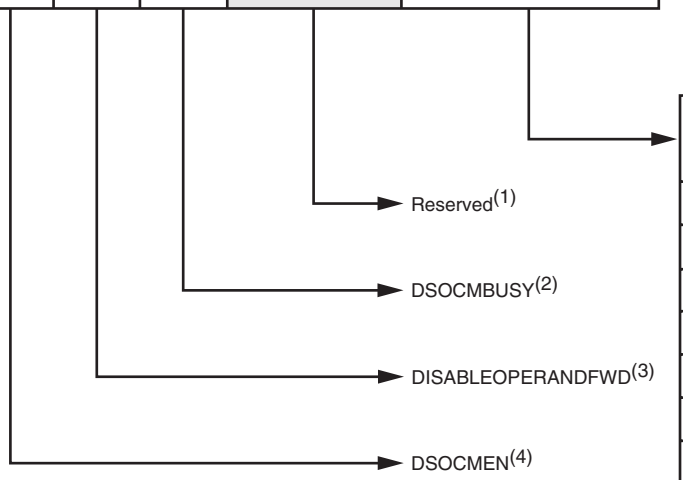
8 bits: Address range compare for DSOCM memory space. They are also configurable via FPGA, through the DSARCVALUE inputs to the processor block.

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

DSCNTL (DCR Control Register)							
0	1	2	3	4	5	6	7
D0/P	D1/P	D2/P	D3/P	D4/P	D5/P...		D7/P

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

(P indicates that this bit can be configured during FPGA power up)



DSOCMMCM[0:2]	CPMC405CLOCK: BRAMDSOCMCLK Ratio
000	N/A
001	1:1
010	N/A
011	2:1
100	N/A
101	3:1
110	N/A
111	4:1

2*n* - 1

where ***n*** = number of processor clocks in one BRAM clock cycle. Must be an integer.

Notes:
1. Reserved bits; will read 0.
2. See section "DSOCM Ports" in the text.
3. DISABLEOPERANDFWD:
   When DISABLEOPERANDFWD is asserted, load data from the DSOCM goes directly into a latch in the processor block. This causes an additional cycle (a total of two cycles) of latency between a load instruction which is followed by an instruction that requires the load data as an operand.

   When DISABLEOPERANDFWD is *not* asserted, load data from the DSOCM must pass through steering logic before arriving at a latch. This causes a single cycle of latency between a load instruction which is followed by an instruction that requires the load data as an operand.
4. DSOCMEN:
   Enables the DSOCM address decoder.

UG018_46_042304

Figure 3-11: **Virtex-II Pro DSOCM DCR Registers**

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

DSARC (DSOCM 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

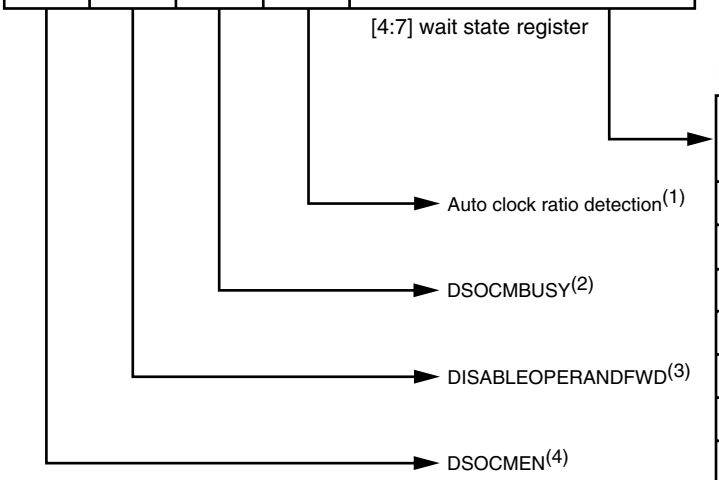
8 bits: Address range compare for DSOCM memory space. They are also configurable via FPGA, through the DSARCVALUE inputs to the processor block.

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

DSCNTL (DCR Control Register)							
0	1	2	3	4	5	6	7
D0/P	D1/P	D2/P	D3/P	D4/P	. . .		D7/P

[4:7] wait state register

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



Legacy support for backward compatibility with Virtex-II Pro

DSOCMMCM[0:3]	CPMC405CLOCK: BRAMDSOCMCLK Ratio
0000	Not supported
0001	1:1
0010	Not supported
0011	2:1
0100	Not supported
0101	3:1
0110	Not supported
0111	4:1
1000	Not supported
1001	5:1
1010	Not supported
1011	6:1
1100	Not supported
1101	7:1
1110	Not supported
1111	8:1

2*n* - 1

where ***n*** = number of processor clocks in one BRAM clock cycle. Must be an integer.

Notes:
1. Recommend 1 for auto clock ratio detection. Additionally, when DSOCMMCM is read back, the value of the auto-detected clock ratio is reflected in terms of the wait state value.
2. See section "DSOCM Ports" in the text.
3. DISABLEOPERANDFWD:
   When DISABLEOPERANDFWD is asserted, load data from the DSOCM goes directly into a latch in the processor block. This causes an additional cycle (a total of two cycles) of latency between a load instruction which is followed by an instruction that requires the load data as an operand.

   When DISABLEOPERANDFWD is *not* asserted, load data from the DSOCM must pass through steering logic before arriving at a latch. This causes a single cycle of latency between a load instruction which is followed by an instruction that requires the load data as an operand.
4. DSOCMEN:
   Enables the DSOCM address decoder.

UG018_46b_042304

Figure 3-12: **Virtex-4 DSOCM DCR Registers**

User Programmable 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.

(P indicates that this bit can be configured during FPGA power up)

Reserved(1)

ISOCMEN(2)

ISOCMMCM[0:2]	CPMC405CLOCK: BRAMISOCMCLK Ratio
000	N/A
001	1:1
010	N/A
011	2:1
100	N/A
101	3:1
110	N/A
111	4:1

2n - 1

where ***n*** = number of processor clocks in one OCM clock cycle. Must be an integer.

Notes:
1. Reserved bits; will read 0.
2. ISOCMEN:
   Enables the ISOCM address decoder.

UG018_47_042304

Figure 3-13: **Virtex-II Pro ISOCM DCR Registers**

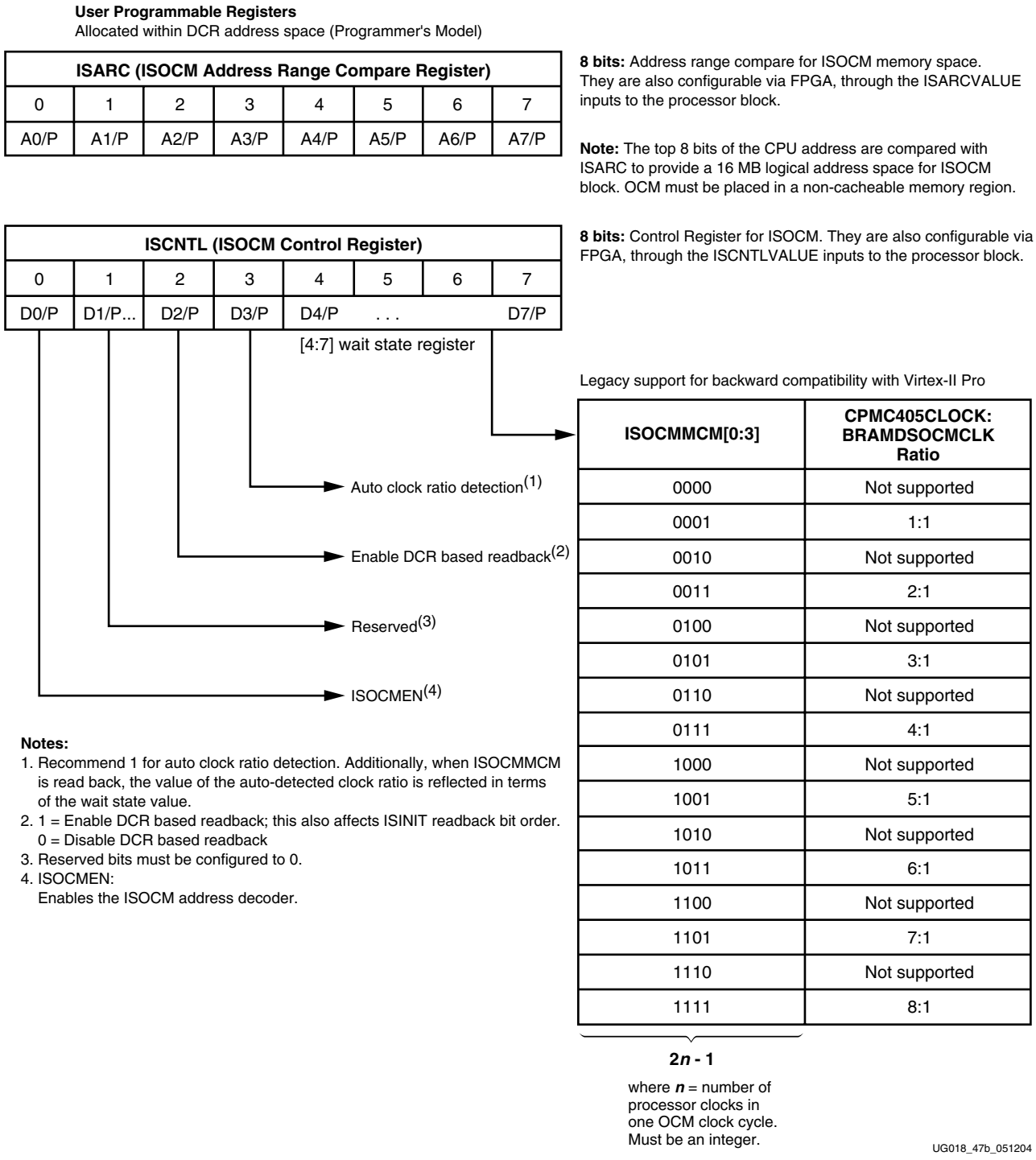


Figure 3-14: **Virtex-4 ISOCM DCR Registers for**

The following section describes the DCR bit mapping during read/write operations on the ISINIT and ISFILL registers.

DCR Write Access

As shown in Figure 3-15, ISINIT is a 22-bit register (A8-A29) that is mapped to DCR write data bus bits D8-D29. The write address on the memory interface is A8-A28, and address bit A29 is used to control the ISOCMBRAMODDWRITEEN and ISOCMBRAMEVENWRITEEN signals. Additionally, in Virtex-4 devices, the ISOCMDCRBRAMEVENEN and ISOCMDCRBRAMODDEN signals can be used to select the corresponding block RAMs in which to write. Each time register ISFILL is written, there is one 32-bit instruction written into the block RAM (odd or even, depending on the value of address bit A29).

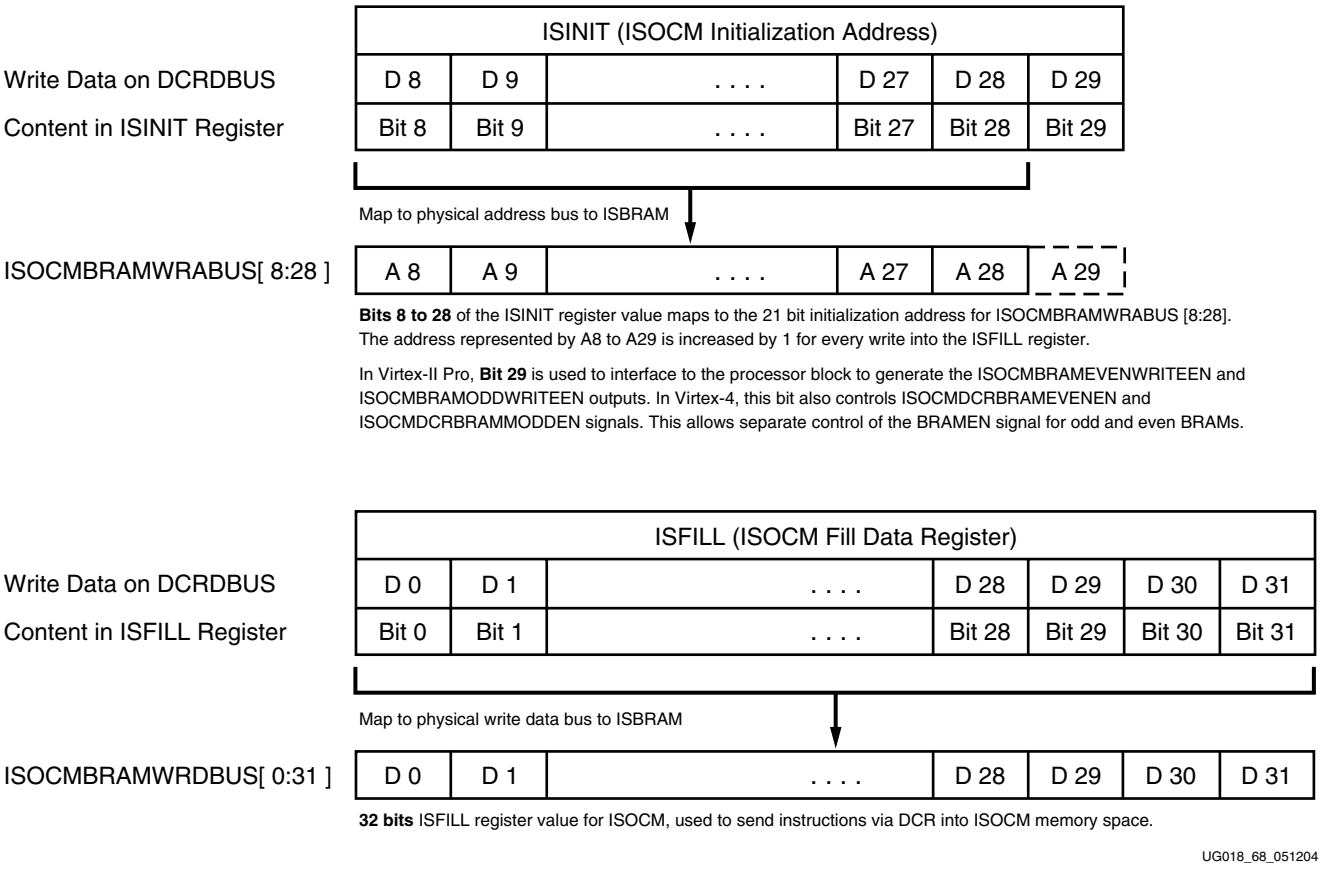


Figure 3-15: **Virtex-II Pro and Virtex-4 ISOCM: ISINIT and ISFILL Descriptions (Write Access)**

DCR Read Access

If the ISINIT register is read back on the DCR:

- For Virtex-II Pro devices, bits A8-A29 are mapped onto DCR read data bus bits D0-D21 as shown in Figure 3-16, please note that the mapping for read access is different from write.
- For Virtex-4 devices, if bit 2 of ISENTL is set to 1, bits A8-A29 are mapped onto DCR read bus bits D8-D29, as shown in Figure 3-17. This helps to eliminate bit shifting in software for further operation on the DCR read value of the ISINIT register. The read address on the memory interface is A8 to A28. Address bit A29 is used to control the ISOCMDCRBRAMEVENEN and ISOCMDCRBRAMODDEN signals. Each time register ISFILL is written, there is one 32-bit instruction written into the block RAM (odd or even, depending on the value of address bit A29). Otherwise, if bit 2 of ISCNTL is set to 0, ISINIT is mapped the same way as it is in Virtex-II Pro devices during DCR read.

If the ISFILL register is read back on the DCR:

- For Virtex-II Pro devices, the current content stored in the ISFILL register will be returned as DCR read data. The actual content of ISOCM addressed by the ISINIT register will not be loaded.
- For Virtex-4 devices, if the DCR-Based Read Back feature is enabled (bit 2 of ISCNTL in a Virtex-4 device is set to 1), the actual content of ISOCM addressed by ISINIT register will be loaded, otherwise, the current content stored in the ISFILL register will be returned as DCR read data.

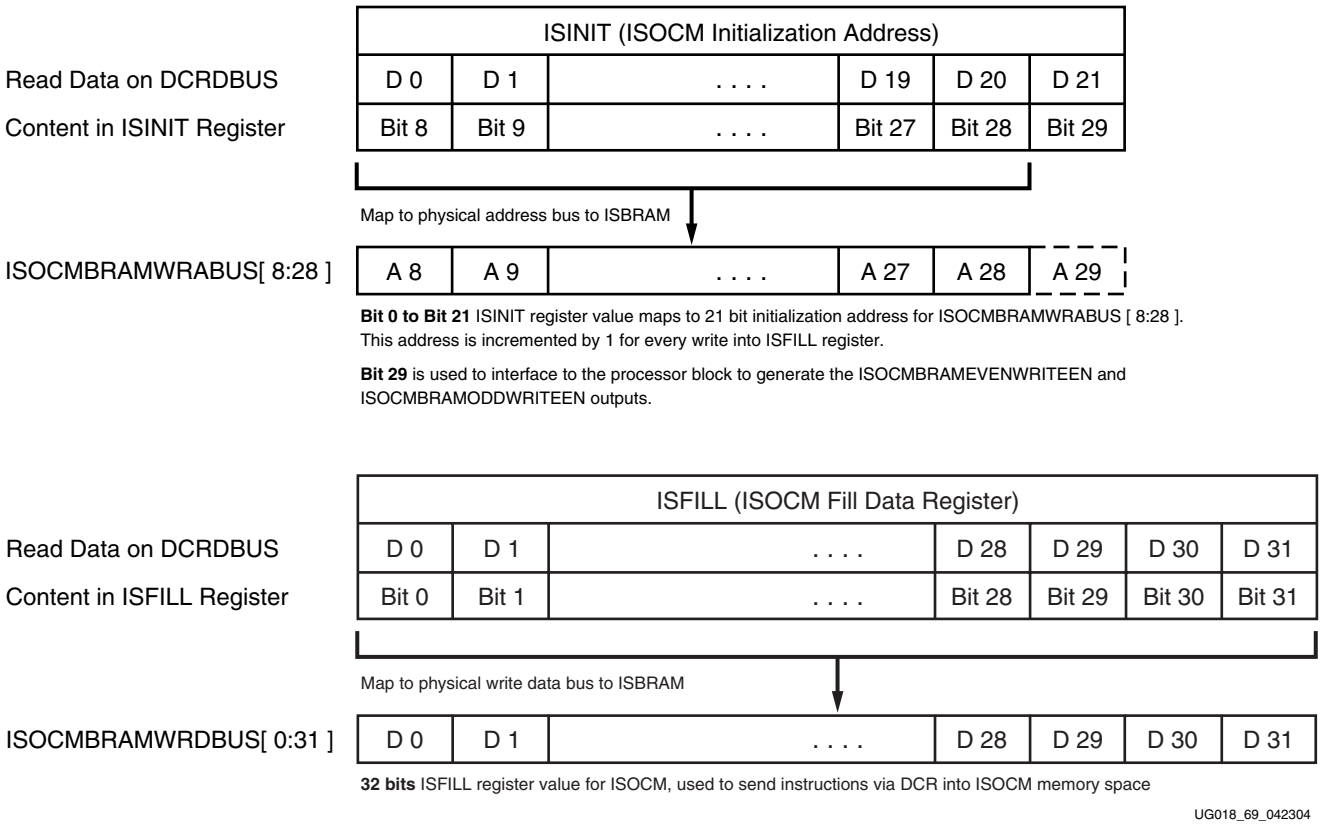
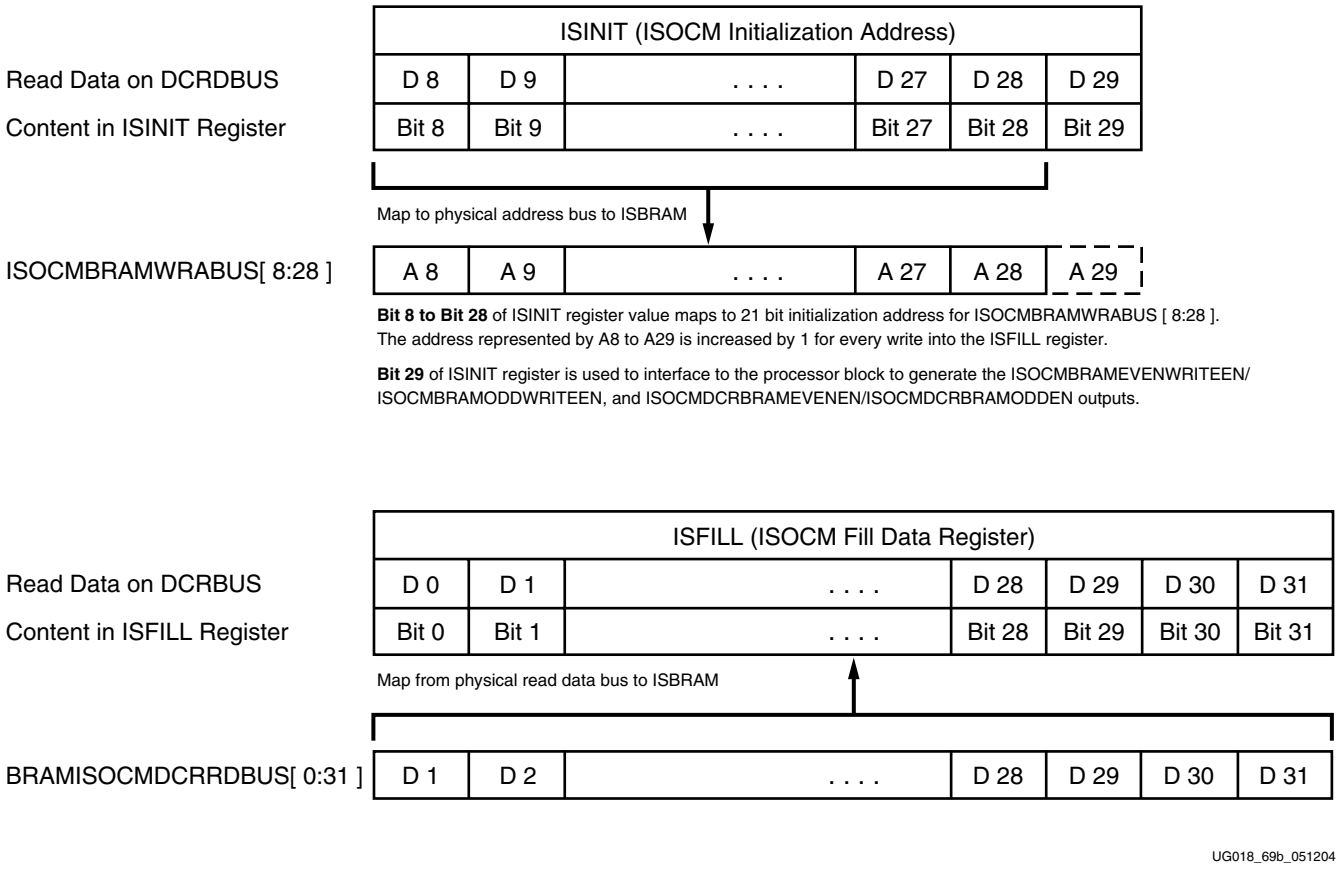


Figure 3-16: **Virtex-II Pro ISOCM: ISINIT and ISFILL Descriptions (Read Access)**



Note: DCR-based readback requires that the Readback bit (ISCNTL[2]) is enabled.

Figure 3-17: **Virtex-4 ISOCM: ISINIT and ISFILL Descriptions (Read Access)**

Block RAMs that interface with the ISOCM controller can also be initialized through the configuration bit-stream, during FPGA configuration. The Data2MEM software utility in the design flow tools can be used to load ISBRAM and DSBRAM with instructions and data respectively.

Timing Specification for Fixed Latency (Virtex-4 and Virtex-II Pro)

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 quantity of block RAMs. As additional block RAMs 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 must be aligned on rising edges of their respective clocks.

The frequency of the OCM to block RAM 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 block RAM organization, signal

routing delays, signal loading, block RAM memory access time, clock to output times, and setup and hold times of the block RAM and processor blocks. Users may need to go through multiple iterations of evaluating OCM block RAM size versus OCM clock frequency in order to achieve the optimum performance.

The Virtex-4 clock ratio between the block RAM clock and the PowerPC 405 is auto-detected in when control register bit 3 is set to 1 (DSCNTL and ISCNTL). The Virtex-II Pro clock ratio is set using bits 5 to 7. Refer to the "Programmer's Model" section for further details.

Single-Cycle Mode

In single-cycle mode, the CPU core, OCM controllers, and block RAMs 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 block RAM 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 block RAMs 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 end 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 CPU core, OCM controllers, DSBRAMs, and ISBRAMs. Additionally, an identical clock must be applied to an OCM controller (DSOCM or ISOCM) and its corresponding block RAMs for any mode described above. Each controller (DSOCM or ISOCM) can be clocked at a frequency independent of the other.

ISOCM Instruction Fetching

The figures below show two back to back instruction fetches for single-cycle mode (Figure 3-18) and multi-cycle mode with CPMC405CLOCK:BRAMISOCMCLK ratio of 2:1 (Figure 3-19). 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, note that the rising edge of the BRAMISOCMCLK defines the bus cycle, as the timing diagram illustrates.

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:

1. The CPU launches the instruction fetch address.
2. The OCM controller translates the CPU order and routes the address and control signals onto the ISOCM bus.
3. One wait state is introduced to permit the synchronous block RAM to access the data.
4. The CPU stores the data.

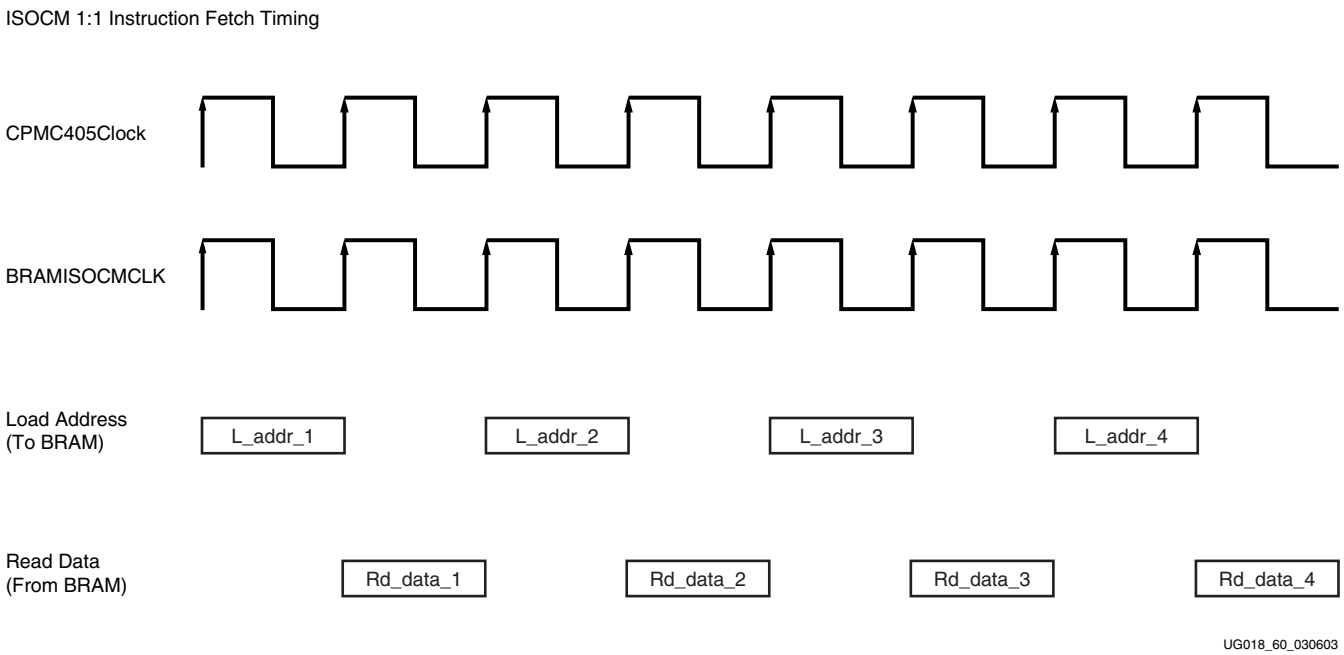


Figure 3-18: **Instruction Fetch Timing**

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 block RAM 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.

In order to estimate the theoretical maximum number of instruction fetches per second on the OCM interface, measure the period of the block RAM clock cycle to determine the maximum throughput.

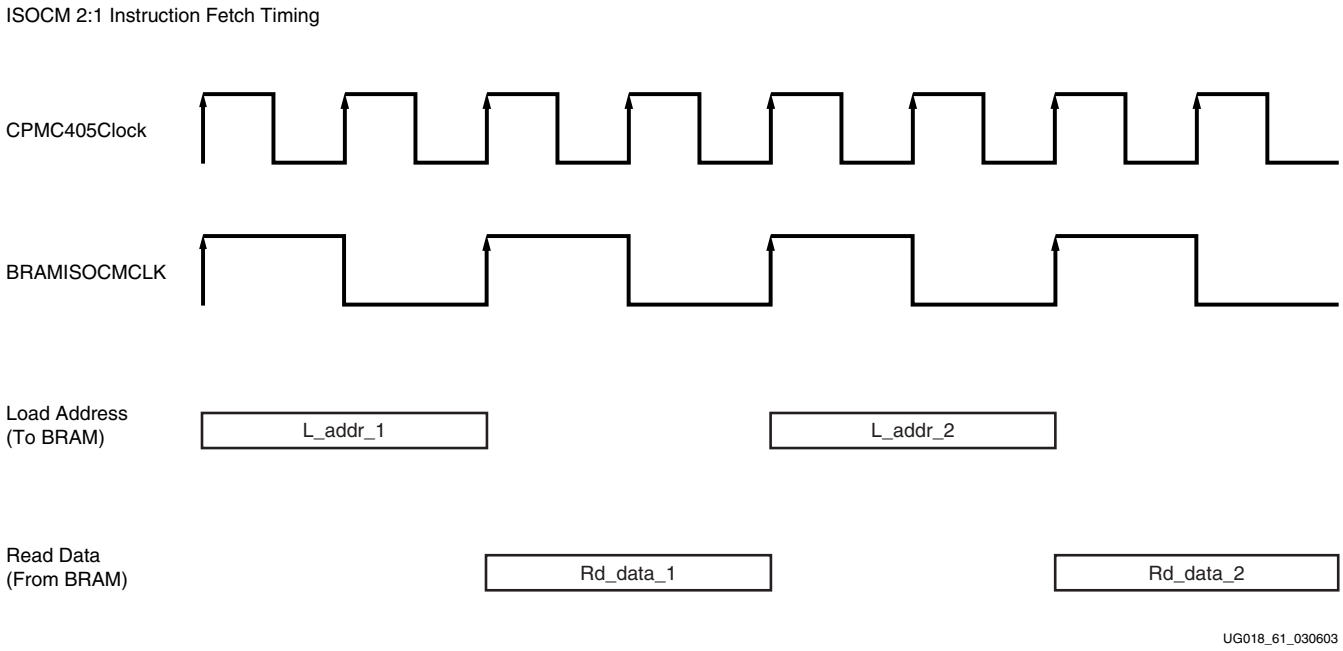


Figure 3-19: **Multi-Cycle Mode (2:1) Instruction Fetch Timing**

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.

Writing to ISBRAM

There are two methods used to write to the instruction side memory. Typically, the block RAM is initialized in the device configuration bitstream. The Data2MEM software utility in the design implementation tools is used to load the block RAM with instructions as well as data. If the application code is static, this eliminates the need to use the DCR based writes through the ISOCM controller.

Write accesses to the ISOCM-attached memory can be performed using the DCR bus. The DCR ISINIT register is first initialized with a start address, then every DCR write to the ISFILL register results in a write into the block RAM. 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. Figure 3-20 and Figure 3-21 show the timing diagrams for a write to instruction memory in single-cycle mode and multi-cycle Mode. The timing interface between the OCM controller and the memory is always with respect to the BRAMISOCMCLK.

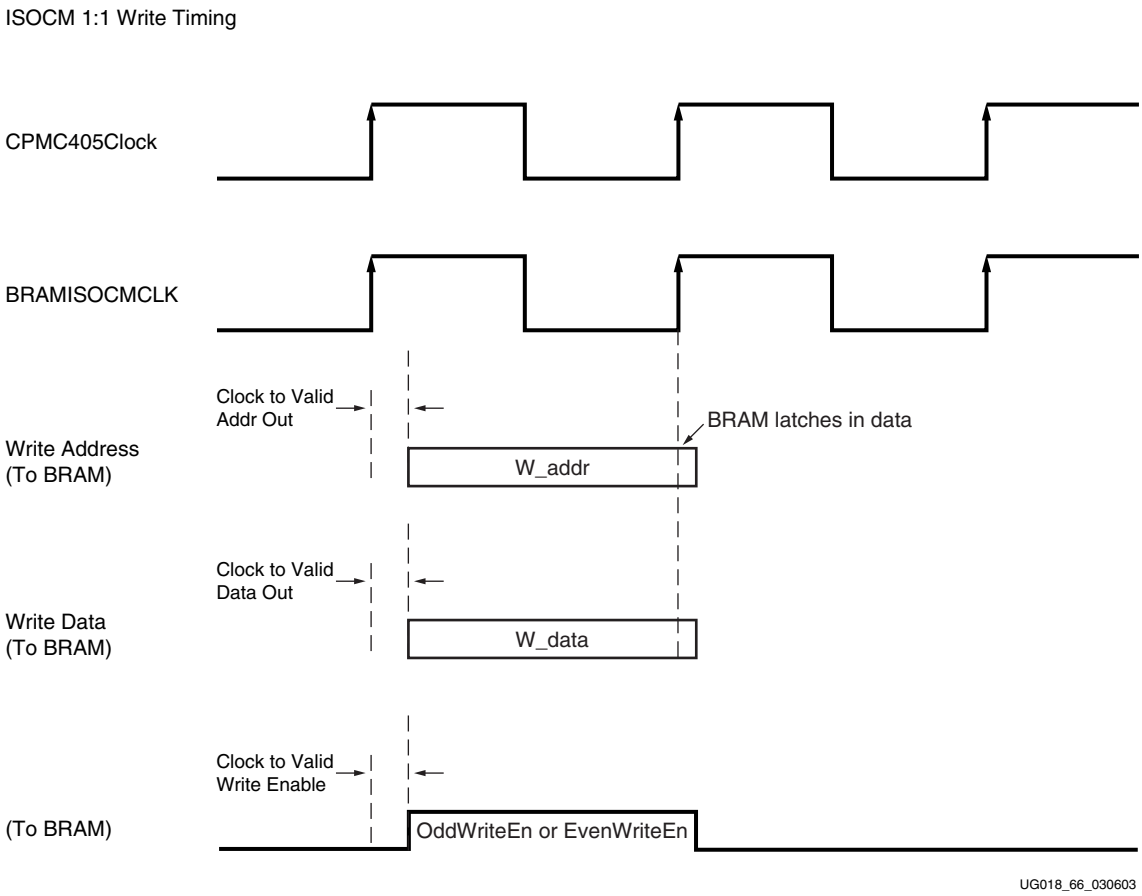


Figure 3-20: **Single Cycle Mode (1:1) ISOCM Write Timing**

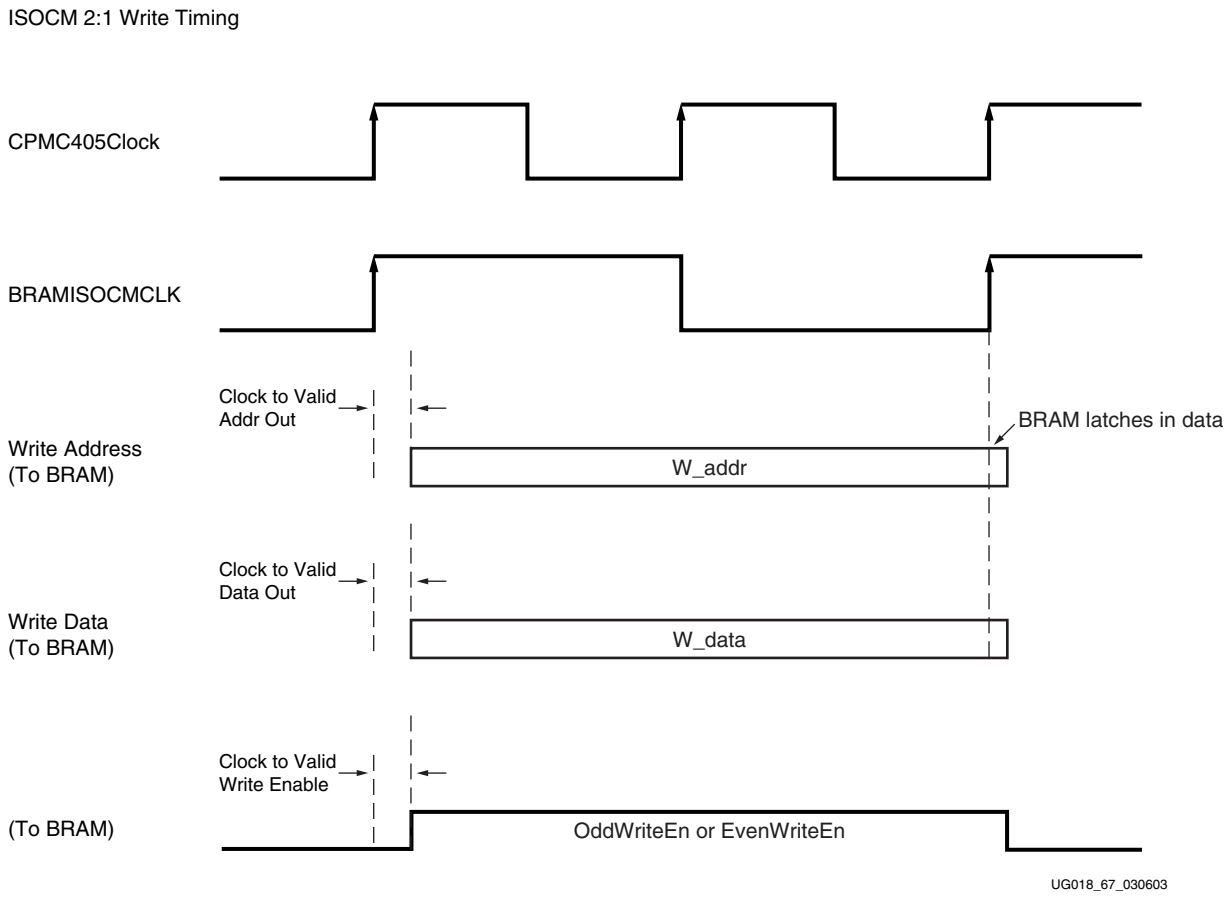


Figure 3-21: **Multi Cycle Mode (2:1) ISOCM Write Timing**

DSOCM Data Load, Fixed Latency

Figure 3-22 and Figure 3-23 show two back-to-back loads for single-cycle mode and multi-cycle mode with a 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 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 sequence:

1. The CPU launches the load address.
2. The OCM controller translates the CPU order and routes the address and control signals onto the DSOCM bus.
3. One wait state is introduced to permit the synchronous block RAM to access the data.
4. The CPU stores the data into a general-purpose register.

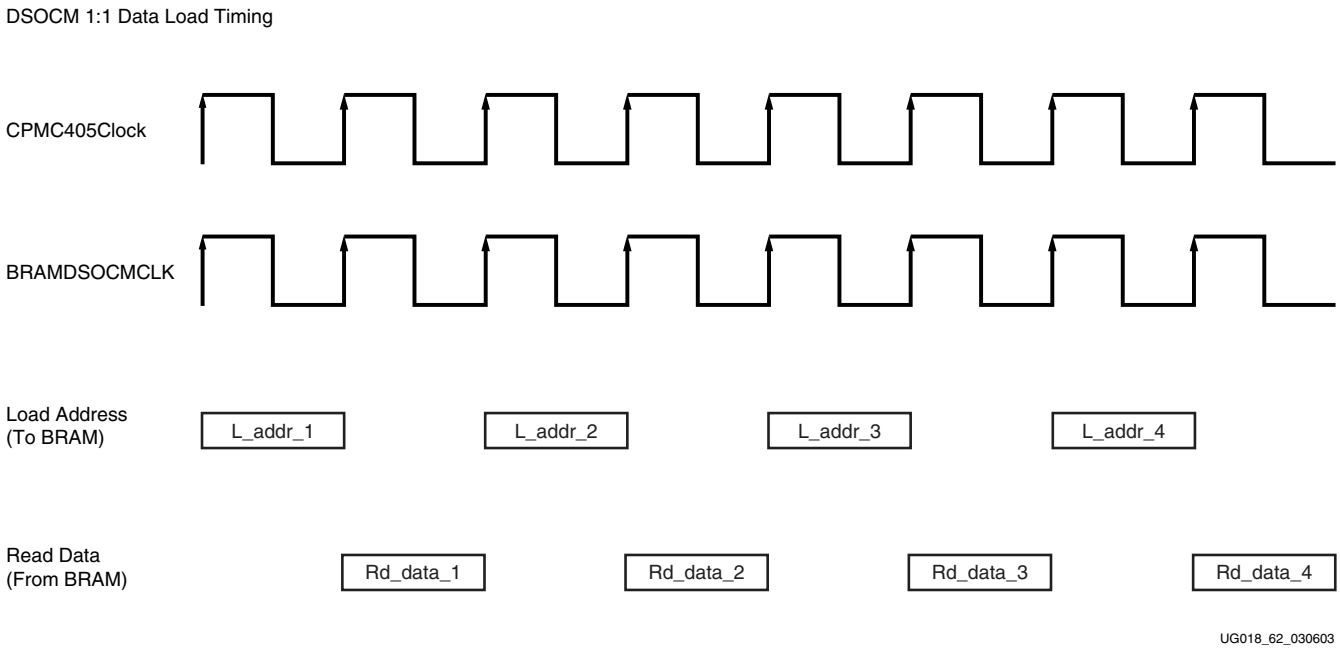


Figure 3-22: **Single Cycle Mode (1:1) Data Load Timing**

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 the block RAM clock should be used to establish throughput. Note that this is only an estimate for load performance.

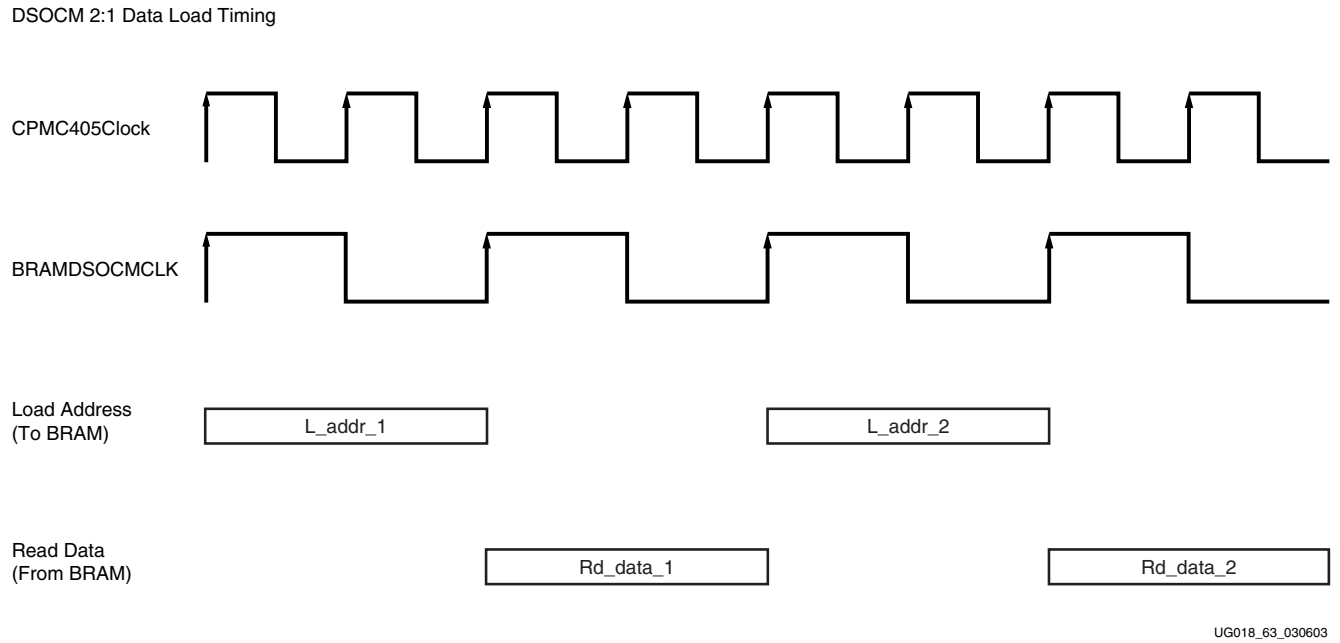


Figure 3-23: **Multi Cycle Mode (2:1) Data Load Timing**

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, Fixed Latency

Figure 3-24 and Figure 3-25 below show two back-to-back stores for single-cycle mode and multi-cycle mode with a 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 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 sequence:

1. The CPU launches the store address.
2. The OCM controller translates the CPU order and routes the address, data, and control signals onto the DSOCM bus.
3. The block RAM stores the data.

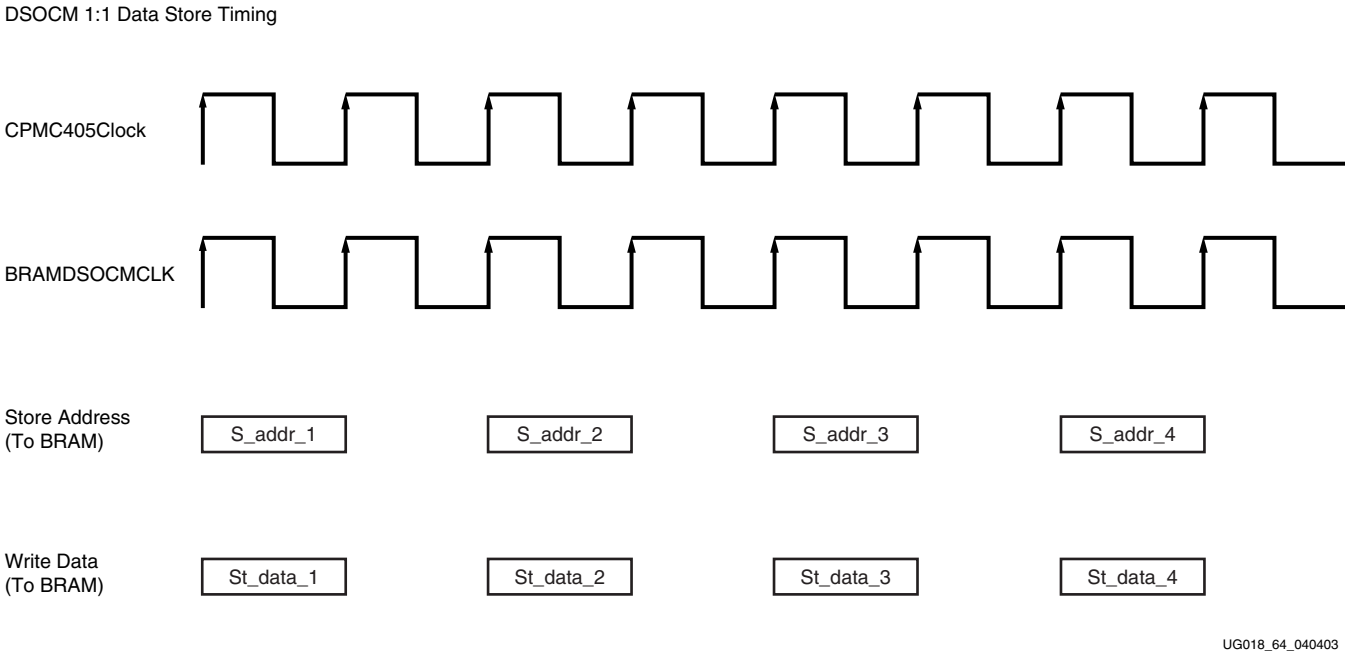


Figure 3-24: **Single Cycle Mode (1:1) Data Store Timing**

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 two block RAM 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 block RAM

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

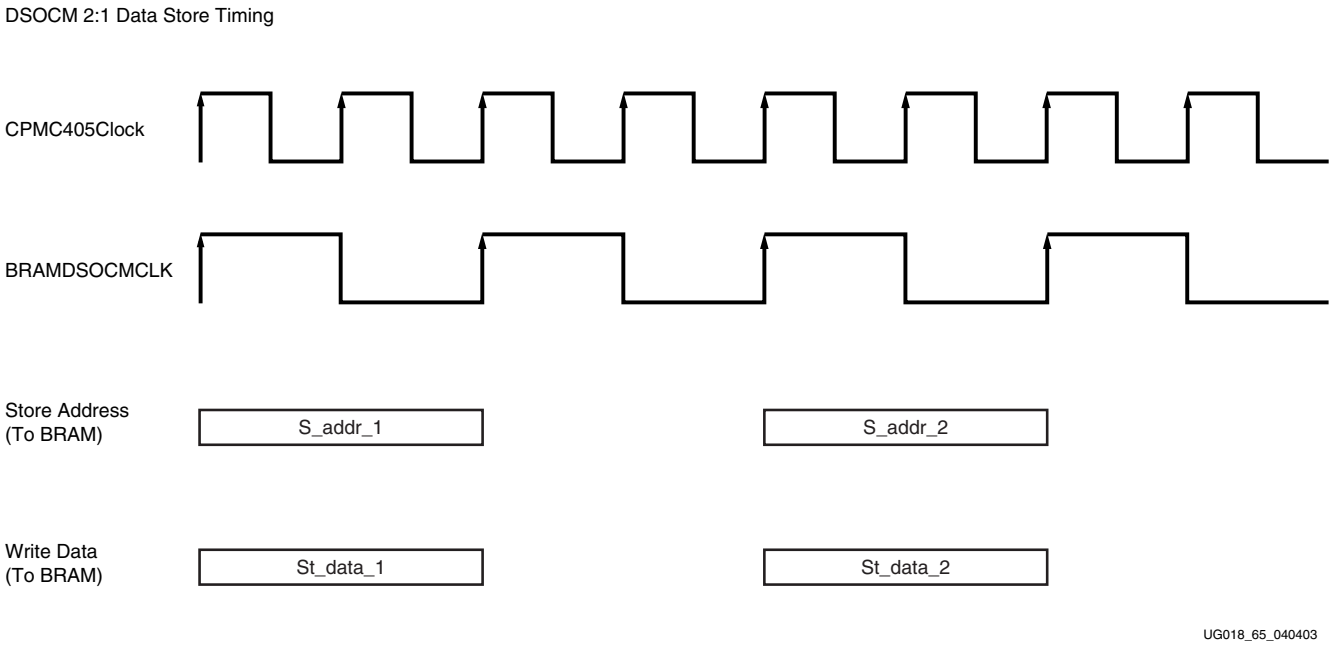


Figure 3-25: **Multi Cycle Mode (2:1) Data Store Timing**

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.

Timing Specification for Variable Latency (Virtex-4 DSOCM Controller Only)

In Virtex-4, the DSOCM controller supports variable latency bus operations, which provides the flexibility to attach one or more memory-mapped slave peripherals to the interface. The variable latency feature allows the FPGA fabric interface to take multiple clocks (BRAMDSOCMCLK) before a load or store operation can be completed. This allows different slave peripheral devices to respond based on the application's requirement and not based on a pre-defined number of BRAMDSOCM clock cycles. Both the DSOCM controller and the slave peripheral attached to the OCM still run at the BRAMDSOCMCLK frequency.

A new completion signal, DSOCMRWCOMPLETE, is introduced in Virtex-4. This signal must be driven by the DSOCM memory-mapped slave peripheral. For a list of use models and applications, see "References."

As in Virtex-II Pro, the PowerPC 405 and DSOCM controller would still operate in either a 1:1 clock ratio (single-cycle mode) or *N*:1 clock ratio (multi-cycle mode, where *N*=2 to 8. The following sections show examples of load and store instructions, in both single-cycle mode and multi-cycle mode.

DSOCM Data Load, Variable Latency

Figure 3-26 and Figure 3-27 show two load operations with variable latency for single cycle mode and multi-cycle mode with a CPMC405CLOCK:BRAMDSOCMCLK ratio of 2:1.

In both single-cycle mode and multi-cycle mode, the data load operation consists of the following sequence:

1. The CPU launches the load request to the OCM controller.
2. The OCM controller translates the CPU order, routes the address, and asserts all of the necessary control signals.

   Note: Read control signals (DSOCMBRAMEN, DSOCMRDADDRVALID) are active for only one BRAMDSOCMCLK cycle and must be registered in the FPGA fabric if they are required for further processing.

   Note: DSOCMRDADDRVALID indicates a valid read address on the DSOCMRDABUS. DSOCMBRAMEN is also asserted for both read or write requests. However, one can choose to ignore this signal if the design does not use block RAMs.
3. The slave waits for multiple BRAMDSOCMCLK cycles–the number of clock cycles depends on the application–and then asserts DSOCMRWCOMPLETE, which must be accompanied by valid read data.
4. The DSOCM controller sees the completion signal (DSOCMRWCOMPLETE) and latches the read data driven by the slave on BRAMDSOCMRDDBUS.
5. The DSOCM controller forwards the data back to the PowerPC 405.

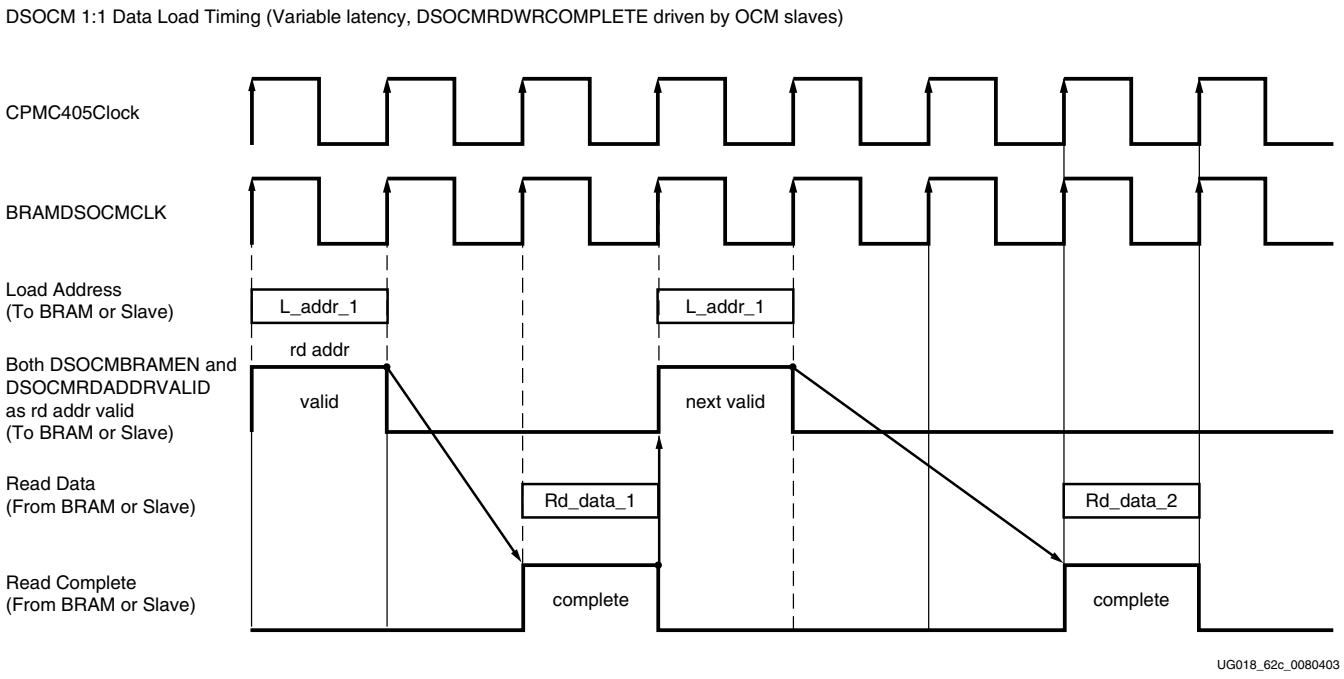


Figure 3-26: **Single Cycle Mode (1:1) DSOCM Read Variable Latency for Virtex-4**

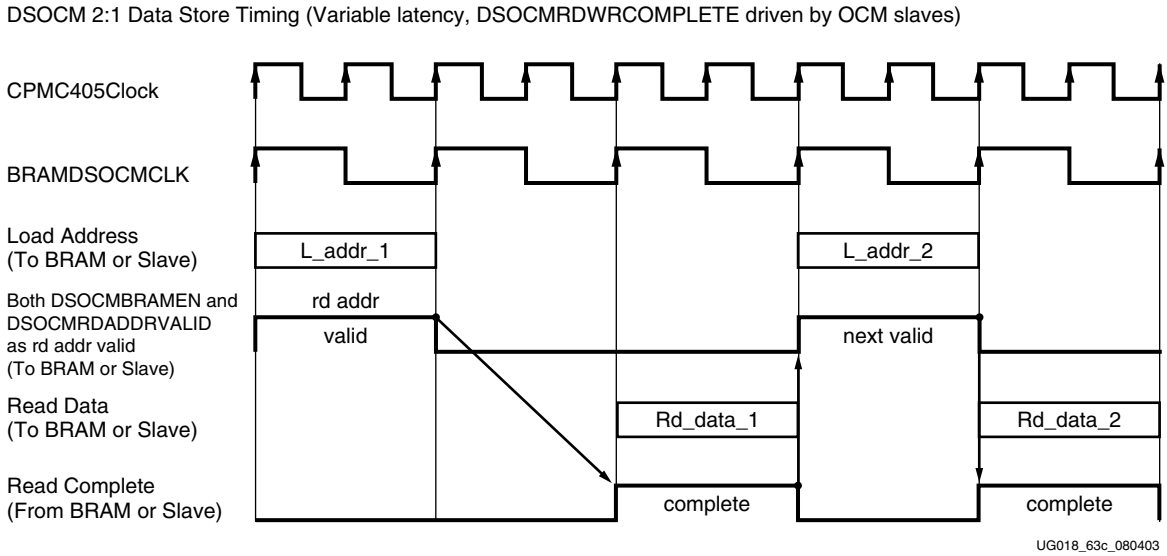


Figure 3-27: **Multi Cycle Mode (2:1) DSOCM Read Variable Latency Virtex-4**

DSOCM Data Store, Variable Latency

Figure 3-28 and Figure 3-29 show two store operations with variable latency for single-cycle mode and for multi-cycle mode with a CPMC405CLOCK:BRAMDSOCMCLK ratio of 2:1.

In both single-cycle mode and multi-cycle mode, the access consists of the following sequence:

1. The CPU launches the store request to the OCM controller.
2. The OCM controller translates the CPU order, routes address and write data, and asserts all of the necessary output control signals.

   Note: Write control signals (DSOCMWRADDRVALID, DSOCMBRAMEN, DSOCMBRAMBYTEWRITE) are active for only one BRAMDSOCMCLK cycle and must be registered in the FPGA fabric if they are required for further processing.

   Note: DSOCMBRAMBYTEWRITE indicates a valid write address and write data on the DSOCMWRABUS. The DSOCMBRAMEN is also asserted for both read or write requests. However, one can choose to ignore this signal if the design does not use block RAMs.
3. The slave waits for multiple BRAMDSOCMCLK cycles (the number of clock cycles depends on the application) and then asserts DSOCMRWCOMPLETE, which signifies a completion of write data store.
4. The DSOCM controller sees the completion signal (DSOCMRWCOMPLETE) and allows the internal state machine to move forward for the next request on the DSOCM bus.

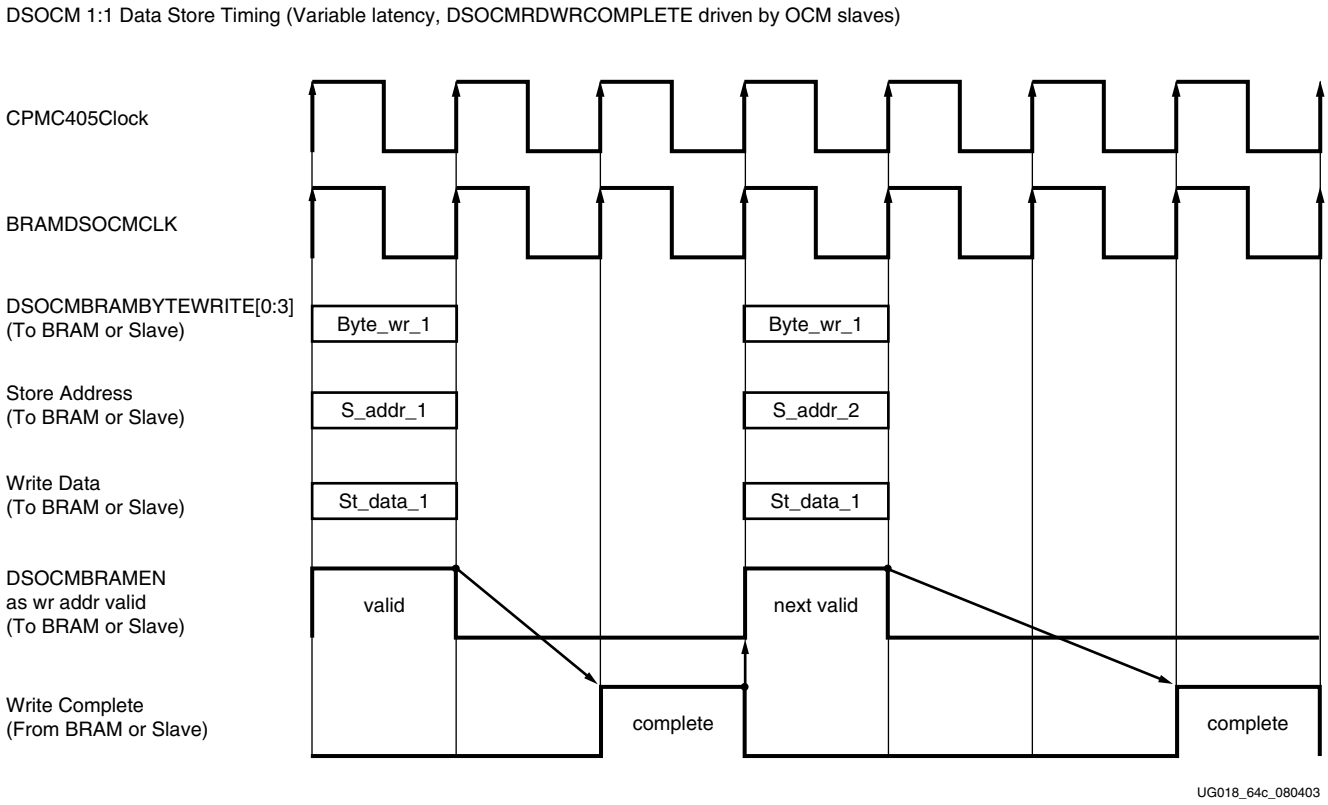


Figure 3-28: **Single Cycle Mode (1:1) DSOCM Write Variable Latency Virtex-4**

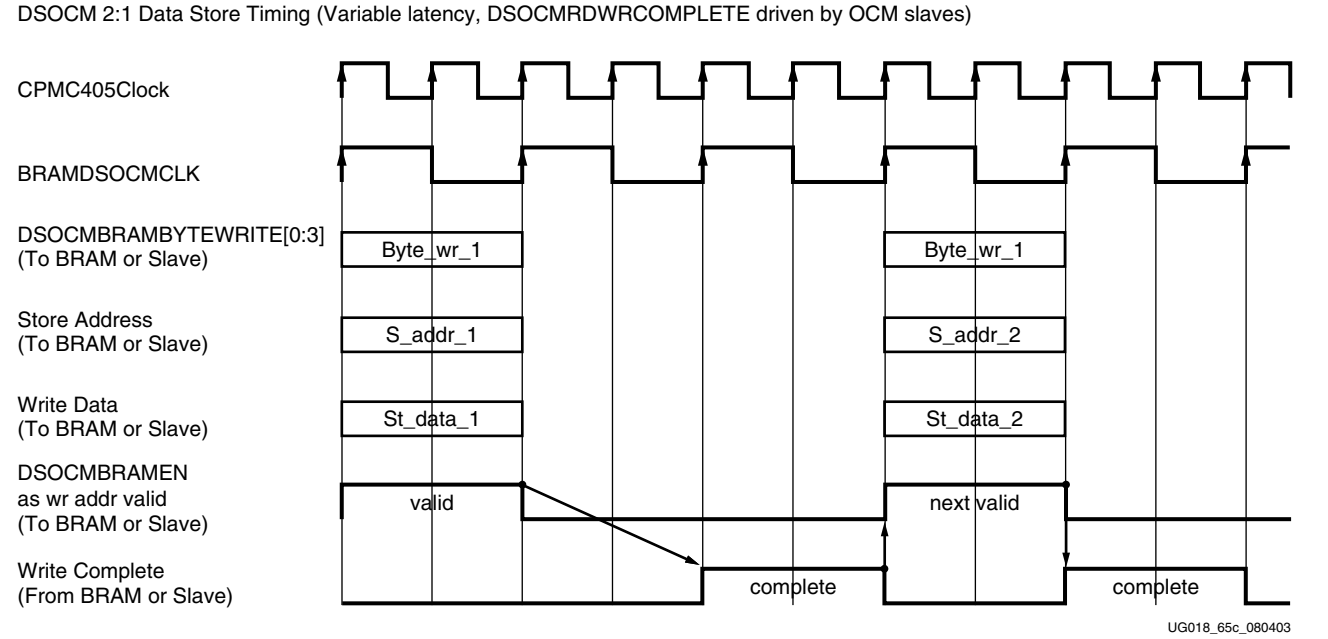


Figure 3-29: **Multi Cycle Mode (1:2) DSOCM Write Variable Latency Virtex-4**

Application Notes and Reference Designs

Xilinx provides several application notes and reference designs utilizing the OCM controllers. Design examples include:

- Booting the PowerPC 405 from on-chip memory.
- Using the dual port feature of the DSOCM in eight, sixteen and thirty-two bit fabric interfaces
- A comparison of PLB versus OCM performance using a software example.

For Virtex-II Pro designs the following application notes and reference designs are available from the Xilinx web site:

- XAPP644: "PLB vs. OCM Comparison Using the Packet Processor Software"
- XAPP660: "Partial Reconfiguration of RocketIO Pre-emphasis and Differential Swing Control Attributes"
- XAPP669: "PowerPC 405 PPE Reference System Using Virtex-II Pro RocketIO Transceivers"
- XAPP672: "The UltraController Solution: A Lightweight PowerPC Microcontroller"
- XAPP699:"A Software UART for the UltraController GPIO Interface"

References

- *Virtex-II Pro User Guide*
- *Virtex-4 User Guide*
- *PowerPC Processor Reference Guide*

Chapter 4

PowerPC 405 APU Controller

This chapter only applies to the PowerPC 405 processor in the Virtex-4 FX family and covers the following topics:

- "FCM Instruction Processing"
- "APU Controller Configuration"
- "Interface Definition"
- "FCM Interface Timing Specification"

Note: The Auxiliary Processor Unit (APU) controller is not available in the Virtex-II Pro family.

Introduction

The Auxiliary Processor Unit (APU) controller allows the designer to extend the native PowerPC 405 instruction set with custom instructions that are executed by an FPGA Fabric Co-processor Module (FCM). This enables a much tighter integration between an application-specific function and the processor pipeline than is possible using, for example, a bus peripheral. Figure 4-1 shows the pipeline flow between the PowerPC 405 Core, the APU controller, and the Fabric Co-processor Module.

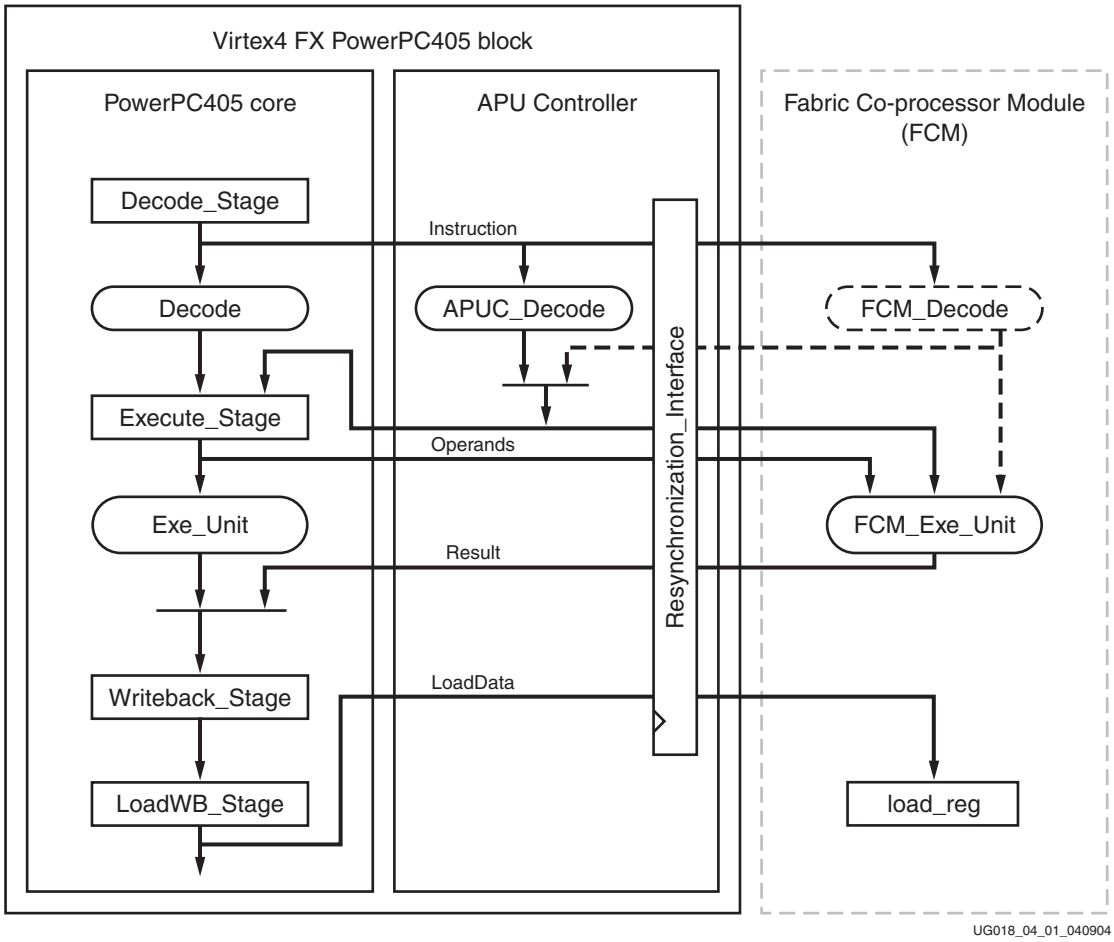


Figure 4-1: **Pipeline Flow Diagram**

The APU controller serves two purposes: to perform clock domain synchronization between the fast PowerPC clock and the slow FCM interface clock, and to decode certain FCM instructions and notify the CPU of the CPU resources needed by the instruction (for example source data from the CPU's GPR). Depending on the FCM application, the APU controller can decode all instructions or no instructions at all, or decode some while the FCM decodes others. A floating-point unit (FPU) is an example of a good FCM candidate. In the case of an FCM FPU, the APU controller is capable of decoding all PowerPC floating point instructions.

The FCM interface is a Xilinx adaptation of the native APU interface implemented on the IBM processor core. The hard core APU controller bridges the PowerPC 405 APU interface and the external FCM interface.

FCM Instruction Processing

FCM instruction decoding can be done by the APU controller or by the FCM. APU controller decoding means the APU controller determines what CPU resources are needed for the instruction execution and passes this information to the CPU. For example, the APU controller will determine if an instruction is a load, a store, or if it needs source data from the GPR, etc. The FCM can also perform this part of decoding and pass the needed

information to the APU controller. However, in order to determine the complete function of the instruction, the FCM must perform a full decode as well as handle all instruction execution. There are two types of instructions that can be executed by an FCM: pre-defined and user-defined (UDI). A pre-defined instruction has its format defined by the PowerPC instruction set (for example, floating point), and the FCM is simply a co-processor performing the ISA-defined execution. A user-defined instruction has a configurable format and is a true extension of the PowerPC instruction set architecture (ISA).

Enabling the APU Controller

The PowerPC MSR register must be configured before the processor can use the APU controller. Table 4-1 describes the APU controller-related bits in the MSR.

Table 4-1: **APU Controller-Related MSR Bits**

Bit(s) in MSR	Description
6	APU present (1=true, 0=false)
12	Enable APU exception (1=true, 0=false)
18	FCM floating point unit present (1=true, 0=false)
(20,23)	Floating point exception mode (FE0,FE1): • (0,0) Ignore FP exceptions • (1,0) Imprecise recoverable mode • (0,1) Imprecise non-recoverable mode • (1,1) Precise mode

Instruction Classes

The ISA extensions to the PowerPC are defined by their interaction with the normal processor pipeline execution. This leads to three different instruction classes: autonomous, non-autonomous blocking, and non-autonomous non-blocking.

Autonomous Instructions

Instructions in the autonomous class do not stall the pipeline of the PowerPC. They are typically fire-and-forget type instructions that are not expected to return any state (for example, overflow) or data to the processor pipeline. An example is a user-defined `UDI_FCM_Read` instruction(1), where an FCM register is loaded with the contents of one of the PowerPC GPR registers without returning any data to the processor. Although autonomous instructions do not stall execution of native instructions, they can stall execution of subsequent FCM instructions in case the FCM is not done with an earlier instruction.

Non-autonomous Instructions

A non-autonomous instruction will stall normal instruction execution in the PowerPC pipeline until the FCM instruction is done. This is typical for instructions that are expected to return some state (e.g. overflow) or data to the PowerPC. For example a user-defined

1. Note that this would not be the same as the "Load" instructions that operate on the storage hierarchy, such as caches, OCM, or PLB.

UDI_FCM_Write instruction that takes data from the FCM and writes it to a PowerPC GPR location.

Blocking Instructions

Any non-autonomous instruction that cannot be predictably aborted and later re-issued must be blocking. Once a blocking instruction has completed the first Execute cycle in the PPC pipeline (the same cycle source operands are ready), all interrupts and exceptions to the PowerPC are blocked, so as not to prevent the instruction from completing. This is, for example, true of the UDI_FCM_Write instruction if the source of the data is a FIFO inside the FCM. If aborted after the FIFO pointer has been changed, but before the data has been stored in the PowerPC register file, such instruction could not be re-issued predictably.

Non-blocking Instructions

Any non-autonomous instruction that can be aborted and predictably re-issued later can be defined as non-blocking. A non-blocking instruction allows the processor to terminate the FCM execution, service interrupts and exceptions, and subsequently re-issue the terminated instruction, with predictable results. If we replace the FIFO in the blocking example above with a traditional random access memory, the aborted UDI_FCM_Write instruction could be predictably re-issued (with no remaining side-effects associated with a FIFO read pointer).

Instruction Format

All FCM instructions conform to the general format shown in Figure 4-2.

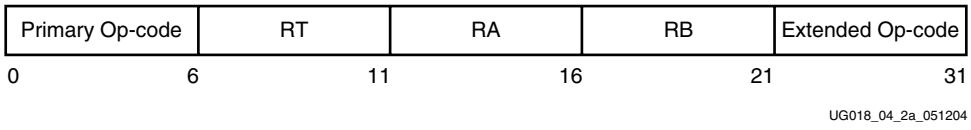


Figure 4-2: **FCM Instruction Format**

Generally speaking, the PowerPC uses both primary and extended op codes to identify potential FCM instructions. The op codes are decoded by the APU controller or the FCM to identify uniquely the specific FCM instruction. For all pre-defined instructions, the RA and RB fields specify operand registers, and the RT field the target register. User-defined instructions (UDI) can be configured to interpret these bit fields as, for instance, immediate values instead.

The primary and secondary op-codes shown in Table 4-2 can be used as APU instructions:

Table 4-2: **APU Op-codes**

Primary Op-code	Extended Op-code	Description
0 (= 0b000000)	0b00000000000	Illegal
	all except above	Available for UDIs(1) that do not set PPC405(CR) bits

Table 4-2: **APU Op-codes *(Cont'd)***

Primary Op-code	Extended Op-code	Description
4 (= 0b000100)	0b------1--0-	MAcc and Xilinx reserved
	0b1----000110	Available for UDIs that need to set PPC405(CR) bits
	all except above	Available for UDIs that do not set PPC405(CR) bits
31 (= 0b011111)	0b-----001110	Pre-defined FCM Load/Store
	0b-111-010-1-	FCM integer divide
(= 0b110---) (2)	0b----------	Pre-defined FPU Load/Store
32 (= 0b011111)	0b1----1-111-	Pre-defined FPU Load/Store
59 (= 0b111011)	0b----------	Pre-defined PowerPC FPU instructions
62 (= 0b111110)	0b--------1--	Pre-defined FPU Load/Store
63 (= 0b111111)	0b----------	Pre-defined PowerPC FPU instructions

Notes:

1. User-defined Instruction. For details refer to the "User-Defined Instruction Decoding" section of this chapter.
2. In this case, the first three bits are defined and the last three will change depending on the FPU instruction.

Instruction Decoding

FCM instructions can be decoded either by the APU controller or by the FCM itself. As mentioned earlier in the "Introduction", the main purpose of APU controller decoding is to pass needed resource information to the CPU.

APU controller decoding benefits from the higher clock frequencies possible inside the hard core. This results in a minimum of latency overhead in the decode stage, improving overall performance. The APU controller can decode two types of FCM instructions: pre-defined instructions that are hard coded in the APU controller and a limited number of user-defined instructions.

FCM decoding, although slower than its counterpart, allows many more user-defined instructions to be implemented.

There are also several signals sent to the FCM when the APU controller decodes an instruction. The first signal is called APUFCMDECODED. When this signal is gated with APUFCMINSTRVALID it indicates the APU controller has notified the CPU of the instruction and the resources required. The signal is especially useful in the situation where the FCM receives an FCM instruction that it cannot execute (for example if only part of the FPU instruction set is implemented). The "Floating Point Instructions" section gives more details on this situation. There is one exception to APUFCMDECODED. Due to the

nature of store instructions, the user should not expect to receive APUFCMDECODED when the APU controller decodes a store. Instead the FCM must decode for a store instruction.

The two other signals sent to the FCM after an APU controller decode are APUFCMDECUDI[0:2] and APUFCMDECUDIVALID. These signals are explained further in the "User-Defined Instruction Decoding" section.

APU Controller Pre-Defined Instruction Decoding

Two types of pre-defined instructions can be decoded by the APU controller: Floating point and FCM Load/Store.

Floating Point Instructions

The APU controller can be enabled to decode all PowerPC floating-point instructions. In addition to this, three groups of floating point instructions can be selectively disabled: the complex arithmetic, conversion, and estimates groups.

Complex Arithmetic Group

- `fdiv`
- `fdivs`
- `fsqrt`
- `fsqrts`
- `fdiv.`
- `fdivs.`
- `fsqrt.`
- `fsqrts.`

Conversion Group

- `fcfid`
- `fctidz`
- `fctiw.`
- `fctiwz.`
- `fctid`
- `fctiw`
- `fctiwz`

Estimates Group

- `fres`
- `fres.`
- `frsqrte`
- `frsqrte.`

The decoded instructions require an FCM floating point unit to be used. FPU instructions that return results to the PowerPC will default to execute as non-autonomous, non-blocking. All other FPU instructions default to execute as autonomous. The user can force FPU instructions to be non-blocking in an APU controller configuration register.

If the FCM is designed such that there are FPU instructions that cannot be executed and are not included in one of the three groups users can selectively disable (listed above), the FCM must make use of the APUFCMDECODED signal. If the FPU receives an instruction it cannot execute and APUFCMDECODED is High (along with APUFCMINSTRVALID), the FPU must generate an exception by setting FCMAPUEXCEPTION High. It is assumed that all FPU store instructions can be executed and therefore the FPU should not expect to receive APUFCMDECODED along with a store instruction. Instead, if an FPU cannot execute an FPU store instruction the FPU is expected to decode the store instruction and generate an exception by setting FCMAPUEXCEPTION High.

Note: While the APU controller decodes these instructions, the FCM has to decode them independently for its own execution. The APU can send the 32-bit instruction, but it cannot tell the FCM which FPU instruction it decoded.

FCM Load/Store Instructions

FCM Load/Store instructions transfer data between the PowerPC processor's data memory system (D-Cache or DSPLB/DSOCM addressable memory) and the FCM. An FCM Load transfers data from a memory location to a destination register in the FCM and vice-versa for an FCM Store. All Load/Store instructions are of indexed format, that is, RA stores the base address and RB the offset. Furthermore, RA can either be the contents of RA or the number 0. R0 cannot be used as RA for loads or stores.

FCM Load/Store should not be confused with user-defined FCM read/write instructions. A user-defined FCM read that transfers data from the PowerPC to the FCM will access data from the PowerPC GPR operand registers not from DSOCM or DSPLB memory. The same is true for a user-defined FCM write instruction.

The FCM Load/Store instructions behave somewhat differently in comparison with other FCM instructions. In a way, they are semi-autonomous because the PowerPC CPU is responsible for performing the necessary memory access involved. That is, the processor pipeline is executing, but it is executing a memory access related to the Load/Store instruction. Since FCM Load/Store instructions can be flushed from the CPU's pipeline, the APU controller is responsible for signaling the FCM when it is safe to commit internal state changes.

For details regarding instruction flushing refer to the "FCM Instruction Flushing" section of this chapter.

Note: While the APU controller decodes Load/Store instructions, the FCM has to decode them independently for its own execution. The APU can send the 32-bit instruction, but it cannot tell the FCM which FPU instruction it decoded.

The extended op-code for Load/Store operations are described in Table 4-3.

Table 4-3: **Load/Store Extended Op-code**

Field	Bit position	Description
U	21	Update: If 1 then load RA with effective address: RA <- (RA|0)+(RB)
W[0:2]	(22,24:25)	0b000 = Byte 0b001 = Half-word 0b010 = Word 0b-11 = Quad-word 0b100 = Double-word 0b101, 0b110 = illegal
L/S	23	0 = Load 1 = Store
-	(26:31)	hard coded 0b001110

APU controller Load/Store instruction decoding can be disabled in the APU controller configuration register.

The PowerPC405 native VMX instructions are a subset of the supported FCM Load/Store instructions.

User-Defined Instruction Decoding

In addition to the pre-defined instructions described previously, the user can also define up to eight custom instructions to be decoded by the APU controller. The instructions conform to the same standard FCM format presented earlier, however, the interpretation of the RA, RB, and RT fields are up to the FCM. The UDI interaction with the PowerPC 405 pipeline is defined in the APU controller UDI configuration registers. When there are user instructions being decoded by the APU, the FCM will receive two important signals (along with the 32-bit instruction). The FCM will receive APUFCMDECUDI[0:2] (the bit encoded UDI register number) and APUFCMDECUDIVALID the same cycle as APUFCMINSTRVALID. APUFCMDECUDI[0:2] indicates which UDI register was just decoded in the APU controller and the APUFCMDECUDIVALID signal tells the FCM this bus is valid. These signals allow the FCM to simply decode 3 bits to determine the instruction instead of decoding all 32 bits of instruction. For details refer to the "APU Controller Configuration" section in this chapter.

FCM Pre-Defined Instruction Decoding

There is one group of pre-defined PowerPC instructions that can be configured to be decoded in the FCM: integer divide instructions.

Integer Divide Instructions

The PowerPC integer divide instruction constitutes a special case. While it would normally be executed in the PowerPC natively (consuming 35 cycles), the APU controller can be configured to give the FCM ownership of decoding and executing integer divide instructions (listed below). See the section "APU Controller Configuration," page 196, for details on enabling the FCM divide.

- divd
- divdo
- divdu
- divduo
- divw
- divw.
- divwo
- divwo.
- divwu
- divwu.
- divwuo
- divwuo.

FCM User-Defined Instruction Decoding

User-defined instructions that are not recognized (i.e., decoded) by the APU controller are passed to the FCM for decoding in fabric logic. While this allows for more custom instructions than the eight APU controller decoded UDIs to be defined, additional instructions come at an execution speed penalty. Decoding in the FCM is not as efficient as in the APU controller.

FCM decoded UDI instructions adhere to the same configuration rules as those decoded by the APU controller.

FCM Exceptions

The FCM can signal an exception (FCMAPUEXCEPTION) to the APU controller while executing store, blocking, or non-blocking instructions. This causes the CPU to flush the instruction from the pipeline, and the APU controller to flush the FCM instruction (see "FCM Instruction Flushing"), and the PowerPC to launch the appropriate exception handler, provided the PowerPC MSR enables APU exceptions (see "Enabling the APU Controller").

To execute the exception routine, the PowerPC saves the return program counter in its SRR0 register and the current value of MSR in the SRR1 register. The exception vector used for FCM exceptions is 0x700. When an exception occurs during the execution of a floating point instruction, bit 12 in the PowerPC ESR register is asserted. For exceptions during all other types of instructions, bit 13 in the ESR is asserted instead.

To return from the exception the FCM must provide the processor some way to strike down the FCMAPUEXCEPTION signal from the exception handler. This could be done using, for example, a UDI or an external DCR bus access.

In the FPU decode section it stated that if an FPU instruction was decoded by the APU controller but cannot be executed in the FPU, the FPU should respond by signaling FCMAPUEXCEPTION. This exception will only be precise if the instruction is blocking, non-blocking, or a store. In this situation the user can force all FPU non-storage instructions to be non-blocking (see "APU Controller Configuration" for more details on how to set this option). If an autonomous or load instruction causes an exception FCMAPUEXCEPTION can be set High, but the exception will not be precise. The CPU will not see the exception signal until the next FCM/FPU instruction is decoded.

CPU Exceptions

An FCM instruction can cause several different CPU exceptions. The possible exceptions are shown in Table 4-4.

Table 4-4: **CPU Exceptions Due to FCM Instructions**

Exception	Interrupt Type	Description
Illegal Instruction	Program	Attempted execution of an instruction not recognized by either the CPU or the FCM. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0700}, ESR[4] set.
Privileged Instruction	Program	Occurs when the processor is operating in the problem state (user mode MSR[17] = 1'b1) and the execution of a privileged instruction is attempted. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0700}, ESR[5] set.
APU Enabled	Program	Occurs when MSR[12] = 1'b1 and the FCM is asserting FCMAPUEXCEPTION. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0700}, ESR[13] set.
FPU Enabled Exception	Program	Occurs when MSR[20] and/or MSR[23] are set and the FPU instruction generated FCMAPUEXCEPTION. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0700}, ESR[12] set.
APU Unavailable	APU Unavailable	Occurs when attempting to execute a UDI while MSR[6] = 1'b0. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0F20}, no ESR bit set.

Table 4-4: **CPU Exceptions Due to FCM Instructions**

Exception	Interrupt Type	Description
FPU Unavailable	FPU Unavailable	Occurs when attempting to execute an FPU instruction while MSR[18] = 1'b0. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0800}, no ESR bit set.
Operand	Alignment	Occurs when there is any misaligned data access by FCM storage instruction. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0600}, DEAR loaded with data address that caused access violation, no ESR bit set.
Unsupported Endian	Alignment	Occurs when an FCM Load/Store attempts access with an endian attribute not supported by hardware (for example using option TrapLE or TrapBE). Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0600}, DEAR loaded with data address that caused access violation, no ESR bit set.
Write Fault	Data Storage	Occurs when any store with an effective address with the WR bit clear and ZPR field != 11 while in the problem state with data translation enabled (MSR[27] = 1'b1). Also occurs when in the supervisor state with data translation enabled (MSR[27] = 1'b1) and any store with an effective address with the WR bit clear and ZPR field other than 11 or 10. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0300}, DEAR loaded with data address that caused access violation, ESR[8] set.
Zone Fault	Data Storage	Occurs when in the problem state with data translation enabled (MSR[27] = 1'b1) and any user-mode Load/Store with an effective address with ZPR field = 00. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0300}, DEAR loaded with data address that caused access violation, ESR[9] set.
U0 Fault	Data Storage	Occurs on a Store with an effective address with the U0 bit set and CCR0[u0ex] = 1. Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 0300}, DEAR loaded with data address that caused access violation, ESR[16] set.
Data TLB Miss	Data TLB Miss	Occurs when desired data storage access (Load/Store) to the effective address does not match any valid TLB entry if the data translation is enabled in the CPU (MSR[27] = 1'b1). Current instruction pointer saved in SRR0, current MSR saved in SRR1, vector to 32'x{EVPR[0:15], 1100}, DEAR loaded with data address that caused access violation, ESR[8] set.

FCM Instruction Flushing

The CPU pipeline can be flushed for various reasons. If an FCM instruction is in the CPU pipeline when this occurs, it is said to be flushed. The APU controller will notify the FCM of a flushed instruction by asserting APUFCMFLUSH. If the FCM instruction is flushed from the CPU pipeline, the FCM must be able to re-issue the same instruction without corrupting its internal state. For each FCM instruction, including blocking non-autonomous instructions, the APU controller signals when the point-of-no-return has been reached (APUFCMWRITEBACKOK asserted), after which the instruction cannot be flushed. The conditions where APUFCMWRITEBACKOK is asserted are as follows:

- The instruction is a non-blocking, multi-cycle operation and is currently in the last cycle of execution (two FCM clock cycles after FCMAPUDONE asserted).
- The instruction is a Blocking or Autonomous multi-cycle in the first cycle of execution (same cycle as APUFCMOPERANDVALID is asserted).
- Executing an FCM Load and the last word is in the PowerPC LoadWB stage.
- Executing an FCM Store with the APU controller configuration register bit StoreWBOK set, and return data has been committed to the PowerPC WriteBack stage.

If the APU controller configuration register bit StoreWBOK is not set, the APUFCMWRITEBACKOK will not be asserted when a Store is executed.

Normally APUFCMWRITEBACKOK is not asserted for store instructions for two reasons:

- In most solutions the FCM will not alter internal registers when executing a store and therefore it does not matter if the instruction is re-issued.
- Store instructions that do not wait for APUFCMWRITEBACKOK have a performance benefit.

However, if the FCM store instruction alters any internal registers and could cause errors if re-issued (for example if the FCM uses a FIFO), the user should set the StoreWBOK bit in the APU controller configuration register. This will force the APU controller to generate the APUFCMWRITEBACKOK signal for all store instructions.

Execution Hazards

The APU controller ensures that there are no data or structural hazards with regard to the PowerPC405 pipeline execution.

FCM internal data hazards such as read-after-write (RAW) and write-after-write (WAW) are eliminated if the designer ensures that all FCM instructions complete in order. This can be done conservatively by asserting FCMAPUDONE only after each instruction has completed. This is, however, incompatible with execution pipelining. A pipelined FCM must handle all possible hazards internally.

APU Controller Configuration

General Configuration Register

The general configuration register defines the APU controller's behavior. The register is 32 bits wide. Individual bits are described in Table 4-5. For reset values, refer to Table 4-11, page 203.

Table 4-5: **APU Controller Configuration Register Bit Description**

Name	Bit	Description
RstUDICfg	0	Reset the UDI configuration registers by loading attribute interface signals (TIEAPUUDI*n*). Reset the APU controller configuration register by loading TIEAPUCONTROL.
-	(1:4)	Not used.
LdStDecDis	5	Disable Load/Store instruction decoding only in APU controller.
UDIDecDis	6	Disable UDI instruction decoding in APU controller. This bit also disables load store instruction decoding.
ForceUDINonB	7	Force all UDI instructions to execute as if Non-Blocking.
FPUDecDis	8	Disable FPU instruction decoding in APU controller.
FPUCArithDis	9	Disable decoding of FPU complex arithmetic instruction group (see "Floating Point Instructions").
FPUConvIDis	10	Disable decoding of FPU conversion instruction group (see "Floating Point Instructions").
FPUEstimIDis	11	Disable decoding of FPU estimation instruction group (see "Floating Point Instructions").
-	(12:14)	Not used.
ForceFPUNonB	15	Force all FPU instructions to execute as if they are non-blocking.
StoreWBOK	16	Enable generation of the APUFCMWRITEBACKOK signal for FCM Store operations (see "FCM Instruction Flushing").
LdStPrivOp	17	Execute Load/Store operations only in priviliged mode.
-	(18:19)	Not used.
ForceAlign	20	Force address alignment for FCM Load/Store data. Forces alignment on the natural boundary depending on the transfer size (word boundary for words, quad word boundary for quad words, etc.) Used for load/store with Update.

Table 4-5: **APU Controller Configuration Register Bit Description *(Cont'd)***

Name	Bit	Description
LETrap	21	Enable little-endian Traps for FCM Load/Store. If the accessed memory is little-endian (APUFCMENDIAN=1), an alignment exception will be cast.
BETrap	22	Enable big-endian Traps for FCM Load/Store. If the accessed memory is big-endian (APUFCMENDIAN=0), an alignment exception will be cast.
BESteer	23	Forces big-endian steering of FCMAPURESULT for FCM Store. PowerPC internally byte-flips little-endian results.
APUDiv	24	Perform PPC integer divide operations in FCM.
-	(25:30)	Not used.
FCMEn	31	Enable FCM usage.

UDI Configuration Registers

The APU controller includes eight UDI configuration registers. This allows the user to define as many custom instructions and have them decoded in the fast APU controller, rather than out in the slower FCM. The 32-bit-wide registers define the PowerPC related behavior of the UDI execution. The individual bits are described in Table 4-6.

Table 4-6: **UDI Configuration Register Bit Description**

Name	Bit	Description

PriOpCodeSel	0	Select primary op-code for instruction:
		`0b0 select 0 (= 0b000000)`
		`0b1 select 4 (= 0b000100)`

ExtOpCode	(1:11)	Extended op-code of instruction.
PrivOp	12	Execute only in priviliged mode.
RaEn	13	Requires operand from GPR(RA).
RbEn	14	Requires operand from GPR(RB).
GPRWrite	15	Write back result to GPR(RT) .
XerOVEn	16	Enable return of overflow status.
XerCAEn	17	Enable return of carry status.
CRFieldEn	(18:20)	Select which field in the PowerPC CR the instruction should affect (only applies to UDI op-codes that can set CR bits, see table Table 4-2, page 188).
-	(21:25)	Hard coded `0b0000`.

Table 4-6: **UDI Configuration Register Bit Description *(Cont'd)***

Name	Bit	Description
Type	(26:27)	Instruction class definition, and reserved DCR use: 0b00 = Blocking 0b01 = Non-blocking 0b10 = Autonomous 0b11 = reserved for UDI register selection for DCR read operations (see "DCR Access to the Configuration Registers").
DCRRegPtr	(28:30)	reserved for DCR UDI register addressing (see "DCR Access to the Configuration Registers")
UDIEn	31	Enable APU controller decoding of this UDI configuration.

The reset value of the individual UDI registers can be defined using attribute inputs to the APU controller. For details see the "APU Controller Attributes" section in this chapter.

DCR Access to the Configuration Registers

The APU controller general configuration register has its own DCR address and can be read and written using normal DCR accesses. Refer to the section "Internal Device Control Register (DCR) Interface" in Chapter 2 for address mapping.

The eight UDI registers share a single DCR address for accessing. A UDI register pointer allows individual access to the different registers.

When performing a DCR write to the UDI configuration register address, the DCRRegPtr field of the write data is used to select which UDI register to write, that is, if DCRRegPtr=3, then the DCR write will affect the configuration register associated with UDI number 3. For this DCR write operation, the Type filed should be one of the following: autonomous, blocking or non-blocking.

A DCR read from the UDI configuration register address uses a 3-bit read pointer register in the APU controller to select which specific UDI configuration to return. This pointer auto-increments after each DCR read operation. To load the read pointer with a specific value, the user must perform a "ghost" write to the UDI configuration DCR address. This write will not affect the contents of any UDI configuration registers, only the read pointer. The data used for a "ghost" write has two significant fields: the Type field and the DCRRegPtr field. All other data fields are ignored. The Type field must be set to 0b11, and the DCRRegPtr should be set to the desired read pointer value. A DCR read performed to the UDI configuration address after such "ghost" write will return the contents of the desired UDI configuration register.

FCM/APU Controller Clocking

The APU controller internally runs at the PowerPC clock speed. It receives as an input the FCM clock, CPMFCMCLK, in order to synchronize signals to/from the FCM. The APU controller will automatically detect the clock ratio between the two clocks. The APU controller to FCM interface clock ratio can be any integer between 1:1 and 16:1. The clocks must be rising-edge aligned. The actual FCM frequency depends on the complexity of the logic designed in the user's FCM.

Interface Definition

The tables below describe all I/O ports related to the APU controller. They connect the APU controller in the PowerPC 405 block to the FCM in the FPGA fabric. The naming convention implies the direction of the data flow: “APUFCM” signifies “from APU controller to FCM”, and “FCMAPU” represents “from FCM to APU controller.”

APU Controller Input Signals

All APU controller input signals should be synchronized on the FCM clock (CPMFCMCLK).

Table 4-7: **FCM Interface Input Signals**

Signal	Function
FCMAPUINSTRACK	Valid instruction decoded in FCM. Must be asserted the first cycle in which FCMAPUDECODEBUSY is Low, after APUFCMINSTRVALID has been asserted. All instruction decode signals from the FCM to APU controller must be valid when asserted. If the instruction is decoded by the APU controller, this signal should remain Low.
FCMAPURESULT[0:31]	FCM execution result being passed to the CPU through the APU controller.
FCMAPUDONE	Indicates the completion of the instruction in the FCM to the APU controller. In the case of an autonomous instruction, FCMAPUDONE simply means that the FCM can receive another instruction.
FCMAPUSLEEPNOTREADY	Indicates to the APU controller that the FCM is still executing. It is used to determine when the CPU is allowed to enter sleep mode.
FCMAPUDECODEBUSY	Allows FCM to do a multi-cycle instruction decode before returning FCMAPUINSTRACK. Two modes: with or without instruction hold. If this signal is Low when APUFCMINSTRVALID asserts, the APUFCMINSTRUCTION data is only valid for that cycle; if, on the other hand, FCMAPUBUSYDECODE is High then APUFCMINSTRUCTION is held until FCMAPUDECODEBUSY is lowered.
FCMAPUDCDGPRWRITE	FCM decoded instruction must write back to the GPR.
FCMAPUDCDRAEN	FCM decoded instruction need data from GPR(Ra).
FCMAPUDCDRBEN	FCM decoded instruction need data from GPR(Rb).
FCMAPUDCDPRIVOP	FCM decoded instruction executes in privileged mode.
FCMAPUDCDFORCEALIGN	FCM decoded load/store instruction with forced address alignment.
FCMAPUDCDXEROVEN	FCM decoded instruction returns overflow status.
FCMAPUDCDXERCAEN	FCM decoded instruction returns carry status.
FCMAPUDCDCREN	FCM decoded instruction sets condition register (CR) bits.
FCMAPUEXECRFIELD[0:2]	FCM decoded instruction selects which of the eight PowerPC CR field to update: 0=CR0, 1=CR1, etc.
FCMAPUDCDLOAD	FCM decoded load instruction.

Table 4-7: **FCM Interface Input Signals *(Cont'd)***

Signal	Function
FCMAPUDCDSTORE	FCM decoded store instruction.
FCMAPUDCDUPDATE	FCM decoded load/store instruction should update Ra with effective address.
FCMAPUDCDLDSTBYTE	FCM decoded load/store instruction does byte transfer.
FCMAPUDCDLDSTHW	FCM decoded load/store instruction does half word transfer.
FCMAPUDCDLDSTWD	FCM decoded load/store instruction does word transfer.
FCMAPUDCDLDSTDW	FCM decoded load/store instruction does double word transfer.
FCMAPUDCDLDSTQW	FCM decoded load/store instruction does quad word transfer.
FCMAPUDCDTRAPLE	FCM decoded load/store instruction will cause alignment exception if the storage endian attribute is 1'b1.
FCMAPUDCDTRAPBE	FCM decoded load/store instruction will cause alignment exception if the storage endian attribute is 1'b0.
FCMAPUDCDFORCEBESTEERING	FCM decoded store instruction will force big-endian steering.
FCMAPUFPUOP	FCM decoded FPU instruction.
FCMAPUEXEBLOCKINGMCO	FCM decoded instruction for multi cycle operation of blocking class.
FCMAPUEXENONBLOCKINGMCO	FCM decoded instruction for multi cycle operation of non-blocking class.
FCMAPULOADWAIT	FCM is not yet ready to receive next load data.
FCMAPURESULTVALID	Values on the FCMAPURESULT[0:31], FCMAPUXEROV, FCMAPUXERCA and FCMAPUCR[0:3] are valid.
FCMAPUXEROV	FCM execution overflow status bit.
FCMAPUXERCA	FCM execution carry status bit.
FCMAPUCR[0:3]	Condition result bits to set in the PowerPC CR field selected by FCMAPUEXECRFIELD: • Bit 0 = set LT-bit, meaning result is less than zero • Bit 1 = set GT-bit, meaning result is greater than 0 • Bit 2 = set EQ-bit, meaning result is zero • Bit 3 = set SO-bit, meaning Summary Overflow
FCMAPUEXCEPTION	FCM generate program exception on the processor (vector 0x0700). Exception must be enabled by processor to trap.
CPMFCMCLK	FCM interface clock. Used for synchronizing FCM signals to/from APU controller. See "CPMFCMCLK (Input, Virtex-4 FX Only)" in Chapter 2.

APU Controller Output Signals

All APU controller output signals are synchronous with the FCM clock (CPMFCMCLK).

Table 4-8: **FCM Interface Output Signals**

Signal	Function
APUFCMINSTRUCTION[0:31]	Instruction being presented to the FCM. Is valid as long as APUFCMINSTRVALID is High.
APUFCMINSTRVALID	This signal is asserted on two conditions: • A valid APU instruction was decoded by the APU controller • An undecoded instruction passed to FCM for decoding The signal will remain High for one FCM clock cycle, unless FCMAPUDECODEBUSY is High when it asserts. In that case it stays High until FCMAPUDECODEBUSY goes Low.
APUFCMRADATA[0:31]	Instruction operand from GPR(RA).
APUFCMRBDATA[0:31]	Instruction operand from GPR(RB).
APUFCMOPERANDVALID	Instruction operand valid.
APUFCMFLUSH	Flush APU instruction in the FCM. If asserted no APUFCMWRITEBACKOK signal will be generated
APUFCMWRITEBACKOK	Safe for FCM to commit internal state change; the APU controller can no longer flush the instruction. In normal cases, this signal is asserted for one FCM clock cycle. In some cases when a non-blocking multi-cycle operation is followed by an autonomous or blocking multi-cycle operation while using a large clock ratio, the signal may be asserted for two back-to-back FCM clock cycles.
APUFCMLOADDATA[0:31]	Data word loaded from storage to the APU register file.
APUFCMLOADDVALID	When asserted the data word on the APUFCMLOADDATA[0:31] data bus is valid.
APUFCMLOADBYTEEN[0:3]	Specifies the valid bytes for the word on the load data bus APUFCMLOADDATA[0:31].
APUFCMENDIAN	When asserted, the load/store instruction being presented to the FCM has true little-endian storage attribute.
APUFCMXERCA	Reflects the XerCA bit used for extended arithmetic.
APUFCMDECODED	Asserted when the APU controller decoded the instruction being sent to the FCM.
APUFCMDECUDI[0:2]	Specifies which UDI the APU controller decoded (binary encoded).
APUFCMDECUDIVALID	Valid signals for APUFCMDECUDI.

APU Controller Attributes

The following input signals are used as reset values for the APU controller configuration registers. The reset values can be over-written using DCR. For details see the "APU Controller Configuration" section in this chapter.

Table 4-9: **APU Controller Attributes**

Attribute Signal	Function
TIEAPUUDI1[0:23]	Reset value for UDI register 1.
TIEAPUUDI2[0:23]	Reset value for UDI register 2.
TIEAPUUDI3[0:23]	Reset value for UDI register 3.
TIEAPUUDI4[0:23]	Reset value for UDI register 4.
TIEAPUUDI5[0:23]	Reset value for UDI register 5.
TIEAPUUDI6[0:23]	Reset value for UDI register 6.
TIEAPUUDI7[0:23]	Reset value for UDI register 7.
TIEAPUUDI8[0:23]	Reset value for UDI register 8.
TIEAPUCONTROL[0:15]	Reset values for the APU control register.

Table 4-10: **Bit Map Between TIEAPUUDI*n* and UDI Configuration Registers**

UDI Configuration Field	TIEAPUUDI Bits
PriOpCodeSel	0
ExtOpCode	(1:11)
PrivOp	12
RaEn	13
RbEn	14
GPRWrite	15
XerOVEn	16
XerCAEn	17
CRFieldEn	(18:20)
Type	(21:22)
UDIEn	23

Table 4-11: **Bit Map Between TIEAPUCONTROL and APU Configuration Register**

APU Controller Configuration Field	TIEAPUCONTROL Bits
LdStDecDis	0
UDIDecDis	1
ForceUDINonB	2
FPUDecDis	3
FPUCArithDis	4
FPUConvIDis	5
FPUEstimIDis	6
ForceFPUNonB	7
StoreWBOK	8
LdStPrivOp	9
ForceAlign	10
LETrap	11
BETrap	12
BESteer	13
APUDiv	14
FCMEn	15

FCM Instruction Execution

There are two distinct classes of instruction execution, non-storage and storage instructions. Each type has different execution options and certain execution restrictions.

FCM Non-storage Instructions

There are a few options that non-storage instructions can take advantage of. The CPU can send RA and RB source data (from the CPU's GPR) along with a carry-in value for extended arithmetic. The FCM can also return RT result to the CPU's GPR along with an overflow bit, a carry bit, and condition record values. The only restriction to these instructions is that any instruction returning a result or status to the CPU must be non-autonomous.

FCM Storage Instructions

FCM storage instructions can be loads or stores of byte, half word, word, double word, or quad word. Additionally loads and stores can be executed with Update (meaning the RA, base address, will be updated with the new effective address). The CPU actually handles the majority of the FCM loads and stores. The CPU calculates the address, retrieves data from the cache or other memory, and then sends the data to the FCM in the case of a load.

For stores, the CPU receives the data from the FCM and puts it in cache or other memory at the address the CPU calculates. The CPU will also take care of any Update on the RA register. Because the CPU is so involved in the execution of FCM storage instructions, the user must make sure the FCM adheres to formats and regulations of these instructions.

A load or store without Update will automatically align to the transfer's natural address boundary. In other words, a word transfer is aligned on a word boundary, a double word on a double word boundary, and so on. Even if the address given is not aligned, the resulting load or store will be aligned on the natural boundary. In the case of multi-word loads, the CPU sends them in address increasing order. For example, if a quad word load starts at address 0, the FCM will receive the word at address 0 first followed by the next three words. The same ordering is expected for multi-word stores. For byte or half word loads, the aligned word is sent to the FCM along with the corresponding byte enables based on the address given. For stores, a half word is expected to be positioned at FCMAPURESULT[16:31] and a byte is expected to be positioned at FCMAPURESULT[24:31]. The CPU uses the address given to store the data in the correct place in memory.

A load or store with Update is required on a byte boundary for byte transfers, half word boundary for half word transfers, and word boundary for any other transfer. If the address for the load/store is not aligned properly it will cause an Alignment Exception in the CPU (see "CPU Exceptions" for more details). To prevent this exception the instruction can use the ForceAlign option. This will force alignment on the transfer's natural boundary (similar to load/store without Update).

FCM Interface Timing Specification

Autonomous Transactions

For an APU controller decoded UDI instruction, the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID, APUFCMDECODED, APUFCMDECUDI[0:2], and APUFCMDECUDIVALID.

In the example shown in Figure 4-3, APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMWRITEBACKOK are seen one cycle after the instruction is presented. These signals can arrive at the same time as the instruction signals or anytime afterward. The timing of these signals depend both on the current situation in the CPU pipeline and the clock ratio between the FCM and CPU. However, in the case of autonomous instructions APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMWRITEBACKOK, they will all arrive during the same cycle. Not shown in this example, APUFCMXERCA will be valid the same cycle as APUFCMOPERANDVALID. This signal indicates the carry in for any extended arithmetic.

After the FCM has received APUFCMWRITEBACKOK it can respond with FCMAPUDONE in the same cycle or anytime afterward. The Figure 4-3 timing diagram assumes the signals from the FCM are coming off of flip-flops, so FCMAPUDONE arrives the cycle after APUFCMWRITEBACKOK.

Also in this example, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if FCMAPUDONE has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

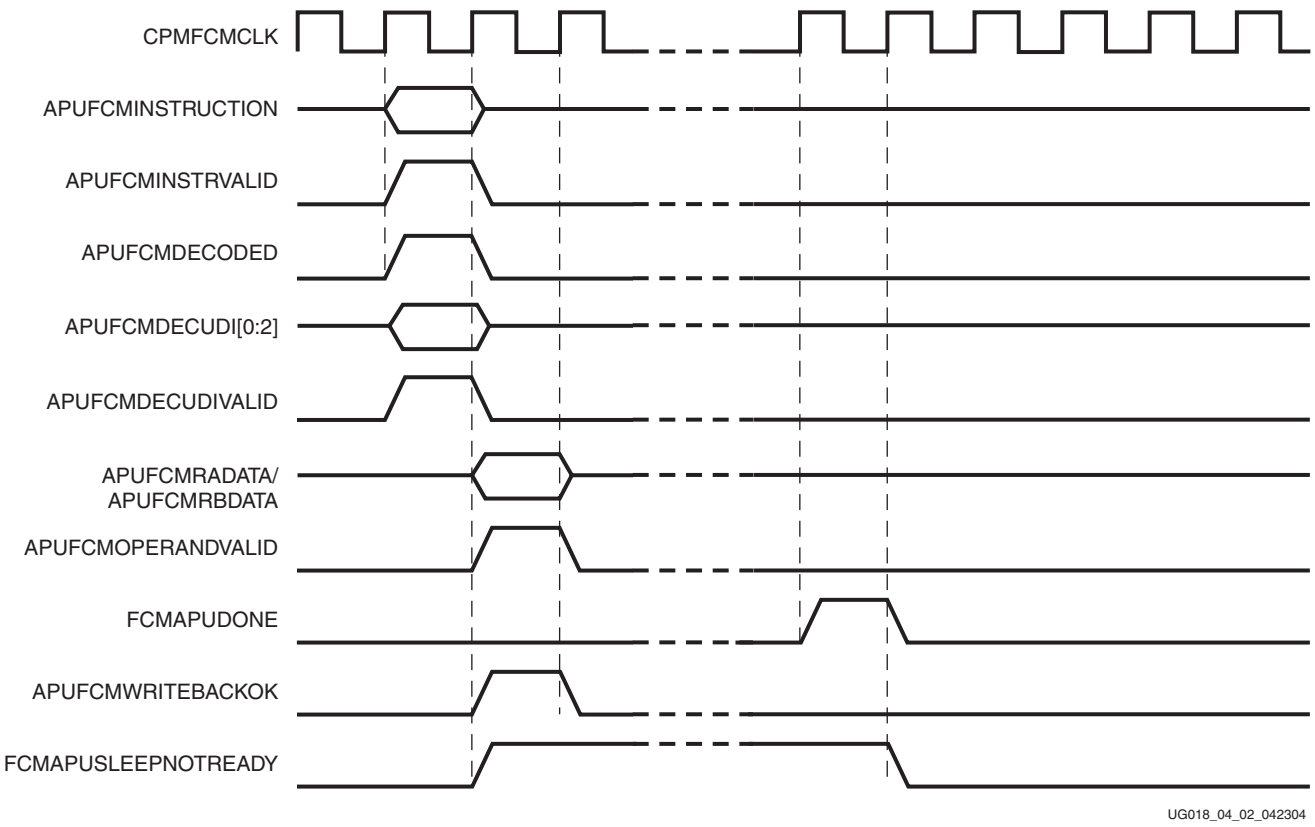


Figure 4-3: **APU Controller Decoded Autonomous Transaction Example**

Note: Actual timing results may vary from those shown in Figure 4-3. For example, the instruction and operands can be valid on the same FCM clock cycle, or they can be many cycles apart.

For an FCM decoded UDI instruction, the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31] and APUFCMINSTRVALID.

In the example shown in Figure 4-4, the FCM responds in the next cycle by setting FCMAPUINSTRACK High and sending any of the FCMAPUOPTIONS that are used by this instruction. To respond in a later cycle, please see Figure 4-12 and Figure 4-13. Table 4-12 shows the FCM signals included in FCMAPUOPTIONAS,

Table 4-12: **FCM Signals Included in FCMAPUOPTIONS**

FCMAPUDCDGPRWRITE	FCMAPUDCDRAEN	FCMAPUDCDRBEN
FCMAPUDCDPRIVOP	FCMAPUDCDFORCEALIGN	FCMAPUDCDXERCAEN
FCMAPUDCDXEROVEN	FCMAPUDCDLOAD	FCMAPUDCDCREN
FCMAPUEXECRFIELD[0:2]	FCMAPUDCDLDSTBYTE	FCMAPUDCDSTORE
FCMAPUDCDUPDATE	FCMAPUDCDLDSTDW	FCMAPUDCDLDSTHW
FCMAPUDCDLDSTWD	FCMAPUDCDTRAPBE	FCMAPUDCDLDSTQW
FCMAPUDCDTRAPLE	FCMAPUEXEBLOCKINGMCO	FCMAPUDCDFORCEBESTEERING
FCMAPUFPUOP	FCMAPUEXENONBLOCKINGMCO	

In the example shown in Figure 4-4, APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMWRITEBACKOK are seen two cycles after the FCM acknowledges the instruction. This is the earliest cycle these signals can be sent to the FCM, but they can also arrive anytime afterward. For autonomous instructions, the signals APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMWRITEBACKOK will all arrive during the same cycle. Not shown in this example, APUFCMXERCA will be valid the same cycle as APUFCMOPERANDVALID. This signal indicates the carry in for any extended arithmetic.

After the FCM has received APUFCMWRITEBACKOK, it can respond with FCMAPUDONE in the same cycle or anytime afterward. This timing diagram assumes the signals from the FCM are coming off of flip-flops, so FCMAPUDONE arrives the cycle after APUFCMWRITEBACKOK.

In the example FCMAPUSLEEPNOTREADY is set High the same cycle the instruction acknowledge is sent. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if FCMAPUDONE has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

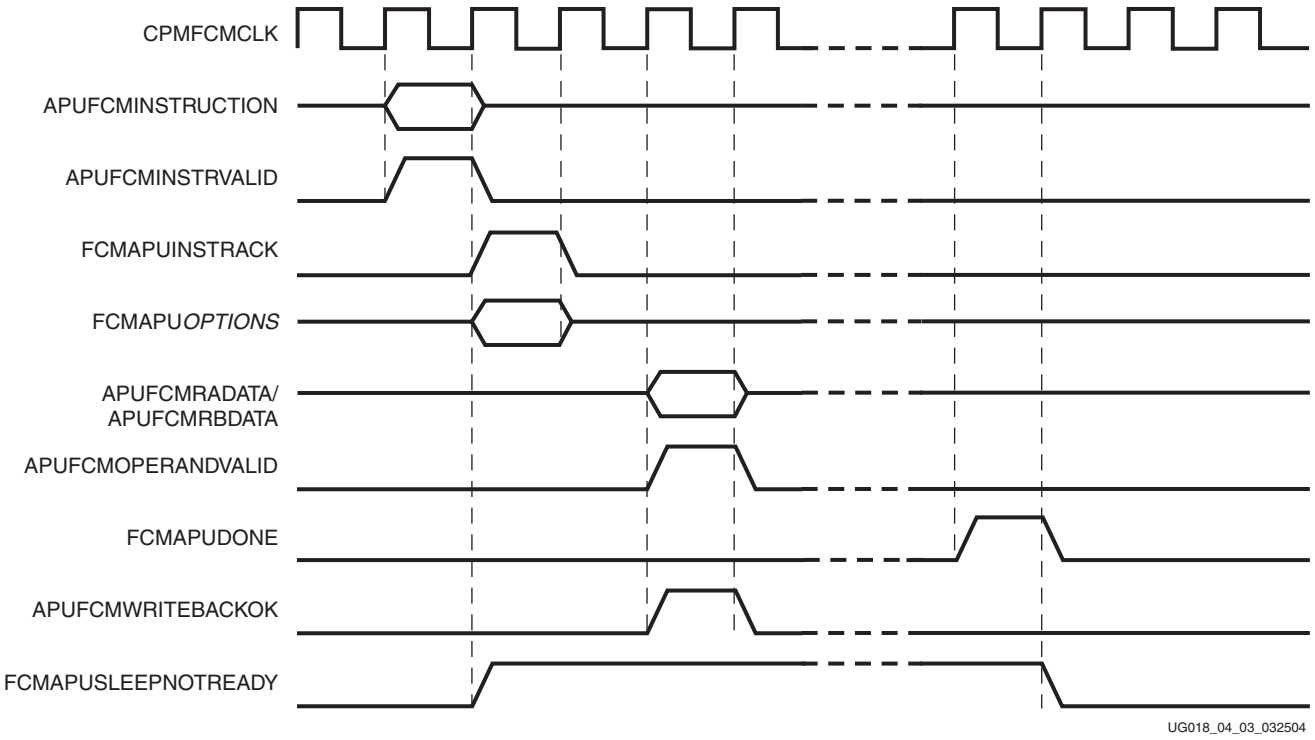


Figure 4-4: **FCM Decoded Autonomous Transaction Example**

Note: Actual timing results may vary from those shown in Figure 4-4. For example, the operands could come later than shown.

Blocking Transactions

For an FCM decoded UDI instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID.

In the example shown in Figure 4-5, the FCM responds in the next cycle by setting FCMAPUINSTRACK High and sending any of the FCMAPUOPTIONS that are used by this instruction. To respond in a later cycle please see Figure 4-12 and Figure 4-13.

In this example APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMWRITEBACKOK are seen two cycles after the FCM acknowledges the instruction. This is the earliest cycle they can be sent to the FCM but can also arrive anytime afterward. In the case of blocking instructions APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMWRITEBACKOK will all arrive during the same cycle. It is not shown in this example, but APUFCMXERCA will be valid the same cycle as APUFCMOPERANDVALID. This signal indicates the carry in for any extended arithmetic.

After the FCM has received APUFCMWRITEBACKOK it can respond with FCMAPURESULT[0:31], FCMAPURESULTVALID, and FCMAPUDONE in the same cycle or anytime after that. The timing diagram in Figure 4-5 assumes the signals from the FCM are coming off of flip-flops, so FCMAPURESULTVALID, etc. arrive the cycle after APUFCMWRITEBACKOK. FCMAPURESULT[0:31] and FCMAPURESULTVALID must arrive the same cycle. FCMAPUDONE can arrive during the same cycle as these or any

cycle later. FCMAPUXEROV, FCMAPUXERCA, and FCMAPUCR[0:3] are not shown in this example, but should be sent at the same cycle as FCMAPURESULT[0:31] and FCMAPURESULTVALID.

In the example shown in Figure 4-5, FCMAPUSLEEPNOTREADY is set High the same cycle the instruction acknowledge is sent. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if FCMAPUDONE has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

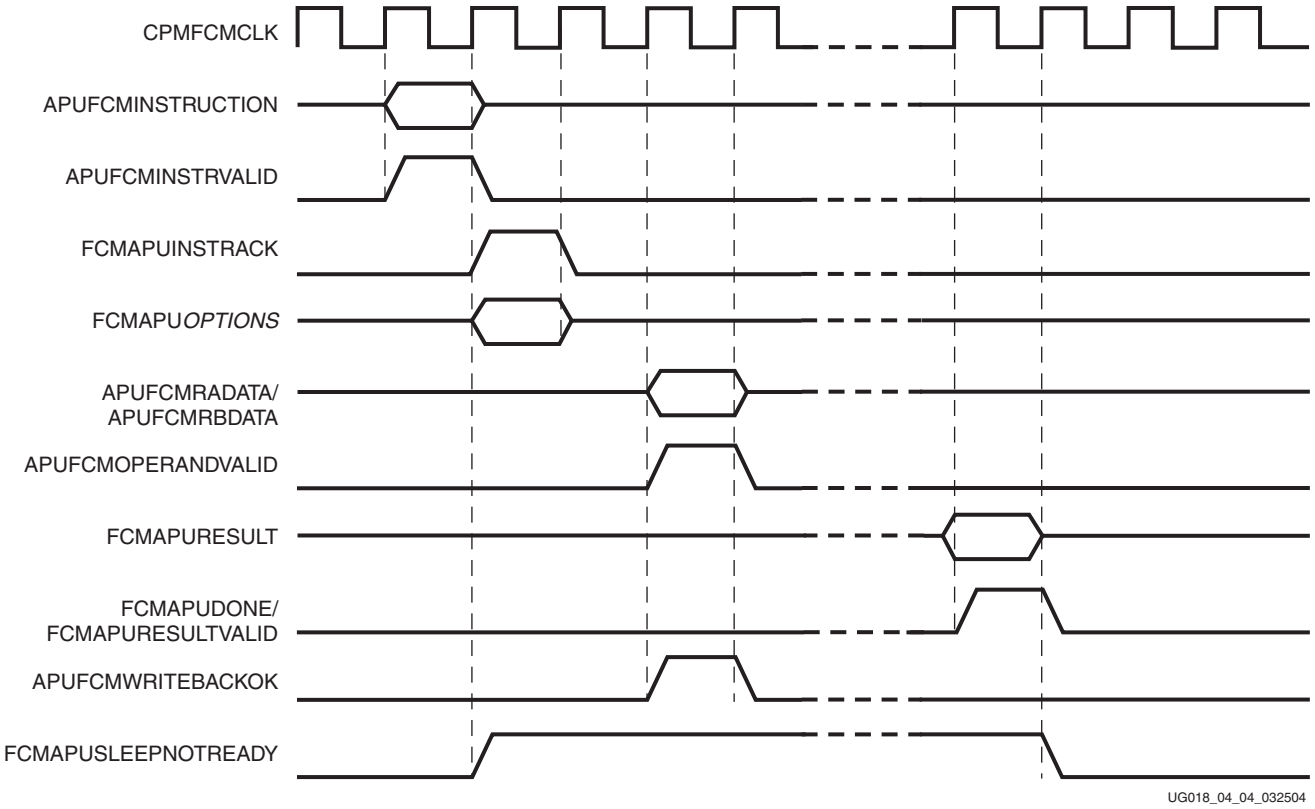


Figure 4-5: **FCM Decoded Blocking Transaction Example**

Note: Actual timing results may vary from those shown in Figure 4-5. For example, the operands could come later than shown.

Non-Blocking Transactions

For an APU controller decoded UDI instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID, and APUFCMDECODED (APUFCMDECUDI[0:2] and APUFCMDECUDIVALID are not shown in Figure 4-6, but they will also arrive the same cycle).

In the example shown in Figure 4-6, APUFCMRADATA/APUFCMRBDATA and APUFCMOPERANDVALID are seen one cycle after the instruction is presented. These signals can arrive at the same time as the instruction signals or anytime afterward. The timing of these signals depend both on the current situation in the CPU pipeline and the clock ratio between the FCM and CPU. However, in the case of non-blocking instructions APUFCMRADATA/APUFCMRBDATA and APUFCMOPERANDVALID will all arrive during the same cycle. It is not shown in this example, but APUFCMXERCA will be valid

the same cycle as APUFCMOPERANDVALID. This signal indicates the carry in for any extended arithmetic.

After the FCM has received APUFCMOPERANDVALID along with the operands it can respond with FCMAPURESULT[0:31], FCMAPURESULTVALID, and FCMAPUDONE in the same cycle or anytime afterward. FCMAPURESULT[0:31] and FCMAPURESULTVALID must arrive the same cycle. FCMAPUDONE can arrive during the same cycle or any cycle later. FCMAPUXEROV, FCMAPUXERCA, and FCMAPUCR[0:3] are not shown in this example, but should be sent at the same cycle as FCMAPURESULT[0:31] and FCMAPURESULTVALID.

APUFCMWRITEBACKOK is shown as arriving two cycles after FCMAPUDONE. This is the earliest cycle it can arrive.

In the example shown in Figure 4-6, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM. The signal should not be lowered until APUFCMWRITEBACKOK has arrived but could remain High afterward.

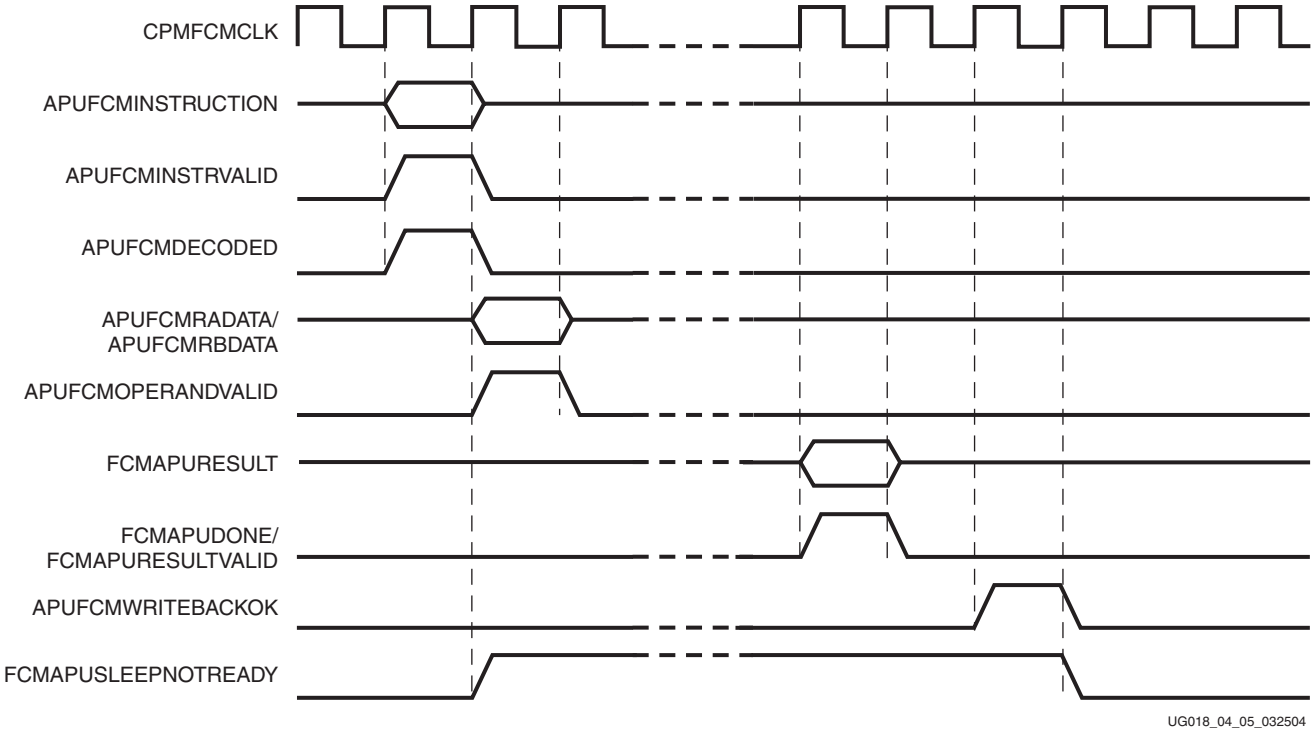


Figure 4-6: **APU Controller Decoded Non-Blocking Transaction Example**

Note: Actual timing results may vary from those shown in Figure 4-6. For example, the operands could come later than shown.

FCM Load Instruction

For an APU controller decoded load instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID, and APUFCMDECODED.

In the example shown in Figure 4-7 (load word), APUFCMLOADDATA[0:31], APUFCMLOADDVALID, and APUFCMWRITEBACKOK are seen one cycle after the instruction is presented. These signals can arrive at the same time as the instruction signals or anytime afterward. The timing of these signals depend both on the current situation in the CPU pipeline and the clock ratio between the FCM and CPU. However, in the case of load byte, load half word, and load word instructions APUFCMAPUFCMLOADDATA[0:31], APUFCMLOADDVALID, and APUFCMWRITEBACKOK will all arrive during the same cycle. For multi-word loads APUFCMWRITEBACKOK will arrive no later than the final word. APUFCMLOADBYTEEN[0:3] is not shown in this example. It will be valid the same cycle as APUFCMLOADDVALID. The byte enables map to the load data bus in the following manner:

APUFCMLOADBYTEEN[0] = APUFCMAPUFCMLOADDATA[0:7]

APUFCMLOADBYTEEN[1] = APUFCMAPUFCMLOADDATA[8:15]

APUFCMLOADBYTEEN[2] = APUFCMAPUFCMLOADDATA[16:23]

APUFCMLOADBYTEEN[3] = APUFCMAPUFCMLOADDATA[24:31]

It is not shown in this example, but APUFCMENDIAN will also be valid the same cycle as APUFCMLOADDVALID.

After the FCM has received APUFCMWRITEBACKOK along with the final word, the FCM can respond with FCMAPUDONE in the same cycle or anytime afterward.

In the example shown in Figure 4-7, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if FCMAPUDONE has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

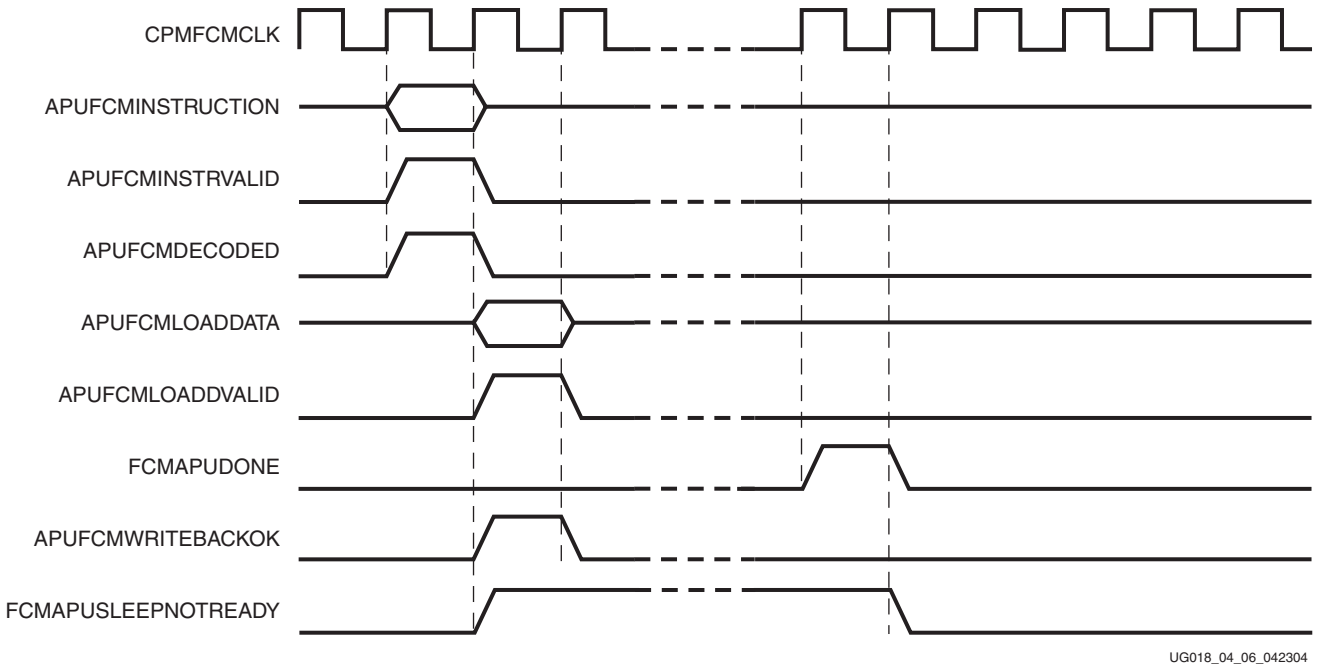


Figure 4-7: **APU Controller Decoded Load Instruction Example**

Note: Load data can arrive at the same time as the instruction or at a later clock cycle than shown in Figure 4-7.

The example shown in Figure 4-8 is similar to Figure 4-7 with two exceptions. This example shows two words being transferred and the FCM is using FCMAPULOADWAIT.

In Figure 4-7, the FCM does not have enough room to accept the second load word and therefore uses FCMAPULOADWAIT. If FCMAPULOADWAIT is set High APUFCMLOADDATA[0:31] and APUFCMLOADDVALID will remain asserted until FCMAPULOADWAIT is deasserted. It should be noted that the second word may arrive later than shown. Multi-words are not necessarily sent back-to-back.

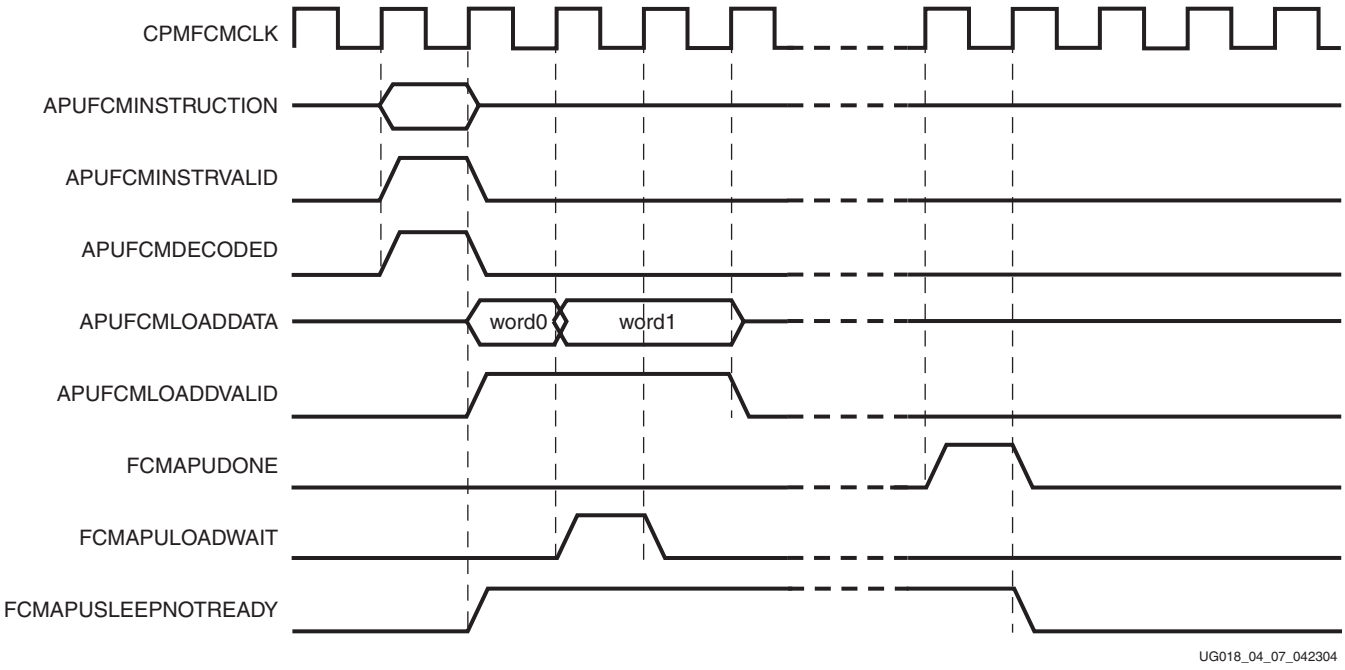


Figure 4-8: **APU Controller Decoded a Double Word Load Instruction with LoadWait Example**

Note: Load data can arrive at the same time as the instruction or at a later clock cycle than shown in Figure 4-8. Also, load data might not be sent back-to-back. Users should look at the valid signal.

FCM Store Instruction

For an APU controller decoded store instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31] and APUFCMINSTRVALID. APUFCMDECODED is shown in Figure 4-9, but the FCM should not expect this signal to arrive with every APU controller decoded store.

After the FCM has received APUFCMINSTRVALID along with the instruction, it can respond with FCMAPURESULT[0:31], FCMAPURESULTVALID, and FCMAPUDONE in the next cycle or anytime afterward. FCMAPURESULT[0:31] and FCMAPURESULTVALID must arrive the same cycle for each word. The words do not have to arrive back-to-back. FCMAPUDONE can arrive during the same cycle as the last word or any cycle later.

APUFCMWRITEBACKOK is not sent in this example. Please see Figure 4-10 for the case when StoreWBOK has been enabled.

In the example shown in Figure 4-9, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if FCMAPUDONE has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

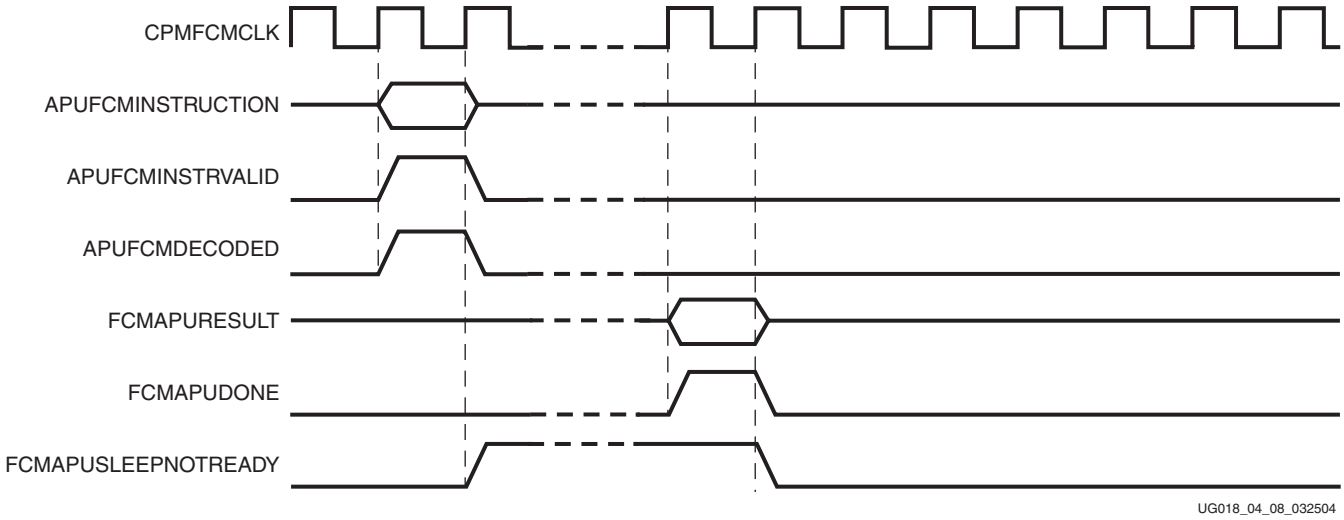


Figure 4-9: **APU Controller Decoded Store Instruction**

The example in Figure 4-10 is very similar to Figure 4-9 except the APU controller has the StoreWBOK bit set to 1. This means the FCM will expect to receive APUFCMWRITEBACKOK after the store instruction is completed. As shown APUFCMWRITEBACKOK will arrive some time after FCMAPUDONE. The timing of these signals depend both on the current situation in the CPU pipeline and the clock ratio between the FCM and CPU.

In the example in Figure 4-10, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM. The signal should not be lowered until APUFCMWRITEBACKOK has arrived but could remain High afterward.

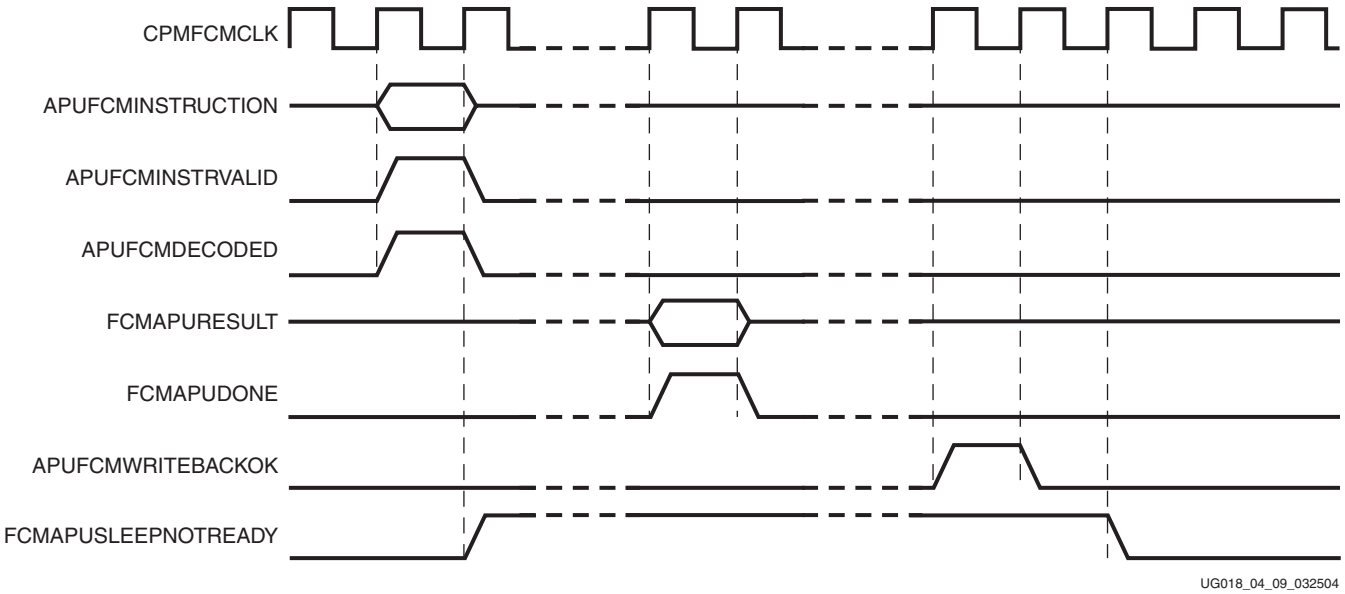


Figure 4-10: **APU Controller Decoded Store Instruction with StoreWBOK=1**

FCM Exception

In the example in Figure 4-11, an FCM instruction has caused an exception in the FCM. As stated in the "FCM Exceptions" section, only blocking and non-blocking instructions will produce precise exceptions.

In this example FCMAPUEXCEPTION is set High one cycle after APUFCMOPERANDVALID. The signal can also be set High the cycle after the instruction is received or any later time (but obviously before FCMAPUDONE is sent). This signal must remain High until it is reset by software.

APUFCMFLUSH will be sent to the FCM after the CPU has seen the exception. This indicates the FCM instruction has been flushed from the CPU pipeline.

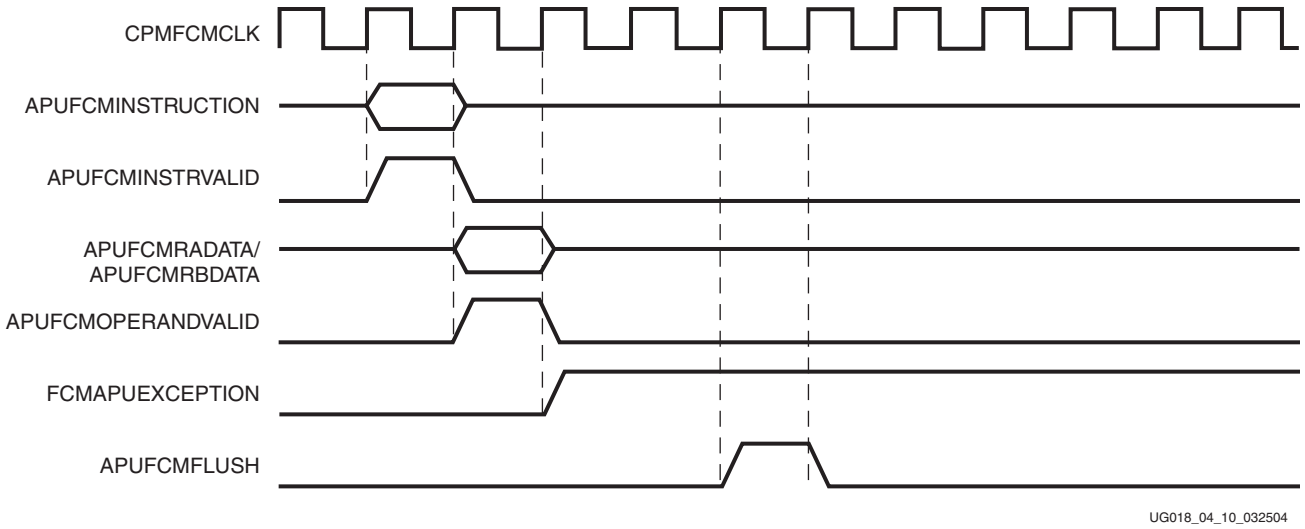


Figure 4-11: **FCM Exception**

Note: FCMAPUEXEPTION may be sent at any time during the execution of a non-autonomous instruction.

FCM Decoding Using Decode Busy Signal

The example in Figure 4-12 only shows the decode part of an FCM decoded instruction using DecodeBusy. In this example, the FCM asserts FCMAPUDECODEBUSY the cycle after it receives the instruction signals. APUFCMINSTRUCTION[0:31] and APUFCMINSTRVALID will not be valid after their one cycle in this case.

The same cycle FCMAPUDECODEBUSY is removed the FCM must send FCMAPUINSTRACK along with any FCMAPUOPTIONS.

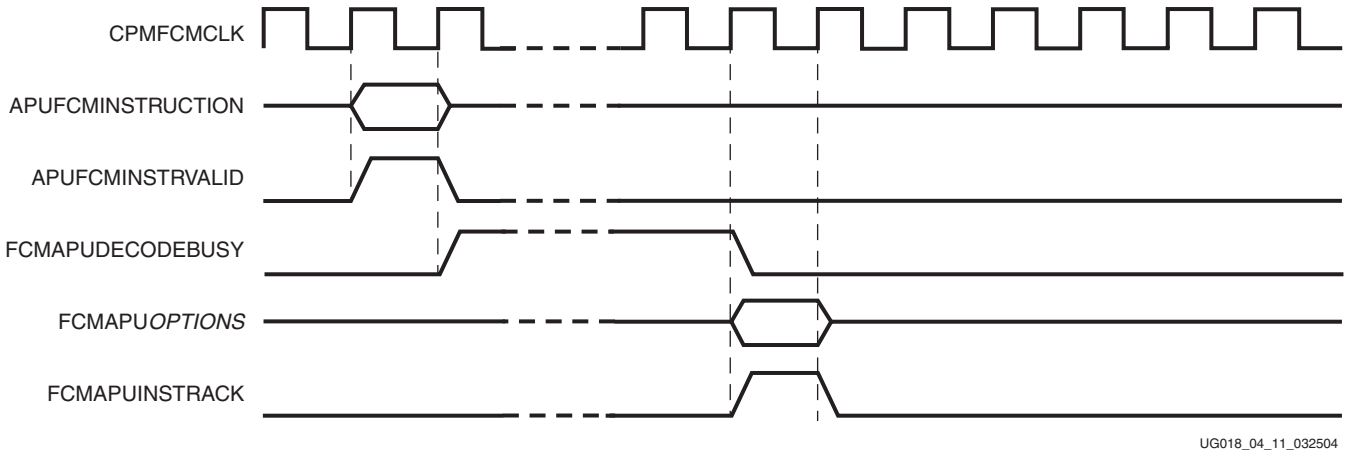


Figure 4-12: **FCM Decode Asserting DecodeBusy**

The example in Figure 4-13 only shows the decode part of an FCM decoded instruction using DecodeBusy. In this example, FCMAPUDECODEBUSY is asserted before it receives

the instruction signals. APUFCMINSTRUCTION[0:31] and APUFCMINSTRVALID will remain valid until FCMAPUDECODEBUSY is deasserted in this case.

The cycle FCMAPUDECODEBUSY is pulsed Low the FCM must send FCMAPUINSTRACK along with any FCMAPUOPTIONS.

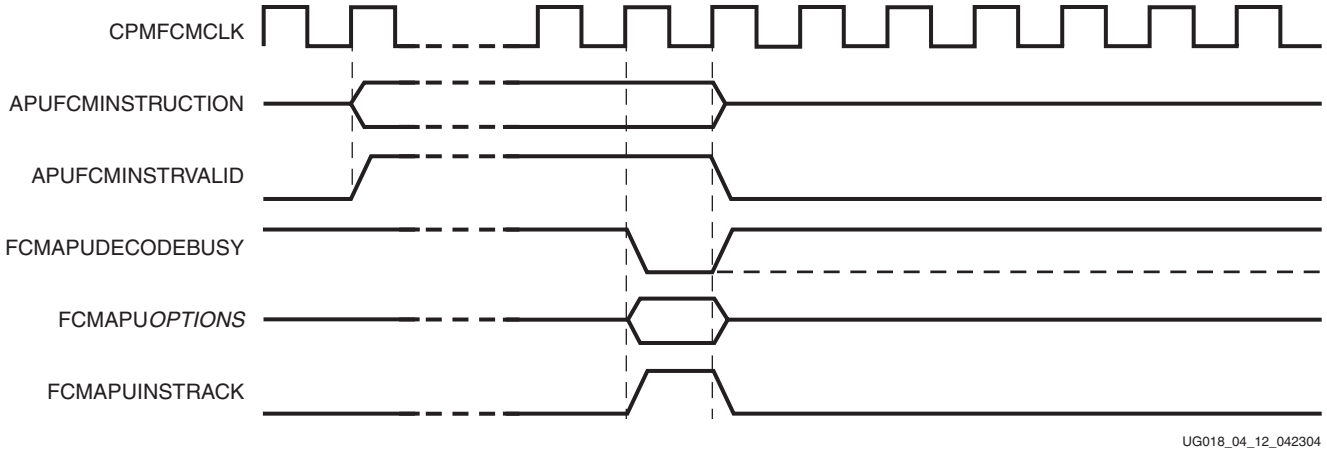


Figure 4-13: **FCM Deasserting DecodeBusy**

Flushed FCM Instructions

The example in Figure 4-14 is very similar to Figure 4-3. The major difference is that this FCM instruction is flushed.

For an APU controller decoded UDI instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID, APUFCMDECODED, APUFCMDECUDI[0:2], APUFCMDECUDIVALID.

In the example in Figure 4-14, APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMXERCA do not arrive until several cycles later. These signals can arrive at the same time as the instruction signals or anytime afterward. The timing of these signals depend both on the current situation in the CPU pipeline and the clock ratio between the FCM and CPU.

APUFCMFLUSH also arrives the same cycle as the operand signals. In the case of an autonomous or blocking instruction, this is the last cycle APUFCMFLUSH can be sent. It could also be sent at the same cycle as APUFCMINSTRVALID. APUFCMWRITEBACKOK will remain Low. Once APUFCMFLUSH has been sent to the FCM, the FCM cannot update its internal registers and must be able to re-execute the same instruction at a later time.

In the example in Figure 4-14, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if

APUFCMFLUSH has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

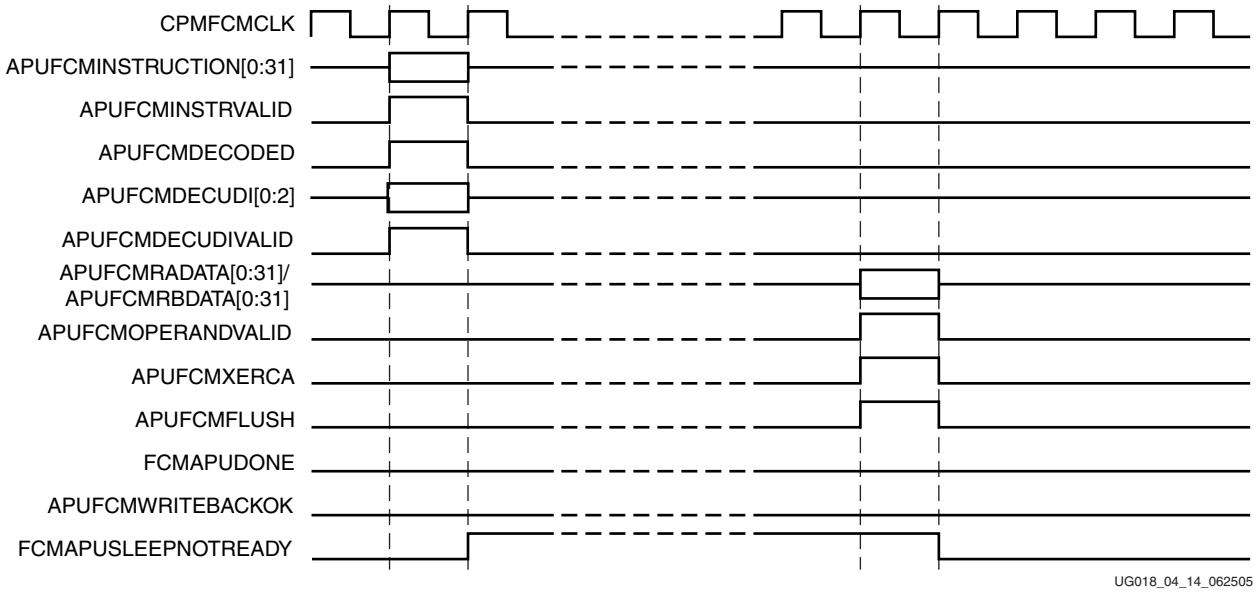


Figure 4-14: **APU Controller Decoded Autonomous Transaction with Flush**

The example in Figure 4-15 is very similar to Figure 4-6. The major difference is that this FCM instruction is flushed.

For an APU controller decoded UDI instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID, APUFCMDECODED, APUFCMDECUDI[0:2], APUFCMDECUDIVALID.

In the example in Figure 4-15, APUFCMRADATA/APUFCMRBDATA, APUFCMOPERANDVALID, and APUFCMXERCA are seen one cycle after the instruction is presented. These signals can arrive at the same time as the instruction signals or anytime afterward.

After the FCM has received APUFCMOPERANDVALID along with the operands it can respond with FCMAPURESULT[0:31], FCMAPURESULTVALID, FCMAPUXEROV, FCMAPUXERCA, FCMAPUCR[0:3], and FCMAPUDONE in the next cycle or anytime afterward.

At this point in the example, APUFCMFLUSH is sent to the FCM. This signal can arrive anytime from the same cycle as the instruction until after FCMAPUDONE has been sent. You will notice that APUFCMWRITEBACKOK remains Low. Once the FCM sees that APUFCMFLUSH has been asserted it cannot update internal registers and must be able to re-execute the same instruction at a later time.

In the example in Figure 4-15, FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM. The signal should not be lowered until APUFCMWRITEBACKOK or APUFCMFLUSH has arrived but could remain High afterward.

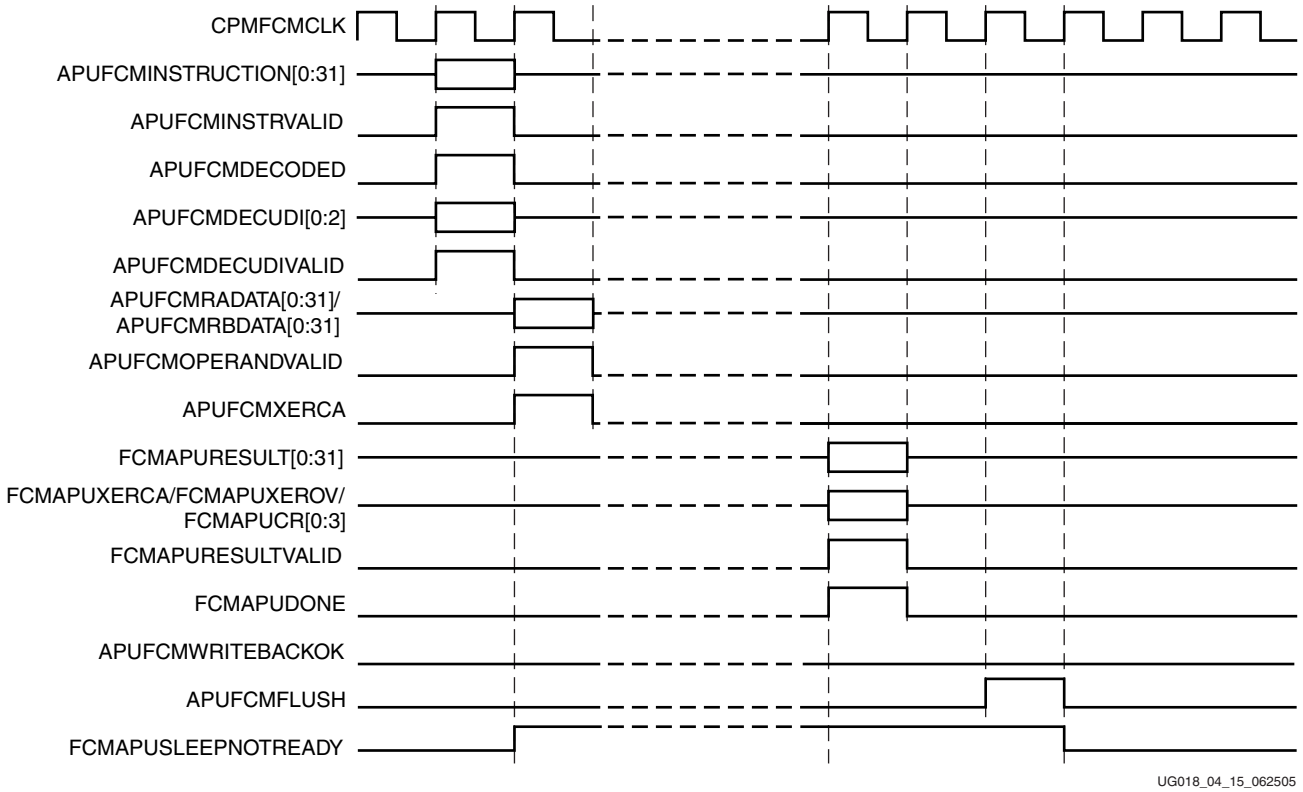


Figure 4-15: **APU Controller Decoded Non-blocking Transaction with Flush**

For an APU controller decoded load instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31], APUFCMINSTRVALID, and APUFCMDECODED.

In the example in Figure 4-16 (load quad word), APUFCMLOADDATA[0:31], APUFCMLOADDVALID, and APUFCMLOADBYTEEN[0:3] are seen two cycles after the instruction is presented. These signals can arrive at the same time as the instruction signals or anytime afterward. The timing of these signals depend both on the current situation in the CPU pipeline and the clock ratio between the FCM and CPU.

In the example in Figure 4-16, APUFCMFLUSH is asserted when the second of the four words is sent to the FCM. At this point, the FCM will not receive more words. The FCM cannot update its internal registers and must be able to re-execute the quad word load at a later time. APUFCMWRITEBACKOK remains Low.

FCMAPUSLEEPNOTREADY is set High the cycle after the instruction signals are received. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM (even if APUFCMFLUSH has been sent). FCMAPUSLEEPNOTREADY will not prevent a new instruction from being sent to the FCM.

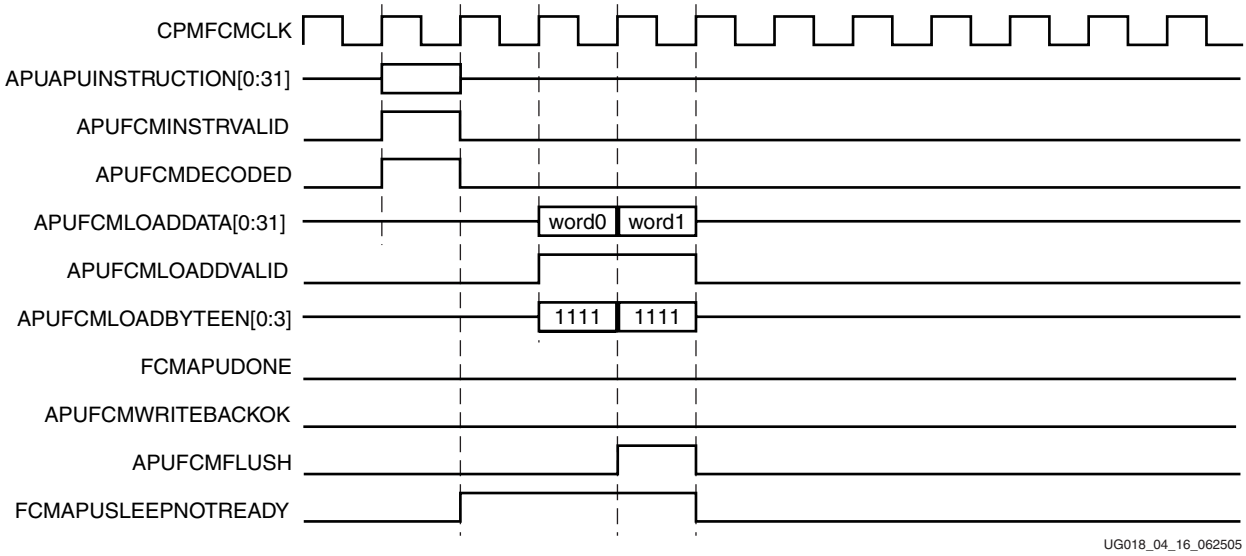


Figure 4-16: **APU Controller Decoded Quad Word Load Transaction with Flush**

For an FCM decoded store instruction the following signals will always arrive at the same cycle: APUFCMINSTRUCTION[0:31] and APUFCMINSTRVALID. In the example in Figure 4-17, the FCM uses FCMAPUDECODEBUSY before responding with FCMAPUINSTRACK and the execution options.

After the FCM has sent FCMAPUINSTRACK it can send FCMAPURESULT[0:31] and FCMAPURESULTVALID for the first word in the next cycle or anytime after that. FCMAPURESULT[0:31] and FCMAPURESULTVALID must arrive the same cycle for each word. The words do not have to arrive back-to-back as shown in this example. FCMAPUDONE can arrive during the same cycle as the last word or any cycle later.

Because the APU controller has the StoreWBOK bit set to 1, the FCM must wait for APUFCMWRITEBACKOK. However, in this case the store was flushed from the CPU pipeline and APUFCMFLUSH is sent. The FCM cannot update its internal registers and must be able to re-execute the store at a later time.

In the example in Figure 4-17, FCMAPUSLEEPNOTREADY is set High the same cycle FCMAPUINSTRACK is sent. This is the latest cycle the signal should go High and it should remain High until the instruction is completely finished in the FCM. The signal should not be lowered until APUFCMWRITEBACKOK or APUFCMFLUSH has arrived but could remain High afterward.

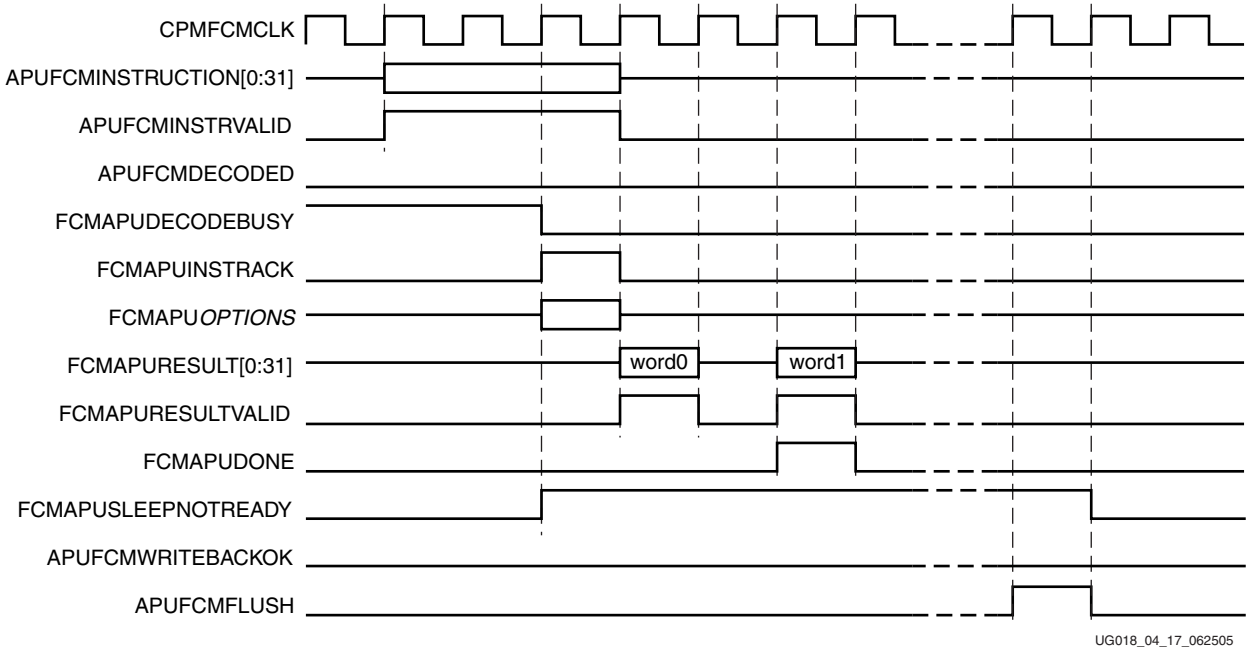


Figure 4-17: **FCM Decoded Double Word Store Transaction with Flush and StoreWBOK = 1**

Appendix A

RISCWatch and RISCTrace Interfaces

This appendix summarizes the interface requirements between the PowerPC 405 processor 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 PowerPC 405 processor 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 PowerPC 405 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.

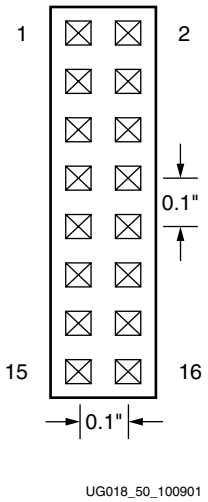


Figure A-1: **JTAG-Connector Physical Layout**

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	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	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 10 KΩ 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 (1 KΩ or less) should be used to provide short-circuit current-limiting protection.
3. A 10 KΩ pull-up resistor must be connected to these signals to ensure proper chip operation when these inputs are not used.

Table A-2: **PowerPC 405 to RISCWatch Signal Mapping**

PowerPC 405		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	TRST	Output	4	21
JTGC405TCK	Input	TCK	Output	7	15
JTGC405TMS	Input	TMS	Output	9	17
DBGC405DEBUGHALT(2)	Input	HALT	Output	11	7

Notes:

1. This signal must be driven by a 3-state device using C405JTGTDOEN as the enable signal.
2. This signal must be inverted between the PowerPC 405 and the RISCWatch.

RISCTrace Interface

The RISCTrace tool communicates with the PowerPC 405 processor 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 PowerPC 405 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.

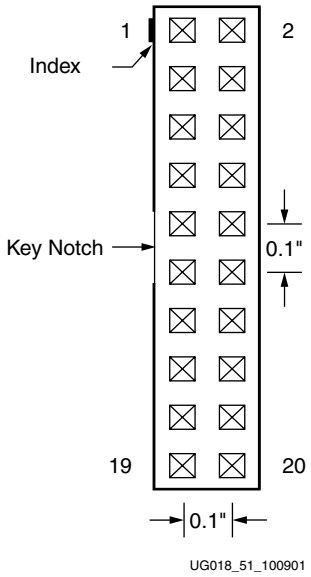


Figure A-2: **Trace-Connector Physical Layout**

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.
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: **PowerPC 405 to RISCTrace Signal Mapping**

PowerPC 405		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 PowerPC 405 interface signals in alphabetical order. A cross reference is provided to each signal description. The signal naming conventions used are described in "Signal Naming Conventions" in Chapter 2.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order**

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
APUFCMDECODED	Virtex-4	O	FCM	No Connect	Indicates APU Controller decoded FCM instruction.
APUFCMDECUDI[0:2]	Virtex-4	O	FCM	No Connect	Indicates which UDI is decoded (binary encoded).
APUFCMDECUDIVALID	Virtex-4	O	FCM	No Connect	Valid signals for APUFCMDECUDI.
APUFCMENDIAN	Virtex-4	O	FCM	No Connect	Indicates load/store instruction has true little-endian storage attribute.
APUFCMFLUSH	Virtex-4	O	FCM	No Connect	Flush APU instruction in the FCM.
APUFCMINSTRUCTION[0:31]	Virtex-4	O	FCM	No Connect	Instruction being presented to the FCM.
APUFCMINSTRVALID	Virtex-4	O	FCM	No Connect	Valid APU instruction decoded by APU Controller, or instruction passed to FCM for decoding.
APUFCMLOADBYTEEN[0:3])	Virtex-4	O	FCM	No Connect	Specifies the valid bytes for the word on APUFCMLOADDATA.
APUFCMLOADDATA[0:31]	Virtex-4	O	FCM	No Connect	Data word loaded from storage to the APU register file.
APUFCMLOADDVALID	Virtex-4	O	FCM	No Connect	Data valid signal for APUFCMLOADDATA.
APUFCMOPERANDVALID	Virtex-4	O	FCM	No Connect	Instruction operand valid.
APUFCMRADATA[0:31]	Virtex-4	O	FCM	No Connect	Instruction operand from GPR(RA).
APUFCMRBDATA[0:31]	Virtex-4	O	FCM	No Connect	Instruction operand from GPR(RB).
APUFCMWRITEBACKOK	Virtex-4	O	FCM	No Connect	Safe for FCM to commit internal state change.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
APUFCMXERCA	Virtex-4	O	FCM	No Connect	Reflects the XerCA bit used for extended arithmetic.
BRAMDSOCMCLK	Virtex-II Pro and Virtex-4	I	DSOCM	1	Clocks the DSOCM controller and the data side interface logic
BRAMDSOCMRDDBUS[0:31]	Virtex-II Pro and Virtex-4	I	DSOCM	0	Read data bus from the FPGA fabric to the DSOCM controller.
BRAMISOCMCLK	Virtex-II Pro and Virtex-4	I	ISOCM	1	Clocks the ISOCM controller and the instruction side memory located in the FPGA fabric.
BRAMISOCMDCRRDDBUS[0:31]	Virtex-4	I	ISOCM	0	Read data from block RAM to ISOCM controller using a DCR-based access.
BRAMISOCMRDDBUS[0:63]	Virtex-II Pro and Virtex-4	I	ISOCM	0	Read data from block RAM to the ISOCM controller
C405CPMCORESLEEPREQ	Virtex-II Pro and Virtex-4	O	CPM	No Connect	Indicates the core is requesting to be put into sleep mode.
C405CPMMSRCE	Virtex-II Pro and Virtex-4	O	CPM	No Connect	Indicates the value of MSR[CE].
C405CPMMSREE	Virtex-II Pro and Virtex-4	O	CPM	No Connect	Indicates the value of MSR[EE].
C405CPMTIMERIRQ	Virtex-II Pro and Virtex-4	O	CPM	No Connect	Indicates a timer-interrupt request occurred.
C405CPMTIMERRESETREQ	Virtex-II Pro and Virtex-4	O	CPM	No Connect	Indicates a watchdog-timer reset request occurred.
C405DBGMSRWE	Virtex-II Pro and Virtex-4	O	Debug	No Connect	Indicates the value of MSR[WE].
C405DBGSTOPACK	Virtex-II Pro and Virtex-4	O	Debug	No Connect	Indicates the PowerPC 405 is in debug halt mode.
C405DBGWBCOMPLETE	Virtex-II Pro and Virtex-4	O	Debug	No Connect	Indicates the current instruction in the PowerPC 405 writeback pipeline stage is completing.
C405DBGWBFULL	Virtex-II Pro and Virtex-4	O	Debug	No Connect	Indicates the PowerPC 405 writeback pipeline stage is full.
C405DBGWBIAR[0:29]	Virtex-II Pro and Virtex-4	O	Debug	No Connect	The address of the current instruction in the PowerPC 405 writeback pipeline stage.
C405DCRABUS[0:9] EXTDCRABUS[0:9]	Virtex-II Pro and Virtex-4	O	DCR	No Connect	Specifies the address of the DCR access request.
C405DCRDBUSOUT[0:31] EXTDCRDBUSOUT[0:31]	Virtex-II Pro and Virtex-4	O	DCR	No Connect or attach to input bus	The 32-bit DCR write-data bus.
C405DCRREAD EXTDCRREAD	Virtex-II Pro and Virtex-4	O	DCR	No Connect	Indicates a DCR read request occurred.
C405DCRWRITE EXTDCRWRITE	Virtex-II Pro and Virtex-4	O	DCR	No Connect	Indicates a DCR write request occurred.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
C405JTGCAPTUREDR (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	Indicates the TAP controller is in the capture-DR state.
C405JTGEXTEST (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	Indicates the JTAG EXTEST instruction is selected.
C405JTGPGMOUT (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	Indicates the state of a general purpose program bit in the JTAG debug control register (JDCR).
C405JTGSHIFTDR (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	Indicates the TAP controller is in the shift-DR state.
C405JTGTDO (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	JTAG TDO (test-data out).
C405JTGTDOEN (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	Indicates the JTAG TDO signal is enabled.
C405JTGUPDATEDR (OUTPUT)	Virtex-II Pro and Virtex-4	O	JTAG	No Connect	Indicates the TAP controller is in the update-DR state.
C405DBGLOADDATAONAPUDBUS	Virtex-4	O	DBG	No Connect	Valid load data from PowerPC 405 core to APU Controller
C405PLBDCUABORT	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the DCU is aborting an unacknowledged data-access request.
C405PLBDCUABUS[0:31]	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Specifies the memory address of the data-access request.
C405PLBDCUBE[0:7]	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Specifies which bytes are transferred during single-word transfers.
C405PLBDCUCACHEABLE	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the value of the cacheability storage attribute for the target address.
C405PLBDCUGUARDED	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the value of the guarded storage attribute for the target address.
C405PLBDCUPRIORITY[0:1]	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the priority of the data-access request.
C405PLBDCUREQUEST	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the DCU is making a data-access request.
C405PLBDCURNW	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Specifies whether the data-access request is a read or a write.
C405PLBDCUSIZE2	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Specifies a single word or eight-word transfer size.
C405PLBDCUU0ATTR	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the value of the user-defined storage attribute for the target address.
C405PLBDCUWRDBUS[0:63]	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	The DCU write-data bus used to transfer data from the DCU to the PLB slave.
C405PLBDCUWRITETHRU	Virtex-II Pro and Virtex-4	O	DSPLB	No Connect	Indicates the value of the write-through storage attribute for the target address.
C405PLBICUABORT	Virtex-II Pro and Virtex-4	O	ISPLB	No Connect	Indicates the ICU is aborting an unacknowledged fetch request.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
C405PLBICUABUS[0:29]	Virtex-II Pro and Virtex-4	O	ISPLB	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	Virtex-II Pro and Virtex-4	O	ISPLB	No Connect	Indicates the value of the cacheability storage attribute for the target address.
C405PLBICUPRIORITY[0:1]	Virtex-II Pro and Virtex-4	O	ISPLB	No Connect	Indicates the priority of the ICU fetch request.
C405PLBICUREQUEST	Virtex-II Pro and Virtex-4	O	ISPLB	No Connect	Indicates the ICU is making an instruction-fetch request.
C405PLBICUSIZE[2:3]	Virtex-II Pro and Virtex-4	O	ISPLB	No Connect	Specifies a four word or eight word line-transfer size.
C405PLBICUU0ATTR	Virtex-II Pro and Virtex-4	O	ISPLB	No Connect	Indicates the value of the user-defined storage attribute for the target address.
C405RSTCHIPRESETREQ	Virtex-II Pro and Virtex-4	O	Reset	Required	Indicates a chip-reset request occurred.
C405RSTCORERESETREQ	Virtex-II Pro and Virtex-4	O	Reset	Required	Indicates a core-reset request occurred.
C405RSTSYSRESETREQ	Virtex-II Pro and Virtex-4	O	Reset	Required	Indicates a system-reset request occurred.
C405TRCCYCLE	Virtex-II Pro and Virtex-4	O	Trace	No Connect	Specifies the trace cycle.
C405TRCEVENEXECUTIONSTATUS[0:1]	Virtex-II Pro and Virtex-4	O	Trace	No Connect	Specifies the execution status collected during the first of two processor cycles.
C405TRCODDEXECUTIONSTATUS[0:1]	Virtex-II Pro and Virtex-4	O	Trace	No Connect	Specifies the execution status collected during the second of two processor cycles.
C405TRCTRACESTATUS[0:3]	Virtex-II Pro and Virtex-4	O	Trace	No Connect	Specifies the trace status.
C405TRCTRIGGEREVENTOUT	Virtex-II Pro and Virtex-4	O	Trace	Wrap to Trigger Event In	Indicates a trigger event occurred.
C405TRCTRIGGEREVENTTYPE[0:10]	Virtex-II Pro and Virtex-4Virtex-4	O	Trace	No Connect	Specifies which debug event caused the trigger event.
C405XXXMACHINECHECK	Virtex-II Pro and Virtex-4	O	Control	No Connect	Indicates a machine-check error has been detected by the PowerPC 405.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
CPMC405CLOCK	Virtex-II Pro and Virtex-4	I	CPM	1 Required	PowerPC 405 clock input (for all non-JTAG logic, including timers).
CPMC405CORECLKINACTIVE	Virtex-II Pro and Virtex-4	I	CPM	0	Indicates the CPM logic disabled the clocks to the core.
CPMC405CPUCLKEN	Virtex-II Pro and Virtex-4	I	CPM	1	Enables the core clock zone.
CPMC405JTAGCLKEN	Virtex-II Pro and Virtex-4	I	CPM	1	Enables the JTAG clock zone.
CPMC405SYNCBYPASS	Virtex-4	I	CPM	1	Bypass PLB re-synchronization for Virtex-II Pro compatibility.
CPMC405TIMERCLKEN	Virtex-II Pro and Virtex-4	I	CPM	1	Enables the timer clock zone.
CPMC405TIMERTICK	Virtex-II Pro and Virtex-4	I	CPM	1	Increments or decrements the PowerPC 405 timers every time it is active with the CPMC405CLOCK.
CPMDCRCLK	Virtex-4	I	CPM	1	DCR bus interface clock for PPC405 synchronization.
CPMFCMCLK	Virtex-4	I	CPM	1	FCM interface clock for the APU Controller.
DBGC405DEBUGHALT	Virtex-II Pro and Virtex-4	I	Debug	0	Indicates the external debug logic is placing the processor in debug halt mode.
DBGC405EXTBUSHOLDACK	Virtex-II Pro and Virtex-4	I	Debug	0	Indicates the bus controller has given control of the bus to an external master.
DBGC405UNCONDDEBUGEVENT	Virtex-II Pro and Virtex-4	I	Debug	0	Indicates the external debug logic is causing an unconditional debug event.
DCRC405ACK EXTDCRACK	Virtex-II Pro and Virtex-4	I	DCR	0	Indicates a DCR access has been completed by a peripheral.
DCRC405DBUSIN[0:31] EXTDCRDBUSIN[0:31]	Virtex-II Pro and Virtex-4	I	DCR	0 or attach to output bus	The 32-bit DCR read-data bus.
DCREMACENABLER	Virtex-4	O	EMAC	No Connect	Always asserted. Used to enable the DCR host bus interface in the embedded Tri-mode Ethernet MAC. Connect to the DCREMACENABLE input of the EMAC.
DSARCVALUE[0:7]	Virtex-II Pro and Virtex-4	I	DSOCM	0	Power-on base address for the data-side on-chip memory
DSCNTLVALUE[0:7]	Virtex-II Pro and Virtex-4	I	DSOCM	Bit 3=1 All others=0	Power-on configuration of the DSOCM controller
DSOCMBRAMABUS[8:29]	Virtex-II Pro and Virtex-4	O	DSOCM	No Connect	Address from the DSOCM controller to FPGA fabric
DSOCMBRAMBYTEWRITE[0:3]	Virtex-II Pro and Virtex-4	O	DSOCM	No Connect	Indicates a write access

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
DSOCMBRAMEN	Virtex-II Pro and Virtex-4	O	DSOCM	No Connect	Block RAM enable signal asserted on accesses
DSOCMBRAMWRDBUS[0:31]	Virtex-II Pro and Virtex-4	O	DSOCM	No Connect	Write data from DSOCM to the data-side memory interface
DSOCMBUSY	Virtex-II Pro and Virtex-4	O	DSOCM	No Connect	Value of the DSOCM DCR control register DSCNTL[2] bit
DSOCMWRADDRVALID	Virtex-4	O	DSOCM	No Connect	The signal indicates a valid read access and read address
DSOCMRWCOMPLETE	Virtex-4	I	DSOCM	0	Indicates that a read access or a write access is complete
DSOCMWRADDRVALID	Virtex-4	O	DSOCM	No Connect	The signal indicates a write and that write address is valid
EICC405CRITINPUTIRQ	Virtex-II Pro and Virtex-4	I	EIC	0	Indicates an external critical interrupt occurred.
EICC405EXTINPUTIRQ	Virtex-II Pro and Virtex-4	I	EIC	0	Indicates an external noncritical interrupt occurred.
FCMAPUCR[0:3]	Virtex-4	I	FCM	0	Condition result bits to set in the PowerPC CR field
FCMAPUDCDCREN	Virtex-4	I	FCM	0	FCM decoded instruction sets condition register (CR) bits.
FCMAPUDCDFORCEALIGN	Virtex-4	I	FCM	0	FCM decoded load/store instruction with forced word alignment
FCMAPUDCDFORCEBESTEERING	Virtex-4	I	FCM	0	FCM decoded store instruction will force Big-Endian steering.
FCMAPUDCDGPRWRITE	Virtex-4	I	FCM	0	FCM decoded instruction must write back to the GPR.
FCMAPUDCDLDSTBYTE	Virtex-4	I	FCM	0	FCM decoded load/store instruction does byte transfer.
FCMAPUDCDLDSTDW	Virtex-4	I	FCM	0	FCM decoded load/store instruction does double word transfer.
FCMAPUDCDLDSTHW	Virtex-4	I	FCM	0	FCM decoded load/store instruction does half word transfer.
FCMAPUDCDLDSTQW	Virtex-4	I	FCM	0	FCM decoded load/store instruction does quad word transfer.
FCMAPUDCDLDSTWD	Virtex-4	I	FCM	0	FCM decoded load/store instruction does word transfer.
FCMAPUDCDLOAD	Virtex-4	I	FCM	0	FCM decoded load instruction.
FCMAPUDCDPRIVOP	Virtex-4	I	FCM	0	FCM decoded instruction executes in privileged mode.
FCMAPUDCDRAEN	Virtex-4	I	FCM	0	FCM decoded instruction need data from GPR(Ra).
FCMAPUDCDRBEN	Virtex-4	I	FCM	0	FCM decoded instruction need data from GPR(Rb).
FCMAPUDCDSTORE	Virtex-4	I	FCM	0	FCM decoded store instruction.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
FCMAPUDCDTRAPBE	Virtex-4	I	FCM	0	FCM decoded load/store instruction will cause alignment exception if the storage Endian attribute is 1'b0.
FCMAPUDCDTRAPLE	Virtex-4	I	FCM	0	FCM decoded load/store instruction will cause alignment exception if the storage Endian attribute is 1'b1.
FCMAPUDCDUPDATE	Virtex-4	I	FCM	0	FCM decoded load/store instruction should update Ra with effective address.
FCMAPUDCDXERCAEN	Virtex-4	I	FCM	0	FCM decoded instruction returns carry status.
FCMAPUDCDXEROVEN	Virtex-4	I	FCM	0	FCM decoded instruction returns overflow status.
FCMAPUDECODEBUSY	Virtex-4	I	FCM	0	Allows FCM to do a multi-cycle instruction decode
FCMAPUDONE	Virtex-4	I	FCM	0	Indicates the completion of the instruction in the FCM to the APU Controller
FCMAPUEXCEPTION	Virtex-4	I	FCM	0	FCM generate program exception on the processor (vector 0x0700).
FCMAPUEXEBLOCKINGMCO	Virtex-4	I	FCM	0	FCM decoded multi cycle operation of blocking class.
FCMAPUEXECRFIELD[0:2]	Virtex-4	I	FCM	0	FCM decoded instruction selects which of the eight PowerPC CR
FCMAPUEXENONBLOCKINGMCO	Virtex-4	I	FCM	0	FCM decoded multi cycle operation of non-blocking class.
FCMAPUFPUOP	Virtex-4	I	FCM	0	FCM decoded FPU instruction.
FCMAPUINSTRACK	Virtex-4	I	FCM	0	Valid instruction decoded in FCM
FCMAPULOADWAIT	Virtex-4	I	FCM	0	FCM is not yet ready to receive next load data.
FCMAPURESULT[0:31]	Virtex-4	I	FCM	0	FCM execution result passed to the CPU
FCMAPURESULTVALID	Virtex-4	I	FCM	0	Values on the FCMAPURESULT[0:31], FCMAPUXEROV, FCMAPUXERCA and FCMAPUCR[0:3] are valid.
FCMAPUSLEEPNOTREADY	Virtex-4	I	FCM	0	Indicates to the APU Controller that the FCM is still executing
FCMAPUXERCA	Virtex-4	I	FCM	0	FCM carry status bit.
FCMAPUXEROV	Virtex-4	I	FCM	0	FCM overflow status bit.
ISARCVALUE[0:7]	Virtex-II Pro and Virtex-4	I	ISOCM	0	Power-on base address for the instruction-side on-chip memory
ISCNTLVALUE[0:7]	Virtex-II Pro and Virtex-4	I	ISOCM	Bit 3=1 All others=0	Power-on configuration of the ISOCM controller
ISOCMBRAMEN	Virtex-II Pro and Virtex-4	O	ISOCM	No Connect	Block RAM read enable from the ISOCM controller

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
ISOCMDCRBRAMEVENEN	Virtex-4	O	ISOCM	No Connect	Even word write enable to block RAM via a DCR-based access
ISOCMDCRBRAMODDEN	Virtex-4	O	ISOCM	No Connect	Odd word write enable to block RAM via a DCR-based access
ISOCMBRAMRDABUS[8:28]	Virtex-II Pro and Virtex-4	O	ISOCM	No Connect	Read address from ISOCM to block RAM
ISOCMBRAMWRABUS[8:28]	Virtex-II Pro and Virtex-4	O	ISOCM	No Connect	Write address from the ISOCM to block RAM via a DCR-based access
ISOCMBRAMWRDBUS[0:31]	Virtex-II Pro and Virtex-4	O	ISOCM	No Connect	Write data from the ISOCM to block RAM via a DCR-based access
ISOCMDCRBRAMEVENEN	Virtex-4	O	ISOCM	No Connect	Block RAM enable (even bank) for a DCR-based access
ISOCMDCRBRAMODDEN	Virtex-4	O	ISOCM	No Connect	Block RAM enable (odd bank) for a DCR-based access
ISOCMDCRBRAMRDSELECT	Virtex-4	O	ISOCM	No Connect	Select between even and odd instruction words from DCR access
JTGC405BNDSCANTDO (INPUT)	Virtex-II Pro and Virtex-4	I	JTAG	0	JTAG boundary scan input from the previous boundary scan element TDO output.
JTGC405TCK (INPUT)	Virtex-II Pro and Virtex-4	I	JTAG	1 (See IEEE 1149.1)	JTAG TCK (test clock).
JTGC405TDI (INPUT)	Virtex-II Pro and Virtex-4	I	JTAG	1	JTAG TDI (test-data in).
JTGC405TMS (INPUT)	Virtex-II Pro and Virtex-4	I	JTAG	1	JTAG TMS (test-mode select).
JTGC405TRSTNEG (INPUT)	Virtex-II Pro and Virtex-4	I	Reset	1 Required	Performs a JTAG test reset (TRST).
JTGC405TRSTNEG (INPUT)	Virtex-II Pro and Virtex-4	I	JTAG	1 Required	JTAG $\overline{\text{TRST}}$ (test reset).
MCBCPUCLKEN (INPUT)	Virtex-II Pro and Virtex-4	I	FPGA	1	Indicates the PowerPC 405 clock enable should follow GWE during a partial reconfiguration.
MCBJTAGEN (INPUT)	Virtex-II Pro and Virtex-4	I	FPGA	1	Indicates the JTAG clock enable should follow GWE during a partial reconfiguration.
MCBTIMEREN (INPUT)	Virtex-II Pro and Virtex-4	I	FPGA	1	Indicates the timer clock enable should follow GWE during a partial reconfiguration.
MCPPCRST (INPUT)	Virtex-II Pro and Virtex-4	I	FPGA	1	Indicates the PowerPC 405 should be reset when GSR is asserted during a partial reconfiguration.
PLBC405DCUADDRACK (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	Indicates a PLB slave acknowledges the current data-access request.
PLBC405DCUBUSY (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	Indicates the PLB slave is busy performing an operation requested by the DCU.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
PLBC405DCUERR (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	Indicates an error was detected by the PLB slave during the transfer of data to or from the DCU.
PLBC405DCURDDACK	Virtex-II Pro and Virtex-4	I	DSPLB	0	Indicates the DCU read-data bus contains valid data for transfer to the DCU.
PLBC405DCURDDBUS[0:63] (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	The DCU read-data bus used to transfer data from the PLB slave to the DCU.
PLBC405DCURDWDADDR[1:3] (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	Indicates which word or doubleword of an eight-word line transfer is present on the DCU read-data bus.
PLBC405DCUSSIZE1 (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	Specifies the bus width (size) of the PLB slave that accepted the request.
PLBC405DCUWRDACK (INPUT)	Virtex-II Pro and Virtex-4	I	DSPLB	0	Indicates the data on the DCU write-data bus is being accepted by the PLB slave.
PLBC405ICUADDRACK (INPUT)	Virtex-II Pro and Virtex-4	I	ISPLB	0	Indicates a PLB slave acknowledges the current ICU fetch request.
PLBC405ICUBUSY (INPUT)	Virtex-II Pro and Virtex-4	I	ISPLB	0	Indicates the PLB slave is busy performing an operation requested by the ICU.
PLBC405ICUERR (INPUT)	Virtex-II Pro and Virtex-4	I	ISPLB	0	Indicates an error was detected by the PLB slave during the transfer of instructions to the ICU.
PLBC405ICURDDACK (INPUT)	Virtex-II Pro and Virtex-4	I	ISPLB	0	Indicates the ICU read-data bus contains valid instructions for transfer to the ICU.
PLBC405DCURDDBUS[0:63]	Virtex-II Pro and Virtex-4	I	ISPLB		The ICU read-data bus used to transfer instructions from the PLB slave to the ICU.
PLBC405ICURDWDADDR[1:3] (INPUT)	Virtex-II Pro and Virtex-4	I	ISPLB	0	Indicates which word or doubleword of a four-word or eight-word line transfer is present on the ICU read-data bus.
PLBC405ICUSSIZE1 (INPUT)	Virtex-II Pro and Virtex-4	I	ISPLB	0	Specifies the bus width (size) of the PLB slave that accepted the request.
PLBCLK (INPUT)	Virtex-II Pro and Virtex-4	I	FPGA	1 Required	PLB clock.
RSTC405RESETCHIP (INPUT)	Virtex-II Pro and Virtex-4	I	Reset	0 Required	Indicates a chip-reset occurred.
RSTC405RESETCORE (INPUT)	Virtex-II Pro and Virtex-4	I	Reset	0 Required	Resets the PowerPC 405 core logic, data cache, instruction cache, and the on-chip memory controller (OCM).
RSTC405RESETSYS (INPUT)	Virtex-II Pro and Virtex-4	I	Reset	0 Required	Indicates a system-reset occurred. Resets the logic in the PowerPC 405 JTAG unit.

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
TIEAPUCONTROL[0:15]	Virtex-4	I	FCM	0	Reset values for the APU control register.
TIEAPUUDI1[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 1.
TIEAPUUDI2[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 2.
TIEAPUUDI3[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 3.
TIEAPUUDI4[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 4.
TIEAPUUDI5[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 5.
TIEAPUUDI6[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 6.
TIEAPUUDI7[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 7.
TIEAPUUDI8[0:23]	Virtex-4	I	FCM	0	Reset value for UDI register 8.
TIEC405DETERMINISTICMULT (INPUT)	Virtex-II Pro and Virtex-4	I	Control	0 Required	Specifies whether all multiply operations complete in a fixed number of cycles or have an early-out capability.
TIEC405DISOPERANDFWD (INPUT)	Virtex-II Pro and Virtex-4	I	Control	0 Required	Disables operand forwarding for load instructions.
TIEC405MMUEN (INPUT)	Virtex-II Pro and Virtex-4	I	Control	0 Required	Enables the memory-management unit (MMU)
TIEDCRADDR[0:5]	Virtex-4	I	DCR	0	Location of PPC internal DCR registers in DCR address space
TIEDSOCMDCRADDR[0:7]	Virtex-II Pro	I	DSOCM	0	Location of PPC DSOCM DCR registers in DCR address space
TIEISOCMDCRADDR[0:7]	Virtex-II Pro	I	ISOCM	0	Location of PPC ISOCM DCR registers in DCR address space
TIEPVRBIT10	Virtex-4	I	PVR	0	Set bit 10 in Processor Version Register (OWN field)
TIEPVRBIT11	Virtex-4	I	PVR	0	Set bit 11 in Processor Version Register (OWN field)
TIEPVRBIT28	Virtex-4	I	PVR	0	Set bit 28 in Processor Version Register (AID field)
TIEPVRBIT29	Virtex-4	I	PVR	0	Set bit 29 in Processor Version Register (AID field)
TIEPVRBIT30	Virtex-4	I	PVR	0	Set bit 30 in Processor Version Register (AID field)
TIEPVRBIT31	Virtex-4	I	PVR	0	Set bit 31 in Processor Version Register (AID field)
TIEPVRBIT8	Virtex-4	I	PVR	0	Set bit 8 in Processor Version Register (OWN field)
TIEPVRBIT9	Virtex-4	I	PVR	0	Set bit 9 in Processor Version Register (OWN field)

Table B-1: **PowerPC 405 Interface Signals in Alphabetical Order *(Cont'd)***

Signal	FPGA Type	I/O Type	Interface	If Unused Ties To:(1)	Function
TRCC405TRACEDISABLE	Virtex-II Pro and Virtex-4	I	Trace	0	Disables trace collection and broadcast.
TRCC405TRIGGEREVENTIN	Virtex-II Pro and Virtex-4	I	Trace	0 Wrap to Trigger Event Out	Indicates a trigger event occurred and that trace status is to be generated.

Notes:

1. The ISE® design tools assign drivers automatically.

Appendix C

Processor Block Timing Model

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 or the *Virtex-4 Data Sheet* and the Timing Analyzer (TRCE) report from Xilinx software. For specific timing parameter values and clocking considerations, refer to the appropriate data sheet(s).

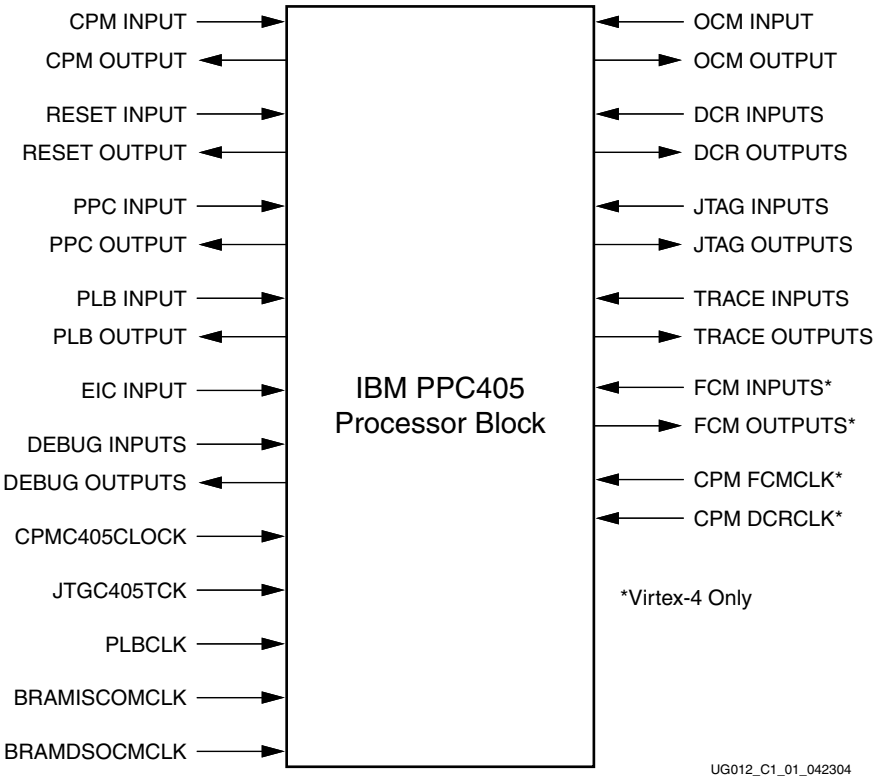


Figure C-1: **PowerPC 405 Processor Block (Simplified)**

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 included in the processor block I/O timing. Signals are characterized with setup and hold times for inputs and clock to valid output times for outputs. Signals are grouped by the interface block from which they originate: 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), Auxiliary Processor Unit Controller (APU, Virtex-4 only), and Fabric Coprocessor Module (FCM, Virtex-4 only).

Table C-1 associates five clocks (Virtex-II Pro) or seven clocks (Virtex-4) 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
CPMDCRCLK (Virtex-4 only)	Device Control Register Bus Clock	EXTDCR
CPMFCMCLK (Virtex-4 only)	Fabric Coprocessor Module Clock	APU, FCM

Timing Parameter Tables and Diagram

The following seven 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.

- "Parameters Relative to the Core Clock (CPMC405CLOCK)", Table C-2, page 241.
- "Parameters Relative to the DCR Bus Clock (CPMDCRCLK, Virtex-4 Only)", Table C-3, page 242.
- "Parameters Relative to the FCM Clock (CPMFCMCLK, Virtex-4 Only)", Table C-4, page 243.
- "Parameters Relative to the PLB Clock (PLBCLK)", Table C-5, page 244.
- "Parameters Relative to the JTAG Clock (JTAGC405TCK)", Table C-6, page 245.
- "Parameters Relative to the ISOCM Clock (BRAMISOCMCLK)", Table C-7, page 245.
- "Parameters Relative to the DSOCM Clock (BRAMDSOCMCLK)", Table C-8, page 246.

Table C-2: **Parameters Relative to the Core Clock (CPMC405CLOCK)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PCCK\_}DCR/T_{PCKC\_}DCR$(1)	Control Inputs	DCRC405ACK
$T_{PDCK\_}DCR/T_{PCKD\_}DCR$(1)	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:		
$T_{PCKCO\_}DCR$(1)	Control Outputs	C405DCRREAD C405DCRWRITE
$T_{PCKAO\_}DCR$(1)	Address Outputs	C405DCRABUS[0:9]
$T_{PCKDO\_}DCR$(1)	Data Outputs	C405DCRDBUSOUT[0:31]
$T_{PCKCO\_}CPM$	Control Outputs	C405CPMMSREE C405CPMMSRCE C405CPMTIMERIRQ C405CPMTIMERRESETREQ C405CPMCORESLEEPREQ
$T_{PCKCO\_}RST$	Control Outputs	C405RSTCHIPRESETREQ C405RSTCORERESETREQ C405RSTSYSRESETREQ
$T_{PCKCO\_}DBG$	Control Outputs	C405DBGMSRWE C405DBGSTOPACK C405DBGWBCOMPLETE C405DBGWBFULL C405DBGWBIAR[0:29]
$T_{PCKCO\_}PPC$	Control Outputs	C405XXXMACHINECHECK

Table C-2: **Parameters Relative to the Core Clock (CPMC405CLOCK) *(Cont'd)***

Parameter	Function	Signals
$T_{PCKCO_}$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

Notes:

1. Virtex-II Pro only. See Table C-3 for Virtex-4 DCR bus timing parameters.

Table C-3: **Parameters Relative to the DCR Bus Clock (CPMDCRCLK, Virtex-4 Only)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PPCDCK_}$EXDCRACK $T_{PPCCKD_}$EXDCRACK	Control Inputs	EXTDCRC405ACK
$T_{PPCDCK_}$EXDCRDBUS $T_{PPCCKD_}$EXCDRDBUS	Data Inputs	EXTDCRC405DBUSIN[0:31]
Clock to Out:		
$T_{PPCCKO_}$EXDCRRD	Control Outputs	EXTDCRREAD
$T_{PPCCKO_}$EXDCRWR		EXTDCRWRITE
$T_{PPCCKO_}$EXDCRABUS	Address Outputs	EXTDCRABUS[0:9]
$T_{PPCCKO_}$EXDCRDBUSO	Data Outputs	EXTDCRDBUSOUT[0:31]
Clock:		
T_{DCRPWH}	Clock Pulse Width, High State	CPMDCRCLK
T_{DCRPWL}	Clock Pulse Width, Low State	CPMCDCRCLK

Table C-4: **Parameters Relative to the FCM Clock (CPMFCMCLK, Virtex-4 Only)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PCCK_}$FCM/$T_{PCKC_}$FCM	Control Inputs	FCMAPUINSTRACK FCMAPUDONE FCMAPUSLEEPNOTREADY FCMAPUDECODEBUSY FCMAPUDCDGPRWRITE FCMAPUDCDRAEN FCMAPUDCDRBEN FCMAPUDCDPRIVOP FCMAPUDCDFORCEALIGN FCMAPUDCDXEROVEN FCMAPUDCDXERCAEN FCMAPUDCDCREN FCMAPUEXECRFIELD[0:2] FCMAPUDCDLOAD FCMAPUDCDSTORE FCMAPUDCDUPDATE FCMAPUDCDLDSTBYTE FCMAPUDCDLDSTHW FCMAPUDCDLDSTWD FCMAPUDCDLDSTDW FCMAPUDCDLDSTQW FCMAPUDCDTRAPLE FCMAPUDCDTRAPBE FCMAPUDCDFORCEBESTEERING FCMAPUFPUOP FCMAPUEXEBLOCKINGMCO FCMAPULOADWAIT FCMAPURESULTVALID FCMAPUXEROV FCMAPUEXENONBLOCKINGMCO FCMAPUXERCA FCMAPUCR[0:3] FCMAPUEXCEPTION
$T_{PDCK_}$FCM/$T_{PCKO_}$FCM	Data Inputs	FCMAPURESULT[0:31]
Clock to Out:		
$T_{PCKCO_}$FCM	Control Outputs	APUFCMINSTRVALID APUFCMOPERANDVALID APUFCMFLUSH APUFCMWRITEBACKOK APUFCMLOADDVALID APUFCMLOADBYTEEN[0:3] APUFCMENDIAN APUFCMXERCA APUFCMDECODED APUFCMDECUDI[0:2] APUFCMDECUDIVALID

Table C-4: **Parameters Relative to the FCM Clock (CPMFCMCLK, Virtex-4 Only) *(Cont'd)***

Parameter	Function	Signals
$T_{PCKDO_}$FCM	Data Outputs	APUFCMINSTRUCTION[0:31] APUFCMRADATA[0:31] APUFCMRBDATA[0:31] APUFCMLOADDATA[0:31]
Clock:		
T_{fcmpwh} and T_{FCMPWL}	Clock High Width Clock Low Width	CPMFCMCLK

Table C-5: **Parameters Relative to the PLB Clock (PLBCLK)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PCCK_}$PLB/$T_{PCKC_}$PLB	Control 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:		
$T_{PCKCO_}$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

Table C-5: **Parameters Relative to the PLB Clock (PLBCLK) *(Cont'd)***

Parameter	Function	Signals
$T_{PCKDO_}$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-6: **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:		
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-7: **Parameters Relative to the ISOCM Clock (BRAMISOCMCLK)**

Parameter	Function	Signals
Setup/Hold:		
$T_{PDCK_}$ISOCM $T_{PCKD_}$ISOCM	Data inputs	BRAMISOCMRDDBUS[0:63]

Table C-7: **Parameters Relative to the ISOCM Clock (BRAMISOCMCLK) *(Cont'd)***

Parameter	Function	Signals
Clock to Out:		
$T_{PCKCO_}$ISOCM	Control outputs	ISOCMBRAMEN ISOCMBRAMODDWRITEEN ISOCMBRAMEVENWRITEEN ISOCMDCRBRAMEVENEN (Virtex-4 only) ISOCMDCRBRAMODDEN (Virtex-4 only) ISOCMDCRBRAMRDSELECT (Virtex-4 only)
$T_{PCKAO_}$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-8: **Parameters Relative to the DSOCM Clock (BRAMDSOCMCLK)**

Parameter	Function	Signals
Setup/Hold:		
TBD Parameter	Control Inputs	DSOCMRDWRCOMPLETE (Virtex-4 only)
$T_{PDCK_}$DSOCM/$T_{PCKD_}$DSOCM	Data inputs	BRAMDSOCMRDDBUS[0:31] BRAMISOCMDCRRDDBUS[0:31] (Virtex-4 only)
Clock to Out:		
$T_{PCKCO_}$DSOCM	Control outputs	DSOCMBRAMEN DSOCMBRAMBYTEWRITE[0:3] DSOCMBUSY DSOCMRDADDRVALID (Virtex-4 only) DSOCMWRADDRVALID (Virtex-4 only)
$T_{PCKDO_}$DSOCM	Data outputs	DSOCMBRAMWRDBUS[0:31]
$T_{PCKAO_}$DSOCM	Address outputs	DSOCMBRAMABUS[8:29]
Clock:		
T_{DPWH}	Clock, Pulse Width High	BRAMDSOCMCLK
T_{DPWL}	Clock, Pulse Width Low	BRAMDSOCMCLK

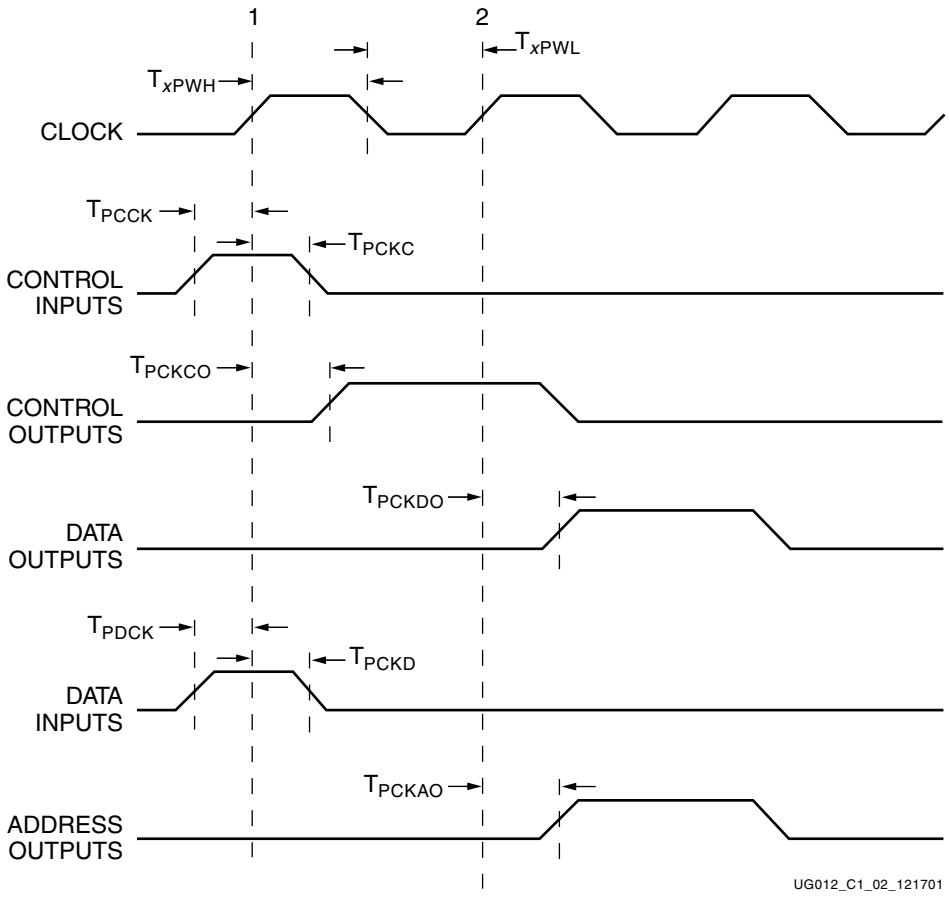


Figure C-2: **Processor Block Timing Relative to Clock Edge**

Index

S

T

U

V

W