Spartan-6 FPGA Memory Controller

User Guide

UG388 (v2.3) August 9, 2010

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/28/09	1.0	Initial Xilinx release.

08/18/09	1.1	• Removed references to MCB per-bit deskew calibration.
		• Chapter 1:
		  • Added XC6SLX75 and XC6SLX75T devices and CPG196, CSG484, and FG(G)900 packages to Table 1-2, page 13.
		• Chapter 2:
		  • In Figure 2-2, page 18, changed Configuration 5 to 128-bit bidirectional.
		  • Added note regarding board design requirements under Table 2-9, page 30.
		• Chapter 3:
		  • Updated first paragraph in Supported Memory Devices, page 35.
		  • Added note to Clocking, page 37.
		  • Added subsection Additional Board Design Requirements, page 42.
		• Chapter 4:
		  • Moved Note 1 from Figure 4-1, page 46 to below the figure.
		  • Added Note 2 about calibration logic.
		• Appendix A:
		  • Updated JEDEC specification links in Memory Standards, page 65.

Date	Version	Revision

12/02/09	2.0	• Moved Chapter 3, "Getting Started," and Chapter 6, "Debugging MCB Designs," and to UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*.
		• Changed introduction in About This Guide, page 7.
		• Chapter 1:
		  • Revised Note 1 in Table 1-1, page 12 to refer to the data sheet for specific values.
		  • Added Note 2 to Table 1-2, page 13.
		• Chapter 2:
		  • In Table 2-3, changed the description and values of the C_MC_CALIBRATION_MODE attribute on page 25.
		  • Appended two sentences to exception (a) on page 30.
		• Chapter 3:
		  • Replaced text regarding the speed of the calibration clock, calib_clk, on page 39.
		• Chapter 4:
		  • Rephrased Note 1 under Figure 4-1, page 46.
		  • In the third paragraph after the Notes on page 46, removed the sentence about calibration logic.
		  • Added note after first paragraph of Calibration, page 47.
		  • Removed portion of sentence about calibration logic in first paragraph of Phase 2: DQS Centering, page 48.
		  • Added paragraph above Figure 4-13, page 59.
		  • Added note on page 62 before Table 4-5.

01/05/10	2.0.1	Revised document hyperlinks.
03/04/10	2.1	Chapter 1: In the Features and Benefits section, added bullet for input termination automatic calibration to section. In Table 1-1, added parameters in Data Rate Minimum column and updated table note 2. In Table 1-2, revised table note 1. Chapter 2: In Table 2-2, added "THREEQUARTERS" as a possible value for the Memory Drive Strength attribute, and modified the description for Memory Burst Length attribute, indicating that DDR3 is always set to 8. In the Clock, Reset, and Calibration Signals section, added calibration to the heading name, introductory text, and caption of Table 2-4. In Table 2-4, changed the signal name BUFPLL to BUFPLL_MCB, changed the signal name sys_rst to async_rst, and added signals mcb_drp_clk and calib_done. In the Memory Device Interface section, modified descriptive text related to RZG and ZIO pins. Added clarifying text in Note (page 30) relating to unused pins from an active MCB reverting to general-purpose I/O. In Table 2-9, modified descriptions for the rzq and zio signals. Chapter 3: In Table 3-1, removed memory devices MT41K128M8xx-25 and MT41K256M4xx-25. In the Clocking section, added text related to MIG/EDK generation of clocking infrastructure. Clarified text related to location of externally driven PLL. Revised text related to calibration clock. In Figure 3-3, changed signal name from calib_clk to mcb_drp_clk. Changed the end of the first sentence after Figure 3-3. Changed the first sentence about the calibration related clock on page 39. Under Figure 3-4, added clarifying text related to using bank1 MCB pins as BPI pins. In the Additional Board Design Requirements section on page 43, clarified requirements for pull-down resistors on the RESET, CKE, and ODT signals. Added Simultaneous Switching Output Considerations section. Chapter 4: In the Phase 1: Input Termination section, added clarifying text related to input termination of the RZQ and ZIO pins. In the Addressing section, added clarifying text related to offsetting the starting address location using the write data mask inputs. Added the Read Latency and Suspend sections. Appendix A: Updated JEDEC URLs.

Date	Version	Revision
06/14/10	2.2	XCN10024, *MCB Performance and JTAG Revision Code for Spartan-6 LX16 and LX45 FPGAs,* addresses these changes: Chapter 1: Added an important note about Standard and Extended performance modes. Chapter 2: In Table 2-4, included the BUFPLL_MCB block name in the pll_lock description and changed the clock frequency example in the sysclk_2x description. Chapter 3: In Clocking, removed LOCKED and pll_lock from the PLL block and MIG Wrapper blocks, respectively, and changed the clock frequency examples in the second and fourth paragraphs under Figure 3-3 on page 38.
08/09/10	2.3	Chapter 1: In Table 1-1, changed the minimum data rate value for LPDDR and indicated that -3N speed-grade devices do not support the MCB in table note 1. Added table note to Table 1-3. Chapter 2: In Table 2-4, added italicized sentence to the calib_done signal description. In Table 2-5, Table 2-6, and Table 2-7, added sentence about reset being required to recover to the pX_cmd_error, pX_wr_error, and pX_rd_error descriptions, respectively. Chapter 3: Added BUFG in Figure 3-3. Added sentences about preferred PLL location to the end of the first paragraph under Figure 3-3. Added sentences about driving MCBs on both sides of the device to the end of the second paragraph under Figure 3-3. Added Modifying the Clock Setup section. Added fourth bullet about V_{REF} to Additional Board Design Requirements. Chapter 4: In the second to the last paragraph of Phase 1: Input Termination, replaced sentence about V_{REF} source still being provided for different I/O standards when a calibrated input termination is desired with sentence about LPDDR memory not requiring V_{REF}. Added sentence about resulting input termination to the last paragraph of Phase 1: Input Termination on page 47. Appendix A: Removed obsolete link.

Table of Contents

Chapter 4: MCB Operation

Appendix A: References

About This Guide

This document describes the Spartan®-6 FPGA memory controller block (MCB). Complete and up-to-date documentation of the Spartan-6 family of FPGAs is available on the Xilinx website at http://www.xilinx.com/products/spartan6/index.htm.

To implement an MCB based memory interface, one of the two supported design tool flows must be followed:

1. Memory Interface Generator (MIG)

   For traditional (non-embedded) FPGA designs, refer to UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide* for information on implementing an MCB based memory interface using the MIG tool within the CORE Generator™ software. This document also contains information on debugging MCB interfaces.

2. Embedded Development Kit (EDK)

   For embedded designs, refer to DS643, *Multi-Port Memory Controller (MPMC)* for details on how the MCB is used to implement the MPMC within the EDK environment.

Guide Contents

This manual contains the following chapters:

- Chapter 1, Memory Controller Block Overview, introduces the Spartan-6 FPGA MCB.
- Chapter 2, MCB Functional Description, describes the architecture, signal interface, and possible configurations of the MCB.
- Chapter 3, Designing with the MCB, provides details on how to incorporate the MCB into a Spartan-6 design, with specifics on how to customize the block for a given application.
- Chapter 4, MCB Operation, explains how the MCB functions in various operational modes: startup, calibration, refresh, precharge, standard read/write transactions, etc.
- Appendix A, References, contains links to additional documentation relevant to memory interface design.

Additional Documentation

The following documents are also available for download at http://www.xilinx.com/products/spartan6/index.htm.

- Spartan-6 Family Overview

  This overview outlines the features and product selection of the Spartan-6 family.

- Spartan-6 FPGA Data Sheet: DC and Switching Characteristics

  This data sheet contains the DC and Switching Characteristic specifications for the Spartan-6 family.

- Spartan-6 FPGA Packaging and Pinouts Product Specification

  This specification includes the tables for device/package combinations and maximum I/Os, pin definitions, pinout tables, pinout diagrams, mechanical drawings, and thermal specifications.

- Spartan-6 FPGA Configuration User Guide

  This all-encompassing configuration guide includes chapters on configuration interfaces (serial and parallel), multi-bitstream management, bitstream encryption, boundary-scan and JTAG configuration, and reconfiguration techniques.

- Spartan-6 FPGA SelectIO Resources User Guide

  This guide describes the SelectIO™ resources available in all Spartan-6 devices.

- Spartan-6 FPGA Clocking Resources User Guide

  This guide describes the clocking resources available in all Spartan-6 devices, including the DCMs and the PLLs.

- Spartan-6 FPGA Block RAM Resources User Guide

  This guide describes the Spartan-6 device block RAM capabilities.

- Spartan-6 FPGA Configurable Logic Block User Guide

  This guide describes the capabilities of the configurable logic blocks (CLBs) available in all Spartan-6 devices.

- Spartan-6 FPGA GTP Transceivers User Guide

  This guide describes the GTP transceivers available in Spartan-6 LXT FPGAs.

- Spartan-6 FPGA DSP48A1 Slice User Guide

  This guide describes the architecture of the DSP48A1 slice in Spartan-6 FPGAs and provides configuration examples.

- Spartan-6 FPGA PCB and Pin Planning Design Guide

  This guide provides information on PCB design for Spartan-6 devices, with a focus on strategies for making design decisions at the PCB and interface level.

- Spartan-6 FPGA Power Management User Guide

  This guide provides information on the various hardware methods of power management in Spartan-6 devices, primarily focusing on the suspend mode.

Additional Support Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support.

XILINX®

Chapter 1

Memory Controller Block Overview

Scope

This chapter provides an overview of the Spartan®-6 FPGA memory controller block (MCB). It contains these sections:

- Introduction
- Features and Benefits
- Block Diagram
- Performance
- Device Family Support
- Supported Memory Configurations
- Software and Tool Support

Introduction

The MCB is a dedicated embedded block multi-port memory controller that greatly simplifies the task of interfacing Spartan-6 devices to the most popular memory standards. The MCB provides significantly higher performance, reduced power consumption, and faster development times than equivalent IP implementations. The embedded block implementation of the MCB conserves valuable FPGA resources and allows the user to focus on the more unique features of the FPGA design.

Features and Benefits

The key features and benefits of the Spartan-6 FPGA memory controller block are:

- DDR, DDR2, DDR3, and LPDDR (Mobile DDR) memory standards support
- Up to 800 Mb/s (400 MHz double data rate) performance
- Up to four MCB cores in a single Spartan-6 device. Each MCB core supports:
  - 4-bit, 8-bit, or 16-bit single component memory interface
  - Memory densities up to 4 Gb
  - Up to 12.8 Gb/s aggregate bandwidth
- Configurable dedicated multi-port user interface to FPGA logic
  - 1 to 6 ports per MCB depending on configuration
  - 32-, 64-, or 128-bit data bus options
  - Bidirectional (R/W) or unidirectional (W only or R only) port options

- Memory Bank Management
  - Up to eight memory banks open simultaneously for greater controller efficiency
- Embedded controller and physical interface (PHY), providing:
  - Predictable timing
  - Low power
  - Guaranteed performance
- Predefined pinouts (I/O locations) for each MCB
  - Simplified board design
  - Predefined I/Os not used in an MCB interface become general-purpose I/Os (see page 30 for details).
- Common memory device options and attributes support
  - Programmable drive strength
  - On-Die Termination (ODT)
  - CAS latency
  - Self refresh (including partial array)
  - Refresh interval
  - Write recovery time
- Automatic delay calibration of memory strobe and read data inputs
  - Adjusts DQS (strobe) to DQ (data) timing relationship for optimal read performance
- Optional automatic calibration of FPGA on-chip input termination for optimal signal integrity
- Supported by Xilinx® CORE Generator™ and Embedded Development Kit (EDK) design tools
  - Memory Interface Generator (MIG) tool within the CORE Generator software simplifies the MCB design flow
  - Embedded designs can also access the MCB via the multi-port memory controller (MPMC) IP in the EDK tool

Block Diagram

The block diagram in Figure 1-1 shows the major architectural components of the MCB core. Throughout this document, the MCB is described as provided to the user by the memory IP tools within the CORE Generator software or EDK environment. These tools typically produce top-level "wrapper" files that incorporate the embedded block memory controller primitive and any necessary soft logic and port mapping required to deliver the complete solution. For example, in Figure 1-1, the physical interface of the MCB uses the capabilities of the general I/O block (IOB) to implement the external interface to the memory. General I/O clock network resources are also used.

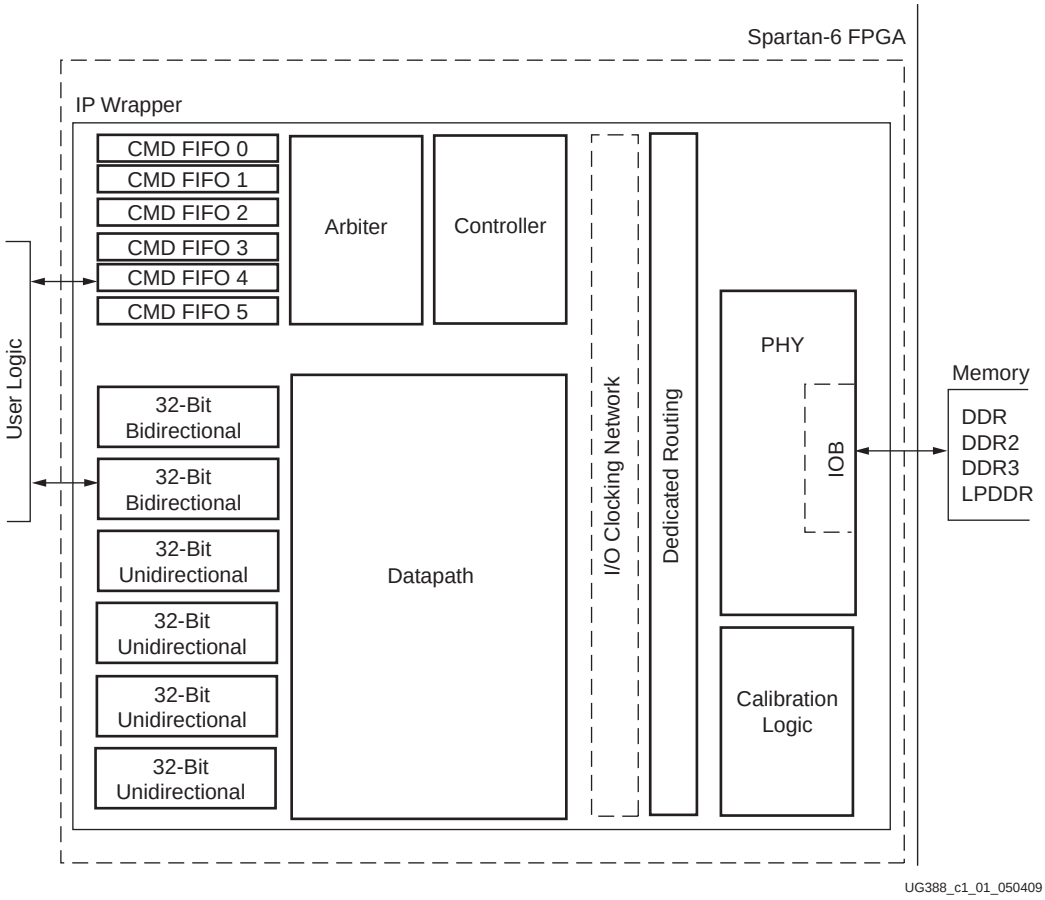

Figure 1-1: **Spartan-6 FPGA Memory Controller Block (IP Wrapper View)**

The single data rate (SDR) user interface to the MCB inside the FPGA can be configured for one to six ports, with each port consisting of a command interface and a read and/or write data interface. The two 32-bit bidirectional and four 32-bit unidirectional hardware-based ports inside the MCB can be grouped to create five different port configurations.

Other major components of the MCB include:

- Arbiter

  Determines which port currently has priority for accessing the memory device.

- Controller

  Primary control block that converts the simple requests made at the user interface into the necessary instructions and sequences required to communicate with the memory.

- Datapath

  Handles the flow of write and read data between the memory device and the user logic.

- Physical Interface (PHY)

  Converts the controller instructions into the actual timing relationships and DDR signaling necessary to communicate with the memory device.

- Calibration Logic

  Calibrates the PHY for optimal performance and reliability.

Performance

The dedicated MCB cores in Spartan-6 devices enable significantly higher performance levels than equivalent IP solutions implemented in the FPGA logic. Because memory bandwidth is often the bottleneck in overall system performance, the MCB cores were specifically engineered for users looking to maximize memory performance in a low-cost, low-power FPGA device.

Each MCB core supports the memory interface data rates and total memory bandwidth specifications shown in Table 1-1. Peak bandwidth for a single MCB memory interface is calculated for the three supported interface widths.

Note: The MCB supports Standard and Extended performance modes depending on the selected V_{CCINT} operating conditions. Peak data rates shown in Table 1-1 represent maximum performance when using the V_{CCINT} range in Extended performance mode. Refer to Table 2 (Recommended Operating Conditions) and the Performance Characteristics section in DS162, *Spartan-6 FPGA Data Sheet: DC and Switching Characteristics* for V_{CCINT} operating conditions and performance specifications for Standard and Extended modes.

Table 1-1: **Memory Interface Data Rates and Peak Bandwidth for Each MCB**

Memory Type	Data Rate: Mb/s DDR (MHz Clock)		Peak Bandwidth per MCB Interface (Gb/s)		
	Minimum	Maximum(1)	4-Bit	8-Bit	16-Bit
DDR	167 Mb/s(2) (83.3 MHz)	400 Mb/s (200 MHz)	1.6 Gb/s	3.2 Gb/s	6.4 Gb/s
DDR2	250 Mb/s(2) (125 MHz)	800 Mb/s (400 MHz)	3.2 Gb/s	6.4 Gb/s	12.8 Gb/s
DDR3	606 Mb/s(2) (303 MHz)	800 Mb/s (400 MHz)	3.2 Gb/s	6.4 Gb/s	12.8 Gb/s
LPDDR	60 Mb/s(2) (30 MHz)	400 Mb/s (200 MHz)	1.6 Gb/s	3.2 Gb/s	6.4 Gb/s

Notes:

1. The maximum MCB data rate shown does not apply to all speed grades. Refer to DS162, *Spartan-6 FPGA Data Sheet: DC and Switching Characteristics*, for performance by speed grade. The -3N speed-grade devices do not support the MCB.
2. The minimum frequency requirement of the MCB is dictated by the minimum frequency specification for the memory standard. See Memory Standards in Appendix A for links to the relevant JEDEC specifications.

Device Family Support

The number of MCBs available in a given Spartan-6 device is determined by the density range that the device falls within. The smallest device (XC6SLX4) contains no MCBs, mid-range density devices contain two MCBs, and the largest devices contain four MCBs. Table 1-2 shows the number of MCBs supported in each device/package combination.

Note: The MCB is designed to interface to a single x4, x8, or x16 memory component. Multiple component interfaces to a single MCB (for example, two x8 memories interfacing to an MCB in x16 mode) are not supported.

Table 1-2: **MCB Support by Device / Package Combination**

Device	Package								
	TQG144	CPG196	CSG225	FT(G)256	CSG324	FG(G)484	CSG484	FG(G)676	FG(G)900
XC6SLX4	0	0	0						
XC6SLX9	0	0	2(1)	2	2				
XC6SLX16		0	2(1)	2	2				
XC6SLX25				2	2	2			
XC6SLX45					2	2	2	2	
XC6SLX75						2(2)	2(2)	4	
XC6SLX100						2(2)	2(2)	4	
XC6SLX150						2(2)	2(2)	4	4
XC6SLX25T					2	2			
XC6SLX45T					2	2	2		
XC6SLX75T						2(2)	2(2)	4	
XC6SLX100T						2(2)	2(2)	4	4
XC6SLX150T						2(2)	2(2)	4	4

Notes:

1. For devices in the CSG225 package, the MCBs support only the x4 and x8 memory interface width options, meaning LPDDR devices cannot be supported. In addition, there are only 13 MCB address bits available in this package, which limits the maximum memory density to 256 Mb for DDR2 and 512 Mb for DDR and DDR3.
2. For devices with four MCBs, only two MCBs are bonded out in the FGG484 and CSG484 packages.

Supported Memory Configurations

The Spartan-6 FPGA MCB supports a wide range of common memory types, configurations, and densities, as shown in Table 1-3.

Table 1-3: **Supported Memory Configurations**

Memory Density	Width (# DQ bits)	Memory Type			
		LPDDR	DDR	DDR2	DDR3
128 Mb	x16	X	X		
	x8		X		
	x4		X		
256 Mb	x16	X	X	X	
	x8		X	X	
	x4		X	X	
512 Mb	x16	X	X	X	X
	x8		X	X	X
	x4		X	X	X
1 Gb	x16	X	X	X	X
	x8		X	X	X
	x4		X	X	X
2 Gb	x16			X	X
	x8			X	X
	x4			X	X
4 Gb(1)	x16				X

Notes:

1. The MCB supports single-die, 4 Gb memory components (when available from memory suppliers) but not dual-die, 4 Gb memory components.

Software and Tool Support

The Spartan-6 FPGA MCB is supported by standard software and tool flows like other soft and embedded IP blocks offered by Xilinx. For conventional (i.e., non-embedded) FPGA designs, the MCB can be integrated into a design using the Memory Interface Generator (MIG) tool, available in the CORE Generator tool.

The MIG tool is used to generate memory interfaces for all Xilinx FPGAs. It produces the necessary RTL design files, user constraints files (UCFs), and script files for simulation and implementation of memory solutions offered by Xilinx. The Getting Started chapter in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*, contains detailed step-by-step instructions on how to use the MIG tool to implement memory interfaces based on the MCB.

For embedded designs (e.g., MicroBlaze™ processor designs), the IP configurator GUI found in the Xilinx Platform Studio tool within the EDK environment can be used to specify the memory interface characteristics. In this flow, the MCB serves as the underlying hardware implementation of the MPMC IP block, available in the EDK library. In addition to setting up the controller and memory attributes, the tool generates the necessary soft bridges to the PLB bus, Xilinx Cache Link (XCL), LocalLink (LL), or other specified interface for connecting EDK peripherals to the resulting memory controller ports.

XILINX®

Chapter 2

MCB Functional Description

This chapter provides a detailed functional description of the Spartan®-6 FPGA MCB. It contains the following sections:

- Architecture Overview
- Port Configurations
- Arbitration
- Programmability
- Interface Details

Architecture Overview

The MCB provides a simple, reliable means of interfacing to a single component memory device. The MCB User Interface removes the complexities of DDR memory interfacing so that more engineering resources can be directed to the unique aspects of the FPGA design.

The MCB can operate at speeds considerably faster than a comparable "soft" solution implemented in the FPGA logic. With data rates up to 800 Mb/s, the MCB more than doubles the performance of prior generation low-cost FPGA memory interface solutions, allowing higher levels of bandwidth and/or narrower memory buses. This provides the significant benefit of conserving valuable FPGA logic and I/O resources that are otherwise required to communicate with the memory device.

Figure 2-1 expands on the MCB block diagram introduced in Chapter 1 to show the major signals associated with the User Interface internal to the FPGA as well as the I/O signals connected to the external memory device. While the User Interface can be configured to support up to six ports, for simplicity, Figure 2-1 shows only the signals for a single bidirectional port.

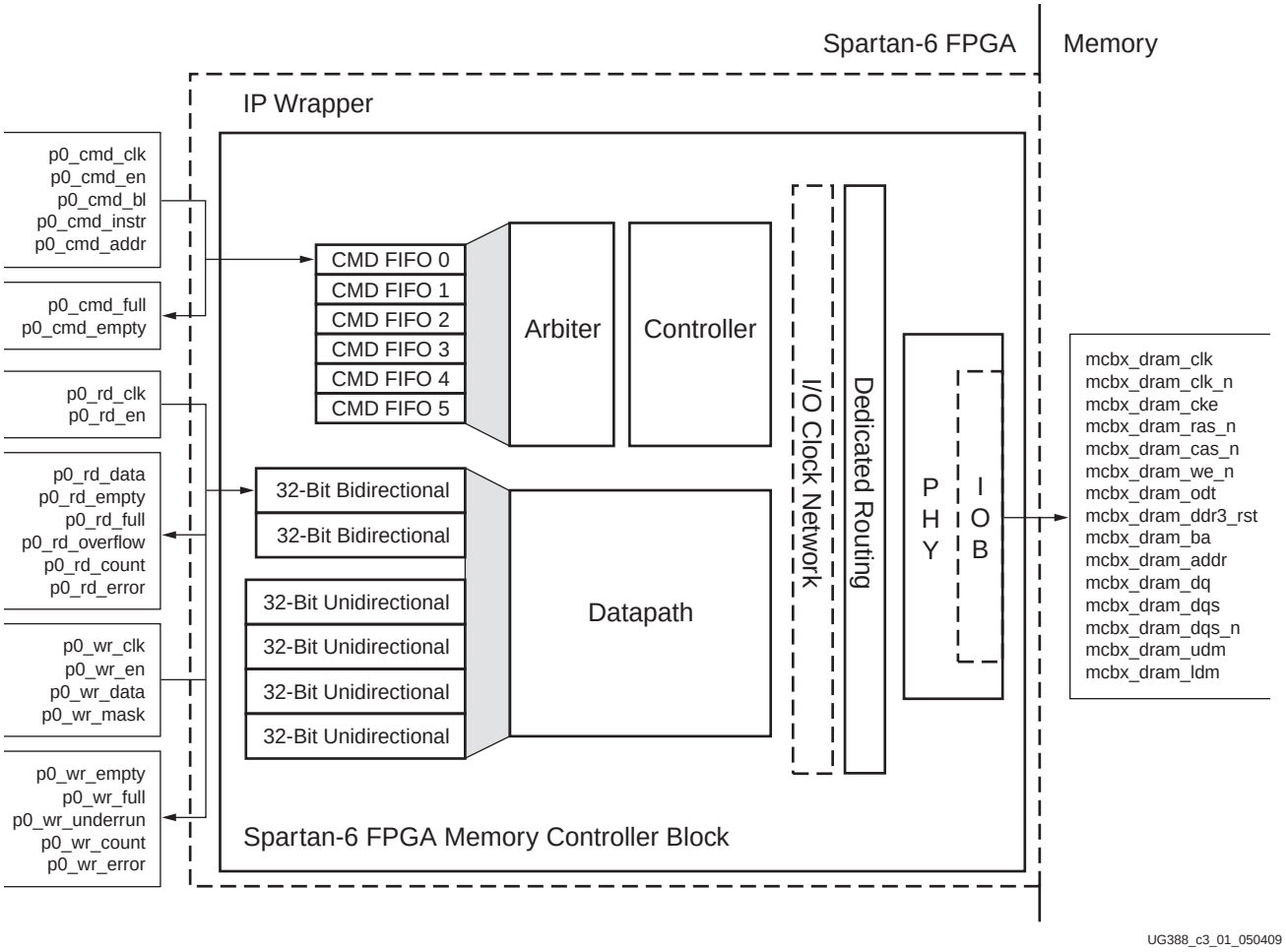

Figure 2-1: **MCB Architecture with Major Internal and I/O Signals**

There are three basic types of ports that can be established at the User Interface:

- Read port (unidirectional)
- Write port (unidirectional)
- Read and Write port (bidirectional)

Each port contains a command path and a datapath. For a unidirectional port, a command path is paired with a single read-only or a single write-only datapath. However, for a bidirectional port, a single command path is shared by both the read and write datapaths associated with that port. FIFOs are used at the User Interface of the command path and datapath to queue up memory requests and to manage the transfer from the user clock domain to the memory controller clock domain.

The command path signals for a port are used to issue requests to the command FIFOs. The command FIFOs have a user-programmable depth up to four. They store the instruction type (read, write, refresh, etc.), address, and burst length associated with a requested memory transaction. The command path also includes full and empty status flag outputs from the command FIFOs, indicating whether new requests can be accepted. There are six command FIFOs available in hardware; the port configuration determines how many are accessible to the User Interface (see Port Configurations). For more details on the command path signals, refer to Interface Details, page 25.

In the datapath, the underlying hardware contains six 32-bit ports, two of which are inherently bidirectional. The other four ports are inherently unidirectional but can be combined to create bidirectional ports as well. There are five possible port configurations that combine these six hardware ports to implement the desired User Interface (see Port Configurations). The width of the read and write data word fields of the User Interface are naturally determined by the chosen configuration.

The datapath FIFOs are 64 deep, allowing for burst lengths of up to 64 data words from a given start address. In addition to the data word field, the write path FIFOs contain mask bit fields that allow optional masking of write data on a per byte basis. Full, empty, underrun, count, and error outputs indicate the current status of the write data FIFOs. The read data FIFOs have a similar set of status outputs. For more details on the read and write datapath signals, refer to Interface Details, page 25.

The arbiter inside the MCB uses a time slot based arbitration mechanism to determine which of the one to six ports of the User Interface currently has access to the memory. There are also methods for allowing some ports greater priority, and thus frequent access to the memory, as discussed in Arbitration.

Bank management logic in the MCB allows up to eight memory banks to be open simultaneously, allowing the controller to maintain high efficiency levels when accessing data spread across multiple banks. In addition, read and write requests to the memory can include an optional auto-precharge to automatically close a bank upon completion of the transaction to improve the efficiency of random data accesses within a bank. The MCB does not perform any reordering of transactions.

Port Configurations

The five possible port configurations for the User Interface are shown in Figure 2-2. In Configuration 1, the user ports essentially map directly to the underlying six physical hardware ports. For the other configurations, the diagram shows how the physical ports are concatenated to create different user port combinations. As shown in Figure 2-2, the MIG tool always sequentially numbers ports for the User Interface starting from 0, regardless of the underlying physical port numbers.

In all five port configurations, the command path, write datapath, and read datapath within a given port all have separate clocks and therefore can be connected to independent clock domains. However, it is recommended that all paths related to a given port be kept on a single clock domain to simplify the interface requirements.

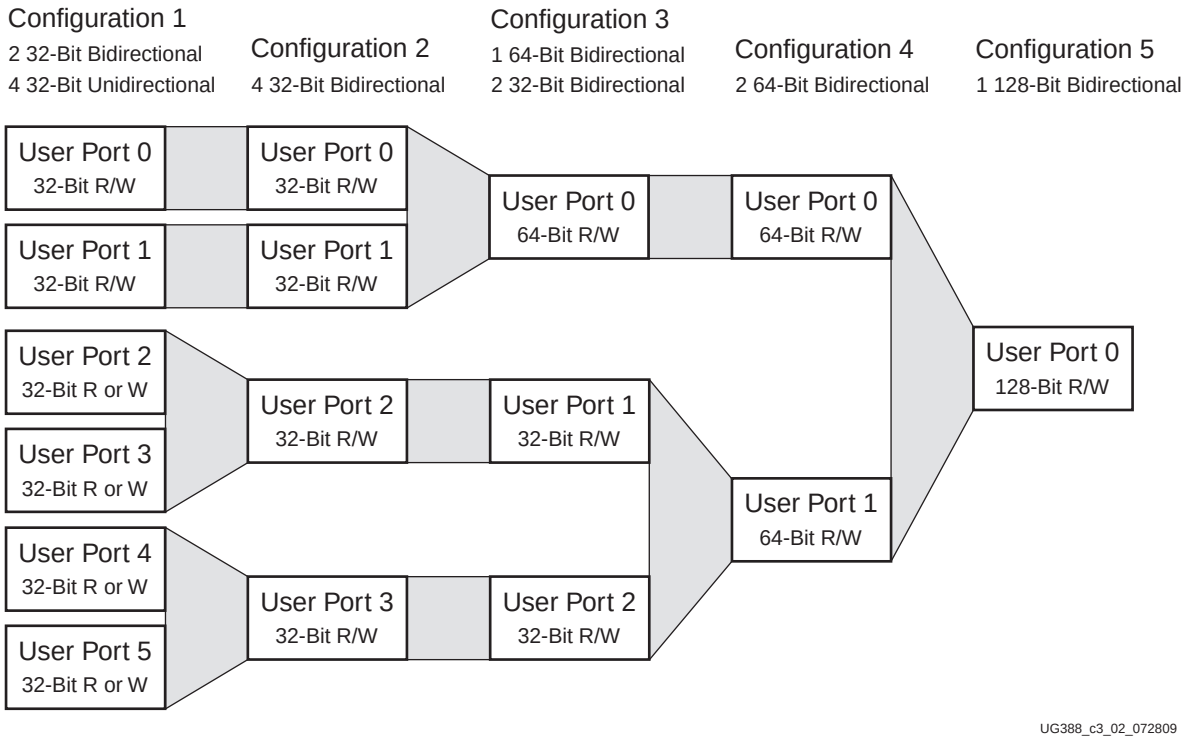

Figure 2-2: **Possible Port Configurations for the User Interface**

Selecting a Port Configuration

The MIG tool in the CORE Generator™ tool provides a simple graphical interface for setting up the number and type of ports required for a specific application. For designs that require less than the full width or functionality of the User Interface, unused ports can simply be disabled through the MIG interface. In the event that additional ports are required beyond the six ports provided in the MCB, port bridges with additional arbitration mechanisms can be implemented in the FPGA logic to expand the MCB port capabilities.

Arbitration

The arbiter inside the MCB uses a time slot based arbitration mechanism to determine which port of the User Interface currently has access to the memory. There are 12 time slots in the arbitration table as shown in Table 2-1. Each time slot corresponds to a single memory clock cycle. The order of port priority in a given time slot is determined by the port numbers entered into the Priority 1 through 6 columns moving from left to right across the table.

Table 2-1 shows the case where the User Interface is configured for the maximum six ports. If the MCB is configured to have fewer than six ports, the arbitration table automatically adjusts to have priority columns only for the selected number of ports.

Table 2-1: **MCB Arbitration Table with Round Robin Configuration**

Time Slot	Priority 1	Priority 2	Priority 3	Priority 4	Priority 5	Priority 6
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4
6	0	1	2	3	4	5
7	1	2	3	4	5	0
8	2	3	4	5	0	1
9	3	4	5	0	1	2
10	4	5	0	1	2	3
11	5	0	1	2	3	4

During a given clock cycle, the arbiter determines which port to service in that time slot. It moves left to right across the priority columns to find the first port in that row that has a command pending in its command FIFO. That port is then serviced with execution of the pending command, and the arbiter moves on to the next time slot on the following clock cycle. If no port has a command pending for that row, no action occurs for that time slot and a clock cycle is lost.

The order of port priorities in the arbitration table is fully programmable. The MIG tool provides a default round-robin scheme as illustrated in Table 2-1, where all ports are given the highest priority in 2 of the 12 available time slots. However, the MIG tool also provides a custom option where the user can define any arbitration table. This allows for some ports to be given greater overall access to the memory device. However, care should be exercised when using this option to ensure that the assigned priorities do not prevent any active ports from receiving access to the memory device.

It is possible to configure the User Interface to have five ports (two 32-bit bidirectional ports and three 32-bit unidirectional ports, with one 32-bit unidirectional port disabled). In this case, the arbitration table is reduced to 10 time slots. When the number of time slots is evenly divisible by the number of ports, each port is ensured to receive equal access to the memory device, if desired.

Programmability

The MCB is highly configurable through a set of memory device and controller attributes, allowing it to support multiple memory standards and configurations. The MIG tool within the CORE Generator tool and the IP Configurator in the Xilinx Platform Studio tool within the EDK environment provide a simple means of configuring the MCB attributes to implement the desired memory interface (for example, see the "Getting Started" chapter in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*).

Table 2-2 and Table 2-3 list the memory device and controller attributes, respectively, supported by the MCB. The specific HDL parameter names, possible values, and descriptions associated with each of the attributes are provided. In general, the MIG tool or IP Configurator tools are responsible for setting all parameter values, so the values should not be modified directly.

Memory timing parameters are taken from the vendor data sheets, and are automatically assigned by the tools when a supported device is selected. Timing parameters can be specified when creating a custom device (see the "Setting Controller Options" section in the *Spartan-6 FPGA Memory Interface Solutions User Guide*).

Table 2-2: **Memory Device Attributes**

Memory Attributes	Parameter Name(s)	Description / Possible Values
Memory Type	C_MEM_TYPE	This attribute sets the memory standard implemented by the MCB. Possible values: DDR, DDR2, DDR3, LPDDR.
Memory Data Bus Width	C_NUM_DQ_PINS	This attribute sets the bit width of the DQ bus. Possible values: "4", "8", "16".
Memory Address Bus Width	C_MEM_ADDR_WIDTH	This attribute sets the memory address bus width (the total number of address bits). Possible values: Based on device selection in the MIG tool.
Memory Bank Address Bus Width	C_MEM_BANKADDR_WIDTH	This attribute sets the number of bank address bits. Possible values: Based on device selection in the MIG tool.
Memory Column Address Bus Width	C_MEM_NUM_COL_BITS	This attribute sets the number of column address bits. Possible values: Based on device selection in the MIG tool.
Memory Burst Length	C_MEM_BURST_LEN	This attribute sets the memory burst length to be used. The MIG tool determines the value based on the memory standard, port configuration, and interface width. DDR3 will always be set to 8. Possible values: "4", "8".

Table 2-2: **Memory Device Attributes *(Cont'd)***

Memory Attributes	Parameter Name(s)	Description / Possible Values
Memory CAS Latency	DDR, DDR2, LPDDR: C_MEM_CAS_LATENCY DDR3: C_MEM_DDR3_CAS_LATENCY, C_MEM_DDR3_CAS_WR_LATENCY	This attribute sets the CAS latency (the delay in clock cycles between the READ command and the first output data) for memory. DDR3 has separate Read and Write CAS latency values. Possible values: 2, 3, 4, 5, 6, 7, 8, 9, 10, depending on memory type.
Partial Array Self-Refresh Size	C_MEM_MOBILE_PA_SR	For LPDDR: This attribute sets the array size for self-refresh operation. Possible values: LPDDR: Full, Half
Memory Drive Strength	DDR, DDR2: C_MEM_DDR1_2_ODS DDR3: C_MEM_DDR3_ODS LPDDR: C_MEM_MDDR_ODS	This attribute sets output drive strength of memory device. Possible values: DDR/DDR2: "FULL", "REDUCED" DDR3: "DIV6" (RZQ/6), "DIV7" (RZQ/7) LPDDR: "FULL", "THREEQUARTERS", "HALF", "QUARTER"
Memory Termination Value (ODT)	DDR2: C_MEM_DDR2_RTT DDR3: C_MEM_DDR3_RTT	This attribute sets on-die termination resistance of the memory device. Possible values: DDR2: "OFF", "50OHMS", "75OHMS", "150OHMS" DDR3: "OFF", "DIV2" (RZQ/2), "DIV4" (RZQ/4), "DIV6" (RZQ/6), "DIV8" (RZQ/8), "DIV12" (RZQ/12) ***Note:*** RZQ = 240Ω
Memory Differential DQS Enable	C_MEM_DDR2_DIFF_DQS_EN	Enables differential DQS strobe use. This attribute is always enabled for DDR3; it is set to "YES" for DDR2 at frequencies above 200 MHz. Possible values: DDR2: "YES", "NO"
Memory Auto Self Refresh	C_MEM_DDR3_AUTO_SR	For DDR3 only: Auto self-refresh allows memory to determine the best refresh interval based on device temperature. If auto self-refresh is not used, the operating temperature range must be indicated using the high-temperature self-refresh register. Possible values: "ENABLED", "MANUAL"
Memory High Temperature Self Refresh	C_MEM_DDR2_3_HIGH_TEMP_SR	For DDR2 and DDR3: Memory can be put in high-temperature self-refresh mode to decrease refresh interval time. Possible values: DDR2/DDR3: "NORMAL" (0–85°C), "EXTENDED" (> 85°C)

Table 2-2: **Memory Device Attributes *(Cont'd)***

Memory Attributes	Parameter Name(s)	Description / Possible Values
Memory Dynamic Output Driver Termination	C_MEM_DDR3_DYN_WRT_ODT	For DDR3: Determines the value of the dynamic output driver termination. Possible values: DDR3: "OFF", "DIV2" (RZQ/2), "DIV4" (RZQ/4)
Memory t_{RAS} Value	C_MEM_TRAS	Minimum Active to Precharge period for memory. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{RCD} Value	C_MEM_TRCD	Minimum Active to the Read or Write command delay for memory. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{REFI} Value	C_MEM_TREFI	Average Periodic Refresh Interval for memory. This attribute is the rate at which the MCB refreshes the memory, not the self-refresh interval. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{RFC} Value	C_MEM_TRFC	Minimum Auto-Refresh to Active or Auto-Refresh command period for memory. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{RP} Value	C_MEM_TRP	Minimum Precharge command period for memory. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{WR} Value	C_MEM_TWR	Minimum Write Recovery time for memory. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{RTP} Value	C_MEM_TRTP	Minimum Read to Precharge command delay for memory. Typically, this parameter is only found in DDR2 and DDR3 devices. Possible values (in picoseconds): Based on device selection in the MIG tool.
Memory t_{WTR} Value	C_MEM_TWTR	Minimum Write to Read command delay for memory. Possible values (in picoseconds): Based on device selection in the MIG tool.

Table 2-3: **Controller Attributes**

Controller Attributes	Parameter Name(s)	Description / Possible Values
Controller Clock Period	C_MEMCLK_PERIOD	This attribute converts memory timing parameters between clock cycles and picoseconds. Possible values (in picoseconds): Based on frequency selection in the MIG tool.
Controller Port Configuration	C_PORT_CONFIG	This attribute sets the port configuration of the User Interface. It determines the port direction (B = Bidirectional, W = Unidirectional Write, R = Unidirectional Read) and data bus width (32, 64, or 128 bits). Possible Values: "B32_B32_W32_W32_W32_W32" "B32_B32_W32_W32_W32_R32" "B32_B32_W32_W32_R32_W32" "B32_B32_W32_W32_R32_R32" "B32_B32_W32_R32_W32_W32" "B32_B32_W32_R32_W32_R32" "B32_B32_W32_R32_R32_W32" "B32_B32_W32_R32_R32_R32" "B32_B32_R32_W32_W32_W32" "B32_B32_R32_W32_W32_R32" "B32_B32_R32_W32_R32_W32" "B32_B32_R32_W32_R32_R32" "B32_B32_R32_R32_W32_W32" "B32_B32_R32_R32_W32_R32" "B32_B32_R32_R32_R32_W32" "B32_B32_R32_R32_R32_R32" "B32_B32_B32_B32" "B64_B32_B32" "B64_B64" "B128"
Port Data Bus Width (Ports 0 and 1)	C_P0_DATA_PORT_SIZE C_P1_DATA_PORT_SIZE	Ports 0 and 1 of the User Interface can vary in data bus width depending on the port configuration selected (Ports 3 through 5, if available, are always 32 bits wide). These parameters set the Port 0 and 1 data width. Possible Values: "32", "64", "128"
Port Data Mask Width (Ports 0 and 1)	C_P0_MASK_SIZE C_P1_MASK_SIZE	This attribute sets the number of mask bits for Ports 0 and 1, depending on the data bus width as determined by the port configuration. Possible Values: "4", "8", "16"

Table 2-3: **Controller Attributes *(Cont'd)***

Controller Attributes	Parameter Name(s)	Description / Possible Values
Controller Port Enable	C_PORT_ENABLE	This six-bit value determines which of the 6 underlying 32-bit hardware ports are used in a given port configuration. Possible Values: For example, `6'b001111` = ports 0 to 3 enabled
Address Mapping Order	C_MEM_ADDR_ORDER	This attribute determines how the byte address presented to the User Interface maps to the physical memory bank, row, and column address bits. This attribute is based on a system addressing scheme. This value should be set to take the most advantage of MCB open bank management capabilities. Possible Values: "BANK_ROW_COLUMN", "ROW_BANK_COLUMN"
Arbitration Time Slot Count	C_ARB_NUM_TIME_SLOTS	This attribute sets the number of time slots in the arbitration table. Most port configurations have 12 time slots, but port configurations with 5 active ports have 10 time slots in the arbitration table to ensure equal arbitration. Possible Values: "12", "10"
Arbitration Time Slot Values	C_ARB_TIME_SLOT[0:11]	These 6-digit octal (18-bit) values set the port priority for each time slot. Possible Values: For example, C_ARB_TIME_SLOT0 = 18'o012345 (sets Port 0 with the highest priority down to Port 5 with the lowest priority).
Controller Calibration Bypass (Simulation)	C_MC_CALIB_BYPASS	Directs the MIG tool to set up simulation files to skip the controller calibration sequence for faster simulation. ***Note:*** This parameter is for simulation only. Possible Values: "YES", "NO"
Reserved Calibration Address Space	C_MC_CALIBRATION_RA C_MC_CALIBRATION_BA C_MC_CALIBRATION_CA	Defines the starting row, bank, and column address reserved for calibration. This attribute is used for training pattern data during recalibration to avoid overwrite of application data. Possible Values (any valid address is okay): Examples: C_MC_CALIBRATION_RA = `15'h0000` C_MC_CALIBRATION_BA = `3'h0` C_MC_CALIBRATION_CA = `12'h000`

Table 2-3: **Controller Attributes *(Cont'd)***

Controller Attributes	Parameter Name(s)	Description / Possible Values
Calibration Mode	C_MC_CALIBRATION_MODE	This attribute determines whether the MCB executes precise alignment and real-time voltage/temperature compensation of the DQS strobe (recommended) or simply uses a fixed ratio of the bit period to offset DQS into the data window. Possible Values: "CALIBRATION" (precise DQS alignment with voltage/temperature compensation), "NOCALIBRATION" (fixed DQS offset delay)
DQS Offset Delay Value	C_MC_CALIBRATION_DELAY	This attribute sets the fixed DQS offset delay as a ratio of the bit period when C_MC_CALIBRATION_MODE = "NO CALIBRATION". Possible Values: "QUARTER", "HALF", "THREEQUARTER", "FULL"

Interface Details

As shown in the architecture block diagram of Figure 2-1, page 16, the MCB has two basic interfaces: the internal User Interface to the FPGA logic and the external interface to the memory device via the predefined I/O pins. The following subsections discuss the details of all signals related to these two interfaces. As in the rest of this document, all descriptions refer to the interface of the IP wrapper delivered by the CORE Generator or EDK tool flows, not the interface of the underlying memory controller block primitive.

User (Fabric Side) Interface

The User Interface contains all the necessary signals for the user logic in the FPGA logic to interact with the command path and datapath of the MCB ports. It also includes the general clock and reset signals for the MCB as well as signals related to calibration, debug, and self-refresh operation. The User Interface can be configured to have anywhere from one to six ports as shown in Port Configurations, page 17.

Clock, Reset, and Calibration Signals

Table 2-4 shows the clock, reset, and calibration related signals of the MCB User Interface.

Table 2-4: **Clock, Reset, and Calbration Signals**

Signal Name	Direction	Description
async_rst	Input	Main system reset for the MCB.
calib_done	Output	This active-High signal indicates the completion of all phases of calibration during the start-up sequence of the MCB. *Transactions should not be submitted to the MCB until this signal goes High to indicate that calibration has completed.* See Calibration in Chapter 4 for more information.

Table 2-4: **Clock, Reset, and Calbration Signals *(Cont'd)***

Signal Name	Direction	Description
mcb_drp_clk	Input	This clock synchronizes the soft calibration module to the sysclk_2x domain. It must be generated by the same PLL as sysclk_2x to ensure that it is phase-synchronized to that domain. See Clocking in Chapter 3 for more information.
pll_ce_0	Input	I/O clock enable strobe from BUFPLL_MCB. This signal pulses High on every other clock cycle of sysclk_2x. It is used for double data rate transfers in the I/O blocks.
pll_ce_90	Input	I/O clock enable strobe from BUFPLL_MCB. This signal pulses High on every other clock cycle of sysclk_2x_180. It is used for double data rate transfers in the I/O blocks.
pll_lock	Input	Lock signal from the BUFPLL_MCB block.
sysclk_2x	Input	Main system clock for the MCB. This signal is generated by the Spartan-6 FPGA PLL block and is rebuffered by the BUFPLL_MCB driver to the I/O clock network. It operates at two times the memory clock frequency (for example, 667 MHz for a 333 MHz memory interface).
sysclk_2x_180	Input	This input is the phase-shifted clock with the same frequency as sysclk_2x. It is generated by the same PLL/BUFPLL_MCB resources.

Command Path

Table 2-5 defines the signals related to the command path of the MCB User Interface. All port signal names have the prefix *pX*, where *X* represents the port number (for example, port 0 signals are prefixed with p0, port 1 with p1, and so forth).

Table 2-5: **Command Path Signals**

Signal Name	Direction	Description
pX_cmd_addr[29:0]	Input	Byte start address for current transaction. Addresses must be aligned to port size: 32-bit ports: Lower two bits must be 0s. 64-bit ports: Lower three bits must be 0s. 128-bit ports: Lower four bits must be 0s.
pX_cmd_bl[5:0]	Input	Burst length in number of user words for the current transaction. Burst length is encoded as 0 to 63, representing 1 to 64 user words (for example, `6'b00011` is a burst length 4 transaction). The user word width equals the port width (for example, a burst length of 3 on a 64-bit port transfers 3 x 64-bit user words = 192 bits total).
pX_cmd_clk	Input	User clock for the Command FIFO. FIFO signals are captured on the rising edge of this clock.
pX_cmd_empty	Output	This active-High empty flag for the Command FIFO indicates no commands are queued in FIFO, although there might be commands in flight.

Table 2-5: **Command Path Signals *(Cont'd)***

Signal Name	Direction	Description
pX_cmd_en	Input	This active-High signal is the write-enable signal for the Command FIFO. This signal is covered in more detail in Chapter 4.
pX_cmd_error	Output	This output indicates a Command Port error occurred because the FIFO pointers were unsynchronized. An MCB reset is required to recover from this condition.
pX_cmd_full	Output	This active-High output is the full flag for the Command FIFO. It indicates the FIFO cannot accept any more commands and blocks writes to the Command FIFO.
pX_cmd_instr[2:0]	Input	Command code for the current instruction. Bit 0 represents the READ/WRITE select, Bit 1 is Auto Precharge enable, and Bit 2 represents Refresh, which always takes priority: Write: `3'b000` Read: `3'b001` Write with Auto Precharge: `3'b010` Read with Auto Precharge: `3'b011` Refresh: `3'b1xx` This signal is covered in more detail in Chapter 4.

Write Datapath

Table 2-6 shows all signals related to the write datapath of the MCB User Interface. All port signal names have the prefix *pX*, where *X* represents the port number (for example, port 0 signals are prefixed with p0, port 1 with p1, and so forth).

Table 2-6: **Write Datapath Signals**

Signal Name	Direction	Description
pX_wr_clk	Input	This signal is the user clock for the Write Data FIFO.
pX_wr_count[6:0]	Output	Count value for Write Data FIFO. This output indicates how many user words are in the FIFO (from 1 to 64). A count value of 0 indicates the FIFO is empty. This signal has a longer latency than the pX_wr_empty flag. Therefore, the FIFO could be empty or experience an underrun even when the count is not 0.
pX_wr_data[PX_SIZE-1:0]	Input	Write Data value to be loaded into Write Data FIFO and sent to memory. PX_SIZE can be 32, 64, or 128 bits, depending on port configuration.
pX_wr_empty	Output	This active-High signal is the empty flag for the Write Data FIFO. It indicates there is no valid data in the FIFO.

Table 2-6: **Write Datapath Signals *(Cont'd)***

Signal Name	Direction	Description
pX_wr_en	Input	This active-High signal is the write enable for the Write Data FIFO. It indicates that the value on pX_wr_data is valid for loading into the FIFO. Data is loaded on the rising edge of pX_wr_clk when pX_wr_en = 1 and pX_wr_full = 0.
pX_wr_error	Output	This signal indicates a Write Data FIFO error occurred because the FIFO pointers were unsynchronized. An MCB reset is required to recover from this condition.
pX_wr_full	Output	This active-High signal is the full flag for the Write Data FIFO. When this signal is high, it prevents data from being loaded into the FIFO.
pX_wr_mask[PX_MASKSIZE-1:0]	Input	Data mask bits for Write Data. This mask is loaded into the FIFO coincident with the associated Write Data (pX_wr_data). One mask bit is associated with each byte of data. When a pX_wr_mask bit is High, the corresponding byte of data is masked (that is, not written to the memory).
pX_wr_underrun	Output	This active-High signal is the underrun flag. It indicates there was not enough data in Write Data FIFO to complete the transaction. The last valid data word is written continuously to finish the burst. To prevent underrun, make sure there is enough data in the FIFO when issuing a Write instruction to the Command FIFO. The sys_rst signal must be asserted to reset this flag and recover from this condition.

Read Datapath

Table 2-7 shows all signals related to the read datapath of the MCB User Interface. All port signal names have the prefix *pX*, where *X* represents the port number (for example, port 0 signals are prefixed with p0, port 1 with p1, and so forth).

Table 2-7: **Read Datapath Signals**

Signal Name	Direction	Description
pX_rd_clk	Input	This input is the user clock for the Read Data FIFO.
pX_rd_en	Input	This active-High signal is the read enable for the Read Data FIFO. Read Data is clocked out of the FIFO on the rising edge of pX_rd_clk when pX_rd_en = 1 and pX_rd_empty = 0.

Table 2-7: **Read Datapath Signals *(Cont'd)***

Signal Name	Direction	Description
pX_rd_data[PX_SIZE-1:0]	Output	Read Data value returning from memory. This signal is driven by the output of the Read Data FIFO into FPGA logic. PX_SIZE can be 32, 64, or 128 bits, depending on the port configuration.
pX_rd_full	Output	This active-High signal is the full flag for the Read Data FIFO. When High, this signal prevents additional data returning from the memory from being loaded into the FIFO.
pX_rd_empty	Output	This active-High signal is the empty flag for the Read Data FIFO. It indicates there is no valid data in the FIFO.
pX_rd_count[6:0]	Output	Count value for Read Data FIFO. This signal indicates how many user words are in the FIFO (from 1 to 64). A count value of 0 indicates the FIFO is empty. This signal has a longer latency than the pX_rd_full flag. Therefore, the FIFO could be full or experience overflow even when the count is less than 64.
pX_rd_overflow	Output	This active-High signal is the overflow flag. It indicates that data was lost due to Read Data continuing to return from the memory after the Read Data FIFO was full. To prevent overflow: • Make sure there is enough room to store the requested Read Data in the FIFO before issuing a Read instruction to the Command FIFO. • Be sure to account for any transactions in flight. The sys_rst signal must be asserted to reset this flag and recover from this condition.
pX_rd_error	Output	This signal indicates a Read Data FIFO error occurred because the FIFO pointers were unsynchronized. An MCB reset is required to recover from this condition.

Self-Refresh Signals

Table 2-8 shows the self-refresh signals accessible through the user interface. Self-refresh is covered in more detail in Chapter 4.

Table 2-8: **Self-Refresh Signals**

Signal Name	Direction	Description
selfrefresh_enter	Input	This input is rising-edge sensitive. When asserted, the MCB requests the memory device to enter self-refresh mode. The signal must remain asserted until the selfrefresh_mode signal goes active.
selfrefresh_mode	Output	This active-High signal indicates the memory device is in self-refresh mode.

Memory Device Interface

The Memory Device Interface contains all the necessary signals to communicate with the external memory device. All these signals (Table 2-9) have predefined pin locations in Spartan-6 devices. See UG385, *Spartan-6 FPGA Packaging and Pinouts Product Specification* for detailed MCB pinout information for each device/package combination. In addition, the soft calibration module generated by MIG requires allocation of an additional pin (RZQ) for all MCB designs. RZQ is a required pin, but its location can be moved within the MCB bank. When Calibrated Input Termination is selected in the MIG tool, a ZIO pin is also generated for use with the soft calibration module. The ZIO location can also be moved but must placed on a bonded I/O (i.e., a valid package pin) within the MCB bank. See the "Setting FPGA Options" section in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*, for more information on the RZQ and ZIO pins.

Note: All predefined pins revert to general-purpose I/Os when an MCB is unused. In addition, unused pins from an active MCB also revert to general-purpose I/Os. This includes higher order address or bank address pins not needed for a particular density device, DQ data bits not needed for a particular interface width, the reset and ODT signals when not needed for a memory standard, and the UDQS / UDQS_n strobes for x4 or x8 interfaces. All other interface pins are required for all MCB based designs. In addition, there are two exceptions to the general-purpose I/O recovery rules:

a. Data mask pins are paired such that if only LDM is used, UDM is lost as a general I/O. The data mask pins are required in all MCB designs to support variable burst length requests at the user interface. Therefore, both LDM and UDM are unavailable as general I/Os whenever the MCB is used.

b. Data strobe pins are paired such that if only DQS is used (single-ended strobe), DQS_n is lost as a general I/O. The same is true for UDQS and UDQS_n.

Table 2-9: **Memory Device Interface Signals**

Signal Name	Direction	Description
mcbx_dram_addr [C_MEM_ADDR_WIDTH–1:0]	Output	Address bus to the memory device. C_MEM_ADDR_WIDTH is set by the MIG tool depending on the memory device configuration (the maximum value is 15).
mcbx_dram_ba[2:0]	Output	Bank Address bus to the memory device. The MCB supports up to eight banks in a memory device.
mcbx_dram_cas_n	Output	This signal is the active-Low column address strobe to the memory device.
mcbx_dram_cke	Output	This active-High signal is the clock enable to the memory device.
mcbx_dram_clk	Output	This output is the differential clock (p output) to the memory device.
mcbx_dram_clk_n	Output	This output is the differential clock (n output) to the memory device.
mcbx_dram_ddr3_rst	Output	This signal is the DDR3 reset to the memory device.
mcbx_dram_dq [C_NUM_DQ_PINS–1:0]	Bidir	Bidirectional data bus to memory device. C_NUM_DQ_PINS is set by the MIG tool depending on the memory device configuration (valid values are 4, 8, and 16).

Table 2-9: **Memory Device Interface Signals *(Cont'd)***

Signal Name	Direction	Description
mcbx_dram_dqs	Bidir	Bidirectional data strobe for DQ[7:0]. This signal is an input during Read transactions and an output during Write transactions.
mcbx_dram_dqs_n	Bidir	Bidirectional complementary data strobe for DQ[7:0]. This signal is an input during Read transactions and an output during Write transactions.
mcbx_dram_ldm	Output	This output is the data mask for the lower data byte (DQ[7:0]) for x16, x8, or x4 configurations.
mcbx_dram_odt	Output	This output is the on-die termination signal. ODT is supported for DDR2 and DDR3.
mcbx_dram_ras_n	Output	This active-Low signal is the row address strobe to the memory device.
mcbx_dram_udm	Output	This output is the data mask for the upper data byte (DQ[15:8]) when interfacing to a x16 device.
mcbx_dram_udqs	Bidir	Bidirectional data strobe for DQ[15:8]. This signal is an input during Read transactions and an output during Write transactions.
mcbx_dram_udqs_n	Bidir	Bidirectional complementary data strobe for DQ[15:8]. This signal is an input during Read transactions and an output during Write transactions.
mcbx_dram_we_n	Output	This signal is the active-Low write enable to the memory device.
rzq	Bidir	Required pin for all MCB designs. When Calibrated Input Termination is selected in the MIG tool, the RZQ pin should have a resistor of value 2R from the pin to ground, where R is the desired input termination value. In all other cases, the RZQ pin should be left as a no-connect pin. The RZQ pin can be moved to any valid package pin within the MCB bank.
zio	Bidir	No Connect signal used with the soft calibration module when Calibrated Input Termination is selected. ZIO must be placed on a valid package pin within the MCB bank, and there should be no board trace attached to this pin (i.e., no connect). ZIO is not generated for designs that do not use Calibrated Input Termination.

Note: Refer to PCB Layout Considerations in Chapter 3 for board design requirements related to the CS#, ODT, and CKE pins of the memory device.

XILINX®

Chapter 3

Designing with the MCB

This chapter provides detailed information on how to design with the Spartan®-6 FPGA MCB. It contains these sections:

- Design Flow
- Supported Memory Devices
- Simulation
- Resource Utilization
- Clocking
- Migration and Banking
- PCB Layout Considerations

Design Flow

There are two supported design flows for the MCB:

- Non-embedded design flow
  - Conventional FPGA design with the Xilinx® ISE® tool flow
  - MIG tool is used within the CORE Generator™ tool for MCB designs
- Embedded design flow
  - Processor-based FPGA system design with EDK tool flow
  - IP Configurator in Xilinx Platform Studio (XPS) is used within the EDK environment for MCB designs

Both tool flows provide a simple method for developing a reliable interface to external memory devices. A step-by-step GUI driven flow allows an MCB based design to be configured and parameterized to meet the precise needs of the application.

The MIG tool flow actually has two "wrapper" levels: a lower-level wrapper (`mcb_raw_wrapper.v`) and the top-level wrapper (for example, `memc3_wrapper.v`). The lower-level wrapper incorporates all of the necessary silicon blocks (MCB, I/O, etc.) and soft logic (soft calibration module), required for the solution. It also provides access to all signals associated with the underlying hardware implementation of the User Interface ports and calibration logic. The top-level wrapper handles signal reassignment, tying off lower-level wrapper signals, as needed, and passing down the parameter values to the lower wrapper based on the user selections in the MIG tool.

The top-level wrapper presents a clean interface of only those signals needed to implement the MCB-based design as configured during the MIG tool flow. For example, while the lower-level wrapper always shows all 6 of the native 32-bit ports on the User Interface, the top-level wrapper reassigns signals, ties off unused ports, and concatenates buses to

present the port interface the user expects, such as a single 64-bit port. The top-level wrapper is the one that is subsequently integrated into the larger FPGA design.

The lower-level wrapper (`mcb_raw_wrapper.v`) is documented throughout this User Guide. The parameter and signal lists in Chapter 2, for example, are all described with respect to this lower-level wrapper. This wrapper does not change based on the choices made in the MIG tool GUI flow, whereas the top-level wrapper is customized as a result of the user selections.

In addition, the embedded design flow (EDK) uses the same lower-level wrapper as the foundation for creating the Multi-Port Memory Controller (MPMC) peripheral. The IP configurator in XPS allows the user to add the necessary soft bridges on top of the lower-level wrapper to create the desired peripheral interfaces, such as:

- PLB interface
- Xilinx Cache Link (XCL) interface
- Local Link (LL) interface
- Other Personality Interface Modules (PIMs) supported by EDK

Figure 3-1 illustrates how the lower-level wrapper is used for both the non-embedded (MIG) and embedded (EDK) design flows.

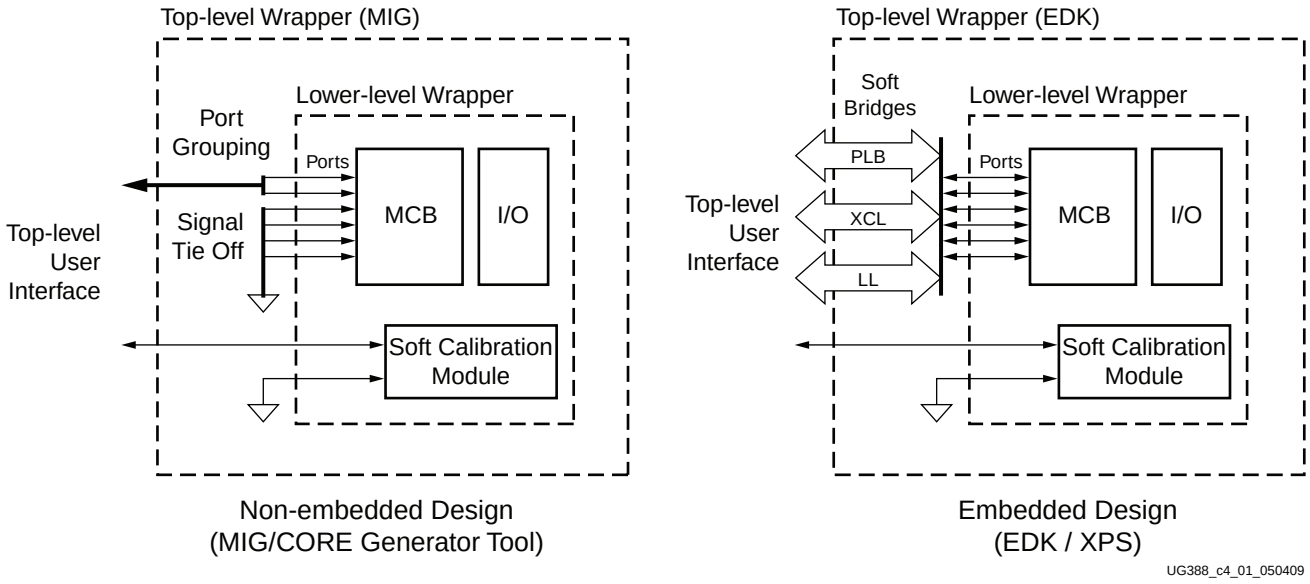

Figure 3-1: **Common Lower-Level Wrapper for Non-embedded and Embedded Design**

CORE Generator Tool

Figure 3-2 shows the high-level design flow for integrating an MCB based memory interface into a non-embedded (conventional) FPGA design. The "Getting Started" chapter in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*, provides a detailed step-by-step guide to Phase 1 of this design flow. Phase 2 and Phase 3 are outside the scope of this document, but detailed instructions on the ISE tool flow can be found elsewhere in the Xilinx Documentation Library.

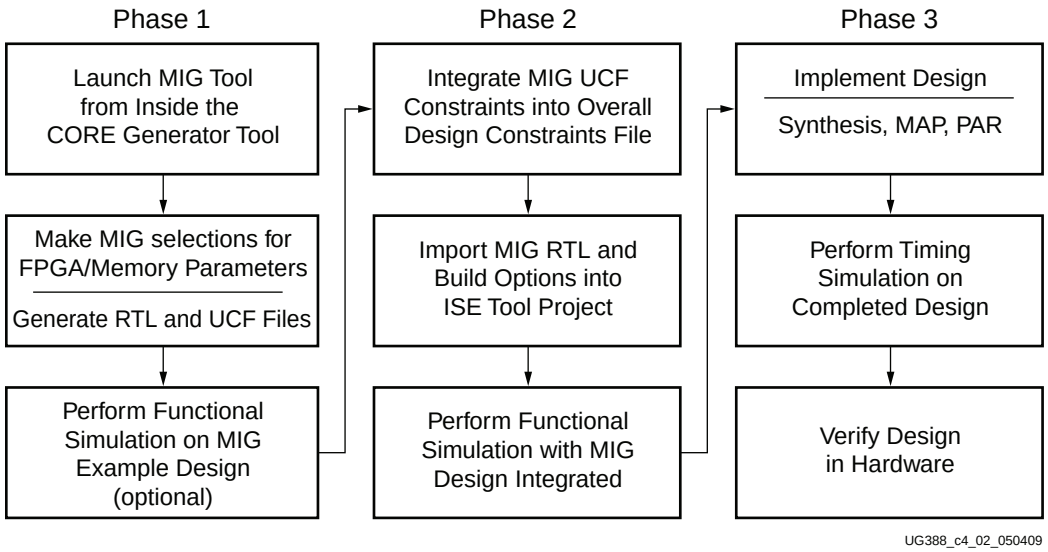

Figure 3-2: **MCB Design Flow for Non-embedded (Conventional) FPGA Applications**

Supported Memory Devices

Table 3-1 provides a list of memory devices to be verified to operate with the MCB on a Xilinx hardware verification platform. These devices can be selected in the MIG tool (or EDK) GUI flow from the drop-down list of supported devices. Xilinx will add devices to the MIG drop-down supported device list in future releases, but these devices will receive "simulation only" verification. Additionally, custom devices can be created by the user in the MIG tool; however, these do not have simulation or hardware verification by Xilinx. See the "Setting Controller Options" section in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*, for more information.

Table 3-1: **Supported Memory Devices for the MCB**

Standard	Vendor	Part Number	Width	Density
DDR3	Micron	MT41J64M16xx-187E	16	1 Gb
DDR3	Micron	MT41J256M8xx-187E	8	2 Gb
DDR3	Micron	MT41J128M8xx-187E	8	1 Gb
DDR3	Micron	MT41J256M4xx-187E	4	1 Gb
DDR3	Micron	MT41J512M4xx-187E	4	2 Gb
DDR2	Micron	MT47H256M4xx-25E	4	1 Gb
DDR2	Micron	MT47H64M8xx-25E-IT	8	512 Mb
DDR2	Micron	MT47H128M8xx-25	8	1 Gb
DDR2	Micron	MT47H128M16xx-3	16	2 Gb
DDR2	Micron	MT47H256M4xx-3	4	1 Gb
DDR2	Micron	MT47H16M16xx-3	16	256 Mb
DDR2	Micron	MT47H32M16xx-37E	16	512 Mb
DDR2	Micron	MT47H32M8xx-37E	8	256 Mb

Table 3-1: **Supported Memory Devices for the MCB *(Cont'd)***

Standard	Vendor	Part Number	Width	Density
DDR2	Micron	MT47J64M16xx-3	16	1 Gb
DDR2	Micron	MT47J256M4xx-37E	4	1 Gb
DDR2	Micron	MT47J128M8xx-3	8	1 Gb
DDR2	Elpida	EDE1116ACBG-8E	16	1 Gb
DDR2	Elpida	EDE5116AJBG-8E	16	512 Mb
DDR2	Hynix	HYB18TC512160B2F-2.5	16	512 Mb
DDR	Micron	MT46V32M16xx-5B-IT	16	512 Mb
DDR	Micron	MT46V32M8xx-5B	8	256 Mb
DDR	Micron	MT46V64M4xx-5B	4	256 Mb
LPDDR	Micron	MT46H32M16xxxx-5	16	512 Mb
LPDDR	Micron	MT46H16M16xxxx-6-IT	16	256 Mb
LPDDR	Micron	MT46H16M16xxxx-75-IT	16	256 Mb
LPDDR	Micron	MT46H64M16xxxx-5L-IT	16	1 Gb
LPDDR	Micron	MT46H64M16xxxx-6L-IT	16	1 Gb

Simulation

The simulation model of the underlying MCB contained within the MIG (or EDK) wrapper is encrypted as specified in Verilog LRM-IEEE Std 1364-2005. This is similar to other IP offered by Xilinx, such as the GTP transceiver and Integrated Endpoint blocks for PCI Express® designs.

Xilinx supports the following simulators for this encryption methodology:

- ModelSim 6.4b and above

The encrypted model of the MCB is automatically compiled when the usual COMPXLIB script is run, provided the appropriate version of the simulator is available on the computer. When running a simulation for a Verilog based design, the following library must be referenced: **`secureip`**.

For most simulators, this can be done by using the **`-L`** switch as an argument to the simulator, such as **`-L secureip`**.

Note: If VHDL is used as the design entry language, a mixed-language license is required for ModelSim to simulate designs that include the MCB.

For more information on simulating IP blocks using the secureip methodology, see UG626, *Synthesis and Simulation Design Guide*.

Resource Utilization

The MIG (or EDK) wrapper produced by the GUI design flow incorporates all of the device resources required for implementation of an MCB based memory interface. For the most part, the wrapper files are simply managing signal name reassignment and connectivity between the silicon resources (for example, MCB to I/O block connections), and thus do not consume any measurable FPGA logic. However, the soft calibration module contained within the wrapper does consume a small amount of FPGA logic resources. In addition, there are specific clocking requirements for the MCB (see Clocking) that result in use of some general clocking resources.

Table 3-2 shows the resource utilization associated with an MCB design, excluding any logic required in the user design to control the User Interface ports. This table uses a DDR3 interface with eight banks, differential strobes, and data masking to calculate a maximum pin count. Other memory standards and configurations use fewer pins.

Table 3-2: **Resource Utilization for Each MCB Based Memory Interface**

Resource	Memory Interface Width		
	x4	x8	x16
Memory Controller Block (MCB)	1	1	1
Predefined I/O Pins: Address, Data, Control, etc.	35	39	50
Soft Calibration Module Logic	< 100 Slices	< 100 Slices	< 100 Slices
PLL Block	1	1	1
BUFPLL_MCB Buffer	1	1	1

Clocking

This section describes the clocking requirements for implementing a memory interface based on the MCB. The MIG (or EDK) tool automatically generates a clocking infrastructure that fully complies with these requirements. The MCB requires three basic types of clocks:

- MCB system clocks determine the operating frequency of the memory controller and physical interface to the external memory device.
- Calibration clock determines the operating frequency of the calibration logic.
- User clocks determine the operating frequency of the User Interface ports. These clocks can be completely asynchronous to the system and calibration clocks. The Command and Data Path FIFOs handle the necessary clock domain transfer from the User Interface to the internal controller logic.

Figure 3-3 shows the recommended clock distribution scheme for the MCB system and calibration clocks. The MCBs are located in the I/O regions on the left and right side of the device, and must therefore be driven by the I/O clock network. The I/O clock network is designed for significantly higher frequencies than the global clock network, allowing memory interfaces to operate at up to 800 Mb/s.

Note: CLKOUT0 and CLKOUT1 are the only outputs of the PLL that can be connected to the BUFPLL_MCB driver. These connections must be made exactly as shown in Figure 3-3.

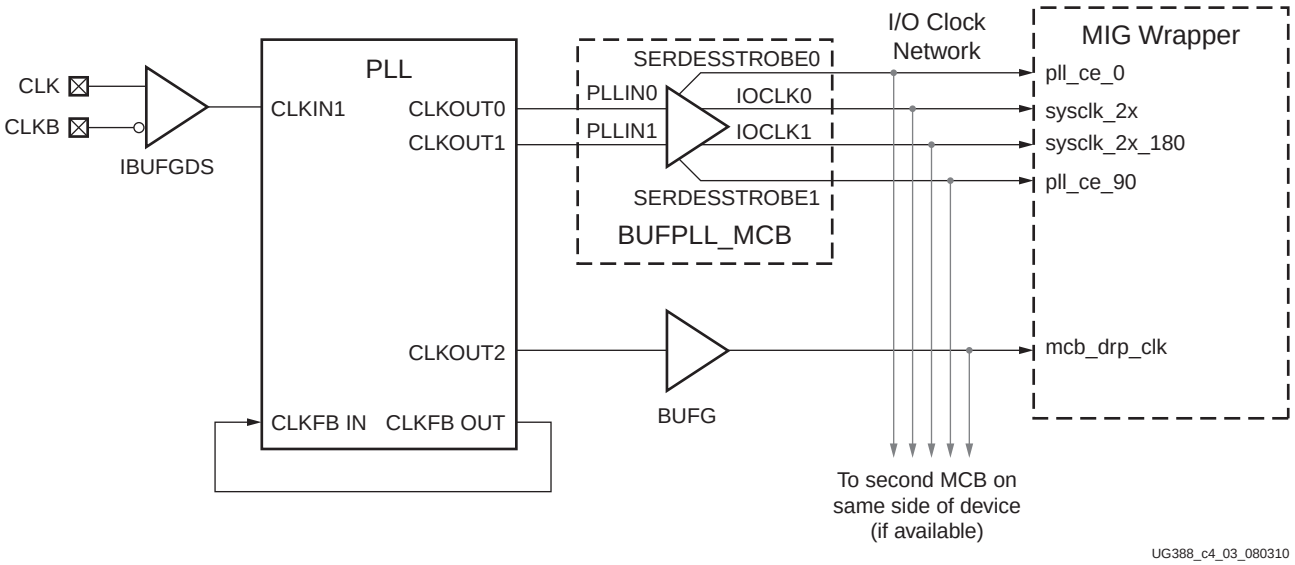

Figure 3-3: **Recommended System and Calibration Clock Distribution**

To create the desired system clock frequency on the I/O clock network, an external clock source drives one of the PLLs in the center column of the device. The external clock frequency is not critical as long as the PLL can synthesize the desired MCB system clocks from it. In general, the preferred PLL location is the one nearest the center of the device that minimizes the physical distance between the PLL and the BUFPLL_MCB block. This PLL location is strongly recommended for larger devices with six PLLs.

The PLL generates two system clock outputs, sysclk_2x and sysclk_2x_180, that are twice the frequency of the desired memory clock (for example, for a 667 Mb/s DDR2 interface with a memory clock equal to 333 MHz, the system clocks are set to 667 MHz) and 180 degrees out of phase from each other. Only two clock lines are available on each side of the device to drive the I/O clock network from the PLLs. The pair of system clocks uses these two clock lines to connect to the MCBs on the left or right side of the device. Thus for devices with four MCBs, the two MCBs on the same side of the device must share the same system clock pair and therefore must run at the same data rate, although the memory standard implemented can be different. DCMs do not have access to the I/O clock network and cannot, therefore, be used to drive MCBs. It is also possible to drive MCBs on both sides of the device from a single PLL. In this case, two BUFPLL_MCB blocks (one on each side of the device) must be driven by the shared PLL.

When the pair of system clocks reaches the I/O clock network, they are rebuffered by a BUFPLL_MCB driver. This driver also creates clock enable strobes required by the MCB: pll_ce_0 and pll_ce_90. The attributes of the BUFPLL_MCB primitive should be set as follows to create the necessary clock enable strobe behavior for the MCBs:

- LOCK_SRC = “LOCK_TO_0”
- DIVIDE = 2

The rebuffered full rate system clocks (2X clocks) are used in the PHY layer of the interface to create the necessary double data rate (DDR) signaling at the I/O pins (for example, a 667 MHz clock is used to generate an effective 667 Mb/s DDR signal at the I/O). A divide-by-two circuit in the MCB creates what is traditionally considered the memory clock frequency (for example, 333 MHz for a 667 Mb/s DDR interface). These 1X clocks drive the controller, arbiter, and other single data rate (SDR) logic.

The calibration related clock, mcb_drp_clk, must be generated by the PLL and must be phase-synchronized (i.e., in phase) with the sysclk_2x domain. The calibration clock rate is limited by normal static timing analysis, with a typical achievable frequency of 100 MHz. In general, a calibration clock frequency of at least 50 MHz should be used to allow the MCB to complete calibration operations in a reasonable period of time.

A set of user clocks is associated with each of the User Interface ports (port number X = 0 to 5) used in a given design, as follows:

- pX_cmd_clk: Command FIFO user clock for clocking in the Address, Instruction, and Burst Length from the FPGA logic into the FIFO.
- pX_wr_clk: Write Data FIFO user clock for loading write data from the FPGA logic into the FIFO in preparation for a burst to memory.
- pX_rd_clk: Read Data FIFO user clock for clocking out data returning from the memory into the FPGA logic.

The user clocks are completely asynchronous from the system and calibration clocks and therefore they can operate at any frequency dictated by the FPGA logic portion of the design. The FIFOs inside the MCB handle the necessary clock domain transfer. For best utilization of the available memory bandwidth, the user clocks should be set at or above the frequency determined by the ratio of the User Interface to the external Memory Device interface. For example:

- For a DDR3 800 Mb/s interface with the memory clock = 400 MHz and a x8 bit memory device:

  The result is 16 bits of data transfer per clock cycle (8 bits on each clock edge)

- For a x64 bit User interface:

  The user clock should be set at or above (16/64) * 400 MHz = 100 MHz

While not technically required, it is also highly recommended that all three user clocks for a port (pX_cmd_clk, pX_wr_clk, and pX_rd_clk) be driven by the same clock source from the FPGA logic to avoid complex timing and synchronization issues in the user design.

Modifying the Clock Setup

By default the MIG tool sets up the clocking infrastructure assuming that the user input clock (CLKIN1 to the PLL) is operating at the memory clock frequency. To modify the clocking setup to create the necessary MCB clocks from a different input clock frequency or to adjust the user or calibration clock frequencies, these PLL parameters can be adjusted at the top level of the MIG example or user design:

- C*x*_CLKFBOUT_MULT
- C*x*_DIVCLK_DIVIDE
- C*x*_CLKOUT0_DIVIDE (for sysclk_2x)
- C*x*_CLKOUT1_DIVIDE (for sysclk_2x_180)
- C*x*_CLKOUT2_DIVIDE (for user clock)
- C*x*_CLKOUT3_DIVIDE (for calibration clock)

where *x* represents the MCB block number.

There are two options for determining the correct values for these parameters:

1. Use the Clocking Wizard found in the CORE Generator tool to determine the appropriate parameter settings based on the desired input and output clock frequencies for the PLL. Choose **Manual Selection** and the **PLL_BASE** primitive on

the opening dialog page to ensure that a PLL is used. Only the listed PLL parameter values produced by the Clocking Wizard should be transferred back into the MIG design. No other output from the Clocking Wizard is needed. The Clocking Wizard also determines the resulting output jitter from a specific PLL configuration that can be used to validate the main MCB system clocks against the memory device input clock jitter requirements.

2. Refer to the "PLL" chapter in UG382, *Spartan-6 FPGA Clocking Resources User Guide* to verify the proper settings of the listed PLL parameters for the desired input and output clock frequencies for the PLL. This method requires a better understanding of aspects such as keeping the PLL VCO operating frequency within the specified limits.

Migration and Banking

The MCB located on the left side (for devices with two MCBs) or lower-left side (for devices with four MCBs) of the device is the most flexible to design with in most situations (see Figure 3-4). The predefined pins for the MCB in this location have the fewest number of "multipurpose" pin functions, while the MCBs on the right side of the device tend to have pins with more shared functionality.

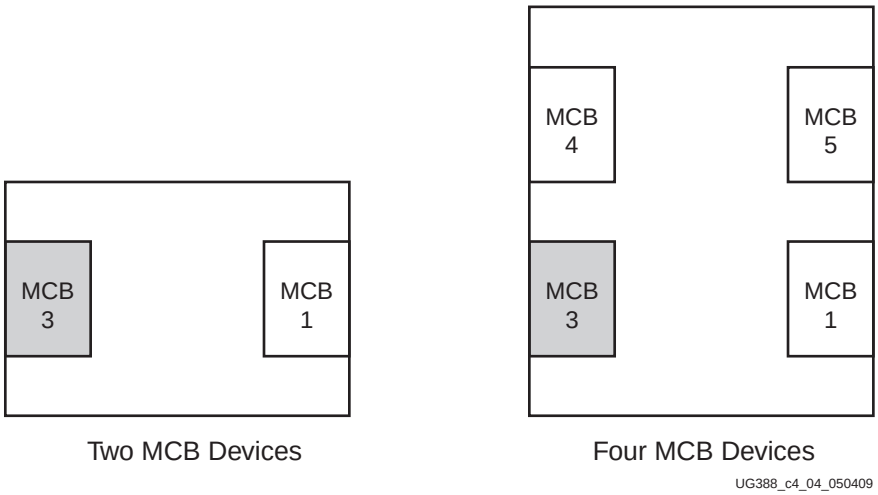

Figure 3-4: **MCB3 is the Preferred Location for Migration and Pin Flexibility**

For example, some bank 1 MCB pins are shared with the Byte-wide Peripheral Interface (BPI) pins that can be used to configure the Spartan-6 device from a parallel flash device. These dual-purpose pins can be used for a BPI or an MCB interface, but not both. Thus, it is necessary to consider what other components will be in the system, and how they will interface to the Spartan-6 device when planning the MCB interfaces. The MCBs on both sides of the device have pins that are shared with global clock (GCLK) pins and PCI pins, but overall the left (or lower left) side MCB has the fewest restrictions related to pin usage.

In addition, it is possible to migrate between Spartan-6 family members in the same package type (for example, migrate from an LX16 device to an LX25 device in the same CSG324 package) while maintaining the same MCB predefined pin locations. This applies to all MCB locations. In general, any particular device can migrate up or down at least one device density in the same package type. Refer to the *Spartan-6 Family Overview* for more details on available devices and package types.

PCB Layout Considerations

This section lists PCB layout considerations, which should be reviewed before beginning board design for an MCB based memory interface. The *Spartan-6 FPGA PCB Designer's Guide* should also be consulted for information regarding proper device decoupling, overall power distribution system design, and other general PCB guidelines. Additional references for PCB layout and signal integrity analysis of DDR memory interfaces can be found in Appendix A, References.

All trace length calculations assume an average 165 ps of electrical delay per inch of signal trace.

General Guidelines

- Only internal PCB layers should be used to route memory interface signals between the FPGA and memory devices. Breakout vias to connect component balls are excluded from this requirement.
- Top or bottom layer routing can be considered for routing to external termination resistors (if used) when placed in a fly-by mode after the memory component.
- Memory interfaces without external terminations should have a maximum of two vias.
- Memory interfaces with external terminations should have a maximum of three vias.
- Once a signal is broken out to an internal signal layer, it must complete its routing on that layer. Terminating the signal to a via permits final routing to the component pad via and connection at the top or bottom layer of the board. PCB layer hopping is not allowed.
- Overall trace length should be minimized. Traces should be 3 inches or less.
- Trace widths should be 3 to 5 mils.
- Trace spacing should be three times the trace width.
- Signals must not be routed over splits or voids.
- Routing of differential pairs adjacent to noisy signal lines or high-speed switching devices such as clock chips should be avoided.
- The spacing between differential clocks/strobes and other signals on the same PCB layer should be 20 mil. The 20 mil spacing should be maintained when using serpentine routing for length matching.
- Differential clocks/strobes are to be routed as 100Ω differential signals. The clock pairs must be routed on the same PCB layer with no layer changes or hops after the initial pad to via breakout.
- Series terminations (if used) should be as close to the FPGA as possible.
- Parallel terminations (if used) should be as close to the DRAM as possible.
- Parallel termination resistors should be placed on a top or bottom layer V_{TT} island.
- Board designers should ensure that clock lines are routed differentially and correct trace widths or clearances maintained to achieve the target differential impedance. Routing the signals differentially reduces the flight time of the clocks when compared to the single-ended signals. Because of this, most DDR2 design guides recommend that clock signals be routed at the same length or longer than the address, control, and command signals to compensate for this timing variation.

Data, Data Mask, and Data Strobe Guidelines

The Data (DQ), Data Mask (DM), and Data Strobe (DQS) signals should receive the highest priority (that is, routed first), because they are the highest speed DDR signals.

- DQ, DM, and DQS signals should be routed in a data group (per byte). Each group should have similar loading and routing to maintain timing and signal integrity.
- The provided spacing should be 20 mil between a data group and any other signals.
- DQS signals should be isolated from other signals by 20 mil to avoid crosstalk.
- There should be a maximum of ±25 ps electrical delay (±150 mil) between any DQ/DM and its associated DQS strobe.
- A data group should be referenced to a GROUND plane.
- DQ bit swapping at the memory interface is permitted to facilitate layout. Swapping should only be done within a data group.
- DQS to DQS_N trace lengths should be matched (±10 mil).
- Memory terminations (if external terminations are used) should be placed after the associated memory component in a fly-by fashion.
- For 16-bit DDR devices, the LDQS/LDQS_N and UDQS/UDQS_N trace lengths should be matched within ±25 ps of the electrical delay (±150 mil).

Address, Control, and Clock Guidelines

When the data groups have been routed, the next highest priority is the differential clock (CK / CK_N). The clock should be routed first because all address and control trace length matching must be referenced to the differential clock PCB trace length, which might need to be adjusted as the layout task proceeds.

- CK to CK_N trace lengths must be matched (±10 mil).
- CK and DQS trace lengths must be matched (±250 mil) to maximize setup and hold margins.
- There must be a maximum ±50 ps electrical delay (±300 mil) between any address/control signals and the associated CK and CK_N differential clock FPGA output.
- Address and control signals can be referenced to a POWER plane if a GROUND plane is not next to this group of signals in the PCB stack-up.
- To avoid crosstalk, address and command signals should be kept on a different routing layer from DQ, DQS, and DM.
- Differential clock terminations (if external terminations are used) must be located as close as possible to the load, after the clock pads of the PCB. PCB trace lengths used in trace length matching must exclude the CLINE length of the PCB trace from memory ball to terminating resistor.
- Memory terminations should be placed (if external terminations are used) after the associated memory component in a fly-by fashion.

Additional Board Design Requirements

In addition to the PCB layout guidelines detailed in this section, these board design requirements must be implemented:

- The active-Low Chip Select (CS#) pin of the target memory device should be connected to ground on the board. Because the MCB only supports connections to a single memory component, it does not provide a signal to control the CS# input. Contact your memory vendor for more information, if needed.
- For DDR3 memory devices, the RESET and CKE signals should each have a 4.7 kΩ resistor to ground to ensure that these signals are Low during memory initialization.
- For DDR2 memory devices, the ODT and CKE signals should each have a 4.7 kΩ resistor to ground to ensure that these signals are Low during memory initialization.
- If the HSWAPEN pin on the Spartan-6 device is Low or grounded during configuration, internal pull-ups to V_{CCO} are enabled on all device I/O pins until configuration completes, including any V_{REF} pins associated with MCB usage. If the V_{REF} level is generated from a resistor divider, the temporary internal pull-ups might elevate the level of the V_{REF} pins during device configuration. The designer should ensure that the MCB is held in reset until the V_{REF} level has returned to a stable level to avoid MCB calibration and operation with an invalid V_{REF}. For more information, see the "Configuration" section in the "I/O Pin and Clock Planning" chapter of UG393, *Spartan-6 FPGA PCB Design and Pin Planning Guide*.

Simultaneous Switching Output Considerations

As noted in Block Diagram in Chapter 1, the MCB utilizes the general I/O blocks (IOBs) associated with the predefined pin locations to create the external interface to the memory device. The MIG tool automatically configures these IOB locations to implement the required I/O standard for the selected memory type (e.g., SSTL18 for DDR2, SSTL15 for DDR3).

Hardware characterization of MCB based memory interfaces indicates there are no Simultaneous Switching Output (SSO) related restrictions when using the predefined IOB locations up to their maximum extent (i.e., the maximum number of data and address pins in the interface). Refer to DS162, *Spartan-6 FPGA Data Sheet: DC and Switching Characteristics* and UG361, *Spartan-6 FPGA SelectIO Resources User Guide* for more information on SSO characteristics. When placing signals on any remaining pins in the bank, it is recommended that they are used as follows:

Single-ended output with the low drive strength, unterminated standards LVCMOS 4 mA or LVCMOS 2 mA

XILINX®

Chapter 4

MCB Operation

This chapter provides detailed information on the operation of the Spartan®-6 FPGA MCB. It contains these sections:

Startup Sequence

Figure 4-1 shows the startup procedure for the MCB. After the FPGA has been fully configured and the PLL providing the system clocks has locked, a number of initialization and calibration steps are automatically performed by the MCB to prepare it for normal operation.

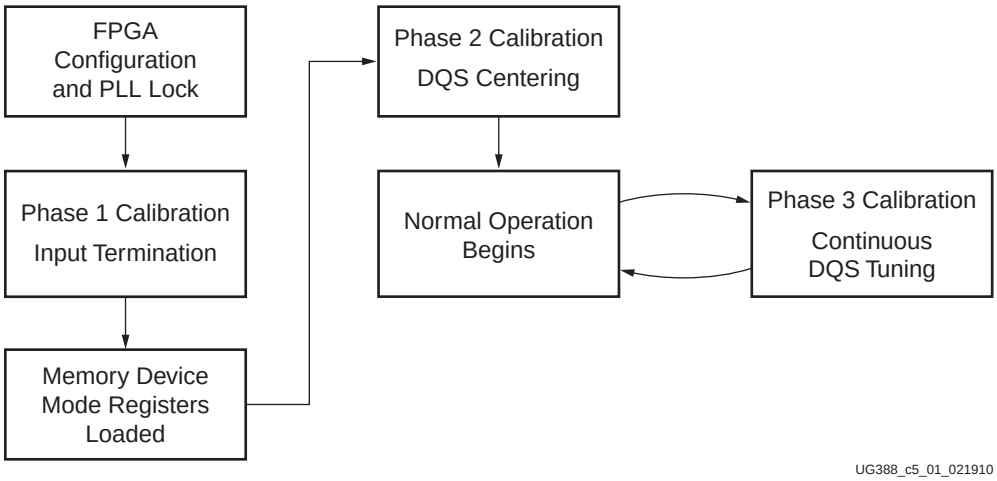

Figure 4-1: **MCB Startup Sequence**

Notes relevant to Figure 4-1:

1. The soft calibration module implements some aspects of Phases 1, 2, and 3 of calibration.
2. The MCB hard calibration logic does NOT perform individual per-bit deskew of the DQ data bus. Follow the guidelines in PCB Layout Considerations, page 41 to ensure that DQ/DQS board traces are properly length matched.

The first major operation is Phase 1 of calibration. In this step, the soft calibration module measures the value of an external resistor on the RZQ pin to determine the desired on-chip Input Termination value for several of the pre-defined MCB pins (e.g., DQ bus). This only occurs if the user selects the Calibrated Input Termination option in the MIG GUI flow (see the "Setting FPGA Options" section in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*). Otherwise an approximate uncalibrated on-chip termination or external termination is assumed, and this startup step is skipped.

The second major step of the startup sequence is to load the memory device mode registers with the desired parameters.

After the memory device has been configured, Phase 2 of calibration occurs. This phase adds delay to the input path of the DQS strobes entering the FPGA. The goal is to shift the DQS strobes into the center of what becomes the Read Data capture window.

Once all of the operations in the startup sequence have completed, the MCB enters normal operation. Commands and Data can be loaded into the User Interface FIFOs while the startup sequence is in progress, but no commands are executed until calibration completes and the block enters normal operation.

During normal operation, the soft calibration module continuously monitors the tap delay values of the IDELAY element used to delay the DQS input paths (for more information on IDELAY, see the *Spartan-6 FPGA SelectIO™ Resources User Guide*). The intent is to measure any change in the per tap delay value due to voltage or temperature variations during operation. If a shift in tap delay value is detected, the tap delay count on the DQS strobe input paths can be adjusted to keep them centered in the Read Data capture window. The update to the IDELAY values is done during memory REFRESH operations to avoid impacting normal data operations and controller efficiency. Phase 3 of calibration is known as continuous DQS tuning.

See Calibration for more details on all phases of calibration.

Calibration

To achieve optimum signal integrity and maximum timing margin (hence, highest performance) for the memory interface, the MCB automatically performs several forms of calibration as briefly outlined in Startup Sequence, page 45. The hard calibration logic in the MCB and the soft calibration module generated by the MIG tool (or EDK) work together to implement a reliable and flexible calibration scheme. Each phase of calibration is discussed in greater detail below.

Note: The descriptions of calibration phases 2 and 3 in this section assume that the C_MC_CALIBRATION_MODE attribute is set to "CALIBRATION" as described in Table 2-2, page 20.

Phase 1: Input Termination

On-chip termination reduces component count and improves signal integrity by moving the termination as close to the endpoint of the signal transmission as possible. The MIG and EDK GUI interfaces allow "Calibrated Input Termination" to be selected for the MCB pre-defined pins. This feature creates an on-chip input termination on MCB pins that has been calibrated based on an external resistor, making it more precise than when using the "Uncalibrated Input Termination" option.

The soft calibration module uses two I/O pins, RZQ and ZIO, generated by the MIG tool (or EDK) to perform calibration of the input termination. RZQ is a required pin for all MCB designs. When Calibrated Input Termination is used, a resistor must be connected between the RZQ pin and ground with a value that is twice (2R) that of the desired input impedance (e.g., a 100Ω resistor to achieve a 50Ω effective input termination). RZQ should be left as a no-connect (NC) pin for designs not using Calibrated Input Termination. In addition, the RZQ pin must be within the same I/O bank as the memory interface pins.

The ZIO pin is only required for designs using Calibrated Input Termination and must be a no-connect pin (i.e., not connected to any PCB trace) assigned to a valid package pin (i.e., bonded I/O) location within the MCB bank. The default locations of the RZQ and ZIO pins can be found in the UCF constraints files.

The soft calibration module relies on the V_{REF} supply required for SSTL I/O standards to perform the necessary input termination calibration. LPDDR memory does not use calibrated input termination or an SSTL type I/O standard and therefore does not require V_{REF}.

Phase 1 of calibration effectively measures the value of the external 2R resistor and programs the I/O blocks of the MCB pins to create a split termination between V_{CCO} and GND. This scheme creates a Thevenin equivalent termination to V_{CCO} / 2 with value R as shown in Figure 4-2. The resulting input termination is dynamically disabled when the MCB pin is driven as an output (e.g., a DQ data pin during a Write transaction) and enabled at all other times to properly terminate incoming signals.

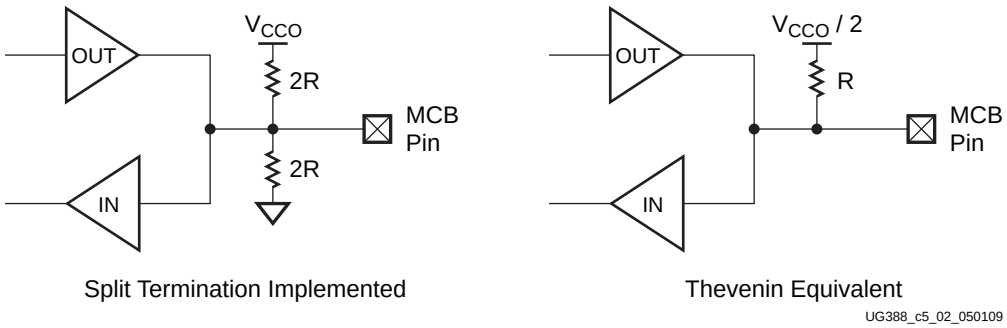

Figure 4-2: **Calibrated Input Termination**

Phase 2: DQS Centering

For optimal performance and maximum timing margin, the DQS strobe edges must be centered in the Read Data capture window with respect to the input capture flip-flop. Phase 2 of calibration is responsible for this DQS centering operation.

The DDR memory device output pins transmit the Read Data (DQ) and DQS strobes edge-aligned to the FPGA input pins as shown in Figure 4-3. For reliable operation, the DQS strobe must be delayed with respect to the DQ bits so that it captures the Read Data away from the transition region of the data bus.

During this phase, the tap delay count of the IDELAY block in the DQS strobe input path is incremented to shift the internal DQS signal at the capture flip-flop into the center of what will become the Read Data capture window as shown in Figure 4-3.

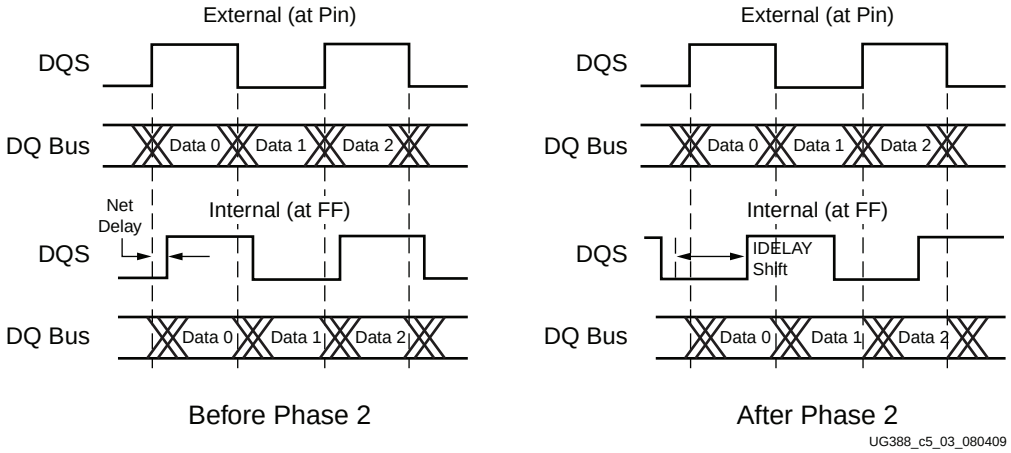

Figure 4-3: **Phase 2 of Calibration - DQS Centering**

Phase 3: Continuous DQS Tuning

Voltage and temperature variations during operation cause changes in the IDELAY tap values. Because the DQS strobe is delayed by half a bit period more than the DQ bits, it uses significantly more IDELAY taps. Therefore, if the per tap delay value of the IDELAY elements changes in response to voltage or temperature drift, the delay on the DQS strobe input path sees a disproportionate shift relative to the DQ bits.

To compensate for voltage and temperature related shift of the DQS strobes, Phase 3 of calibration runs continuously during normal operation. It uses the soft calibration module

to continuously monitor the tap delay values of the IDELAY elements used to delay the DQS input paths. If a shift in tap delay value is detected, the tap delay count on the DQS strobe input paths can be adjusted to keep them centered in the Read Data capture window. The update to the IDELAY values is done during memory REFRESH operations to avoid impacting normal data operations and controller efficiency.

Instructions

Table 4-1 provides detailed descriptions for all memory instructions implemented by the MCB. To load an instruction into the Command FIFO of a User Interface port, the 3-bit code for the instruction is clocked into the pX_cmd_instr[2:0] inputs on the rising edge of pX_cmd_clk.

Table 4-1: **Instructions Implemented by the MCB**

Instruction	Code [2:0]	Description
Write	000	Memory Write. Writes the number of data words specified by pX_cmd_bl[5:0] to the memory device beginning at the byte address specified by pX_cmd_addr[29:0]. Prior to issuing this instruction, sufficient data must be loaded into the Write Data FIFO to complete the transaction. Otherwise a data "underrun' condition occurs. This instruction is valid for write only and bidirectional ports.
Read	001	Memory Read. Reads the number of data words specified by pX_cmd_bl[5:0] from the memory device beginning at the byte address specified by pX_cmd_addr[29:0]. Prior to issuing this instruction, the Read Data FIFO must have enough space to complete the transaction. Otherwise a data "overflow" condition occurs. This instruction is valid for read only and bidirectional ports.
Write with Auto Precharge	010	Memory Write with Auto Precharge. This instruction is the same as the Write instruction but with auto precharge appended after burst completion. Auto precharge closes the DRAM bank where the transaction ended. This can improve latency for applications with more random access patterns that tend to jump between rows in the same bank. **Note:** The MCB looks ahead at subsequent transactions. The auto precharge is skipped if the following transaction is to the same row accessed in the current transaction.
Read with Auto Precharge	011	Memory Read with Auto Precharge. This instruction is the same as the Read instruction but with auto precharge appended after burst completion. Auto precharge closes the DRAM bank where the transaction ended. This can improve latency for applications with more random access patterns that tend to jump between rows in the same bank. **Note:** The MCB looks ahead at subsequent transactions. The auto precharge is skipped if the following transaction is to the same row accessed in the current transaction.
Refresh	1xx	Memory Refresh. Prompts the MCB to issue a refresh command to the memory device. Resets the tREFI counter allowing data to stream uninterrupted for a full refresh cycle. This instruction should only be used for highly customized dataflow structures. In general, the MCB automatically issues refresh commands on its own, which periodically results in increased latency for transactions.

Addressing

From the User Interface perspective, the MCB provides a simple and sequential byte addressing scheme into the physical DRAM. The fact that DRAMs store data in fixed segments is abstracted by this scheme, allowing for a simple SRAM-like address interface. For details on how the bank, row, and column address bits are mapped to the byte address, refer to Byte Address to Memory Address Conversion, page 61.

Table 4-2 shows how the byte address presented to the User Interface must be aligned to the port width. Depending on the number of bytes in the port width, a certain number of the low address bits must be set to 0 to ensure that consecutive addresses fall on data word boundaries. The write data mask inputs (pX_wr_mask) to the User Interface can be used to offset the starting address byte location. For example, to begin writing at byte address `0x01` when using a 32-bit (4-byte) User Interface, the byte address presented to the command port of the User Interface should be `0x00` to meet the requirements of Table 4-2, but the least significant mask bit should be set to 1 such that only bytes at address `0x01` and higher are actually written.

Table 4-2: **Address Requirements for Byte Address Alignment**

Port Width	Bytes per Data Word	Address Requirement
32 bits	4	pX_cmd_addr[1:0] = `2'b00`
64 bits	8	pX_cmd_addr[2:0] = `3'b000`
128 bits	16	pX_cmd_addr[3:0] = `4'b0000`

It is also important to understand the addressing relationship when 32-bit and 64-bit ports are used together in the User Interface (see Port Configurations, page 17). For 32-bit ports the memory appears to be aligned on 4-byte boundaries, while for 64-bit ports the memory appears to be aligned on 8-byte boundaries. Table 4-3 shows how two data words for a 32-bit port map into the address space of a single data word for a 64-bit port.

Table 4-3: **32-bit vs. 64-bit Port Address Relationship**

32-bit Port		64-bit Port	
Address	**Data**	**Address**	**Data**
`0x00`	[31:0]	`0x00`	[31:0]
`0x04`	[31:0]		[63:32]
`0x08`	[31:0]	`0x08`	[31:0]
`0x0C`	[31:0]		[63:32]

Command Path Timing

The command path of the User Interface uses a simple 4-deep FIFO structure to hold pending commands. The instruction type, address, and burst length for the requested transaction are all loaded into this Command FIFO. The full flag (pX_cmd_full) signal from the command FIFO must be Low for a new command to be accepted into the FIFO when pX_cmd_en is asserted during the rising edge of pX_cmd_clk. Otherwise, the command is ignored. Figure 4-4 and Figure 4-5 demonstrate the protocol for loading a command into the FIFO.

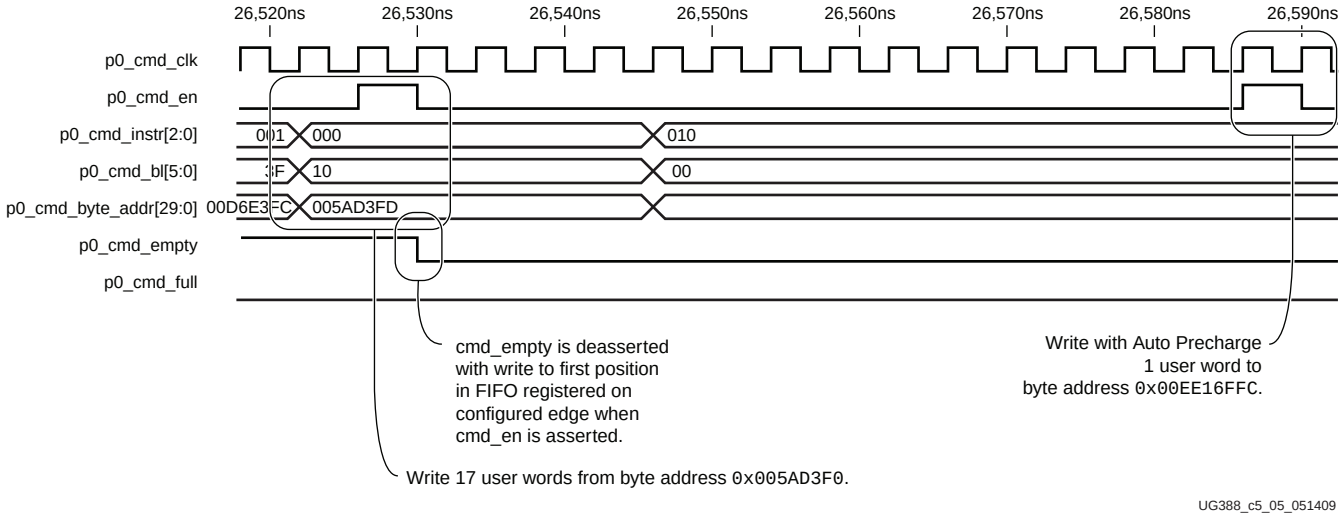

Figure 4-4: **Command Path Timing (Write)**

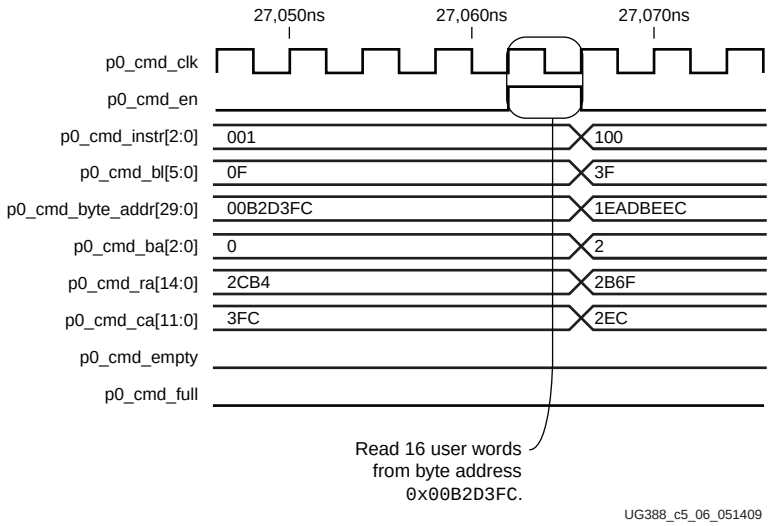

Figure 4-5: **Command Path Timing (Read)**

Write Path Timing

The write path of the User Interface uses a simple 64-deep FIFO structure to hold data in preparation for a Write transaction to memory. Similar to the Command FIFO, the full flag (pX_wr_full) from the Write Data FIFO must be Low for new data to be accepted into the FIFO when pX_wr_en is asserted during the rising edge of pX_wr_clk. Otherwise, the data is ignored. If the full flag is Low, the pX_wr_data bus is captured into the FIFO on the rising edge of pX_wr_clk. For every clock cycle that pX_wr_en is asserted, there must be valid data on the pX_wr_data bus. Figure 4-6 demonstrates the protocol for loading data into the Write Data FIFO.

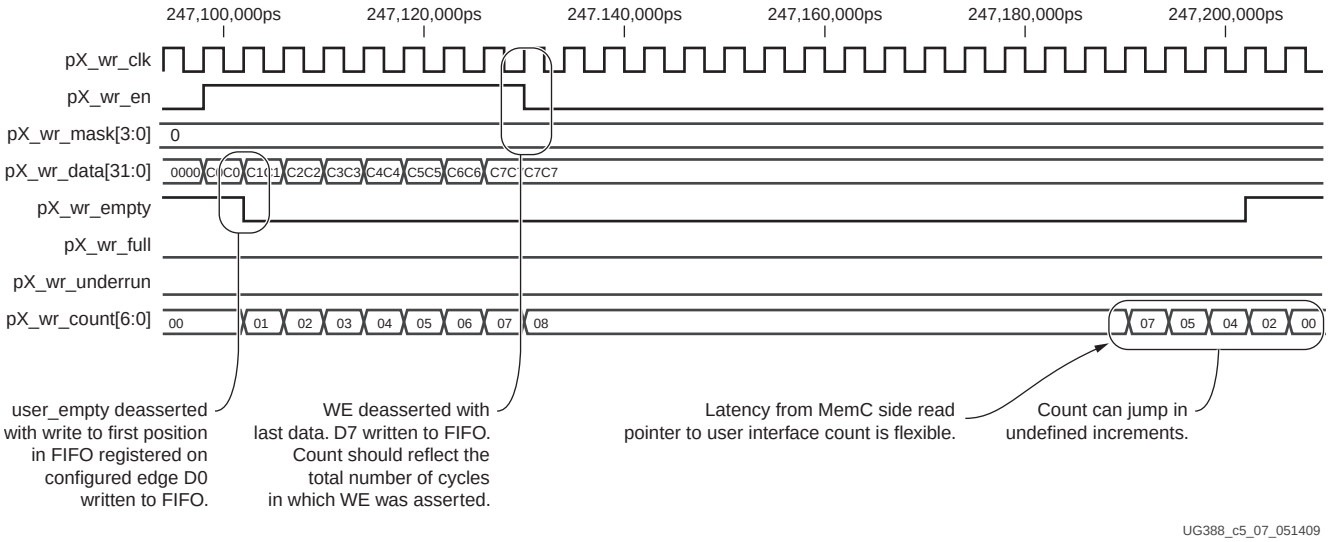

Figure 4-6: **Write Path Timing**

The pX_wr_underrun signal indicates to the user that the memory controller has attempted to send more data than was present in the write data FIFO and that the data which was intended for the memory never reached the memory. This condition must be avoided to guarantee reliable operation. To avoid an underrun condition, the user must guarantee that all necessary data is available in the write data FIFO to accommodate a transaction before committing that transaction to the command FIFO.

The count signal bus (pX_wr_count) provides a count of the number of entries in the FIFO. Due to the asynchronicity of the FIFOs in the MCB, the count signal bus has a longer latency than the empty and full flags. Therefore, this bus should only be used for intermediate references and watermarks. The count will transition immediately with respect to FIFO operations committed by the user. However, it take s longer for operations committed by the controller to be apparent on the count signals than the full or empty signals. Thus for the Write Data FIFO as the FIFO is filling, the count always reports at least as many entries as are in the FIFO.

For example, if the user has written eight words into the FIFO, the count might report eight even though somewhere during the process of writing to the FIFO, the controller could have started pulling data out of the FIFO. Additionally, if the controller continues to transmit the data to the memory, the count could still be showing entries in the FIFO even though the FIFO is already empty. For the Write Data FIFO, it is perfectly suitable to use the count signal bus as an almost full flag because the FIFO will never be full if the count is reporting less than full. However, it is very important to use other methods to ensure underrun conditions do not occur.

Read Path Timing

The read path of the User Interface uses a simple 64-deep FIFO structure to hold data returning from a Read transaction. The empty flag (pX_rd_empty) from the Read Data FIFO can be used as a data valid indicator. Whenever pX_rd_empty is deasserted, there is valid data present on the pX_rd_data bus. To transfer data into the FPGA logic from the Read Data FIFO, the pX_rd_en signal must be asserted on the rising edge of pX_rd_clk. The pX_rd_data bus transitions on the rising edge of pX_rd_clk. The pX_rd_en signal can remain asserted at all times and the pX_rd_empty signal can be used as a data valid indicator, if desired. Figure 4-7 demonstrates the protocol for loading data out of the Read Data FIFO.

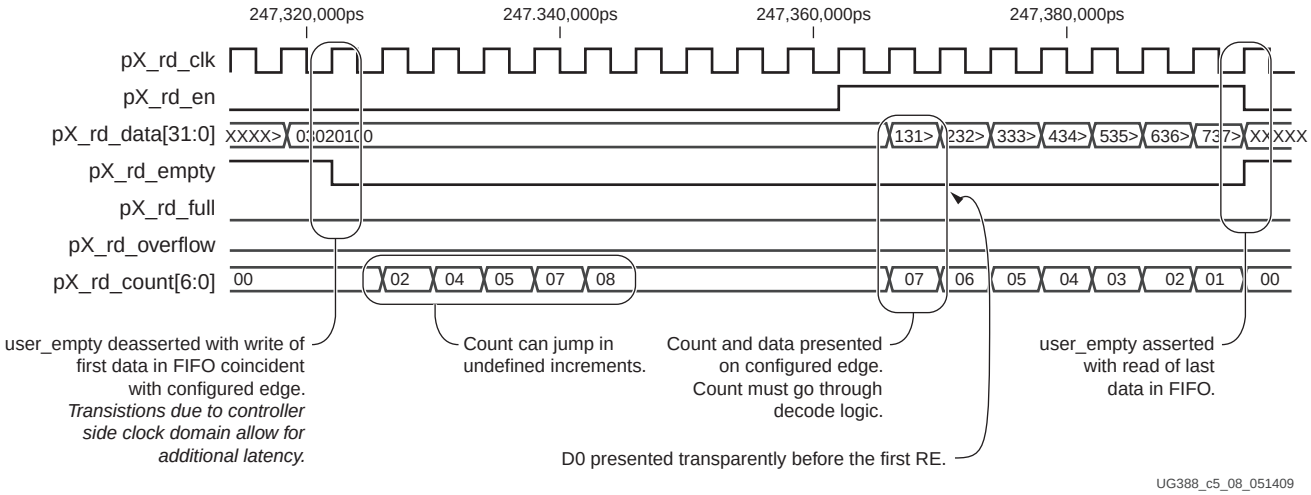

Figure 4-7: **Read Path Timing**

The pX_rd_overflow signal indicates to the user that the memory has returned more data than fits into the read data FIFO and that data was lost. This condition must be avoided to guarantee reliable operation. To avoid an overflow condition, the user must guarantee that there is enough space in the read data FIFO to accommodate a transaction before committing that transaction to the command FIFO.

The count signal bus (pX_rd_count) provides a count of the number of entries in the FIFO. Due to the asynchronicity of the FIFOs in the MCB, the count signal has longer latency than the empty and full flags. Therefore, this bus should only be used for intermediate references and watermarks. The count will transition immediately with respect to FIFO operations committed by the user; however, it takes longer for operations committed by the controller to be apparent on the count signals than the full or empty signals. Thus for the Read Data FIFO as the FIFO is emptying, the count always reports less than or equal to the number of entries that are actually in the FIFO.

For example, if the FIFO contains eight words, the count might report eight even though somewhere during the process of reading from the FIFO, the controller could have started pushing more data into the FIFO. Additionally, if the controller continues to push data into the FIFO, the count could be showing fewer entries in the FIFO even though the FIFO is already full or has even overflowed. For the Read Data FIFO, the count must be used with caution as there will likely be more data in the FIFO than the count is reporting, especially in flight transactions. Count can be used as an almost empty flag, but only to throttle read datapath pipelines, not to throttle commands into the command FIFO.

Memory Transactions

Executing a Write or Read transaction requires proper sequencing between the command and data paths. The following subsections demonstrate the protocols for issuing simple Write and simple Read transactions.

Simple Write

To implement a Write transaction, the Write Data FIFO first must be loaded with sufficient data to complete the request as dictated by the burst length value that is entered into the Command FIFO. Otherwise, an underrun condition occurs when the transaction tries to execute.

Figure 4-8 shows the most basic protocol for loading the Write Data FIFO. The data is presented on the pX_wr_data bus, and pX_wr_en is activated such that the data is written into the FIFO on the rising edge of pX_wr_clk. The pX_wr_empty and pX_wr_count values reflect the fact that data has been loaded into the FIFO. In this example, a total of three data words (32 bits each) are loaded into the FIFO.

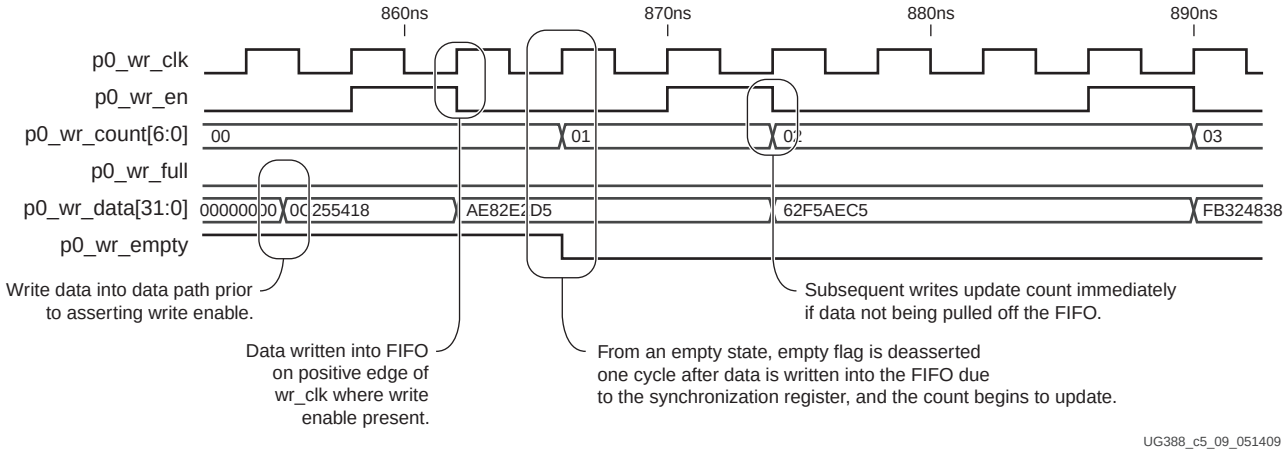

Figure 4-8: **Loading the Write Data FIFO**

Figure 4-9 shows the protocol for entering the Write request into the Command FIFO after the data has been loaded into the Write Data FIFO. The pX_cmd_bl value (`b'10` = burst length 3) is consistent with the number of data words loaded. When the Write request is loaded into the Command FIFO, the MCB automatically executes the transaction to the memory device when the arbiter services this port.

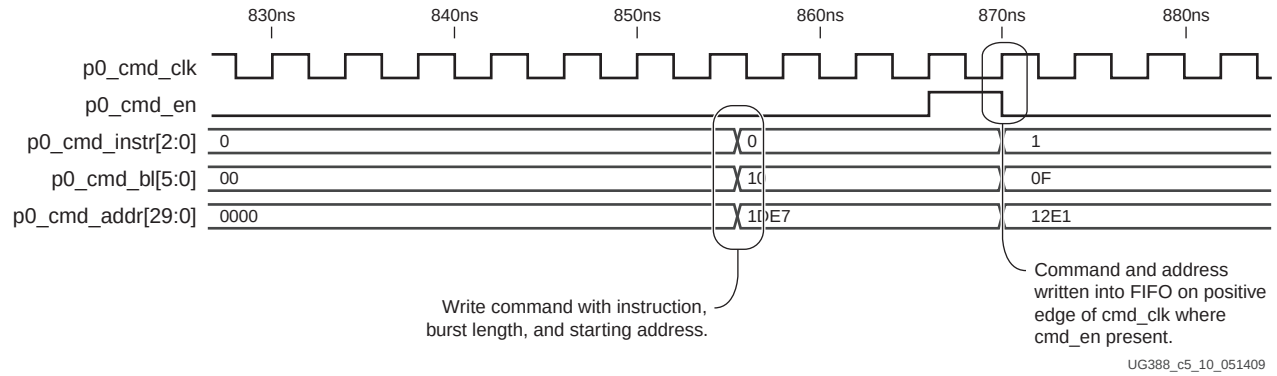

Figure 4-9: **Entering the Write Request into Command FIFO**

Simple Read

To implement a Read transaction, the Read Data FIFO must have enough space to complete the request as dictated by the burst length value that is entered into the Command FIFO. Otherwise, an overflow condition occurs when the transaction tries to execute.

Figure 4-10 shows the protocol for entering the Read request into the Command FIFO. The pX_cmd_bl value specifies the number of data words requested from the memory. When the Read request is loaded into the Command FIFO, the MCB automatically executes the transaction when the arbiter services this port.

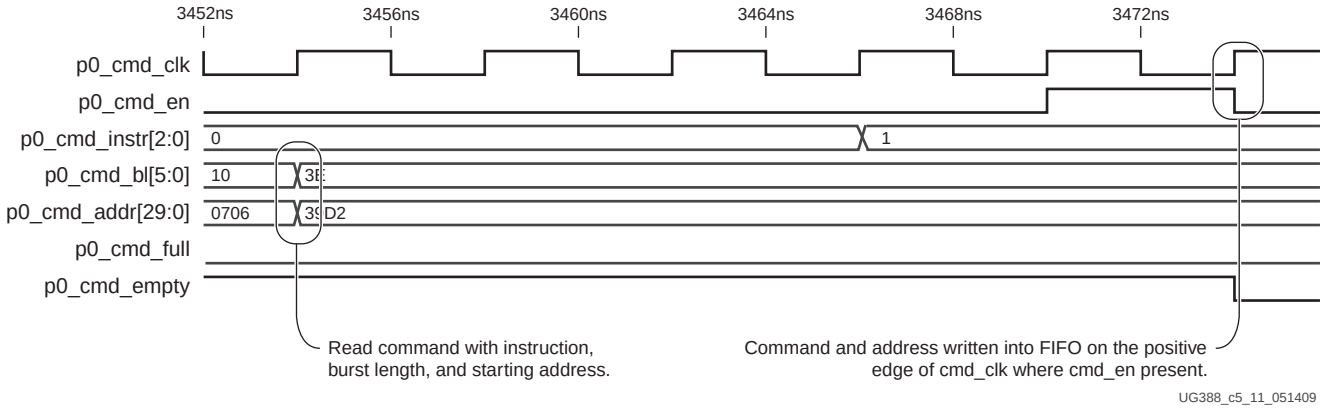

Figure 4-10: **Entering the Read Request into Command FIFO**

Figure 4-11 shows the requested data returning from the memory and being loaded into the Read Data FIFO. The data is then presented on the pX_rd_data bus for access by the FPGA logic. The pX_rd_empty and pX_rd_count values indicate that data has been loaded into the FIFO.

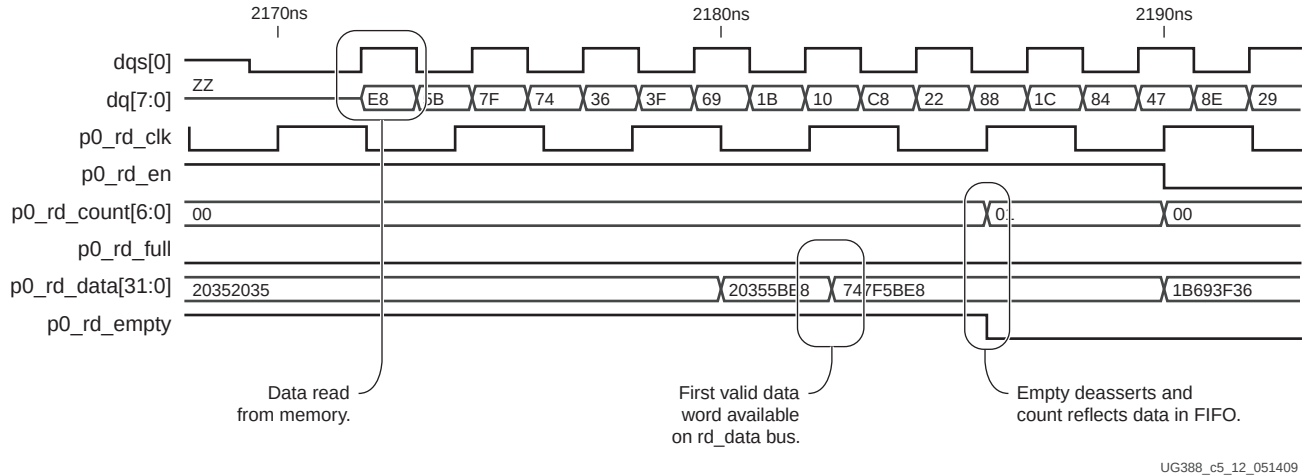

Figure 4-11: **Read Data Returning from the Memory Device**

To transfer data into the FPGA logic from the Read Data FIFO, the pX_rd_en signal is activated during the rising edge of pX_rd_clk as shown in Figure 4-12. The pX_rd_count value updates accordingly.

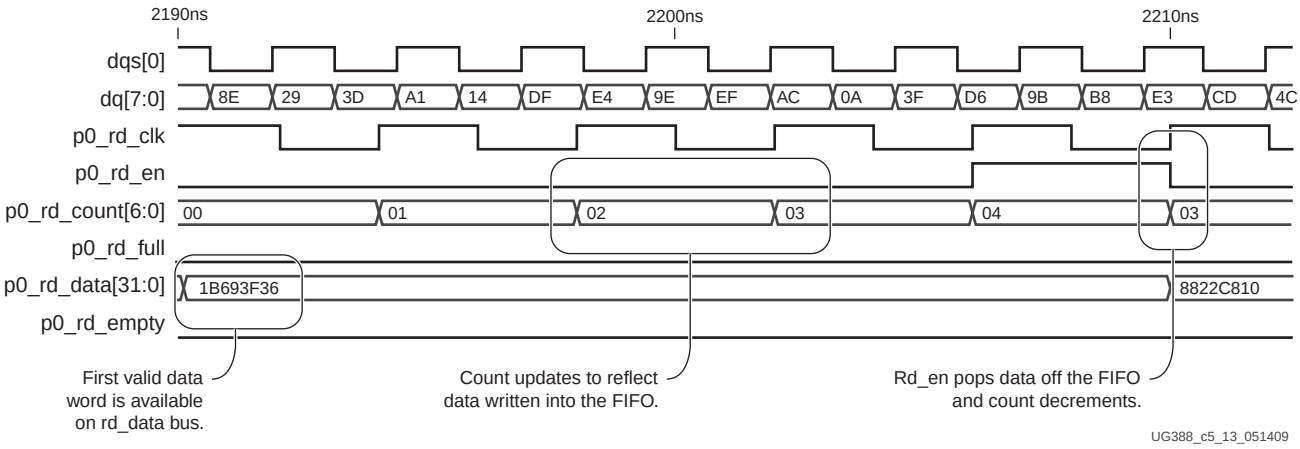

Figure 4-12: **Transferring Read Data into FPGA Logic**

Read Latency

Read latency is defined as the number of memory clock cycles from when the READ command is written to the Command Path FIFO of the User Interface to when the corresponding first data word is available in the Read Data Path FIFOs.

When benchmarking read latencies, it is important to specify the exact conditions under which the measurement occurs. Read latency varies based on the conditions, such as:

- Number of commands already in the FIFO pipeline before the READ command is issued
- Whether an ACTIVATE command needs to be issued to open the new bank/row
- Whether a PRECHARGE command needs to be issued to close a previously opened bank
- Specific timing parameters for the memory, such as t_{RAS} and t_{RCD} in conjunction with the bus clock frequency
- State of the arbiter in multi-port designs
- Memory device CAS latency
- Board-level and chip-level (for both memory and FPGA) propagation delays

Table 4-4 shows MCB read latencies for two different situations at two memory clock frequencies. In the first scenario, the read occurs to a row that is already open in the memory device, meaning no precharge or row activate commands are required prior to accessing the requested data. In the second scenario, the read occurs to a new row address location (bank/row conflict). This requires a precharge to close the previously open row, followed by activation of the new row, which increases read latency. Both scenarios in Table 4-4 assume a single port MCB User Interface with no other commands pending (i.e., the MCB is idle prior to the read request) and a memory device with a CAS latency equal to 5.

Table 4-4: **MCB Read Latency**

Read Latency Scenario	Read Latency (Memory Clock Cycles)	
	MEMCLK = 333 MHz (667 Mb/s)	MEMCLK = 400 MHz (800 Mb/s)
Read from Open Row		
Outbound Command Path	12.5	12.5
Memory CAS Latency (CL)	5	5
Inbound Read Datapath	4.5	4.5
Total Latency in Cycles (Time in ns)	**22 Cycles (66 ns)**	**22 Cycles (55 ns)**
Read from New Row		
Outbound Command Path	12.5	12.5
Precharge/Activate	10	12
Memory CAS Latency (CL)	5	5
Inbound Read Datapath	4.5	4.5
Total Latency in Cycles (Time in ns)	**32 Cycles (96 ns)**	**34 Cycles (85 ns)**

Self Refresh

The self-refresh interface is the mechanism by which the user can request that the memory enter or exit its self-refresh mode. Self refresh is only supported in LPDDR, DDR2, and DDR3 memories. Self refresh allows the memory to conserve power while retaining data when the memory does not need to be actively transmitting data.

The self-refresh interface uses a simple protocol to enter and exit self-refresh mode. A single mode status pin (selfrefresh_mode) indicates whether or not the memory is currently in self-refresh mode. The asynchronous selfresh_enter signal is sampled on the MCB core clock, which is often running at speeds much faster than the User Interface clocks.

To enter self-refresh mode, the selfrefresh_enter signal is asserted until selfrefresh_mode goes High (see Figure 4-13). The selfrefresh_enter signal must remain High to stay in self-refresh mode. To exit the mode, the selfrefresh_enter signal is deactivated (see Figure 4-14). The selfrefresh_mode signal then goes Low indicating that the self-refresh mode has been exited.

The selfresh_enter signal must be maintained in a steady state condition because any glitch on the line can be interpreted as a request. In general, these signals should be registered by the user before going to the MCB to guarantee that these signals only switch when desired.

The Spartan-6 device can be put into suspend mode while the external memory is in self-refresh mode to further reduce system power consumption. However, the Spartan-6 device cannot be reconfigured while the memory device is in self-refresh mode. Reconfiguring causes loss of state in the MCB, preventing proper exiting from the self-refresh mode.

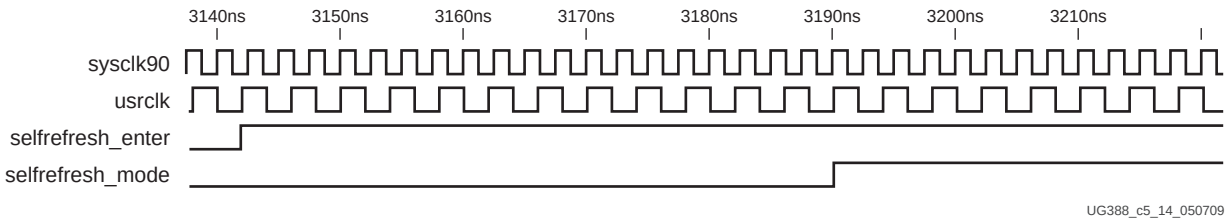

Figure 4-13: **Entering Self-Refresh Mode**

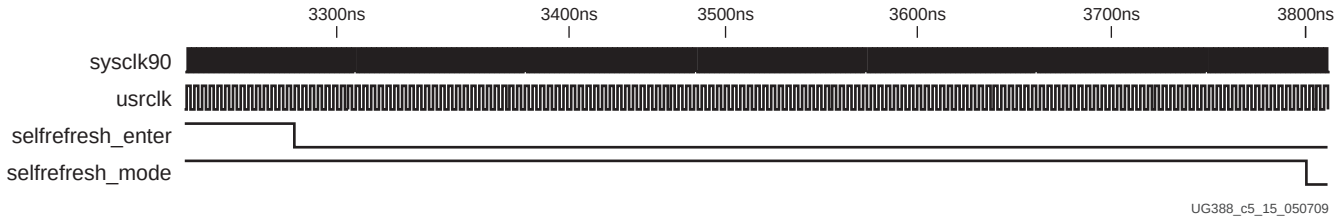

Figure 4-14: **Exiting Self-Refresh Mode**

Suspend

This section describes two recommended methods for using the Suspend Mode capabilities of Spartan-6 devices with designs containing an MCB-based interface.

Suspend Mode without DRAM Data Retention

In cases where it is not important to retain the data stored in the DRAM device, the Suspend pin can simply be brought to an active-High state to enter the suspend mode. Prior to bringing the Suspend pin High, the MCB should be placed in reset by bringing async_rst to an active-High state. While in suspend mode, the MCB is held in reset.

When the Suspend pin is brought Low to exit suspend mode, the MCB is held in reset until the PLL_LOCK signal goes active, indicating a stable clock source to the MCB. The MCB then exits reset and initializes the DRAM using the same startup sequence that occurs during initial power-up or system reset of the MCB. All DRAM data should be considered invalid when exiting suspend mode in this scenario.

Suspend Mode with DRAM Data Retention

In cases where the DRAM data must be retained, the SUSPEND_SYNC primitive must be used in combination with the Self Refresh interface of the MCB to implement suspend mode properly. The SUSPEND_SYNC primitive is used to ensure that the MCB puts the DRAM device into self-refresh mode (see the Self Refresh section) to retain its state prior to putting the FPGA into suspend mode.

Figure 4-15 shows how the SUSPEND_SYNC primitive is connected to the Suspend pin logic and the MCB interface to implement suspend mode with DRAM data retention. The timing diagram in Figure 4-16 illustrates the signal relationships required to successfully take the FPGA into and out of suspend mode in this scenario.

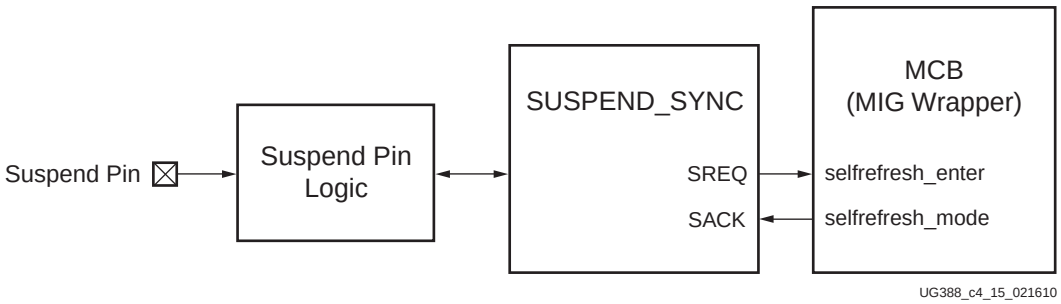

Figure 4-15: **SUSPEND_SYNC Connections**

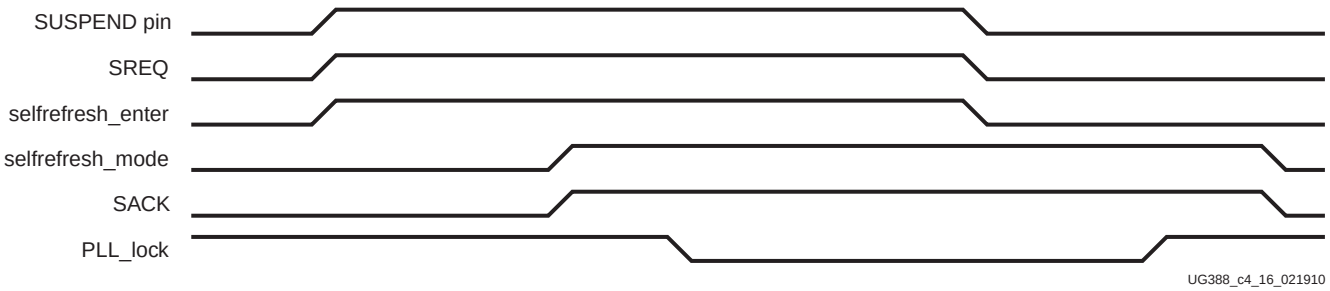

Figure 4-16: **Suspend Mode Timing Diagram**

In response to the active-High Suspend pin, the SUSPEND_SYNC primitive sends a suspend request (SREQ) signal to the MCB to indicate the desire to enter suspend mode. The SREQ signal is connected directly to the selfrefresh_enter input of the top-level MIG (or EDK) wrapper, from which it is routed to the soft calibration module. The soft calibration module completes any current operations before forwarding the self-refresh request to the MCB and from there to the memory device.

Once the MCB has successfully placed the DRAM device in self-refresh mode, the selfrefresh_mode output goes High. This signal is directly connected to the suspend acknowledge (SACK) input of the SUSPEND_SYNC primitive, indicating that the FPGA can now be placed in suspend mode. The PLL_lock signal is lost when the Suspend occurs.

When the Suspend pin goes Low to exit suspend mode, SREQ and therefore the selfrefresh_enter signals go inactive, and the FPGA emerges from the Suspend state. The PLL_lock signal is initially Low as the PLL tries to lock onto the incoming clock again. However, because the selfrefresh_mode signal is active, this Low PLL_lock condition does not cause a system reset of the MCB as it normally would. When the PLL achieves lock, the soft calibration module forwards the request to leave the self-refresh mode to the MCB and from there to the memory device.

When the DRAM device has successfully exited the self-refresh mode, the selfrefresh_mode signal returns to the Low state, and normal MCB operation can resume with no loss of DRAM data.

Additional Suspend Mode Requirements

While the Spartan-6 device is in suspend mode, some critical signals driving the DRAM device must be maintained in a known state. When using suspend mode with memory devices that support the self-refresh mode, the CKE output of the Spartan-6 device should have a constraint added to retain the last state of the pin during the Suspend state. A statement like the following should be added to the user constraints file (UCF):

```
NET "mcbx_dram_cke" SUSPEND="drive_last_value";
```

This ensures that the DRAM device executes the self-refresh mode properly. In addition, for DDR3, the reset signal to the DRAM should have a similar constraint added to the UCF file as follows:

```
NET "mcbx_dram_reset_n" SUSPEND="drive_last_value";
```

This prevents an unintentional reset of the DRAM device during suspend mode.

Byte Address to Memory Address Conversion

From the User Interface perspective, the MCB provides a simple and sequential byte addressing scheme into the physical DRAM. The fact that DRAMs store data in fixed segments is abstracted by this scheme, allowing for a simple SRAM-like address interface. The MCB automatically converts the User Interface byte address into the necessary row, bank, and column address signals required for a particular memory device configuration. To complete the abstraction of the physical memory addressing details, the MCB manages automatic row and bank crossing transparent to the User Interface.

The memory standard, bus width, and density all affect how the User Interface byte address bits map to the respective row, bank, and column address bits. Memory device selection in the MIG tool results in the passing of the necessary parameters to the MCB so that it can create the proper address bit assignments. Table 4-5, page 63 shows how the assignments are made based on a given memory device configuration. These mappings are based on JEDEC standard addressing schemes.

As shown in Table 4-5, the memory width (x4, x8, or x16) affects the mapping of the byte address to the physical address. For x4 devices, the column address LSB is always set to 0 on the external address bus to create byte-aligned addressing into the memory device (User Interface bit 0 maps to column address bit 1). Because x8 devices use native byte addressing, the MCB uses a direct mapping of byte address to physical address bit (User Interface bit 0 maps directly to column address bit 0). For x16 devices, the mapping is shifted to create address alignment on a two-byte boundary (User Interface bit 1 maps to column address bit 0).

The MCB supports two general schemes for mapping the User Interface byte address to the memory interface physical address: ROW_BANK_COLUMN and BANK_ROW_COLUMN. The Port Configuration page in the MIG tool allows selection of the scheme most suited for the particular application (see the “Creating an MCB Design” section in UG416, *Spartan-6 FPGA Memory Interface Solutions User Guide*). Table 4-5 shows the mapping only for ROW_BANK_COLUMN addressing. For BANK_ROW_COLUMN addressing, the position of the Row and Bank address groups are switched such that the Bank address bits are in the MSB position with respect to the User Interface byte address. Column address bit mappings remain unchanged.

The ROW_BANK_COLUMN addressing scheme means that for a transaction occurring over a sequential address space (for example, a long data burst), the MCB automatically opens up the same row in the next bank of the DRAM device to continue the transaction when the end of an existing row is reached. This reduces the overhead caused by closing

down the current row (Precharge command) and opening another row in the same bank (Activate command) to continue the transaction. The ROW_BANK_COLUMN addressing scheme is well suited to applications that require bursting of large data packets to sequential address locations where efficiency can be gained by striping the data across multiple banks.

In contrast, BANK_ROW_COLUMN addressing means that crossing a row boundary results in closing that row and opening another one within the same bank. The Bank address bits reside in the MSB position of the User Interface byte address and can be used to switch between major address spaces that reside in different banks. For example, a microprocessor or microcontroller based application that tends to have shorter, more random transactions to one block of memory for a period of time and then jump to another block (that is, bank) might prefer this address mapping scheme.

The specifics of the application determine whether ROW_BANK_COLUMN or BANK_ROW_COLUMN should be chosen as the address scheme.

Note: When referring to Table 4-5, the user must ensure that the requirements listed in Table 4-2, page 51 are followed to preserve proper data word boundaries.

Table 4-5: **Memory Device Mapping**

Byte Address			29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	V
Type	**Width**	**Depth**																															
DDR	x16	128 Mb																															
		256 Mb																															
		512 Mb																															
		1 Gb																															
	x8	128 Mb																															
		256 Mb																															
		512 Mb																															
		1 Gb																															
	x4	128 Mb																															
		256 Mb																															
		512 Mb																															
		1 Gb																															
DDR2	x16	256 Mb																															
		512 Mb																															
		1 Gb																															
		2 Gb																															
		4 Gb																															
	x8	256 Mb																															
		512 Mb																															
		1 Gb																															
		2 Gb																															
	x4	256 Mb																															
		512 Mb																															
		1 Gb																															
		2 Gb																															
DDR3	x16	512 Mb																															
		1 Gb																															
		2 Gb																															
		4 Gb																															
	x8	512 Mb																															
		1 Gb																															
		2 Gb																															
	x4	512 Mb																															
		1 Gb																															
		2 Gb																															
LPDDR	x16	128 Mb																															
		256 Mb																															
		512 Mb																															
		1 Gb																															

Column Address Bits:

Bank Address Bits: CA[0] tied Low on 4-bit memories

Row Address Bits:

Transaction Ordering and Coherency

In the MCB architecture, transactions are executed to memory in the order that the transactions are acknowledged with respect to a single port. Consequently, on a single port, transactions are completed in the same order as requested.

Across multiple ports of the MCB, there is no guarantee that the transactions issued by different ports will complete in the request order. The arbitration algorithm can be modified so that a given port is favored over another port. This can be used as a mechanism to influence transaction ordering but might not guarantee a specific order.

The MCB allows write transactions to be buffered inside itself. Because of the buffering, there is an undefined time between when a write transaction is accepted into the Command FIFO and when the write completes to memory. Because transaction ordering is not guaranteed across ports, a port doing a read from an address location being written to by another port might read the new or the old memory value.

In some applications, it is important to know that a write has completed to memory before issuing a read of that location. There are three methods that can ensure coherency:

1. Monitor the Command FIFO empty flag:
   - Assuming that only one command is queued for the given port, the user can monitor the Command FIFO empty flag. This flag goes High when the MCB has started to issue the write command. When the command starts, it is guaranteed to finish provided the data is available in the Write Data FIFO.
   - The design can wait for the Command FIFO empty flag to go High before signaling that a read can be performed.
2. The design can take advantage of the fact that transactions complete in order on a given port:
   - After a write to a sensitive part of memory, the device can issue a dummy read and wait for the dummy read to complete and return data.
   - The completion of the dummy read ensures that the previous write has completed to memory.
3. The arbitration algorithm can be adjusted:
   - If the port performing the writes can always be set to have higher priority than the ports doing the reads, this ensures that the write completes before the read across the two ports. Care should be taken with this method if there is a possibility that the ports can have “bubble” cycles between the write and the read request.

Note: Using any of these methods to ensure coherency could result in reduced system performance; these methods should be employed only when necessary.

Appendix A

References

Memory Standards

These links provide more details about each of the memory standards implemented by the MCB:

- JEDEC DDR3 Specification
  http://www.jedec.org/sites/default/files/docs/JESD79-3D.pdf
- JEDEC DDR2 Specification
  http://www.jedec.org/sites/default/files/docs/JESD79-2F.pdf
- JEDEC DDR Specification
  http://www.jedec.org/sites/default/files/docs/JESD79F.pdf
- JEDEC LPDDR Specification
  http://www.jedec.org/sites/default/files/docs/JESD209A.pdf

PCB Layout and Signal Integrity

These references provide additional details regarding PCB layout and signal integrity analysis for DDR memories. Xilinx does not guarantee the accuracy or completeness of any material referenced here.

- *Hardware Tips for Point-to-Point System Design: Termination, Layout, and Routing*
  http://download.micron.com/pdf/technotes/DDR/tn4614.pdf
- *Interfacing the RC32434/5 with DDR SDRAM Memory*
  http://www.idt.com/products/getDoc.cfm?docID=571565
- *DDR-SDRAM Layout Considerations for MCF547x/8x Processors*
  http://www.freescale.com/files/32bit/doc/app_note/AN2826.pdf
- *Hardware and Layout Design Considerations for DDR2 SDRAM Memory Interfaces*
  http://www.freescale.com/files/32bit/doc/app_note/AN2910.pdf
- *DDR System Design Considerations*
  http://download.micron.com/pdf/presentations/dram/plat7justin.pdf
- *DDR2 Package Sizes and Layout Requirements*
  http://download.micron.com/pdf/technotes/ddr2/TN4708.pdf
- *DDR2 (Point-to-Point) Package Sizes and Layout Basics*
  http://download.micron.com/pdf/technotes/ddr2/TN4720.pdf

- *Understanding TI's PCB Routing Rule-Based DDR Timing Specification* http://focus.ti.com.cn/cn/lit/an/spraav0a/spraav0a.pdf
- *Implementing DDR2 PCB Layout on the TMS320C6454/5* http://focus.ti.com/lit/an/spraaa7e/spraaa7e.pdf