Virtex-6 FPGA Memory Interface Solutions

User Guide

UG406 January 18, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/24/09	1.0	Initial Xilinx release.
09/16/09	1.1	MIG 3.2 release. Chapter 1: • Updated screen captures, controller options descriptions, and tables in Getting Started with the CORE Generator Software, page 11. • Updated Table 1-83, page 112 and Table 1-88, page 128. • Removed Table 1-11 that was identical to Table 1-10, page 43. • Updated Figure 1-51, page 106. • Added Debugging Virtex-6 FPGA DDR2/DDR3 SDRAM Designs, page 149. Chapter 2: • Updated Figure 2-2, page 175 to Figure 2-11, page 181. • Added Figure 2-33, page 196. • Updated Table 2-2, page 198, Table 2-6, page 200, Table 2-10, page 211, Table 2-11, page 211, Table 2-13, page 220, and Table 2-14, page 223. • Updated Figure 2-35, page 206 and Figure 2-36, page 207. • Updated Controller Options, page 183 and Calibration, page 216. Added Chapter 3, RLDRAM II Memory Interface Solution.

Date	Version	Revision
12/02/09	1.2	• Added VHDL (`.vhd`) file extensions to existing file names with Verilog (`.v`) file extensions in various places throughout the document, as appropriate, as follows: `<file name>.v/vhd`. • Globally changed version number of ISE® Design Suite from 11.3 to 11.4. Chapter 1: • Added signals app_full and app_wdf_full to Table 1-18, page 68. • Updated Figure 1-9, page 16, Figure 1-12, page 19, Figure 1-13, page 19, and Figure 1-50, page 105. • Added Note for parameter CLK_PERIOD in Table 1-88, page 128. • Added User Interface, page 114, Command Path, page 115, Write Path, page 118, and Read Path, page 122. • Renamed section previously entitled “Interfacing to the Core” to Native Interface, page 123. • Updated DDR3 SDRAM, page 135. • Added Design Rules, page 135, and DDR3 Component PCB Routing, page 135. • Updated DDR2 SDRAM, page 142. • Added Design Rules, page 142, and Pin Assignments, page 142. • Added Supported Devices for Virtex-6 FPGAs, page 170. Chapter 2: • Updated Figure 2-10, page 180, Figure 2-13, page 183 and Figure 2-14, page 183. • Added Note for parameter CLK_PERIOD in Table 2-13, page 220. • Renamed “Debugging the Core” to DEBUG_PORT Signals, and moved it into Debugging Virtex-6 FPGA QDRII+ SRAM Designs, page 224. • Added Debugging Virtex-6 FPGA QDRII+ SRAM Designs, page 224. Chapter 3: • Updated Figure 3-9, page 250, Figure 3-12, page 253, Figure 3-13, page 253 and Figure 2-36, page 207. • Added Note for parameter CLK_PERIOD in Table 3-15, page 296. • Renamed “Debugging the Core” to DEBUG_PORT Signals, and moved it into Debugging Virtex-6 FPGA RLDRAM II Memory Designs, page 301. • Added Debugging Virtex-6 FPGA RLDRAM II Memory Designs, page 301.

Date	Version	Revision
04/19/10	1.3	Updated Introduction, page 11. Updated ISE Design Suite version from 11.4 to 12.1. Replaced app_full with app_rdy, and app_wdf_full with app_wdf_rdy in Chapter 1. Updated Figure 1-9, Figure 1-24, and Figure 1-28. Added Figure 1-27, Figure 1-29 and Figure 1-30. Removed modules `phy_ocb_mon.v/vhd` and `phy_ocb_mon_top.v/vhd` from Table 1-3. Removed `pipeline_inserter.v/vhd` from Table 1-4 and Table 1-14. Replaced app_full with app_rdy, and removed app_wdf_full from Table 1-18. Updated description for app_rdy in User Interface, page 68. Updated Figure 1-48, Figure 1-49, Figure 1-50, Figure 1-51, Figure 1-56, and associated text descriptions. Removed Figure 1-54: OSERDES Clock Monitor Block Diagram and removed OSERDES Clock Phase Monitor subsection. Added signals SDA and SCL to Table 1-83. Updated Figure 1-57, Figure 1-60, Figure 1-61, Figure 1-62, Figure 1-63, Figure 1-65, Figure 1-66, and Figure 1-69. Added Figure 1-64. Updated Phase Detector, page 111 and Write Path, page 118. Added Read Latency, page 125, and Core Constraints, page 125. Updated Table 1-88. Updated DDR3 SDRAM Bank Selection Rules, page 135, and added Configuration, page 139. Updated DDR2 SDRAM Bank Selection Rules, page 142 and added Configuration, page 145. Updated Table 1-98. Updated Figure 2-10, Figure 2-21, Figure 2-35, Figure 2-36, Figure 2-39, Figure 2-40, and Figure 2-41. Added Figure 2-20 and Figure 2-22. Removed `qdr_rld_phy_ocb_mon_top.v/vhd` from and updated description of `qdr_rld_phy_ocb_mon.v/vhd` in Table 2-2 and Table 2-6. Updated Table 2-9, Table 2-10, Table 2-13, Table 2-14, Table 2-17, and Table 2-18. Added Table 2-12. Removed description of two-word burst length protocol, including Figure 2-41, from Interfacing with the Memory Device, page 212. Updated I/O Architecture, page 214. Updated Data Capture, page 215. Removed Output Circular Buffer Monitor subsection. Updated Figure 3-9, Figure 3-26, Figure 3-30, and Figure 3-49. Added Figure 3-29 and I/O Voltage Option description. Removed `qdr_rld_phy_ocb_mon_top.v/vhd` from and updated description of `qdr_rld_phy_ocb_mon.v/vhd` in Table 3-2 and Table 3-6. Updated Table 3-9 and Table 3-11. Updated I/O Architecture, page 291 and Figure 3-50. Updated Data Capture, page 293. Removed Output Circular Buffer Monitor subsection. Updated Table 3-15, Table 3-19, and Table 3-20. Added Appendix B, Simulation Help.
07/23/10	1.4	Updated ISE Design Suite version from 12.1 to 12.2. Updated screen captures in Chapter 1, Chapter 2, and Chapter 3. Removed subsection "Finish" from Creating Virtex-6 FPGA DDR3 Memory Controller Block Design. Replaced app_wdf_full with app_wdf_rdy in Figure 1-46. In Table 1-18, updated description of app_sz and added clk_mem, clk_rd_base, dfi_init_complete, and app_ecc_multiple_err[3:0]. Added clk_mem, clk_rd_base, and phy_init_done. Added description of `example_top` module settings after Figure 1-49. Replaced app_full with app_rdy in Command Path. In Table 1-88, updated Options column for PHASE_DETECT and added STARVE_LIMIT. In Table 1-89, updated Options column for parameter APP_DATA_WIDTH. Added Data/Strobe/Mask Span Allocation Rules. Removed System Control Selection from Bank Selections, page 188. Updated Write Path, page 212, including Figure 2-39. Updated Read Path, page 215 and Dynamic Calibration, page 217. In Table 2-13, updated descriptions of CLK_PERIOD and DATA_WIDTH, and removed MMCM_ADV_PS_WA. Updated PHY_LATENCY. In Table 2-17, updated If Problems Arise column for dbg_phy_status[6]. Added Configuration option and Figure 3-20 to Setting RLDRAM II Memory Parameter Option. Removed System Control Selection from Bank Selections, page 261. Updated signal directions in Figure 3-43. In Table 3-15, removed MMCM_ADV_PS_WA, and updated PHASE_DETECT and note 3. In Table 3-19, updated If Problems Arise column for dbg_phy_status[6].

Date	Version	Revision
09/21/10	1.5	MIG 3.6 release. Updated ISE Design Suite version from 12.2 to 12.3. Updated Introduction, page 11 for AXI4 slave interface support. Changed title of Getting Started section to Getting Started with the CORE Generator Software, page 11 and updated Customizing and Generating the Core, page 11. Updated Creating Virtex-6 FPGA DDR3 Memory Controller Block Design, page 18 and Figure 1-11 for the AXI4 slave interface option. Added AXI Parameter Options, page 23 and Figure 1-22. Updated Figure 1-29 and Figure 1-30. Updated Directory Structure and File Descriptions, page 37. Added Verify UCF and Update Design and UCF Rules, page 47, Modifying the Example Design, page 56, Getting Started with EDK, page 64, AXI4 Slave Interface Block, page 67, and IDELAYCTRL, page 68. Removed Table 1-14, "Test Data Patterns." Updated description of app_rd_data_end in Table 1-18. Added AXI4 Slave Interface Block, page 72. Added last paragraph (about IDELAYCTRL module) to Clocking Architecture, page 104. Added paragraph about timing margin adjustment to Read Leveling, page 110. Updated first paragraph under Interfacing to the Core, page 113 for the AXI4 slave interface option. Added AXI4 Slave Interface, page 113. Added timing for app_sz in Figure 1-66. Added BUFR Resynchronization Full-Cycle Path, page 126. Updated Table 1-88 and Table 1-94. Updated paragraph immediately following Table 1-94. Added Verify UCF and Update Design and UCF Rules, page 202. Modified description of qdr_cq_n in Table 2-11. Updated Figure 2-39. Removed fifth sentence under Read Path, page 215 (about CQ and CQ#). Updated Data Capture, page 215 (removed sentence about CQ/CQ#). Updated Figure 2-41 (removed CQ# IOB block and connected CLKB to same input to CLK). Updated Calibration of CQ/CQ# and Q and Data Realignment, page 217. Updated Dynamic Calibration, page 217. Added IDELAYCTRL, page 219. Updated values and description for Dynamic Calibration, page 217 in Table 2-13. Updated first bullet under Trace Length Requirements, page 222. Updated Pinout Requirements, page 223. Added Verify UCF and Update Design and UCF Rules, page 275 and IDELAYCTRL, page 296. Updated Figure 3-49 and Pinout Requirements, page 300.
12/14/10	1.6	MIG 3.61 release. Added EDK parameter names to Table 1-88 and Table 1-89. Added Noise and DLL Lock, page 142 and Noise and DLL Lock, page 148.

Date	Version	Revision
03/01/11	1.7	MIG 3.7 release. Updated ISE Design Suite version to 13.1. Chapter 1: Updated Figure 1-23. Added paragraph about address mapping scheme under Figure 1-23. In Verify UCF and Update Design and UCF Rules, page 47, replaced bullet about allocation of first memory clock pair with allocation of memory clock signals (CK and CK#). Added new subsections to Getting Started with EDK, page 64. In Table 1-18, updated clk_rd_base definition to indicate it is full frequency. Changed name of dfi_init_complete to phy_init_done throughout. Added OSERDES blocks to Figure 1-50. Rewrote Read Datapath, page 108. Revised Phase Detector, page 111 and User Interface, page 114. Added new section AXI Addressing. Added MEM_ADDR_ORDER (C_MEM_ADDR_ORDER) to Table 1-88. Revised CXT and lower power device bullets in Design Rules, page 135. In Pin Allocation Rules on page 136 (DDR3) and page 143 (DDR2), changed the memory clock signal allocation bullet and deleted the bullet about a single VREF pin being reserved. In Trace Lengths, page 141 for DDR3, added sentences about recommended method for determining delay, replaced "electrical delay" with "skew", and added a new bullet on DQ to DQS matching. Revised CXT bullets to a single bullet for both speed grades in Design Rules, page 142. Added Table 1-90 and Table 1-91. In Trace Lengths, page 147 for DDR2, replaced "electrical delay" with "skew", and added a new bullet on DQ to DQS matching. Added dbg_inc_rd_fps and dbg_dec_rd_fps to Table 1-98. Chapter 2: In Trace Length Requirements, page 222 for QDRII+ SRAM, changed the delay detection to FPGA Editor and replaced "electrical delay" with "skew". Chapter 3: In Trace Length Requirements, page 300 for RLDRAM II, changed the delay detection to FPGA Editor and replaced "electrical delay" with "skew". Removed Figure 3-54, MMCM Placement.
06/22/11	1.8	MIG 3.8 release. Updated ISE Design Suite version to 13.2. Updated various GUI screens throughout Chapter 1, Chapter 2, and Chapter 3. Chapter 1: Added the `user_design/rtl/ecc`, Simulating the Example Design (for Designs with an AXI4 Interface), Simulation Considerations, AXI4-Lite Slave Control/Status Register Interface Block, and Error Correcting Code (ECC) sections. Updated Error Correction Code (ECC) description on page 20. Added traffic generator feature list on page 55. Added ENFORCE_RD_WR, ENFORCE_RD_WR_CMD, ENFORCE_RD_WR_PATTERN, C_EN_WRAP_TRANS, and C_AXI_NBURST_TEST to Table 1-15. Added sentence about traffic generator signals after Table 1-17 on page 63. Added note about data mask option on page 63. Added information about AXI4-Lite interface to AXI4 Interface Connection, page 65. In Table 1-18, added table notes and inserted new app_correct_en signal. Added third paragraph to AXI4 Slave Interface Block, page 72. Under Figure 1-50, replaced text regarding MMCM parameters. Revised the value of DATA_BUF_ADDR_WIDTH in Table 1-89. Added methods for determining delay to the first paragraph of Trace Lengths, page 147. Added the Pin Mapping for x4 RDIMMs and Verifying the Simulation Using the Example Design (for Designs with the AXI4 User Interface) sections. Chapter 2: Revised the description of clk_wr in Table 2-10. Added methods for determining delay to the first paragraph of Trace Length Requirements, page 222. Chapter 3: In Table 3-9, revised the user_afifo_empty description. Added methods for determining delay to the first paragraph of Trace Length Requirements, page 300.

Date	Version	Revision
10/19/11	1.9	MIG 3.9 release. Updated ISE Design Suite version to 13.3. Updated various GUI screens throughout Chapter 1, Chapter 2, and Chapter 3. Removed Preface and added Appendix A, Additional Resources. • Chapter 1: Removed IODELAY Power Versus Performance bullet from Setting the DDR3 Memory Parameter Option, page 24. Updated the rule when the frequency is over 400 MHz in Verify UCF and Update Design and UCF Rules, page 47. Revised app_sz description in Table 1-18. Added Read Priority with Starve Limit (RD_PRI_REG_STARVE_LIMIT), page 76. Expanded AXI Addressing, page 113. Added 2:1 mode example after Figure 1-66. In Table 1-85, removed 1 option from nCK_PER_CLK, removed DISABLED option from RTT_NOM (C_RTT_NOM) for DDR3, and changed RTT_NOM to RTT_WR in RTT_WR (C_RTT_WR). Added sentence about when to tie DM to GND on page 141 and page 147. • Chapter 2: Removed qdr_qvld and Qvld from this chapter. Removed IODELAY Power Versus Performance bullet from Controller Options, page 183. • Chapter 3: Removed rld_qvld and QVLD from this chapter. Removed IODELAY Power Versus Performance bullet from Setting RLDRAM II Memory Parameter Option, page 256.
01/18/12	1.10	MIG 3.91 release. Updated ISE Design Suite version to 13.4. • Updated footnote 2 in Table 1-18. • Added paragraph about calibration completion under Table 1-94. • Removed “Write Priority” section from Arbitration in the AXI Shim, page 75.

Table of Contents

Appendix B: Simulation Help

XILINX®

Chapter 1

DDR2 and DDR3 SDRAM Memory Interface Solution

Introduction

The Virtex®-6 FPGA memory interface solutions core is a combined pre-engineered controller and physical layer (PHY) for interfacing Virtex-6 FPGA user designs and AMBA® advanced extensible interface (AXI4) slave interfaces to DDR2 and DDR3 SDRAM devices. This user guide provides information about using, customizing, and simulating a LogiCORE™ IP DDR2 or DDR3 SDRAM memory interface core for the Virtex-6 FPGA. In the Embedded Development Kit (EDK) this core is provided through the Xilinx® Platform Studio (XPS) as the `axi_v6_ddrx` IP with a static AXI4 to DDR2 or DDR3 SDRAM architecture. The user guide describes the core architecture and provides details on customizing and interfacing to the core. While it might be possible to use the different blocks independently, the supported solution is the combined controller and physical layer.

Getting Started with the CORE Generator Software

This section is a step-by-step guide for using the CORE Generator™ software to generate a DDR2 or DDR3 SDRAM memory interface in a Virtex-6 device, run the design through implementation with the Xilinx tools, and simulate the example design using the provided synthesizable test bench.

System Requirements

The operating system used was Microsoft Windows XP Professional. ISE® Design Suite, version 13.4 was used.

Customizing and Generating the Core

Generation through Graphical User Interface

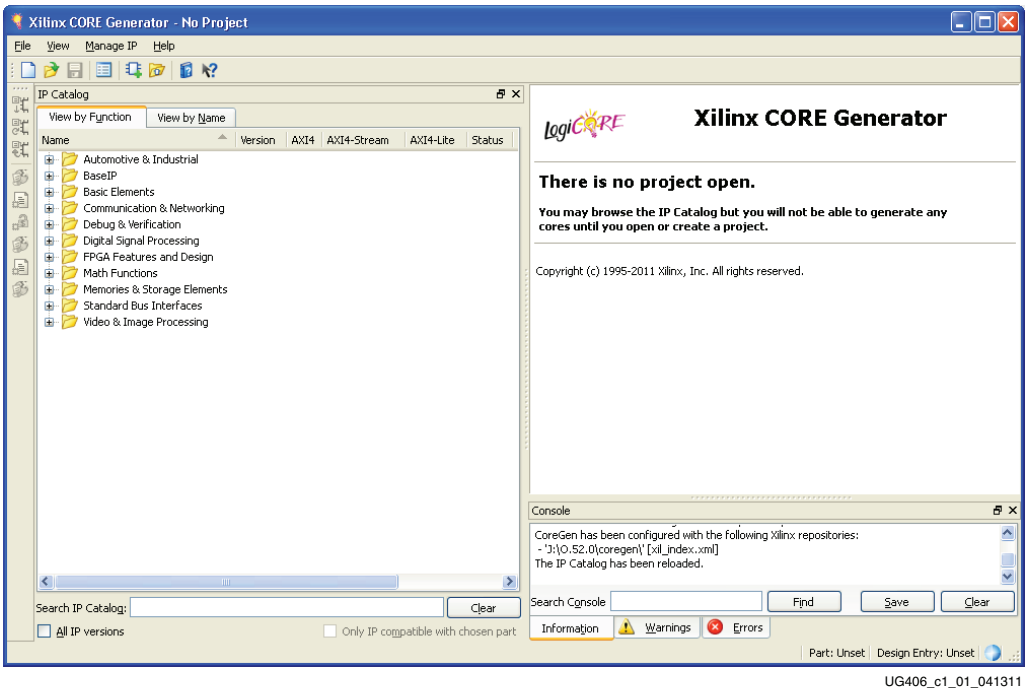
The Memory Interface Generator (MIG) is a self-explanatory wizard tool that can be invoked under the CORE Generator software from XPS. This section is intended to help in understanding the various steps involved in using the MIG tool.

These steps should be followed to generate a Virtex-6 FPGA DDR3 SDRAM design:

Note: The exact behavior of the MIG tool and the appearance of some pages/options might differ depending on whether the MIG tool is invoked from the CORE Generator software or from XPS, and whether or not an AXI interface is selected. These differences are described in the steps below.

1. To invoke the MIG tool from XPS, select **Memory and Memory Controller** → **AXI V6 Memory Controller** from the XPS IP catalog (when adding new IP to the system) or right-click the **axi_v6_ddrx** component in the XPS System Assembly View and select **Configure IP...**. Then skip to MIG Output Options, page 16.

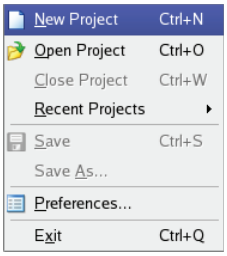
   Otherwise, to launch the MIG tool from the CORE Generator software, select **Start** → **Xilinx ISE Design Suite 13.4** → **ISE** → **Accessories** → **CORE Generator** (Figure 1-1).



UG406_c1_01_041311

Figure 1-1: **Xilinx CORE Generator Software**

2. Choose **File** → **New project** to open the New Project dialog box. Create a new project named Virtex6_MIG_Example_Design (Figure 1-2).



UG406_c1_02_081109

Figure 1-2: **New CORE Generator Software Project**

3. Enter a project name and location. Click **Save** (Figure 1-3).

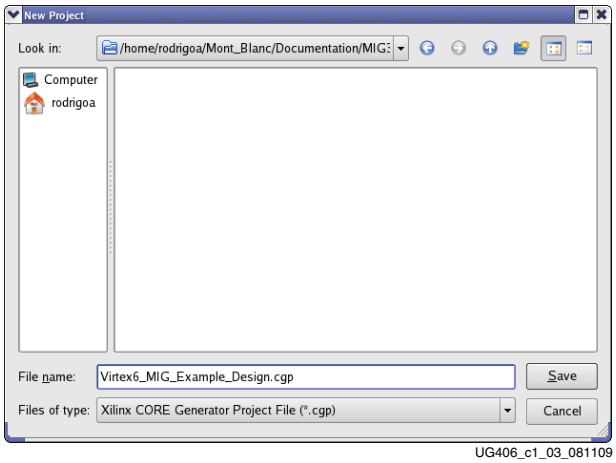


Figure 1-3: **New Project Menu**

4. Select these project options for the part (Figure 1-4):
   a. Family: **Virtex-6**
   b. Device: **xc6vlx240t**
   c. Package: **ff1156**
   d. Speed Grade: **-2**

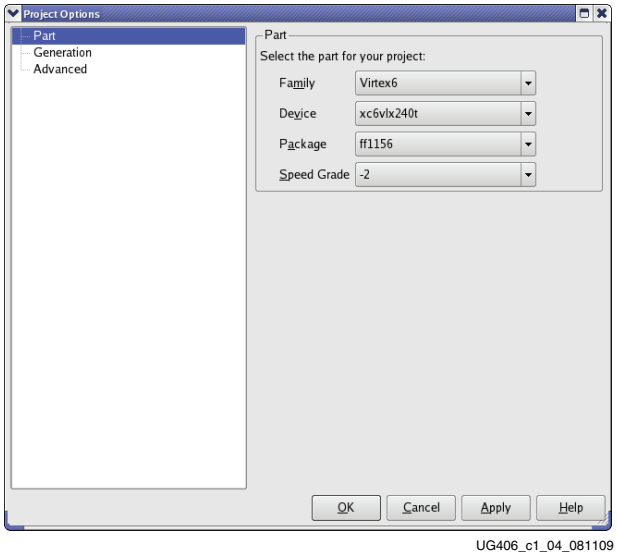
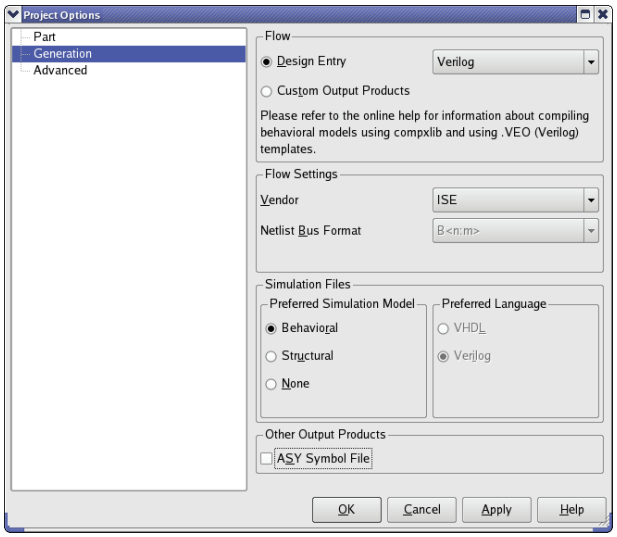


Figure 1-4: **CORE Generator Software Device Selection Page**

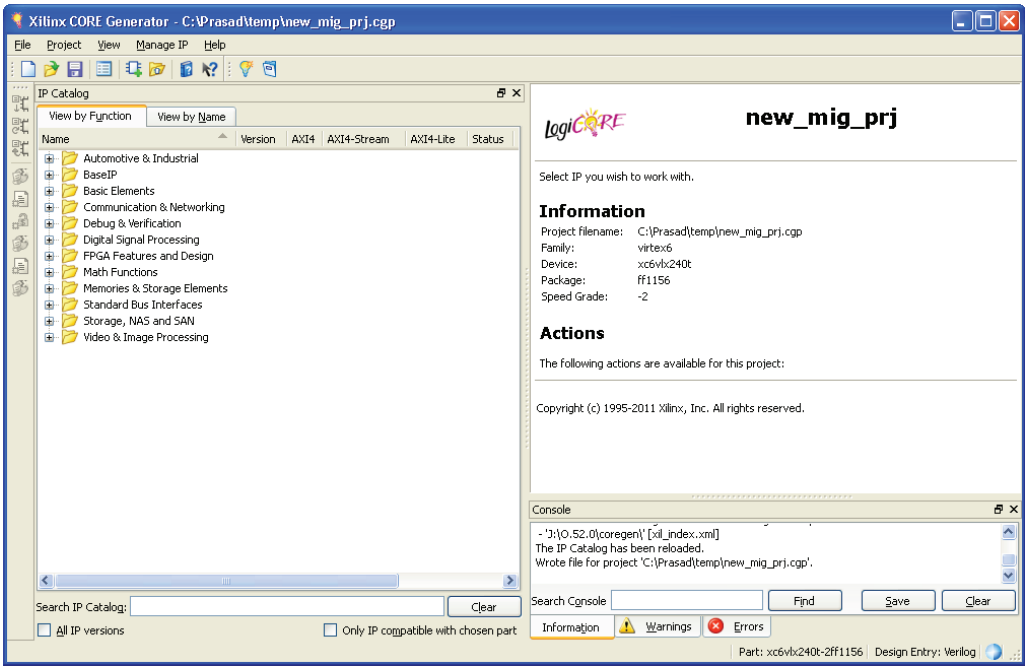
5. Select **Verilog** or **VHDL** as the Design Entry Option and **ISE** for the Vendor Flow Setting. Click **OK** to finish the Project Options setup (Figure 1-5).



UG406_c1_05_081109

Figure 1-5: **CORE Generator Software Design Flow Setting Page**

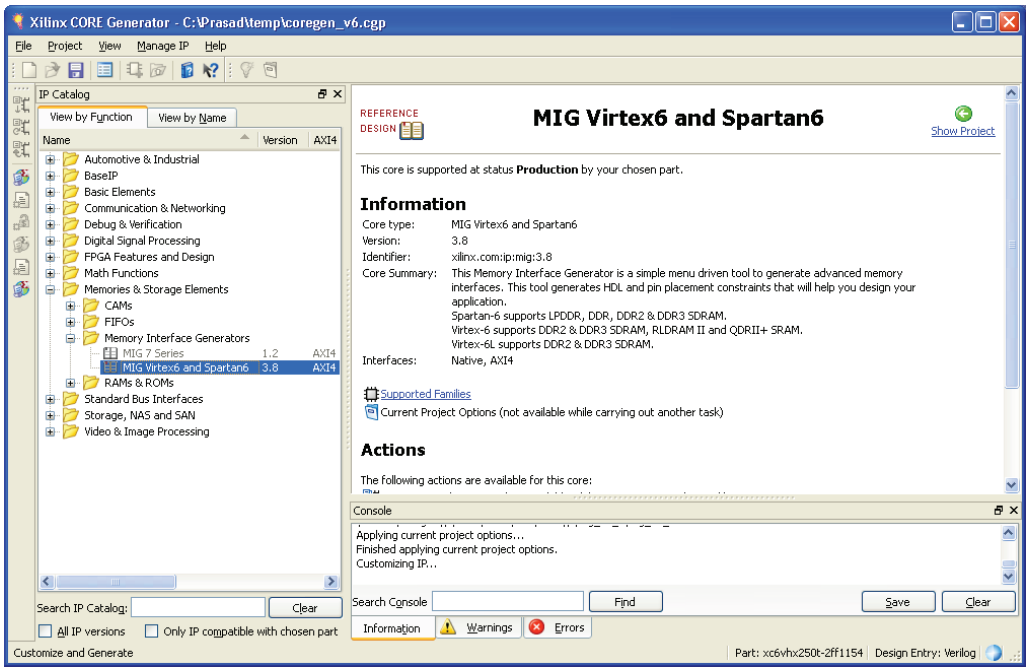
6. Select **Memory Interface Generators (MIG)** by expanding **Memories & Storage Elements** (Figure 1-6).



UG406_c1_06_041311

Figure 1-6: **Virtex_6_MIG_Example_Design Project Page**

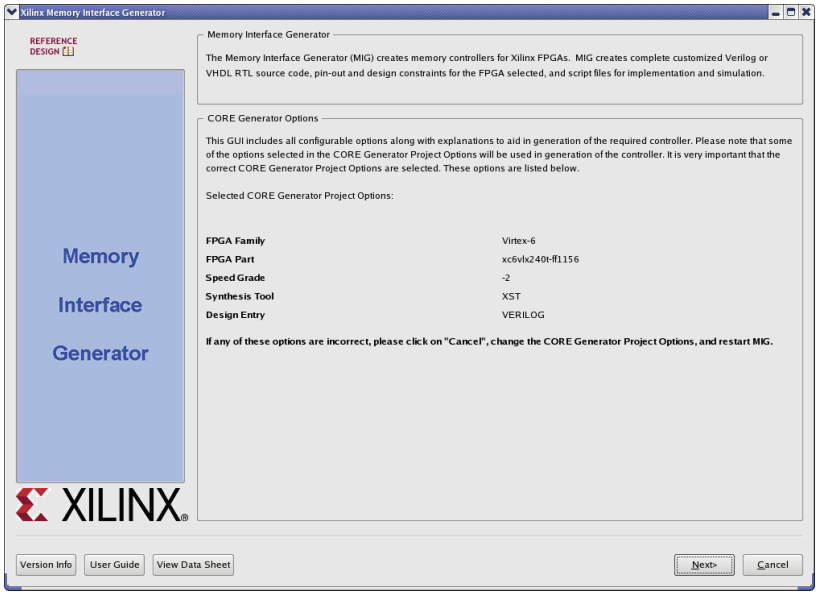
7. Launch the MIG wizard by selecting **Memories & Storage Elements** → **Memory Interface Generator MIG**. Select the latest MIG version under Memory Interface Generators (Figure 1-7).



UG406_c1_07_041311

Figure 1-7: **Starting the MIG Wizard**

8. The options screen in the CORE Generator software displays the details of the selected CORE Generator software options that are selected before invoking the MIG tool (Figure 1-8).



UG406_C1_08_081109

Figure 1-8: **Virtex-6 FPGA Memory Interface Generator Front Page**

9. Click **Next** to display the Output Options window.

MIG Output Options

1. Select the **Create Design** radio button to create a new memory controller design. Enter a component name in the Component Name field (Figure 1-9).

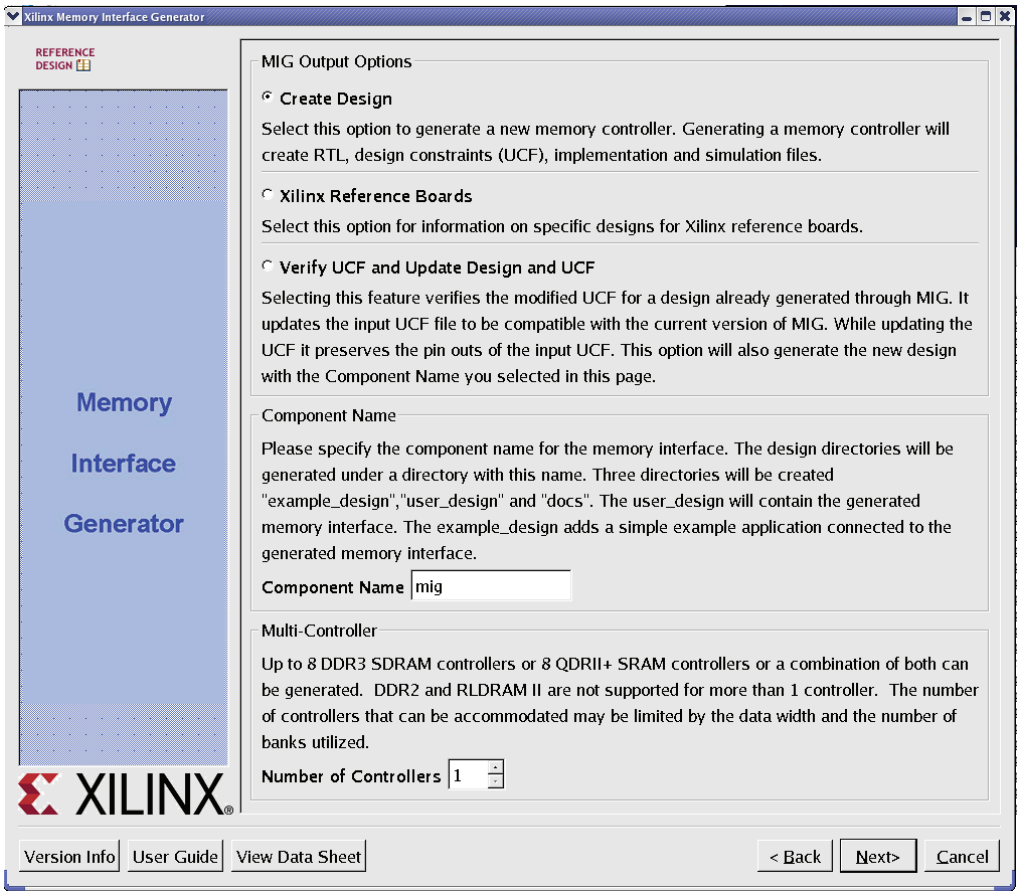


Figure 1-9: **MIG Output Options**

MIG outputs are generated with the folder name `<component name>`.

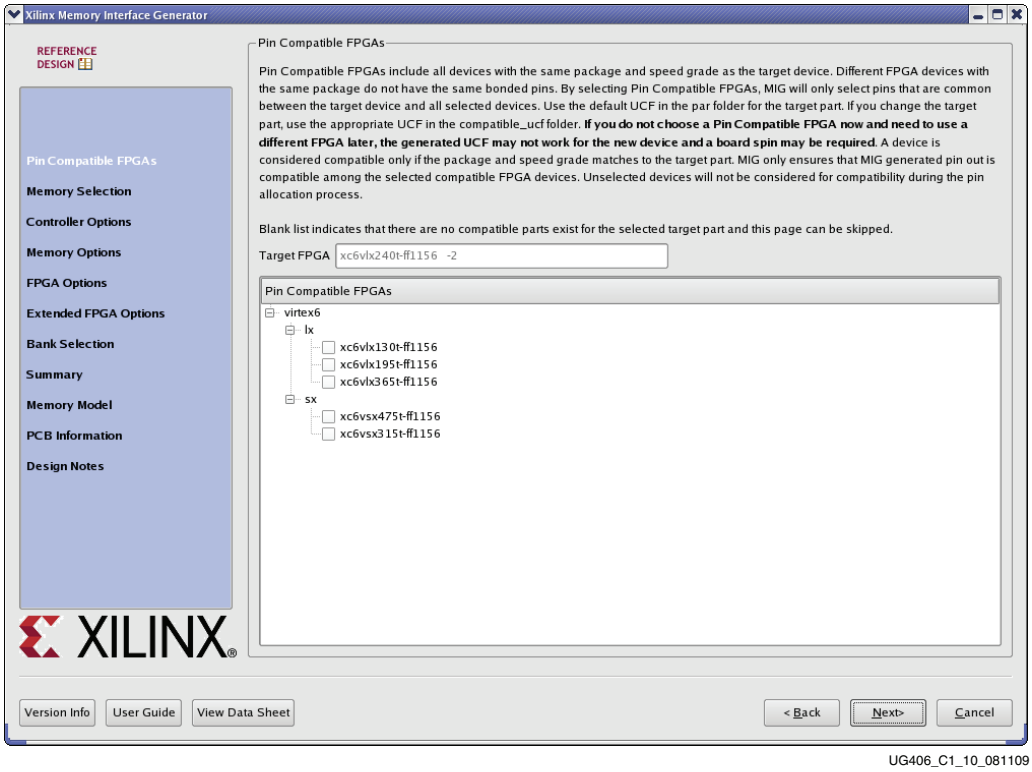
Note: Only alphanumeric characters can be used for `<component name>`. Special characters cannot be used. This name should always start with an alphabetical character and can end with an alphanumeric character.

When invoked from XPS, only the **Create Design** and **Verify UCF and Update Design and UCF** options are available. The component name is also corrected to be the IP instance name from XPS. The **Multi-Controller** option in XPS is not available. Multiple controllers are implemented as multiple separate instances of `axi_v6_ddrx` in XPS.

2. Click **Next** to display the Pin Compatible FPGAs window.

Pin Compatible FPGAs

The Pin Compatible FPGAs window lists FPGAs in the selected family having the same package. If the generated pinout from the MIG tool needs to be compatible with any of these other FPGAs, this option should be used to select the FPGAs with which the pinout has to be compatible (Figure 1-10).



UG406_C1_10_081109

Figure 1-10: **Pin-Compatible Virtex-6 FPGAs**

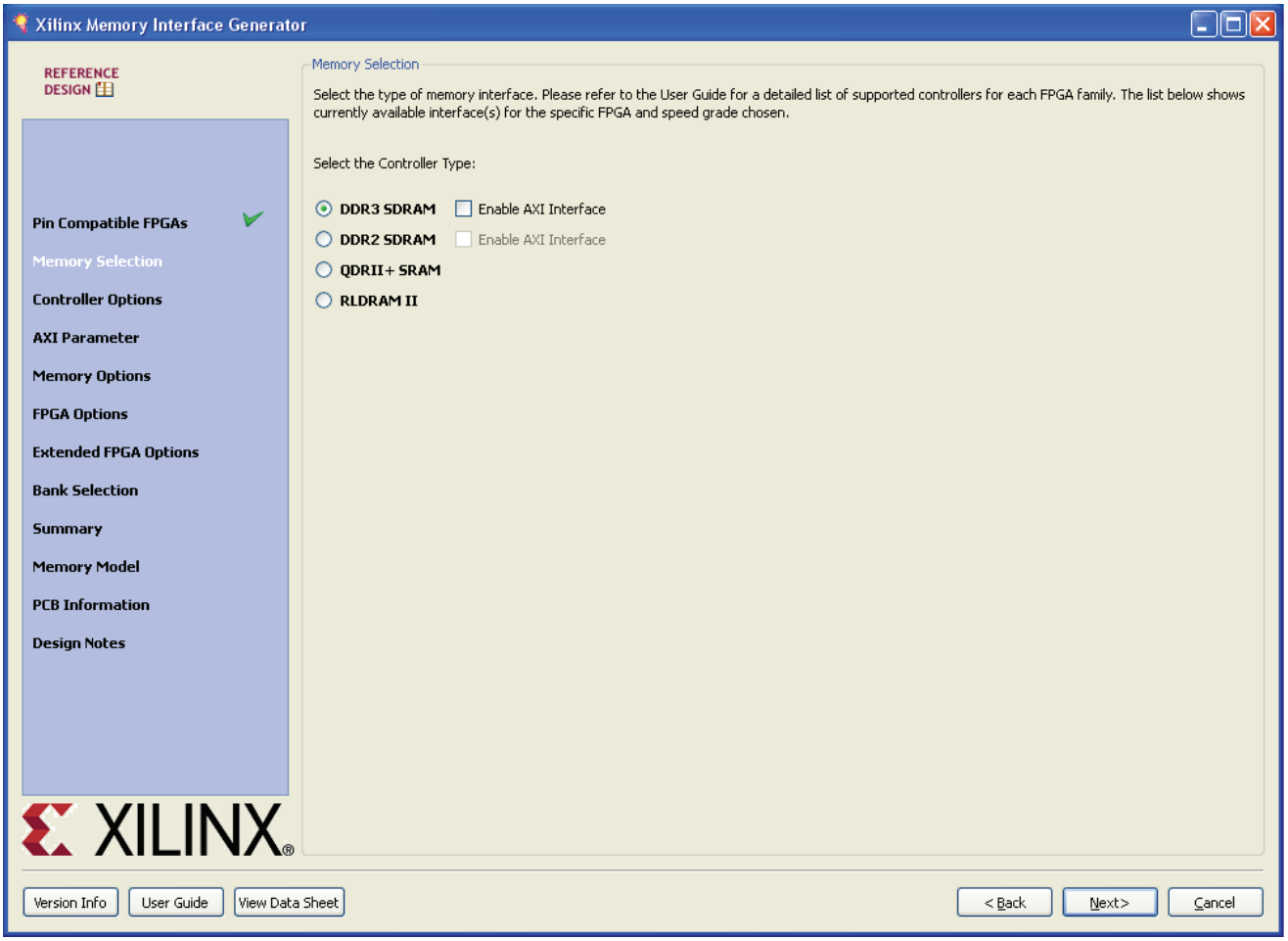
1. Select any of the compatible FPGAs in the list. Only the common pins between the target and selected FPGAs are used by the MIG tool. The name in the text box signifies the target FPGA selected.
2. Click **Next** to display the Memory Selection window.

Creating Virtex-6 FPGA DDR3 Memory Controller Block Design

Memory Selection

This page displays all memory types that are supported by the selected FPGA family.

1. Select the **DDR3 SDRAM** controller type.
2. Click **Next** to display the Controller Options window (Figure 1-11).



UG406_c1_11_073010

Figure 1-11: **Memory Type and Controller Selection**

DDR3 and DDR2 SDRAM designs support memory-mapped AXI4 interfaces. The AXI interface is currently implemented in Verilog only. Users that require the AXI interface should select the language as "Verilog" in the CORE Generator software before invoking the MIG tool. If the AXI4 interface is not selected, the user interface (UI) is the primary interface.

The `axi_v6_ddrx` IP from the EDK flow only supports DDR2 and DDR3 SDRAM memories and has the AXI support always turned on.

Controller Options

This page shows the various controller options that can be selected (Figure 1-12).

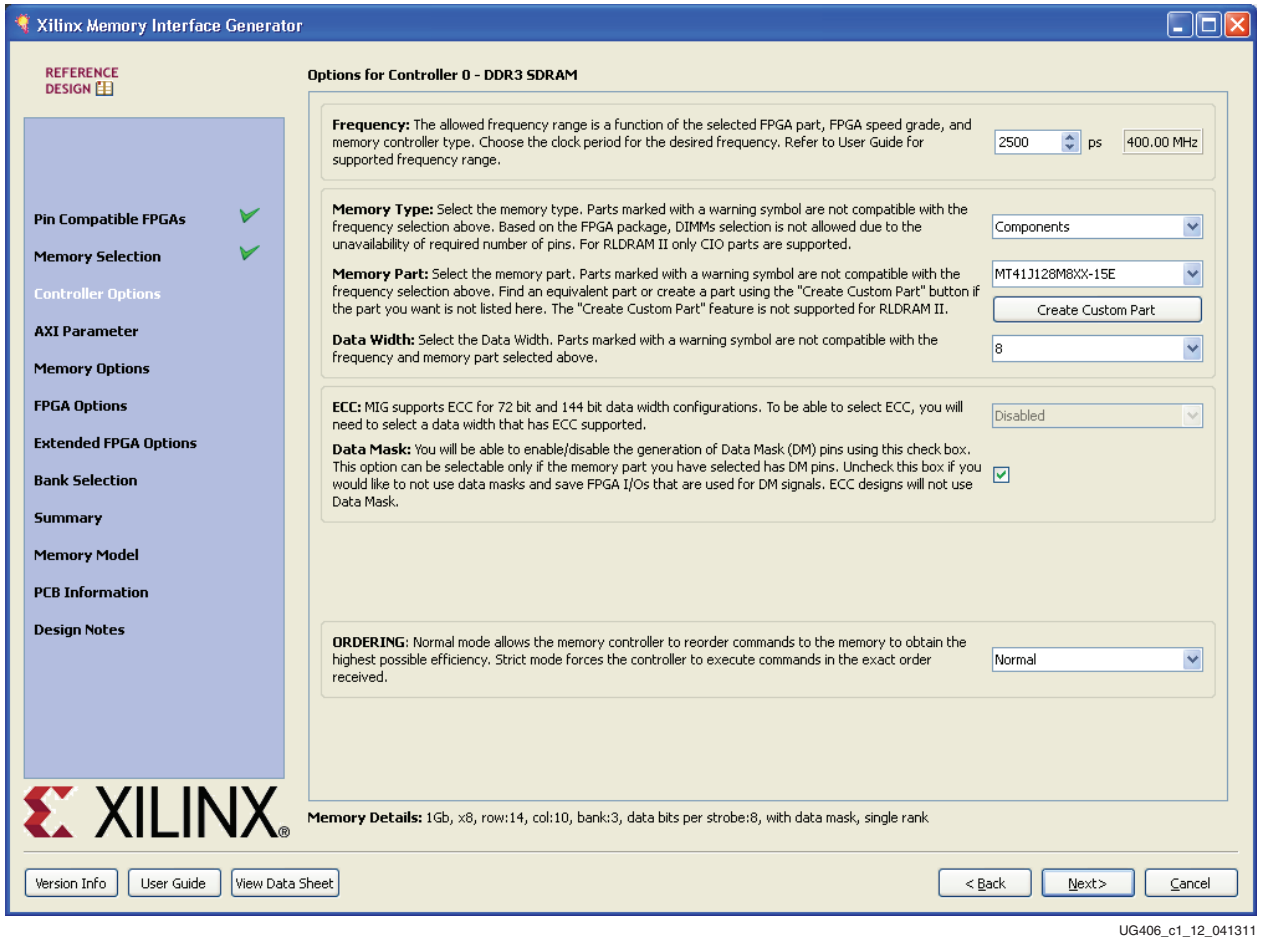


Figure 1-12: **Controller Options Page**

If the design has multiple controllers, the controller options page is repeated for each of the controllers. This page is partitioned into a maximum of nine sections. The number of partitions depends on the type of memory selected. The controller options page also contains these pull-down menus to modify different features of the design:

- **Frequency**: This feature indicates the operating frequency for all controllers (Figure 1-13). The frequency block is limited by factors such as the selected FPGA and device speed grade. In the EDK flow, an extra check box (selected by default) allows the user to specify that the frequency information should be calculated automatically from EDK.

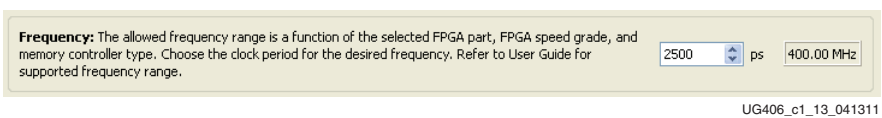


Figure 1-13: **Frequency**

- **Memory Type**: This feature selects the type of memory parts used in the design (Figure 1-14).

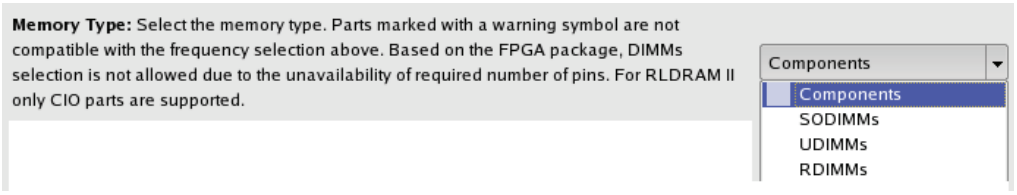


Figure 1-14: **Memory Type**

- **Memory Part**: This option selects a memory part for the design. Selections can be made from the list, or a new part can be created (Figure 1-15).

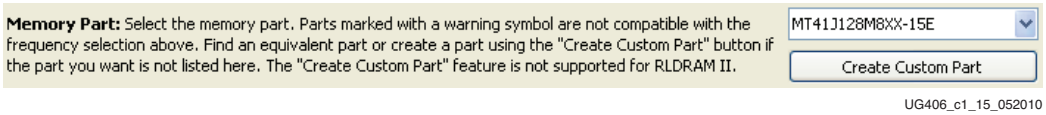


Figure 1-15: **Selecting Memory Type and Memory Part**

- **Data Width**: The data width value can be selected here based on the memory type selected earlier (Figure 1-16). The list shows all supported data widths for the selected part. One of the data widths can be selected. These values are generally multiples of the individual device data widths. In some cases, the width might not be an exact multiple. For example, 16 bits is the default data width for x16 components, but 8 bits is also a valid value.

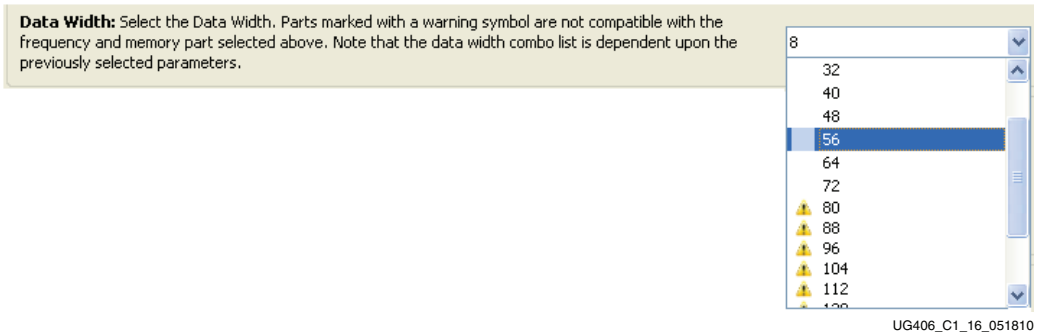


Figure 1-16: **Data Width**

- **Error Correction Code (ECC):** This feature is supported for 72-bit and 144-bit DIMM configurations and allows the memory controller to detect and correct single-bit errors (Figure 1-17). Enabling ECC in AXI mode provides an AXI4-Lite slave interface to interact with the ECC Status/Control registers. This feature is supported in EDK flow only.

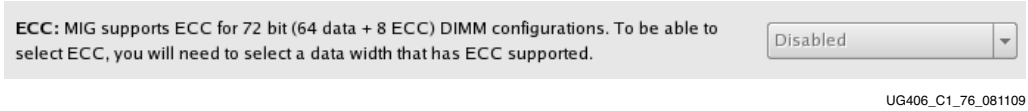


Figure 1-17: **ECC**

- **Data Mask**: This option allocates data mask pins when selected (Figure 1-18). This option should be deselected to deallocate data mask pins and increase pin efficiency. This option is disabled for memory parts that do not support data mask.

Figure 1-18: **Data Mask**

- **Ordering**: This feature allows the memory controller to reorder commands to improve the memory bus efficiency (Figure 1-19).

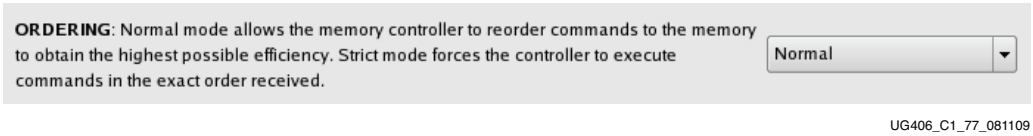


Figure 1-19: **Ordering**

- **Memory Details**: The bottom of the Controller Options page (Figure 1-12, page 19) displays the details for the selected memory configuration (Figure 1-20).

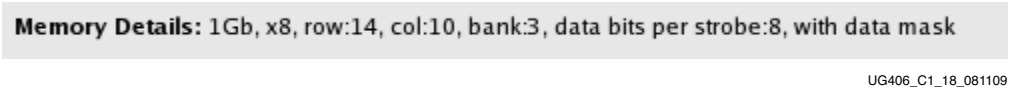


Figure 1-20: **Memory Details**

1. Select the appropriate frequency (Figure 1-13). Either use the spin box or enter a valid value using the keyboard. Values entered are restricted based on the minimum and maximum frequencies supported.

2. Select the appropriate memory part from the list (Figure 1-15). If the required part or its equivalent is unavailable, a new memory part can be created. To create a custom part, click the **Create Custom Part** button below the Memory Part pull-down menu. A new window appears, as shown in Figure 1-21.

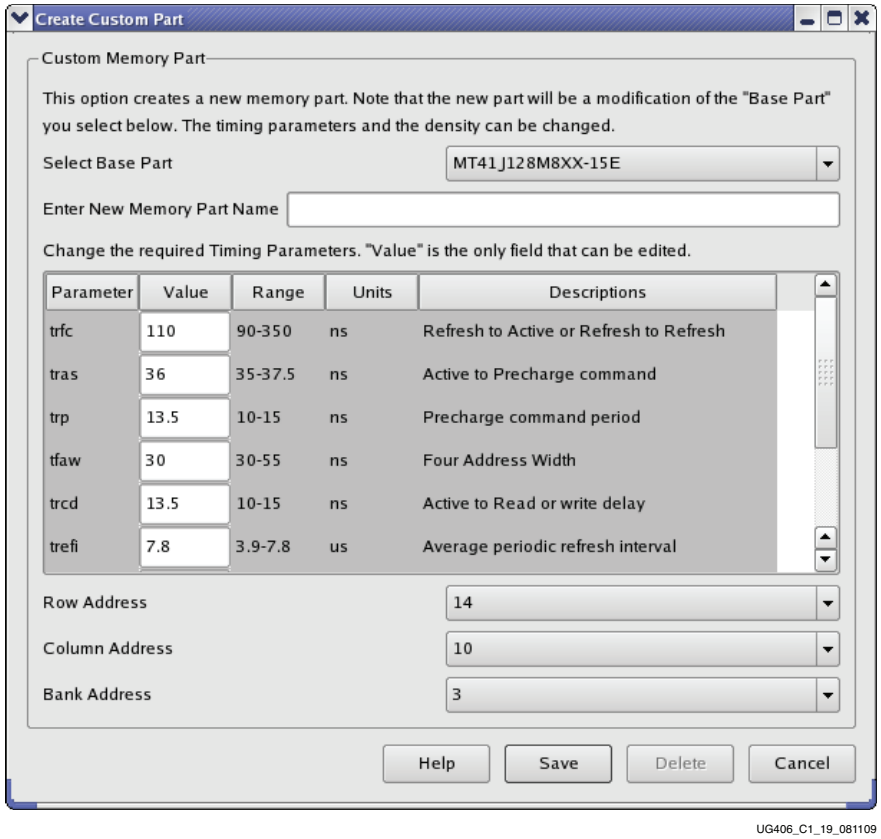


Figure 1-21: **Create Custom Part**

The Create Custom Part window includes all the specifications of the memory component selected in the Select Base Part pull-down menu.

3. Enter the appropriate memory part name in the text box.
4. Select the suitable base part from the Select Base Part list.
5. Edit the value column as needed.
6. Select the suitable values from the Row, Column, and Bank options as per the requirements.
7. After editing the required fields, click the **Save** button. The new part is saved with the selected name. This new part is added in the Memory Parts list on the Controller Options page (Figure 1-13, page 19). It is also saved into the database for reuse and to produce the design.
8. Click **Next** to display the Memory Options window (or the AXI Parameter Options window if AXI Enable is checked on the Memory Type selection page).

AXI Parameter Options

This feature allows the selection of AXI parameters for the controller (Figure 1-22). These are standard AXI parameters or parameters specific to the AXI4 interface. Details are available in the ARM® AMBA® specifications. [Ref 1]

These parameters specific to the AXI4 interface logic can be configured:

- **Address Width** and **AXI ID Width**: When invoked from XPS, address width and ID width settings are automatically set by XPS so the options are not shown.
- **Base** and **High Address**: Sets the system address space allocated to the memory controller. These values must be a power of 2 with a size of at least 4 KB, and the base address must be aligned to the size of the memory space.
- **Narrow Burst Support**: Deselecting this option allows the AXI4 interface to remove logic to handle AXI narrow bursts to save resources and improving timing. XPS normally auto-calculates whether narrow burst support can be disabled based on the known behavior of connected AXI masters.

  Inferred AXI interconnect parameter settings are also available in the EDK flow. Details on the interconnect parameters and how they are handled in XPS are available in the EDK documentation.

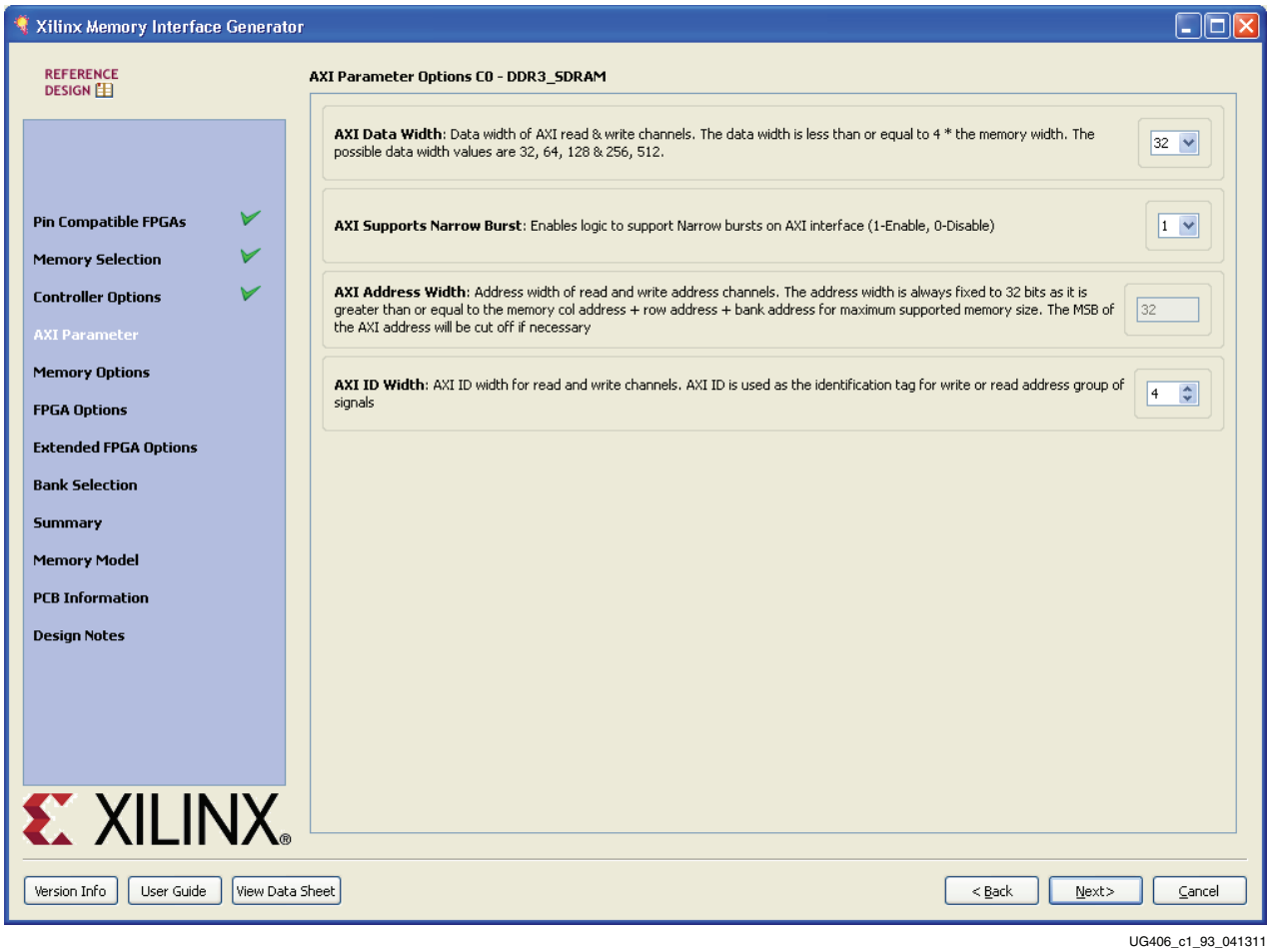


Figure 1-22: **Setting AXI Parameter Options**

Setting the DDR3 Memory Parameter Option

This feature allows the selection of various memory mode register values, as supported by the controller's specification (Figure 1-23).

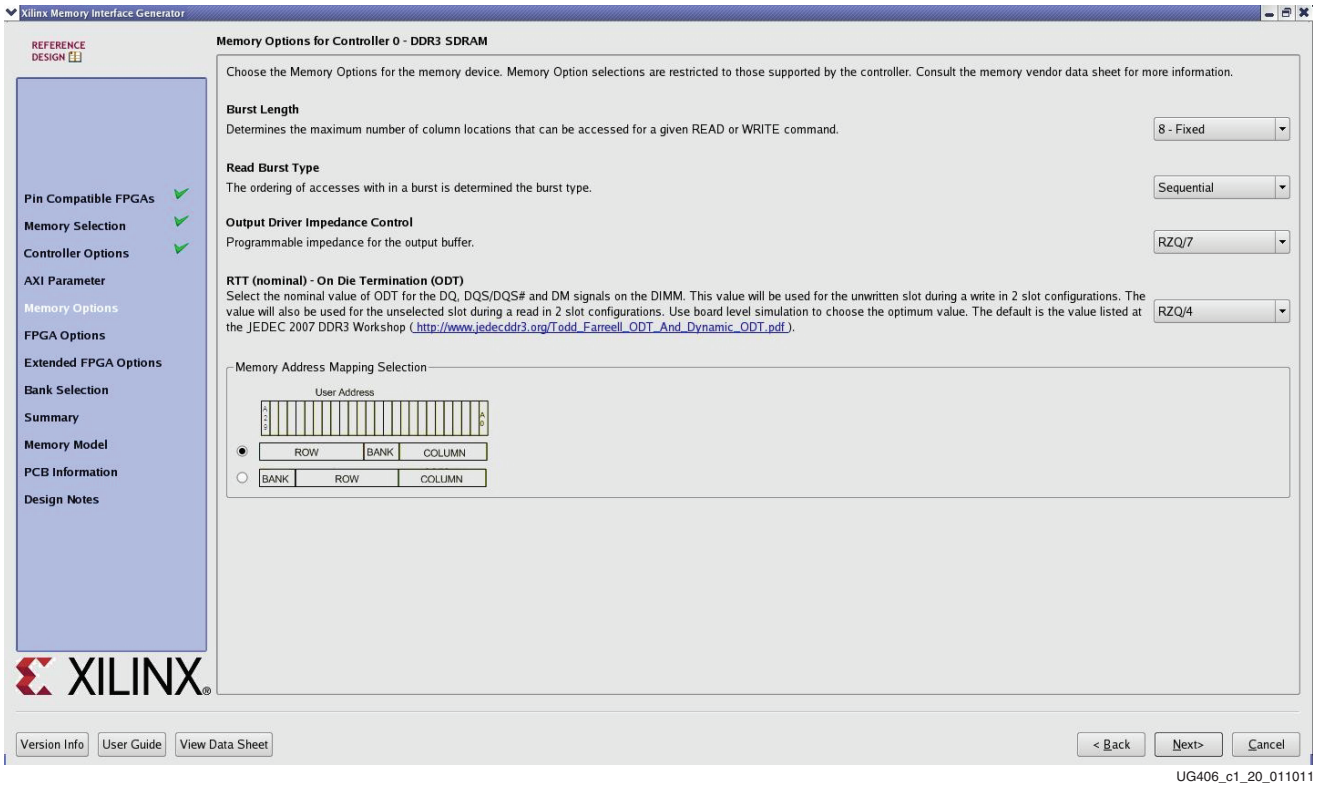


Figure 1-23: **Setting Memory Mode Options**

The mode register value is loaded into the load mode register during initialization. Burst length is automatically set by the MIG tool to provide the best performance.

To optimize controller efficiency, the addressing mapping scheme is user-selectable as **Bank-Row-Column** or **Row-Bank-Column**, depending on organization of application data.

Click **Next** to display the FPGA Options window (Figure 1-24).

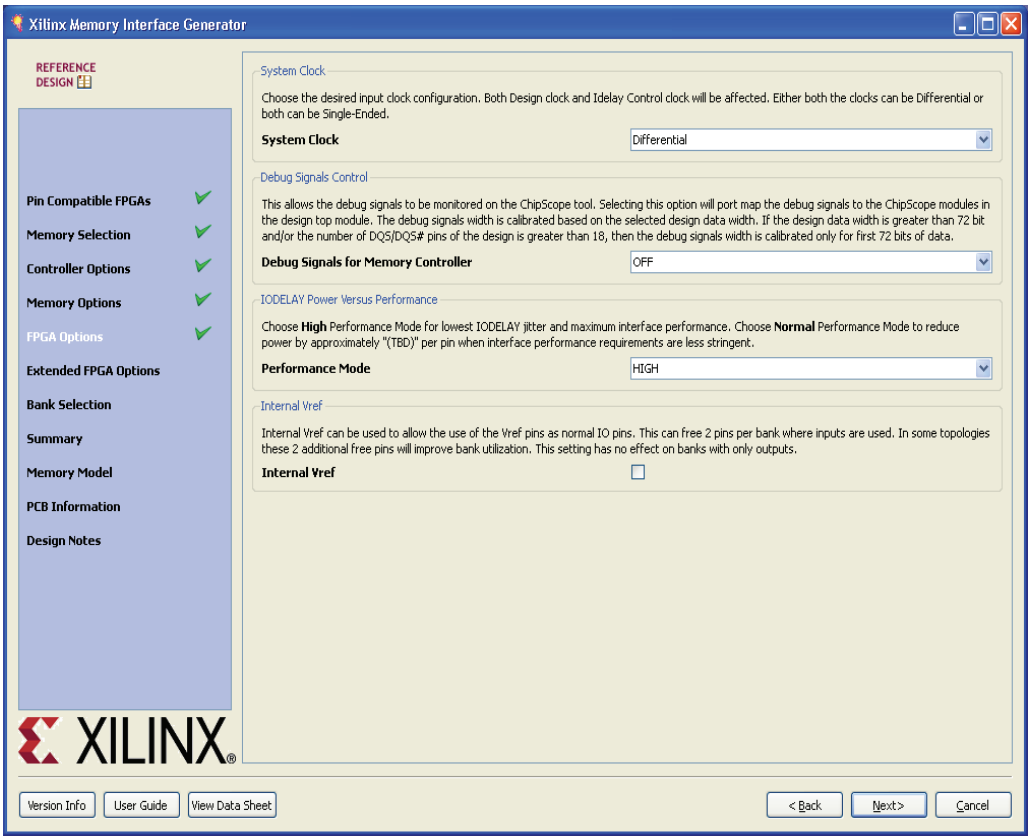


Figure 1-24: **FPGA Options**

- **System Clock.** This option (not available in the EDK flow) selects the input clock type: single-ended or differential (Figure 1-25).

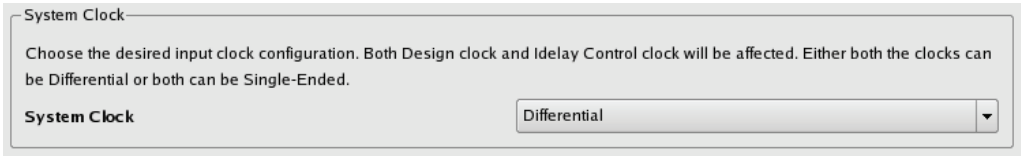


Figure 1-25: **System Clock Selection**

- **Debug Signals Control.** Selecting this option (not available in the EDK flow) enables calibration status and user port signals to be port mapped to the ChipScope™ analyzer modules in the `design_top` module (Figure 1-26). This helps in monitoring traffic on the user interface port with the ChipScope analyzer. When the generated design is run in batch mode using `ise_flow.bat` in the design's `par` folder, the CORE Generator software is called to generate ChipScope analyzer modules (that is, NGC files are generated). Deselecting the Debug Signals Control option leaves the debug signals unconnected in the `design_top` module. No ChipScope analyzer modules are instantiated in the `design_top` module and no ChipScope analyzer modules are generated by the CORE Generator software. The debug port is always disabled for functional simulations.

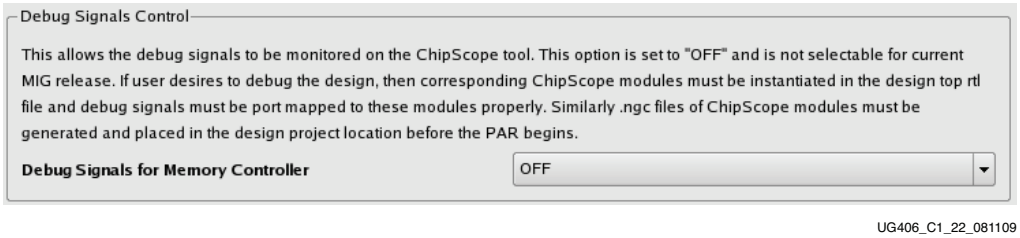


Figure 1-26: **Debug Signals Control**

- **Internal Vref Selection.** Internal Vref can be used for data group banks to allow the use of the VREF pins for normal I/O usage.

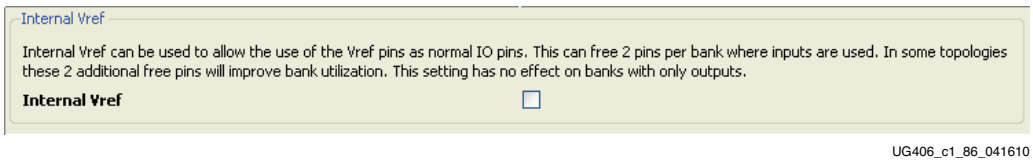


Figure 1-27: **Internal Vref Selection**

Click **Next** to display the DCI description window (Figure 1-28).

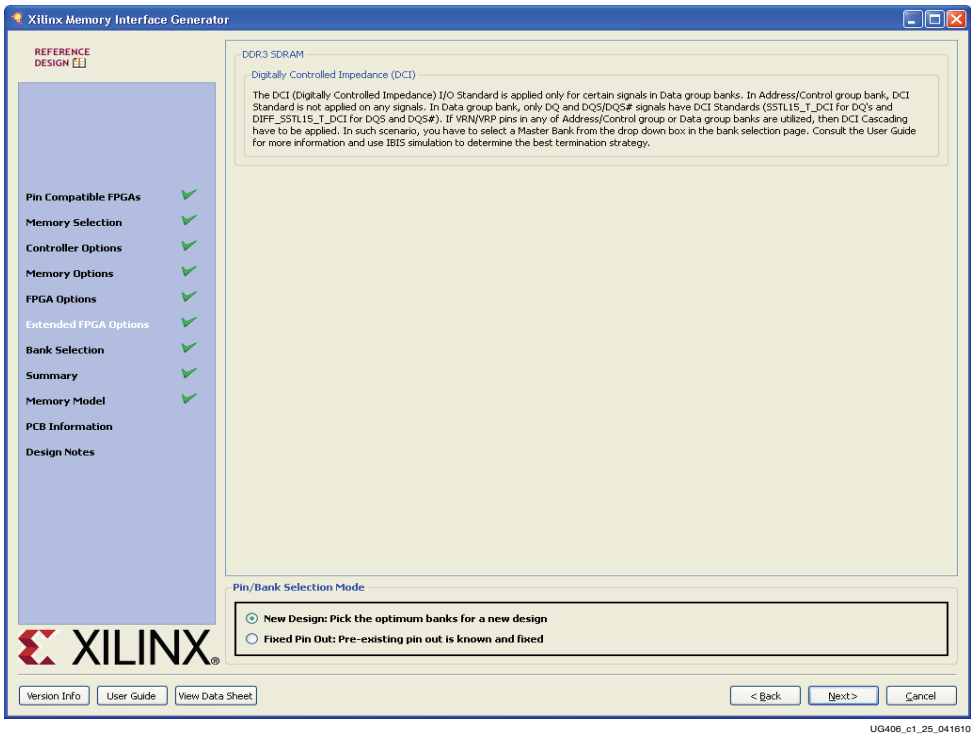
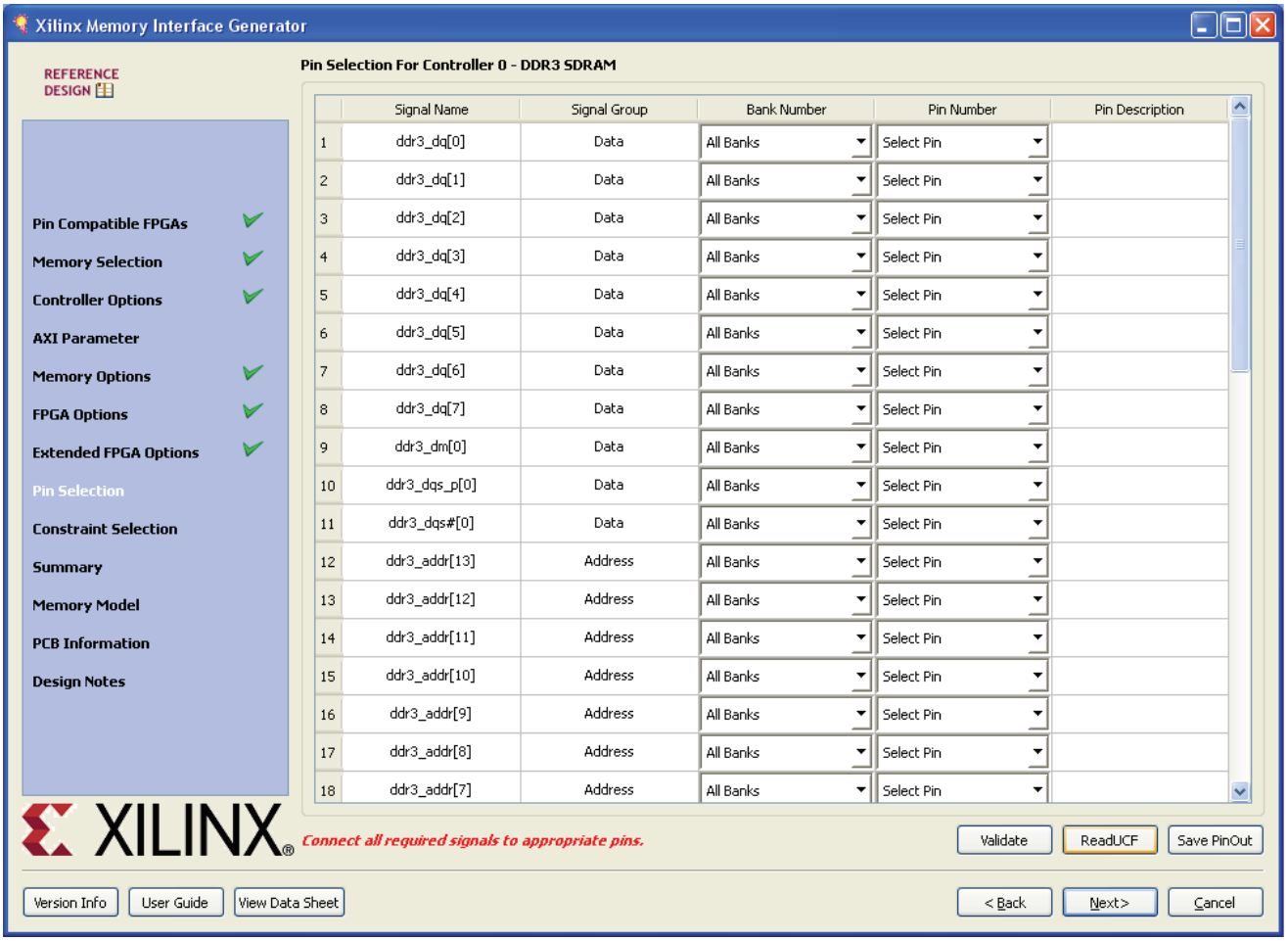


Figure 1-28: **DCI Description**

- **Digitally Controlled Impedance (DCI)**. The DCI option allows the use of the FPGA's on-chip internal resistors for termination. DCI must be used for DQ and DQS/DQS# signals. DCI cascade might have to be used, depending on the pinout and bank selection.

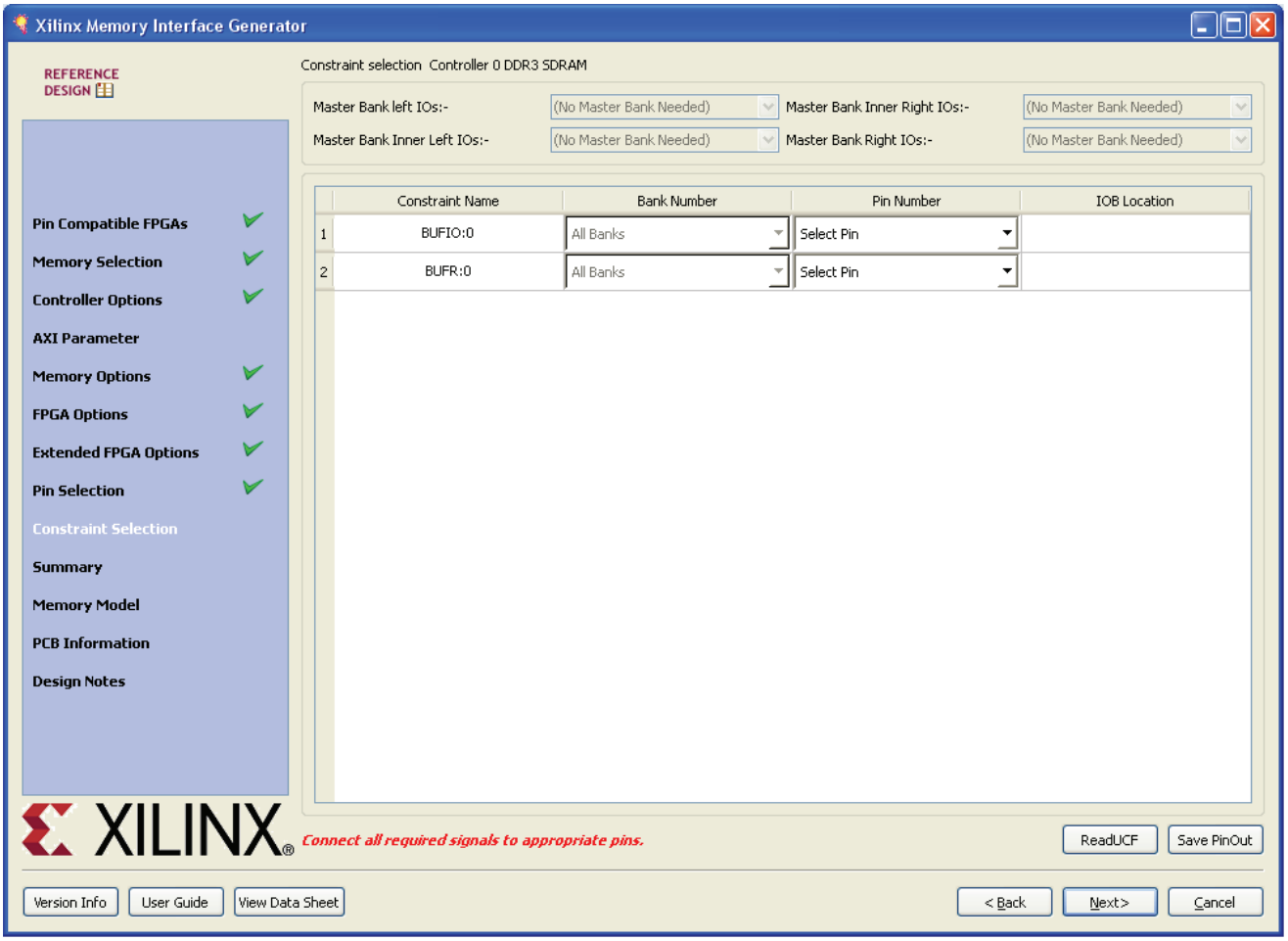
- **Pin/Bank Selection**. The Pin/Bank Selection mode allows the user to specify an existing pinout and generate a new RTL core for this pinout, or pick banks for a new design. Figure 1-29 shows the option for using an existing pinout. The user must assign the appropriate pins for each signal. A choice of banks is available to narrow down the list of pins. It is not mandatory to select the banks prior to selection of the pins. Click **Validate** to check a pinout against the MIG pinout rules.



UG406_c1_88_072310

Figure 1-29: **Pin Selection Using an Existing Pinout**

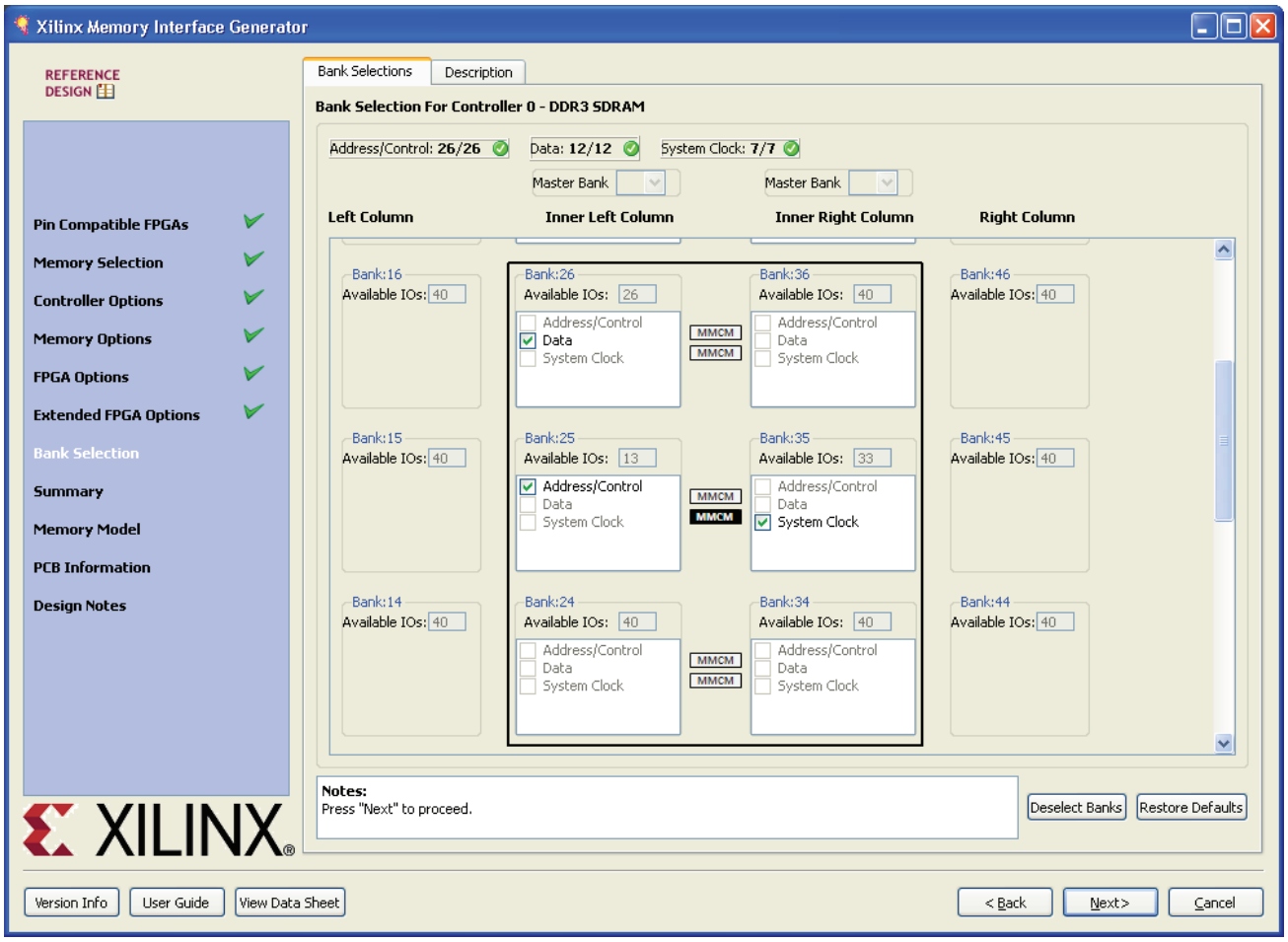
- For Virtex-6 FPGAs, the user must also reserve IOBs for BUFR/BUFIO signals. Figure 1-30 shows how to reserve pins corresponding to the IOBs for such signals. The user can also select the master banks for the I/Os in this window.



UG406_c1_89_072310

Figure 1-30: **Reserving IOBs for BUFR/BUFIO Signals**

Click **New Design** in the Pin/Bank Selection mode, then click **Next** to display the Bank Selection window (Figure 1-31).



UG406_c1_27_041610

Figure 1-31: **Bank Selection**

Bank Selection

This feature allows the selection of banks for the memory interface. Banks can be selected for different classes of memory signals, such as:

- Address and control signals
- Data signals
- System clock

For customized settings, click **Deselect Banks** and select the appropriate bank and memory signals. Click **Next** to move to the next page if the default setting is used.

- **Address Group Selection.** Select the address/control group from one of the white banks. Only inner columns are allowed (Figure 1-32).

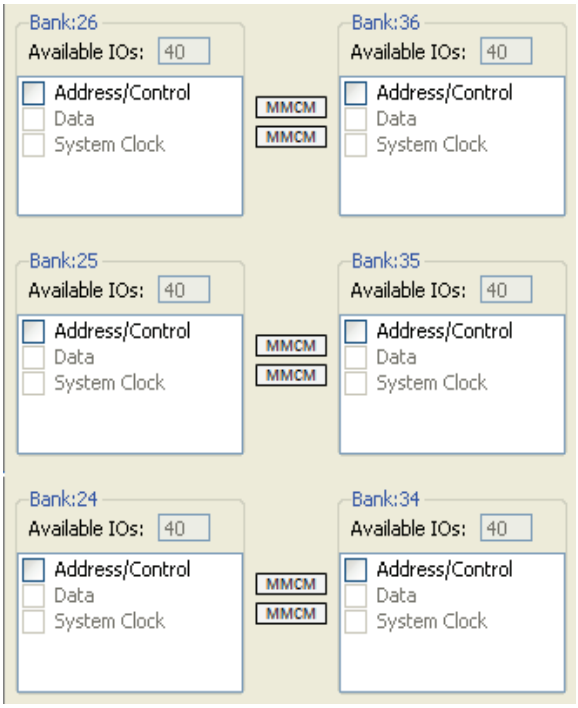


Figure 1-32: **Address Group Selection**

- **Data Group Selection.** After the address group is assigned, the MIG tool only allows the data group to be chosen within the banks inside the black box, as shown in Figure 1-33.

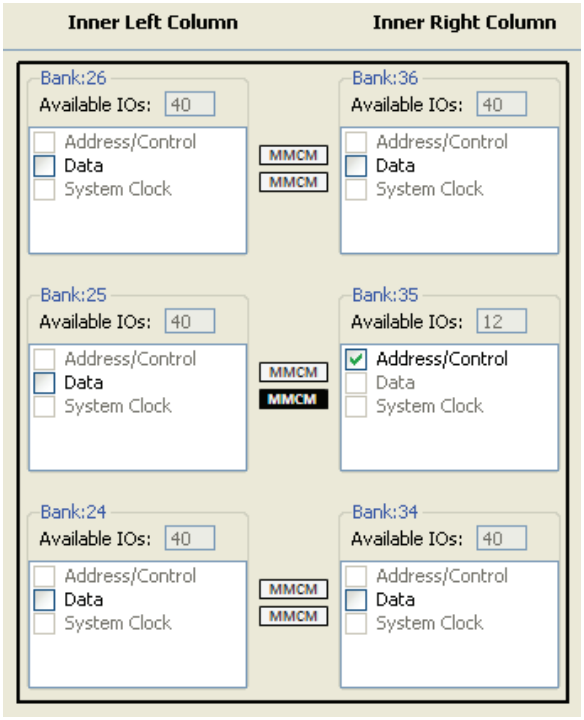


Figure 1-33: **Data Group Selection**

- **Master Bank Selection.** Two extra pins are required to set up a DCI reference that provides better signal integrity. Select the master bank from one of the list of banks shown in the pull-down menu (Figure 1-34).

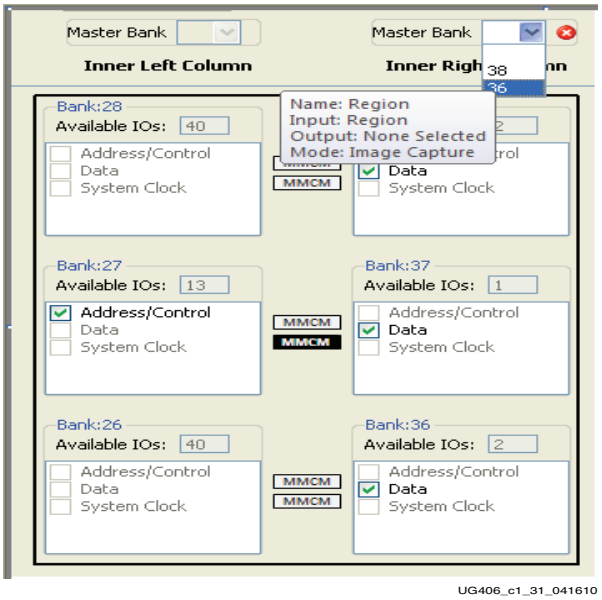


Figure 1-34: **Master Bank Selection**

- **System Clock Group Selection.** Select the system clock group from any one of the enabled banks (Figure 1-35).

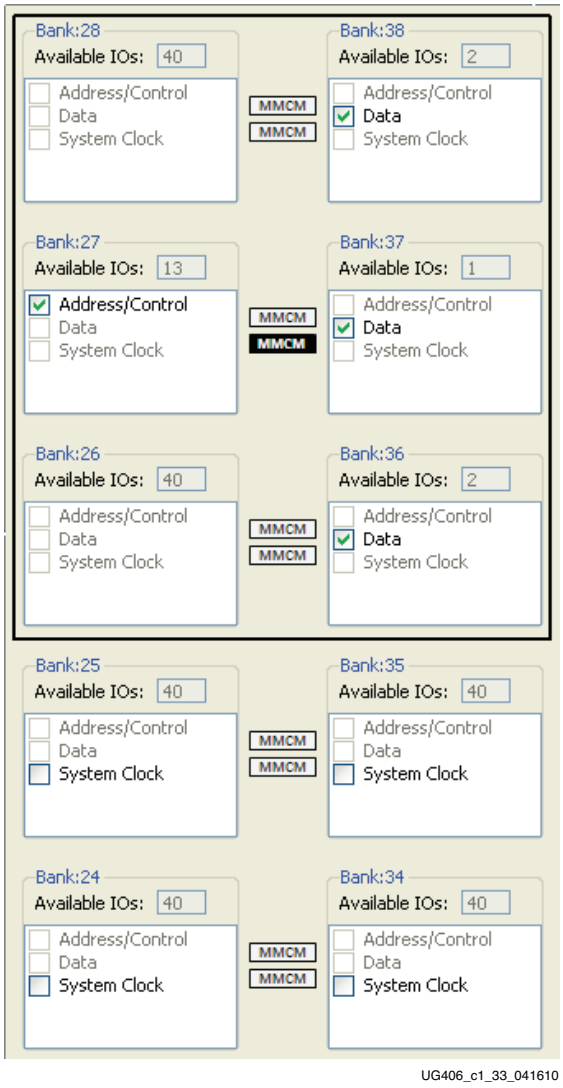


Figure 1-35: **System Clock Group Selection**

To unselect the banks that are selected, click the **Deselect Banks** button. To restore the defaults, click the **Restore Defaults** button.

For detailed bank selection rules, click on the **Description** tab (Figure 1-36).

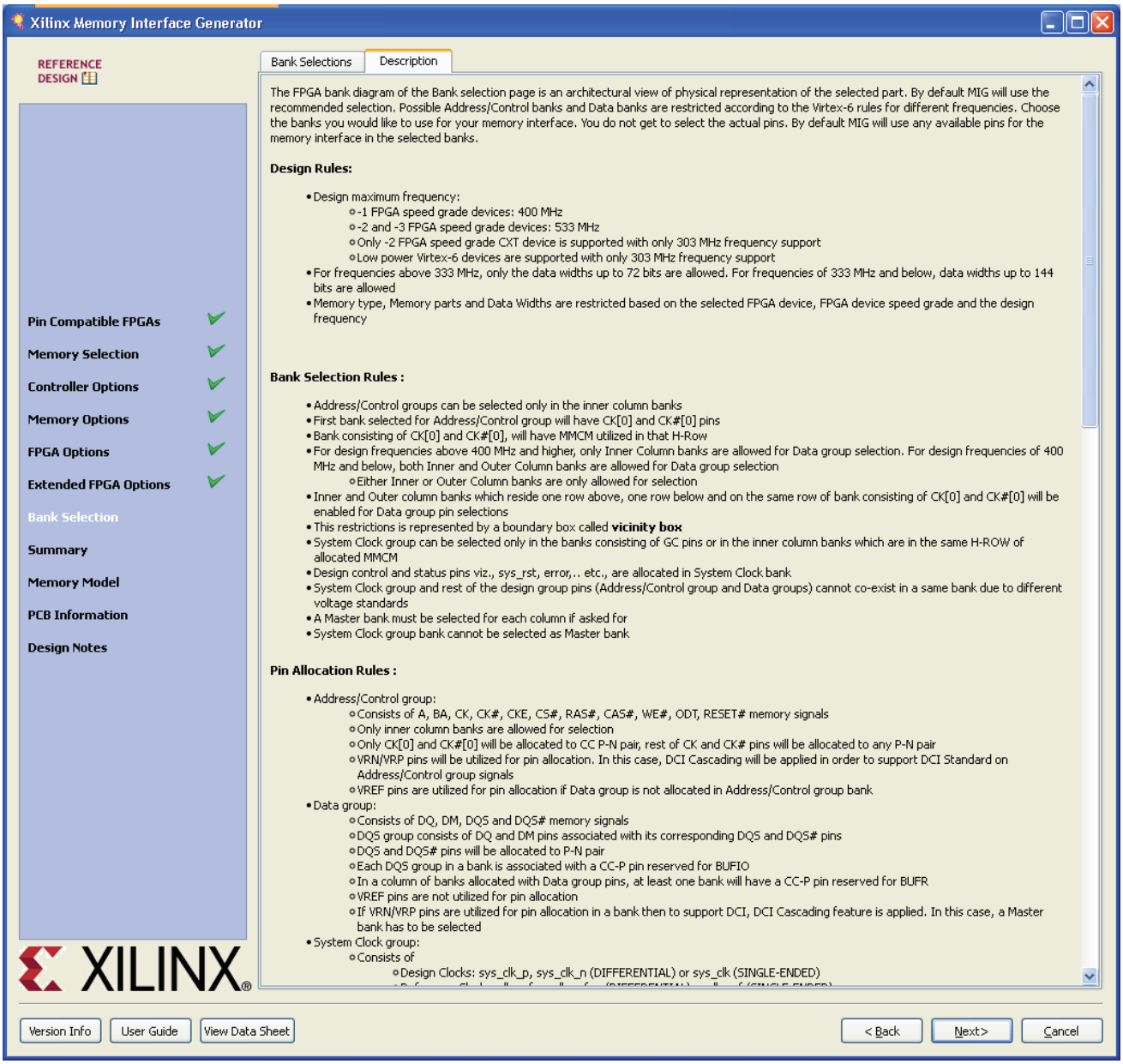
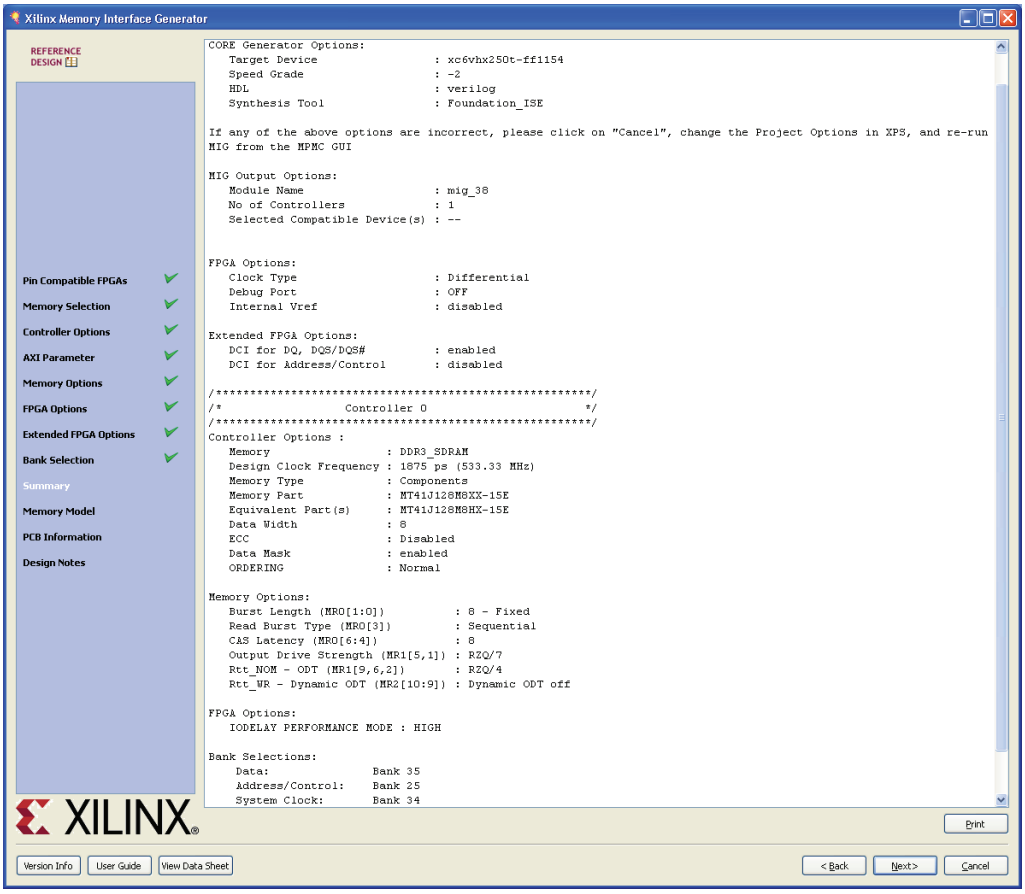


Figure 1-36: **Bank Selection Rule Description**

Click **Next** to display the Summary window.

Summary

This window provides the complete details about the Virtex-6 FPGA memory core selection, interface parameters, CORE Generator software options, and FPGA options of the active project (Figure 1-37). In the EDK flow, this is the last screen, and clicking the **Finish** button (replaces the **Next** button) saves the changes and returns the user to the XPS tool.



UG406_c1_35_041311

Figure 1-37: **Summary**

Memory Model License

The MIG tool can output a chosen vendor's memory model for simulation purposes (not available in the EDK flow) for memories such as DDR, DDR2, and DDR3 SDRAMs. To access the models in the output `sim` folder, click the license agreement (Figure 1-38). Read the license agreement and check the **Accept License Agreement** box to accept it. If the license agreement is not agreed to, the memory model is not made available. A memory model is necessary to simulate the design.

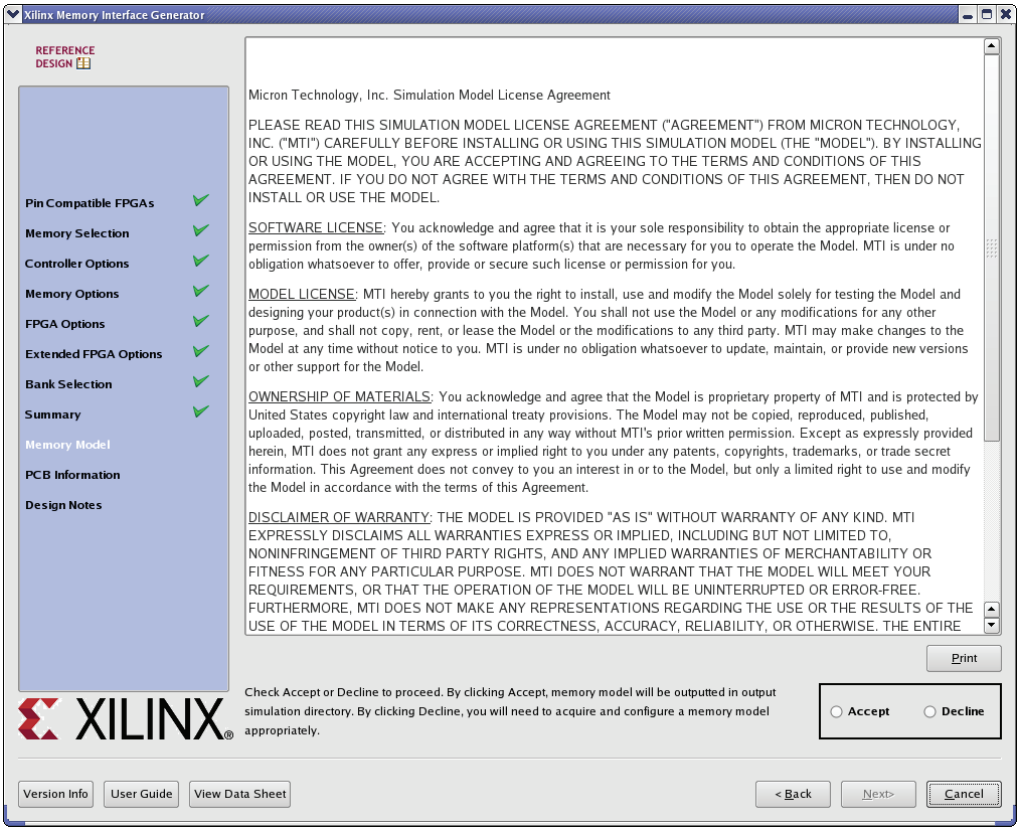


Figure 1-38: **License Agreement**

Click **Next** to move to PCB Information page.

PCB Information

This page displays the PCB-related information to be considered while designing the board that uses the MIG tool generated designs. Click **Next** to move to the Design Notes page.

Design Notes

Click the **Generate** button (not available in the EDK flow) to generate the design files. The MIG tool generates two output directories: `example_design` and `user_design`. After generating the design, the MIG GUI closes.

Directory Structure and File Descriptions

Overview

Output Directory Structure

The MIG tool outputs (non-EDK flow) are generated with folder name `<component name>`.

Note: In the EDK flow, the MIG project file is stored in `<EDK Project Directory>/data/<Instance Name>_mig_saved.prj` and should be retained with the XPS project. The MIG UCF with pin location information is written to `<EDK Project Directory>/__xps/<Instance Name>/mig.ucf` and is translated to an EDK core-level UCF at `<EDK Project Directory>/implementation/<Instance Name>_wrapper/<Instance Name>.ucf` during builds.

Figure 1-39 shows the output directory structure of the selected memory controller design from the MIG tool. In the `<component name>` directory, three folders are created:

- `docs`
- `example_design`
- `user_design`

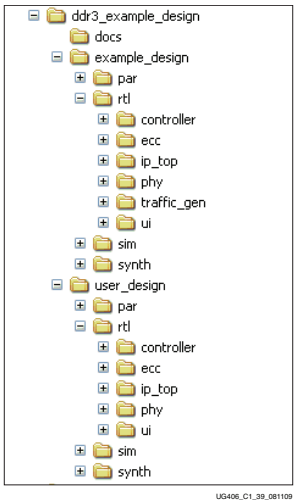


Figure 1-39: **Directory Structure**

Directory and File Contents

The Virtex-6 device core directories and their associated files are listed in this section.

`<component name>/docs`

The `docs` folder contains the PDF documentation.

`<component name>/example_design/`

The `example_design` folder contains three folders, namely, `controller`, `example_top`, and `phy`.

`example_design/rtl/controller`

This directory contains the memory controller that is instantiated in `example_design` (Table 1-1).

Table 1-1: **Modules in `example_design/rtl/controller` Directory**

Name	Description
`arb_mux.v/vhd`	This is the top-level module for arbitration logic.
`arb_row_col.v/vhd`	This block receives requests to send row and column commands from the bank machines and selects one request, if any, for each state.
`arb_select.v/vhd`	This module selects a row and column command from the requested information provided by the bank machines.
`bank_cntrl.v/vhd`	This structural block instantiates the three subblocks that comprise the bank machine.
`bank_common.v/vhd`	This module computes various items that cross all of the bank machines.
`bank_compare.v/vhd`	This module stores the request for a bank machine.
`bank_mach.v/vhd`	This is the top-level bank machine block.
`bank_queue.v/vhd`	This is the bank machine queue controller.
`bank_state.v/vhd`	This is the primary bank state machine.
`col_mach.v/vhd`	This module manages the DQ bus.
`mc.v/vhd`	This is the top-level module of the MC.
`rank_cntrl.v/vhd`	This module manages various rank-level timing parameters.
`rank_common.v/vhd`	This module contains logic common to all rank machines. It contains a clock prescaler and arbiters for refresh and periodic read.
`rank_mach.v/vhd`	This is the top-level rank machine structural block.
`round_robin_arb.v/vhd`	This is a simple round-robin arbiter.

`example_design/rtl/top`

This directory contains the example design (Table 1-2).

Table 1-2: **Modules in `example_design/rtl/top` Directory**

Name	Description
`example_design_top.v/vhd`	This top-level module serves as an example for connecting the user design to the Virtex-6 FPGA memory interface core.
`clk_ibuf.v/vhd`	This module instantiates the input clock buffer.
`iodelay_ctrl.v/vhd`	This module instantiates IDELAYCNTRL primitives needed for IODELAY use.
`infrastructure.v/vhd`	This module helps in clock generation and distribution, and reset synchronization.

Table 1-2: **Modules in `example_design/rtl/top` Directory *(Cont'd)***

Name	Description
`mem_intfc.v/vhd`	This is the top-level memory interface block with the native interface.
`memc_ui_top.v/vhd`	This is the top-level memory interface controller wrapper with the user interface.

`example_design/rtl/phy`

This directory contains the Virtex-6 FPGA memory interface PHY implementation (Table 1-3).

Table 1-3: **Modules in `example_design/rtl/phy` Directory**

Name	Description
`circ_buffer.v/vhd`	This is a circular buffer for synchronizing signals between clock domains.
`phy_ck_iob.v/vhd`	This module provides clock forwarding to memory and pad loopback into the FPGA.
`phy_clock_io.v/vhd`	This is the top-level module for CK/CK# clock forwarding to memory and feedback into the FPGA.
`phy_control_io.v/vhd`	This module instantiates IOBs for output-only control and address signals to the SDRAM.
`phy_data_io.v/vhd`	This is the top-level module for all data-related (DQ, DQS, DM) IOB logic.
`phy_dly_ctrl.v/vhd`	This module provides centralized control for all IODELAY elements in interface IODELAYs.
`phy_dm_iob.v/vhd`	This module places the data mask signals into the IOBs.
`phy_dq_iob.v/vhd`	This module instantiates I/O-related logic for DQ.
`phy_dqs_iob.v/vhd`	This module instantiates I/O-related logic for DQS.
`phy_init.v/vhd`	This module provides memory initialization and overall master state control during initialization and calibration.
`phy_pd.v/vhd`	This module provides phase detector calibration.
`phy_pd_top.v/vhd`	This is the top-level module of the phase detector.
`phy_rdclk_gen.v/vhd`	This module generates and distributes the capture clock.
`phy_rdctrl_sync.v/vhd`	This module synchronizes the read control signal from MC/PHY rdlvl logic to read capture logic.
`phy_rddata_sync.v/vhd`	This module synchronizes captured read data to the core clock domain.
`phy_rdlvl.v/vhd`	This module provides read-leveling calibration logic.
`phy_read.v/vhd`	This is the top-level module for the PHY read logic.
`phy_top.v/vhd`	This is the top-level module for memory in the PHY interface.

Table 1-3: **Modules in `example_design/rtl/phy` Directory *(Cont'd)***

Name	Description
phy_write.v/vhd	This module delays various write control signals based on user-specific timing parameters (for example, CAS write latency).
phy_wrlvl.v/vhd	This module provides calibration for write leveling.
rd_bitslip.v/vhd	This module shifts data captured by the ISERDES in bit time increments to provide aligned data across all DQS groups.

`example_design/rtl/traffic_gen`

This directory contains the traffic generator that provides the stimulus to the Virtex-6 FPGA memory controller (Table 1-4).

Table 1-4: **Modules in `example_design/rtl/traffic_gen` Directory**

Name	Description
mcb_traffic_gen.v/vhd	This is the top level of the traffic generator.
cmd_gen.v/vhd	This is the command generator. This module provides independent control of generating the types of commands, addresses, and burst lengths.
cmd_prbs_gen.v/vhd	This is a pseudo-random binary sequence (PRBS) generator for generating PRBS commands, addresses, and burst lengths.
mcb_flow_control.v/vhd	This module generates flow control logic between the memory controller core and the cmd_gen, read_data_path, and write_data_path modules.
read_data_path.v/vhd	This is the top level for the read datapath.
read_posted_fifo.v/vhd	This module stores the read command that is sent to the memory controller, and its FIFO output is used to generate expect data for read data comparisons.
rd_data_gen.v/vhd	This module generates timing control for reads and ready signals to mcb_flow_control.v/vhd.
write_data_path.v/vhd	This is the top level for the write datapath.
wr_data_g.v/vhden.v/vhd	This module generates timing control for writes and ready signals to mcb_flow_control.v/vhd.
v6_data_gen.v/vhd	This module generates different data patterns.
a_fifo.v/vhd	This is a synchronous FIFO using LUTRAMs.
data_prbs_gen.v/vhd	This is a 32-bit linear feedback shift register (LFSR) for generating PRBS data patterns.
init_mem_pattern_ctr.v/vhd	This module generates flow control logic for the traffic generator.

example_design/rtl/ui

This directory contains the user interface code that mediates between the native interface of the memory controller and user applications (Table 1-5).

Table 1-5: **Modules in `example_design/rtl/ui_top` Directory**

Name	Description
ui_cmd.v/vhd	This is the user interface command port.
ui_rd_data.v/vhd	This is the user interface read buffer. It reorders read data returned from the memory controller back to the request order.
ui_wr_data.v/vhd	This is the user interface write buffer. It reorders write data returned from the memory controller back to the request order.
ui_top.v/vhd	This is the top level of the memory controller user interface.

<component name>/example_design/par

Table 1-6 lists the modules in the example_design/par directory.

Table 1-6: **Modules in `example_design/par` Directory**

Name	Description
<component_name>_example_design.ucf	This is the UCF for the core and the example design.
create_ise.bat	Double-clicking this file creates an ISE tools project that contains the recommended build options for the design. Double-clicking the ISE tools project file opens up the ISE software in GUI mode with all the project settings.
ise_flow.bat	This script file runs the design through synthesis, build, map, and par. This file sets all the required options and should be referred to for the recommended build options for the design.

Caution! The ise_flow.bat file in the par folder of the <component name> directory contains the recommended build options for the design. Failure to follow the recommended build options could produce unexpected results.

<component name>/example_design/sim

Table 1-7 lists the modules in the example_design/sim directory.

Table 1-7: **Modules in `example_design/sim` Directory**

Name	Description
ddr2_model.v ddr3_model.v	These are the DDR2 and DDR3 SDRAM memory models.
ddr2_model_parameters.vh ddr3_model_parameters.vh	These files contain the DDR2 and DDR3 SDRAM memory model parameter setting.

Table 1-7: **Modules in `example_design/sim` Directory *(Cont'd)***

Name	Description
`glbl.v`	This file is used for initializing the simulation environment.
`sim.do`	This SDC file has design constraints for the Synplify Pro synthesis tool.
`sim.exe`	Double-clicking this file causes the design to be automatically simulated using the ModelSim simulator.
`sim_tb_top.v/vhd`	This is the simulation top file.

`<component name>/user_design`

The `example_design` folder contains three folders, namely, `controller`, `example_top`, and `phy`.

`user_design/rtl/controller`

This directory contains the memory controller that is instantiated in the example design (Table 1-8).

Table 1-8: **Modules in `user_design/rtl/controller` Directory**

Name	Description
`arb_mux.v/vhd`	This is the top-level module of arbitration logic.
`arb_row_col.v/vhd`	This block receives requests to send row and column commands from the bank machines and selects one request, if any, for each state.
`arb_select.v/vhd`	This module selects a row and column command from the request information provided by the bank machines.
`bank_cntrl.v/vhd`	This structural block instantiates the three subblocks that comprise the bank machine.
`bank_common.v/vhd`	This module computes various items that cross all of the bank machines.
`bank_compare.v/vhd`	This module stores the request for a bank machine.
`bank_mach.v/vhd`	This is the top-level bank machine block.
`bank_queue.v/vhd`	This is the bank machine queue controller.
`bank_state.v/vhd`	This is the primary bank state machine.
`col_mach.v/vhd`	This module manages the DQ bus.
`mc.v/vhd`	This is the top-level module of the MC.
`mem_intfc.v/vhd`	This top-level memory interface block instantiates the controller and the PHY.
`rank_cntrl.v/vhd`	This module manages various rank-level timing parameters.

Table 1-8: **Modules in `user_design/rtl/controller` Directory *(Cont'd)***

Name	Description
`rank_common.v/vhd`	This module contains logic common to all rank machines. It contains a clock prescaler and arbiters for refresh and periodic read.
`rank_mach.v/vhd`	This is the top-level rank machine structural block.
`round_robin_arb.v/vhd`	This is a simple round-robin arbiter.

`user_design/rtl/top`

This directory contains the user design (Table 1-9).

Table 1-9: **Modules in `user_design/rtl/top` Directory**

Name	Description
`iodelay_ctrl.v/vhd`	This module instantiates IDELAYCNTRL primitives needed for IODELAY use.
`clk_ibuf.v/vhd`	This module instantiates the input clock buffer.
`mem_intfc.v/vhd`	This is the top-level memory interface block that instantiates the controller and the PHY.
`memc_ui_top.v/vhd`	This is the top-level memory controller module.
`infrastructure.v/vhd`	This module helps in clock generation and distribution, and reset synchronization.
`user_design_top.v/vhd`	This top-level module serves as an example for connecting the user design to the Virtex-6 FPGA memory interface core.

`user_design/rtl/phy`

This directory contains the Virtex-6 FPGA memory interface PHY implementation (Table 1-10).

Table 1-10: **Modules in `user_design/rtl/phy` Directory**

Name	Description
`circ_buffer.v/vhd`	This is the circular buffer for synchronizing signals between clock domains.
`phy_ck_iob.v/vhd`	This module provides clock forwarding to memory and pad loopback into the FPGA.
`phy_clock_io.v/vhd`	This is the top-level module for CK/CK# clock forwarding to memory and feedback into the FPGA.
`phy_control_io.v/vhd`	This module instantiates IOBs for output-only control and address signals to the SDRAM.
`phy_data_io.v/vhd`	This is the top-level module for all data-related (DQ, DQS, DM) IOB logic.
`phy_dly_ctrl.v/vhd`	This module provides centralized control for all IODELAY elements in interface IODELAYs.
`phy_dm_iob.v/vhd`	This module places the data mask signals into the IOBs.

Table 1-10: **Modules in `user_design/rtl/phy` Directory *(Cont'd)***

Name	Description
`phy_dq_iob.v/vhd`	This module instantiates I/O-related logic for DQ.
`phy_dqs_iob.v/vhd`	This module instantiates I/O-related logic for DQS.
`phy_init.v/vhd`	This module provides memory initialization and overall master state control during initialization and calibration.
`phy_pd.v/vhd`	This module provides phase detector calibration.
`phy_pd_top.v/vhd`	This is the top-level module of the phase detector.
`phy_rdclk_gen.v/vhd`	This module generates and distributes the capture clock.
`phy_rdctrl_sync.v/vhd`	This module synchronizes the read control signal from MC/PHY rdlvl logic to read capture logic.
`phy_rddata_sync.v/vhd`	This module synchronizes captured read data to the core clock domain.
`phy_rdlvl.v/vhd`	This module provides read-leveling calibration logic.
`phy_read.v/vhd`	This is the top-level module for the PHY read logic.
`phy_top.v/vhd`	This is the top-level module for the memory PHY interface.
`phy_write.v/vhd`	This module delays various write control signals based on user-specific timing parameters (for example, CAS write latency).
`phy_wrlvl.v/vhd`	This module provides calibration for write leveling.
`rd_bitslip.v/vhd`	This module shifts data captured by the ISERDES in bit time increments to provide aligned data across all DQS groups.

`user_design/rtl/ui`

This directory contains the user interface code that mediates between the native interface of the memory controller and user applications (Table 1-11).

Table 1-11: **Modules In `user_design/rtl/ui` Directory**

Name	Description
`ui_cmd.v/vhd`	This is the user interface command port.
`ui_rd_data.v/vhd`	This is the user interface read buffer. It reorders read data returned from the memory controller back to the request order.
`ui_wr_data.v/vhd`	This is the user interface write buffer. It reorders write data returned from the memory controller back to the request order.
`ui_top.v/vhd`	This is the top level of the memory controller user interface.

`user_design/rtl/ecc`

This directory contains the optional error correcting code that implements single bit error correction and two bit error detection (Table 1-12). All modules are instantiated in the `mc.v/vhd` RTL.

Table 1-12: **Modules in `user_design/rtl/ecc`**

Name	Description
`ecc_buf.v/vhd`	This data buffer block temporarily holds the data for the read-modify-write cycles.
`ecc_gen.v/vhd`	This module generates the ECC H matrix.
`ecc_merge_enc.vhd`	This module computes the ECC bits and appends them to data.
`ecc_dec_fix.v/vhd`	This module decodes and fixes the read data.

`<component name>/user_design/par`

Table 1-13 lists the modules in the `user_design/par` directory.

Table 1-13: **Modules in `user_design/par` Directory**

Name	Description
`<component_name>_example_design.ucf`	This is the UCF for the core and the example design.
`ise_flow.bat`	This script file runs the design through synthesis, build, map, and par, and sets all the required options. Refer to this file for the recommended build options for the design. Double-clicking the `create_ise.bat` file creates an ISE tools project that contains the recommended build options for the design. Double-clicking the ISE tools project file opens up the ISE software in GUI mode with all the project settings.

Caution! The `ise_flow.bat` file in the `par` folder of the `<component name>` directory contains the recommended build options for the design. Failure to follow the recommended build options can produce unexpected results.

<component name>/user_design/sim

Table 1-14 lists the modules in the user_design/sim directory.

Table 1-14: **Modules in user_design/sim Directory**

Name	Description
sim.do	This SDC file has design constraints for the Synplify Pro synthesis tool.
sim.exe	Double-clicking this file causes the design to be automatically simulated using the ModelSim simulator.
ddr2_model.v ddr3_model.v	These are the DDR2 and DDR3 SDRAM memory models.
ddr2_model_parameters.vh ddr3_model_parameters.vh	These files contain the DDR2 and DDR3 SDRAM memory model parameter settings.
sim_tb_top.v/vhd	This is the simulation top file.
glbl.v	This file is used for initializing the simulation environment.
mcb_traffic_gen.v/vhd	This is the top level of the traffic generator.
cmd_gen.v/vhd	This is the command generator. This module provides independent control of generating types of commands, addresses, and burst lengths.
cmd_prbs_gen.v/vhd	This is a PRBS generator for generating PRBS commands, addresses, and burst lengths.
mcb_flow_control.v/vhd	This module generates flow control logic between the memory controller core and the cmd_gen, read_data_path, and write_data_path modules.
read_data_path.v/vhd	This is the top level for the read datapath.
read_posted_fifo.v/vhd	This module stores the read command that is sent to the memory controller, and its FIFO output is used to generate expect data for the read data comparison.
rd_data_gen.v/vhd	This module generates timing control for read and ready signals to mcb_flow_control.v/vhd.
write_data_path.v/vhd	This is the top level for the write datapath.
wr_data_gen.v/vhd	This module generates timing control for write and ready signals to mcb_flow_control.v/vhd.
v6_data_gen.v/vhd	This module generates different data patterns.
a_fifo.v/vhd	This is a synchronous FIFO using LUTRAMs.
data_prbs_gen.v/vhd	This is a 32-bit LFSR for generating PRBS data patterns.
init_mem_pattern_ctr.v/vhd	This module generates flow control logic for the traffic generator.

Verify UCF and Update Design and UCF Rules

Verify UCF and Update Design and UCF verifies the input UCF for bank selection, pin allocation, and constraint allocation rules and generates warnings or error reports for any issues. It does not verify the input `.prj` file. This feature is useful to verify any UCF pinout changes after the design is generated from the MIG tool. The user must load the MIG generated `.prj` file (the original `.prj` file) without any modifications. The verification report is not correct if any of the parameters in the original `.prj` file are altered. In the CORE Generator tool, the recustomization option should be selected to reload the project. The design can be generated only when Verify UCF does not report an error in the verification report. Warnings can be ignored while generating a design.

These rules are verified from the input UCF:

- If a pin is allocated to more than one signal, the tool reports an error.
  - Further verification does not occur if the UCF does not adhere to the uniqueness property.
- The pins related to one DQS set should be allocated in the same bank.
- When the frequency of the configuration is more than 400 MHz, the IOB distance calculation is verified according to Data/Strobe/Mask Span Allocation Rules, page 137.
- Banks should be allocated for the address and data within the vicinity arena.
  - An error occurs if a bank is allocated outside the vicinity arena.
- The system clock bank can be selected adjacent to the GC bank (24, 25, 34, and 35) or to the bank adjacent to the capture clock bank.
  - The system clock bank can be selected adjacent to the capture clock bank only when the frequency of this controller is not repeated in any of the other controllers. If the frequency of this controller is repeated in any of the other controllers, the system clock group must be allocated to any of the GC banks (24, 25, 34, and 35) but not to the bank where only CC pins are available (this occurs when a bank adjacent to the capture clock bank is used).
  - The signal pairs sys_clk and clk_ref are allocated to the CC pair or GC pair pins (for a bank adjacent to the capture clock bank) or to the GC pair pins (for GC banks).
- The memory clock signals (CK and CK#) should be allocated to the differential pair pins (P-N pair).
- The DQS pair should be allocated to the differential pair pins.
- The capture clock (BUFIO) and resynchronization clock (BUFR) constraints are verified as follows:
  - The capture clock LOC constraint should be associated with its corresponding strobe set. Otherwise, the tool reports an error and provides the valid pins to correct the constraints and rerun the verification.
  - The resynchronization clock LOC constraint should be associated with the corresponding column where its related strobes are allocated. If the resynchronization clock is not associated with its corresponding strobe pins, the tool reports an error and provides the valid pins to correct the constraints and rerun the verification.
- In the DCI CASCADE syntax, the selected configuration should require the master bank.
  - The slave banks provided should be valid.

- A valid mixed-mode clock manager (MMCM) constraint value should be provided, otherwise a warning is generated.

If the UCF satisfies the above rules, the updated design is generated. The design:

- Provides the latest HDL.
- Updates the UCF with the latest clock constraints or any timing ignores (TIGs) provided by keeping the same pinout.
- Generates even the compatible UCFs if the project loaded contains the compatible FPGA selection.

Error Messages

This section describes the error messages that are generated when verifying the UCF. The reference UCF must follow the MIG naming conventions (refer to the UCF generated by the MIG tool or names used for the ML605 board).

- **Uniqueness**: If two or more signals are allocated to the same pins in the reference UCF, an error message is listed in the directed file with a user-assigned name.

  The error message format is "<*signalname1*> and <*signalname2*> are allocated to the same pin." For example, if ddr3_dq[0] and ddr3_dqs[0] are allocated to the same pin, such as:

  ```
  NET "ddr3_dq[0]" LOC = "D12";
  NET "ddr3_dqs[0]" LOC = "D12";
  ```

  Then this error message is displayed:

  ```
  ERROR: ddr3_dq[0] and ddr3_dqs[0] are allocated to the same pin.
  Pins are not unique.
  ```

- **Association**: Signals related to the same DQS set should be allocated in the same bank, otherwise the MIG tool reports an error message.

  The error message format is "<*signalname1*> and <*signalname2*> are not allocated in the same bank." For example:

  ```
  ERROR: ddr3_dq[16](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[17](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[18](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[19](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[20](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[21](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[22](26) and ddr3_dq[23](25) are not allocated in the
  same bank
  ERROR: ddr3_dq[23](25) and ddr3_dm[2](26) are not allocated in the
  same bank
  ```

- **IOB Distance Verification**: This property is verified when the frequency selected for the configuration is greater than 400 MHz for SDRAM designs.

- A warning message is displayed when the distance between the DQS and its corresponding pins is more than eight IOB pads:

```
WARNING: The IOB distance between the strobe
pins'ddr3_dqs_p[0]','ddr3_dqs_n[0]' and 'ddr3_dq[3]' is greater
than 8 which may not allow the read capture circuit to function
properly above 400 MHz (800 Mbps).

WARNING: The IOB distance between the strobe
pins'ddr3_dqs_p[0]','ddr3_dqs_n[0]' and 'ddr3_dq[2]' is greater
than 8 which may not allow the read capture circuit to function
properly above 400 MHz (800 Mbps).

WARNING: The IOB distance between the strobe
pins'ddr3_dqs_p[0]','ddr3_dqs_n[0]' and 'ddr3_dq[1]' is greater
than 8 which may not allow the read capture circuit to function
properly above 400 MHz (800 Mbps).

WARNING: The IOB distance between the strobe
pins'ddr3_dqs_p[0]','ddr3_dqs_n[0]' and 'ddr3_dq[0]' is greater
than 8 which may not allow the read capture circuit to function
properly above 400 MHz (800 Mbps).
```

- A warning message is displayed when the DQS set is allocated across the clock tree:

```
WARNING: The DQS set "ddr3_dqs_p[1]" is allocated across the clock
tree of the bank "26". Then the pins should be no more distant than
13 IOBs from the clock tree. But the pin "ddr3_dqs_n[1]" is more than
13 IOBs distant from the clock tree which may not allow the read
capture circuit to function properly above 400 MHz (800 Mbps).

WARNING: The DQS set "ddr3_dqs_p[1]" is allocated across the clock
tree of the bank "26". Then the pins should be no more distant than
13 IOBs from the clock tree. But the pin "ddr3_dq[0]" is more than
13 IOBs distant from the clock tree which may not allow the read
capture circuit to function properly above 400 MHz (800 Mbps).
```

- **Vicinity Verification**: Error messages are displayed when the pins are allocated out of the vicinity arena.
  - If the data bank selected is out of the vicinity arena, these error messages are displayed:

```
ERROR: c0_ddr3_dq[0](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dq[1](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dq[2](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dq[3](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dq[4](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dq[5](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.
```

```
ERROR: c0_ddr3_dq[6](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dq[7](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_ddr3_dm[0](Mask) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.
```

- **Differential Pair Verification**: If the system clock pins are not allocated to the differential pairs, these error messages are displayed:

```
ERROR: "sys_clk_p" Should be allocated to either CC P pin or GC P
pin.

ERROR: "sys_clk_n" Should be allocated to either CC N pin or GC N
pin.

ERROR: "sys_clk_p" and "sys_clk_n" Should be allocated to either CC
or GC P/N pair.

ERROR: "clk_ref_p" Should be allocated to either CC P pin or GC P
pin.

ERROR: "clk_ref_n" Should be allocated to either CC N pin or GC N
pin.

ERROR: "clk_ref_p" and "clk_ref_n" Should be allocated to either CC
or GC P/N pair.
```

- **Absence of Signals**: If one or more signal pin pairs is missing and/or commented in the given UCF against the selected inputs, the verification result indicates the absence of these signal pin pairs as a warning.

  The warning message format is "Signal <*signal_name*> is expected, but not present in the UCF." For example:

```
WARNING: Signal "ddr2_dq[15]" expected, but not present in the UCF.

WARNING: Signal "ddr2_dq[16]" expected, but not present in the UCF.

WARNING: Signal "ddr2_dq[17]" expected, but not present in the UCF.

WARNING: Signal "ddr2_dq[18]" expected, but not present in the UCF.
```

- **Capture Clock/Resynchronization Clock Verification**: These constraints are provided only for SDRAM designs.
  - This is the message format for capture clock (BUFIO) constraints:

```
ERROR: <Constraint name> constraint for the Strobe -
<gen_ck_cpt[<vector>]> is not provided or provided constraint may be
invalid. Following is (are) the valid <Constraint> Constraints for
this Capture Clock. But verify whether any of these IOB sites are
utilized by any other constraints or Pin LOC's (Valid constraints
for the strobe).

ERROR: BUFIO Constraint for the Capture Clock - "gen_ck_cpt[0]" is
not provided or provided BUFIO constraint is invalid. Following is
(are) the valid BUFIO Constraints for this Capture Clock. But verify
whether any of these IOB sites are utilized by any other constraints
or Pin LOC's.

AR35 - X1Y183.

AT37 - X1Y181.

AP37 - X1Y179.
```

```
AJ31 - X1Y177.
ERROR: BUFIO Constraint for the Capture Clock - "gen_ck_cpt[1]" is
not provided or provided BUFIO constraint is invalid. Following
is(are) the valid BUFIO Constraints for this Capture Clock. But
verify whether any of these IOB sites are utilized by any other
constraints or Pin LOC's.
AR35 - X1Y183.
AT37 - X1Y181.
AP37 - X1Y179.
AJ31 - X1Y177.
ERROR: BUFIO Constraint for the Capture Clock - "gen_ck_cpt[2]" is
not provided or provided BUFIO constraint is invalid. Following
is(are) the valid BUFIO Constraints for this Capture Clock. But
verify whether any of these IOB sites are utilized by any other
constraints or Pin LOC's.
AR35 - X1Y183.
AT37 - X1Y181.
AP37 - X1Y179.
AJ31 - X1Y177.
ERROR: BUFIO Constraint for the Capture Clock - "gen_ck_cpt[4]" is
not provided or provided BUFIO constraint is invalid. Following
is(are) the valid BUFIO Constraints for this Capture Clock. But
verify whether any of these IOB sites are utilized by any other
constraints or Pin LOC's
AY33 - X1Y103.
AR30 - X1Y101.
AW32 - X1Y99.
AT30 - X1Y97.
```

- This is the message format for resynchronization clock (BUFR) constraints:

```
ERROR: The BUFR Constraint for the FPGA column '1' is not provided
in the UCF or Constraint provided for this column is not valid.
Following is(are) valid BUFR constraints.
AJ41 - X0Y143.
AD36 - X0Y141.
AF36 - X0Y139.
AD37 - X0Y137.
ERROR: The BUFR Constraint for the FPGA column '1' is not provided
in the UCF or Constraint provided for this column is not valid.
Following is (are) valid BUFR constraints.
AT6 - X2Y143.
AT9 - X2Y141.
AV6 - X2Y139.
AW7 - X2Y137.
```

- **Master Bank Verification**: This verifies whether the provided master bank is valid for the selected DCI banks in the column. This error message is displayed when the valid master bank is not provided for the column:

```
ERROR: the master bank "23" provided is not valid master bank.
Following are the valid master bank "24, 25" for the column "1".
```

Quick Start Example Design

Overview

After the core is successfully generated, the example design HDL can be processed through the Xilinx implementation toolset.

Implementing the Example Design

The `ise_flow.bat` script file runs the design through synthesis, translate, map, and par, and sets all the required options. Refer to this file for the recommended build options for the design.

Simulating the Example Design (for Designs with a Standard User Interface)

The MIG tool provides a synthesizable test bench to generate various traffic data patterns to the memory controller. This test bench consists of a `memc_ui_top` wrapper, a `traffic_generator` that generates traffic patterns through the user interface to a `ui_top` core, and an infrastructure core that provides clock resources to the `memc_ui_top` core. A block diagram of the example design test bench is shown in Figure 1-40.

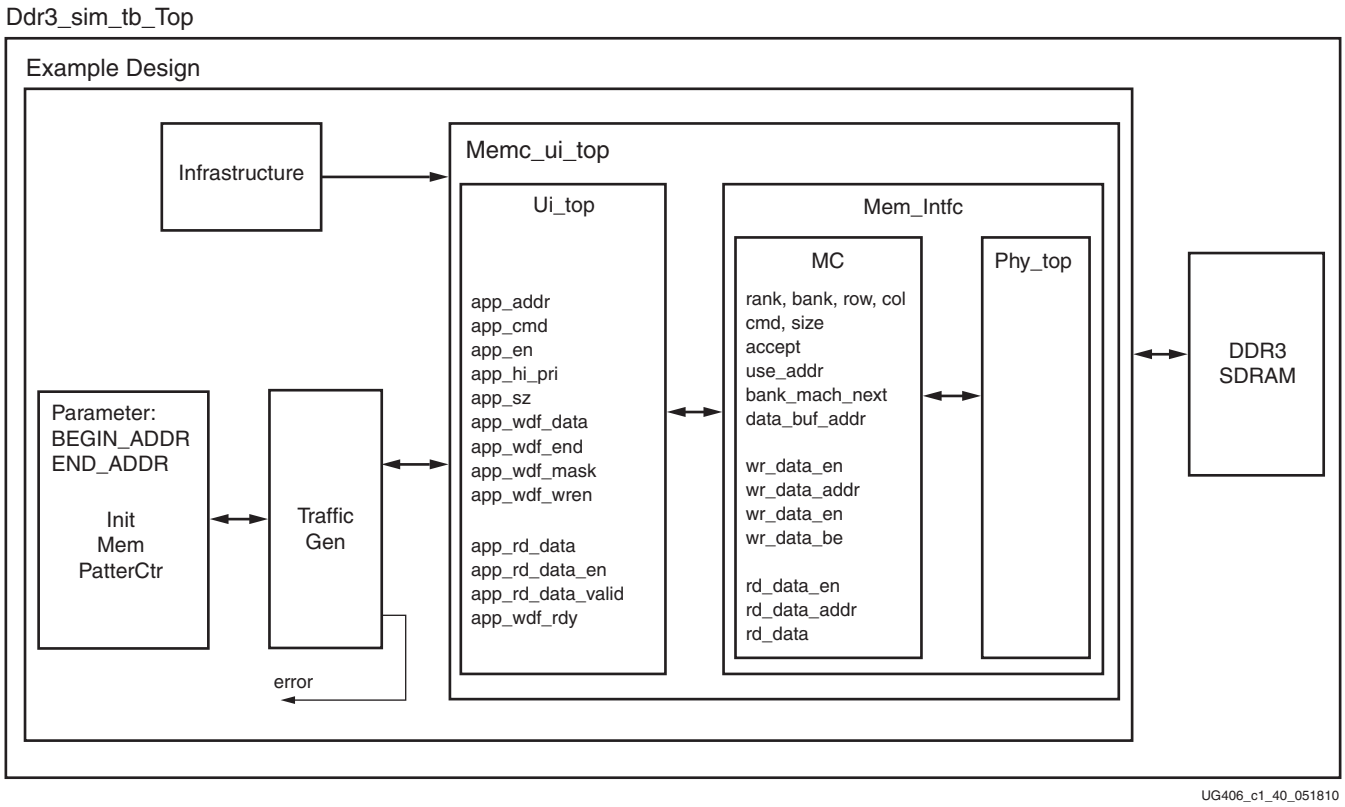
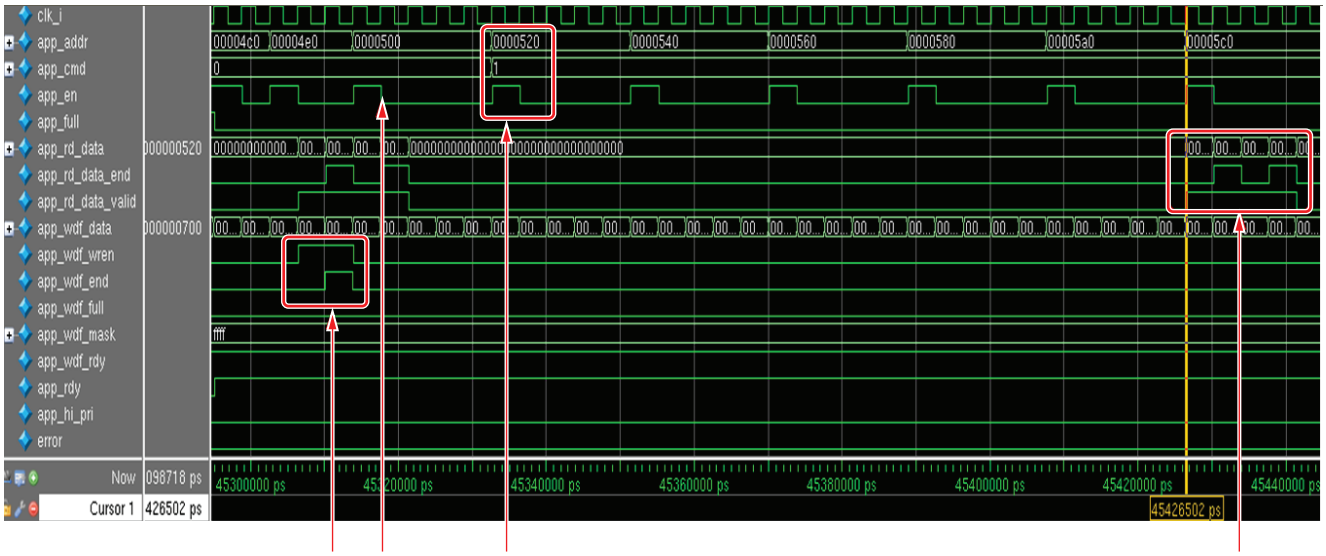


Figure 1-40: **Synthesizable Example Design Block Diagram**

Figure 1-41 shows the simulation result of a simple read and write transaction between the `tb_top` and `memc_intfc` modules.



Address as data (0x...0500) values are written to the data FIFO when app_wdf_wren is asserted and app_wdf_end indicates the last cycle of write input data. A write command is issued after the last write data cycle.

Read command to 0x520 is accepted here.

Memory user interface asserts app_rd_data_valid signal to indicate valid data (0x...0520) returning from memory.

UG406_c1_41_050709

Figure 1-41: **User Interface Read and Write Cycle**

Simulating the Example Design (for Designs with an AXI4 Interface)

The MIG tool provides synthesizable AXI4 test bench to generate various traffic patterns to the memory controller. This test bench consists of a memc_ui_top wrapper, a traffic generator that generates traffic patterns through the user interface to a ui_top core, and an infrastructure core that provides clock resources to the memc_ui_top core. Figure 1-42 shows a block diagram of the example design test bench.

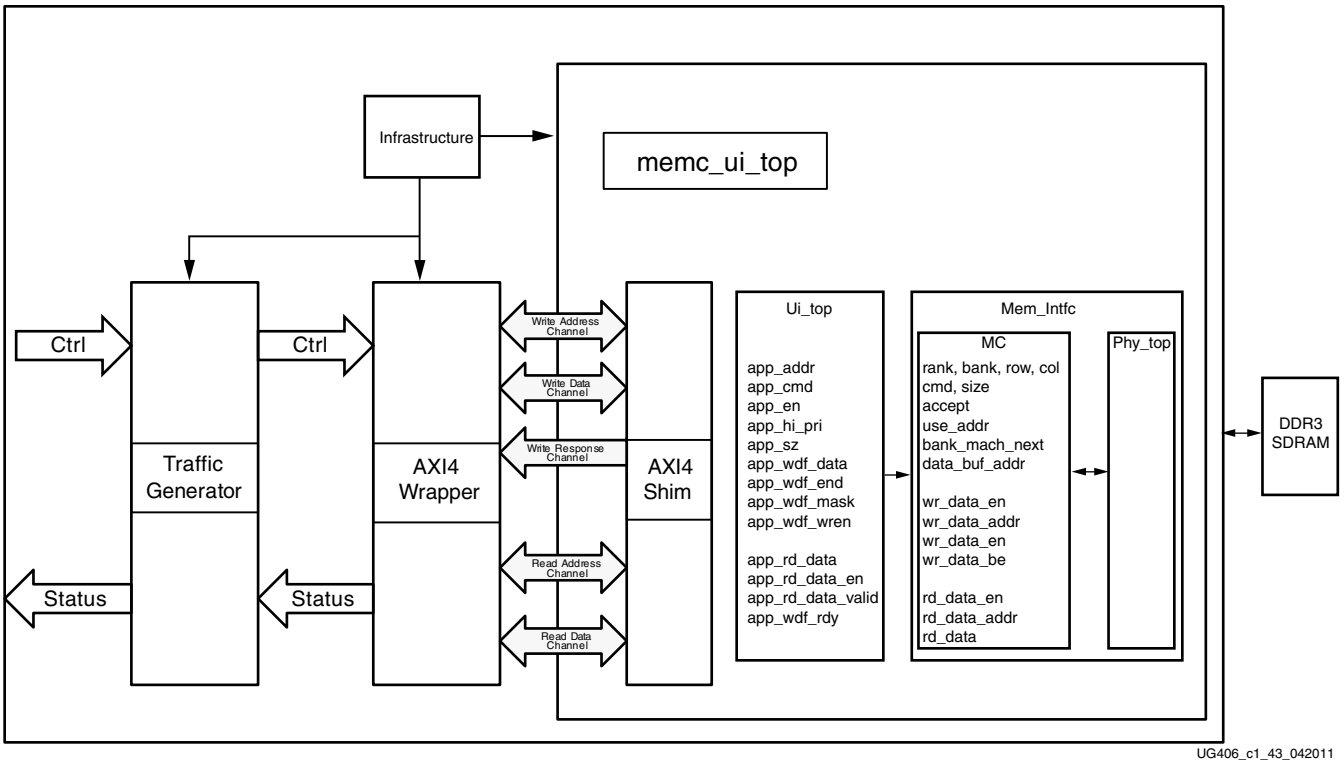


Figure 1-42: **Synthesizable Example Design Block for AXI4 Interface**

Figure 1-43 shows the simple write transaction being performed on the AXI4 interface. This consists of a command phase, data phase and the response phase as shown in Figure 1-43. This follows the standard AXI4 protocol.

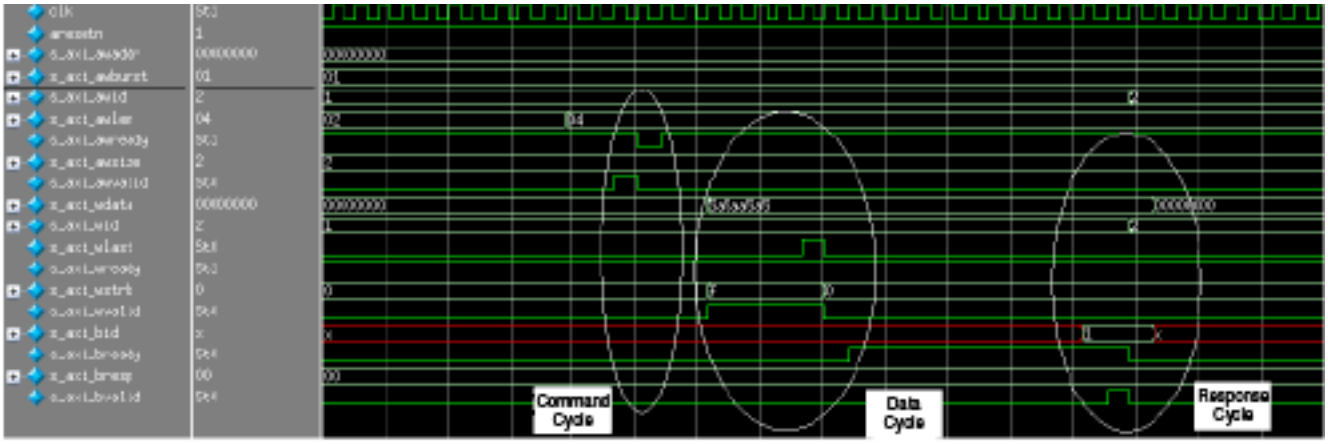


Figure 1-43: **AXI4 Interface Write Cycle**

Figure 1-44 shows the simple read transaction being performed on the AXI4 interface. This consists of a command phase and data phase shown in Figure 1-43. This transaction follows the standard AXI4 protocol.

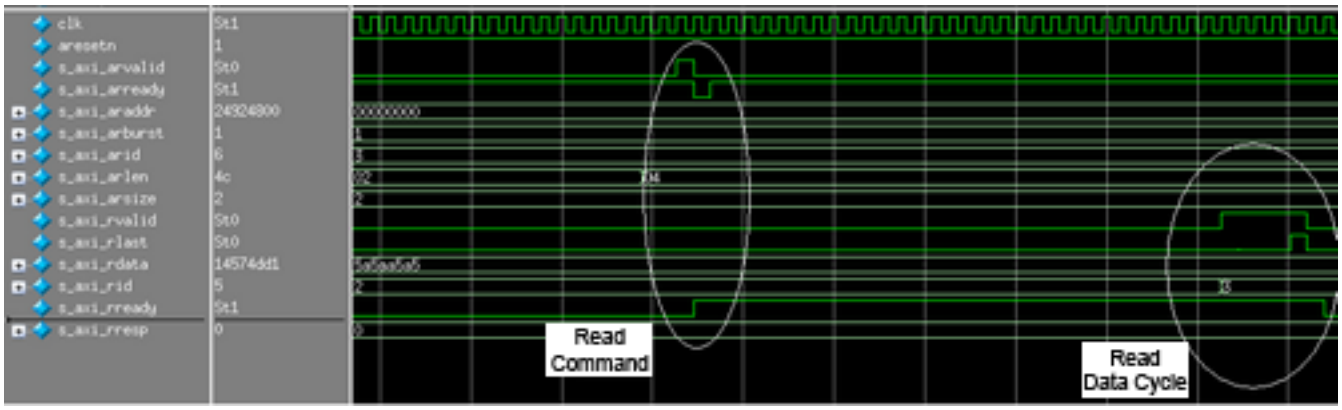


Figure 1-44: **AXI4 Interface Read Cycle**

Traffic Generator Operation

The traffic generator module contained within the synthesizable test bench can be parameterized to create various stimulus patterns for the memory design. It can produce repetitive test patterns for verifying design integrity as well as pseudo-random data streams that model real-world traffic.

The user can define the address range through the BEGIN_ADDRESS and END_ADDRESS parameters. The Init Memory Pattern Control block directs the traffic generator to step sequentially through all the addresses in the address space, writing the appropriate data value to each location in the memory device as determined by the selected data pattern. By default, the test bench uses the address as the data pattern, but the data pattern in this example design can be modified using vio_data_mode signals that can be modified within the ChipScope analyzer.

When the memory has been initialized, the traffic generator begins stimulating the user interface port to create traffic to and from the memory device. By default, the traffic generator sends pseudo-random commands to the port, meaning that the instruction sequences (R/W, R, W, etc.) and addresses are determined by PRBS generator logic in the traffic generator module.

The read data returning from the memory device is accessed by the traffic generator through the user interface read data port and compared against internally generated "expect" data. If an error is detected (that is, there is a mismatch between the read data and expected data), an error signal is asserted and the readback address, readback data, and expect data are latched into the error_status outputs.

The AXI4 traffic generator supports these features:

- Data widths of 32, 64, 128, 256, and 512 bits
- INCR/WRAP only. Does not support FIXED mode. A parameter enables the support for wrapping bursts
- Little-endian mode operation
- Data pattern generation parameterized (changes during compile time)
- Parameter-controlled ID generation. By default, all transactions have IDs that increment with every transaction. The IDs count up to 16 for ID widths greater than 4 and count up to the maximum ID value for ID widths less than 4. For example, if the ID width is 3, it counts up to 8.
- Command interface to control the data flow

- The wrapper and the traffic generator are modular and designed to be used in standalone mode
- Pseudo-random burst lengths and read/write generation
- Watchdog timer that times out if no response is received from the AXI4 slave after a fixed number of clock cycles

Modifying the Example Design

The provided example_top design comprises traffic generator modules and can be modified to tailor different command and data patterns. A few high-level parameters can be modified in the `example_top.v/vhd` module. Table 1-15 describes these parameters.

Table 1-15: **Traffic Generator Parameters Set in the example_top Module**

Parameter	Parameter Description	Parameter Value
FAMILY	Indicates the family type.	The value of this parameter is "VIRTEX6".
PORT_MODE	Sets the port mode.	Valid setting for this parameter is: BI_MODE: Generate a WRITE data pattern and monitor the READ data for comparison.
BEGIN_ADDRESS	Sets the memory start address boundary.	This parameter defines the start boundary for the port address space. The least-significant bits [3:0] of this value are ignored.
END_ADDRESS	Sets the memory end address boundary.	This parameter defines the end boundary for the port address space. The least-significant bits [3:0] of this value are ignored.
PRBS_SADDR_MASK_POS	Sets the 32-bit OR MASK position.	This parameter is used with the PRBS address generator to shift random addresses up into the port address space. The BEGIN_ADDRESS value is ORed with the PRBS address for bit positions that have a "1" in this mask.
PRBS_EADDR_MASK_POS	Sets the 32-bit AND MASK position.	This parameter is used with the PRBS address generator to shift random addresses down into the port address space. The END_ADDRESS value is ANDed with the PRBS address for bit positions that have a "1" in this mask.

Table 1-15: **Traffic Generator Parameters Set in the example_top Module *(Cont'd)***

Parameter	Parameter Description	Parameter Value
CMD_PATTERN	This parameter sets the command pattern circuits to be generated. For a larger device, the CMD_PATTERN can be set to "CGEN_ALL". This parameter enables all supported command pattern circuits to be generated. However, it is sometimes necessary to limit a specific command pattern because of limited resources in a smaller device.	Valid settings for this signal are: • CGEN_FIXED: The address, burst length, and instruction are taken directly from the fixed_addr_i, fixed_bl_i, and fixed_instr_i inputs. • CGEN_SEQUENTIAL: The address is incremented sequentially, and the increment is determined by the data port size. • CGEN_PRBS: A 32-stage linear feedback shift register (LFSR) generates pseudo-random addresses, burst lengths, and instruction sequences. The seed can be set from the 32-bit cmd_seed input. • CGEN_ALL (default): This option turns on all of the above options and allows addr_mode_i, instr_mode_i, and bl_mode_i to select the type of generation during run time.
DATA_PATTERN	This parameter sets the data pattern circuits to be generated. For larger devices, the DATA_PATTERN can be set to "DGEN_ALL". This parameter enables all supported data pattern circuits to be generated. However it is sometimes necessary to limit a specific command pattern because of limited resources in a smaller device.	Valid settings for this parameter are: • ADDR (default): The address is used as a data pattern. • HAMMER: All `1`s are on the DQ pins during the rising edge of DQS, and all `0`s are on the DQ pins during the falling edge of DQS. • WALKING1: Walking `1`s are on the DQ pins and the starting position of `1` depends on the address value. • WALKING0: Walking `0`s are on the DQ pins and the starting position of `0` depends on the address value. • NEIGHBOR: The Hammer pattern is on all DQ pins except one. The address determines the exception pin location. • PRBS: A 32-stage LFSR generates random data and is seeded by the starting address. • DGEN_ALL: This option turns on all the above options and allows data_mode_i to select the type of data pattern generation during run time.
ENFORCE_RD_WR	Enforce read/write sequence from the test bench	This parameter allows the user to control the read and write sequence from the test bench.
ENFORCE_RD_WR_CMD	8-bit value that controls the read and write sequence	This 8-bit parameter controls the read and write sequence. A one indicates a write operation and a zero indicates a read operation. For example, `8'h11` indicates a write followed by three reads, and this operation is repeated twice.

Table 1-15: **Traffic Generator Parameters Set in the example_top Module *(Cont'd)***

Parameter	Parameter Description	Parameter Value
ENFORCE_RD_WR_PATTERN	3-bit parameter that selects the pattern used for the write operation	The parameter selects the pattern for memory writes. • `3'b000` - 'h5A5A_A5A5 pattern • `3'b001` - PRBS pattern • `3'b010` - Walking zeros • `3'b011` - Walking ones • `3'b100` - All ones • `3'b101` - All zeros
C_EN_WRAP_TRANS	Enable code for wrap transactions	This parameter enables code to support the wrap transactions.
C_AXI_NBURST_TEST	Enable code for narrow bursts	This parameter enables code that generates narrow burst transactions on the AXI4 interface.

The command patterns instr_mode_i, addr_mode_i, bl_mode_i, and data_mode_i of the traffic_gen module can each be set independently. The provided init_mem_pattern_ctr module has interface signals that allow the user to modify the command pattern in real time using the ChipScope analyzer virtual I/O (VIO).

This is the varying command pattern:

1. Set vio_modify_enable to `1`.
2. Set vio_addr_mode_value to:

   1: Fixed_address.

   2: PRBS address.

   3: Sequential address.
3. Set vio_bl_mode_value to:

   1: Fixed bl.

   2: PRBS bl. If bl_mode value is set to 2, the addr_mode value is forced to 2 to generate the PRBS address.
4. Set vio_data_mode_value to:

   0: Reserved.

   1: FIXED data mode. Data comes from the fixed_data_i input bus.

   2: DGEN_ADDR (default). The address is used as the data pattern.

   3: DGEN_HAMMER. All `1`s are on the DQ pins during the rising edge of DQS, and all `0`s are on the DQ pins during the falling edge of DQS.

   4: DGEN_NEIGHBOR. All `1`s are on the DQ pins during the rising edge of DQS except one pin. The address determines the exception pin location.

   5: DGEN_WALKING1. Walking `1`s are on the DQ pins. The starting position of `1` depends on the address value.

   6: DGEN_WALKING0. Walking `0`s are on the DQ pins. The starting position of `0` depends on the address value.

   7: DGEN_PRBS. A 32-stage LFSR generates random data and is seeded by the starting address.

Modifying Port Address Space

The address space for a port can be easily modified by changing the BEGIN_ADDRESS and END_ADDRESS parameters found in the top-level test bench file. These two values must be set to align to the port data width. The two additional parameters, PRBS_SADDR_MASK_POS and PRBS_EADDR_MASK_POS, are used in the default PRBS address mode to ensure that out-of-range addresses are not sent to the port. PRBS_SADDR_MASK_POS creates an OR mask that shifts PRBS-generated addresses with values below BEGIN_ADDRESS up into the valid address space of the port. PRBS_SADDR_MASK_POS should be set to a 32-bit value equal to the BEGIN_ADDRESS parameter. PRBS_EADDR_MASK_POS creates an AND mask that shifts PRBS-generated addresses with values above END_ADDRESS down into the valid address space of the port. PRBS_EADDR_MASK_POS should be set to a 32-bit value, where all bits above the most-significant address bit of END_ADDRESS are set to 1 and all remaining bits are set to 0. Table 1-16 shows some examples of setting the two mask parameters.

Table 1-16: **Example Settings for Address Space and PRBS Masks**

SADDR	EADDR	PRBS_SADDR_MASK_POS	PRBS_EADDR_MASK_POS
0x1000	0xFFFF	0x00001000	0xFFFF0000
0x2000	0xFFFF	0x00002000	0xFFFF0000
0x3000	0xFFFF	0x00003000	0xFFFF0000
0x4000	0xFFFF	0x00004000	0xFFFF0000
0x5000	0xFFFF	0x00005000	0xFFFF0000
0x2000	0x1FFF	0x00002000	0xFFFFE000
0x2000	0x2FFF	0x00002000	0xFFFFD000
0x2000	0x3FFF	0x00002000	0xFFFFC000
0x2000	0x4FFF	0x00002000	0xFFFF8000
0x2000	0x5FFF	0x00002000	0xFFFF8000
0x2000	0x6FFF	0x00002000	0xFFFF8000
0x2000	0x7FFF	0x00002000	0xFFFF8000
0x2000	0x8FFF	0x00002000	0xFFFF0000
0x2000	0x9FFF	0x00002000	0xFFFF0000
0x2000	0xAFFF	0x00002000	0xFFFF0000
0x2000	0xBFFF	0x00002000	0xFFFF0000
0x2000	0xCFFF	0x00002000	0xFFFF0000
0x2000	0xDFFF	0x00002000	0xFFFF0000
0x2000	0xEFFF	0x00002000	0xFFFF0000
0x2000	0xFFFF	0x00002000	0xFFFF0000

Traffic Generator Signal Description

Traffic generator signals are described in Table 1-17.

Table 1-17: **Traffic Generator Signal Descriptions**

Signal Name	Direction	Description
addr_mode_i[1:0]	Input	Valid settings for this signal are: 00: Block RAM address mode. The address comes from the bram_cmd_i input bus. 01: FIXED address mode. The address comes from the fixed_addr_i input bus. 10: PRBS address mode (Default). The address is generated from the internal 32-bit LFSR circuit. The seed can be changed through the cmd_seed input bus. 11: SEQUENTIAL address mode. The address is generated from the internal address counter. The increment is determined by the User Interface port width.
bl_mode_i[1:0]	Input	Valid settings for this signal are: 00: Block RAM burst mode. The burst length comes from the bram_cmd_i input bus. 01: FIXED burst mode. The burst length comes from the fixed_instr_i input bus. 10: PRBS burst mode (Default). The burst length is generated from the internal 16-bit LFSR circuit. The seed can only be changed through the parameter section.
bram_cmd_i[38:0]	Input	This bus contains the block RAM interface ports: {BL, INSTR, ADDRESS}.
bram_rdy_o	Output	This block RAM interface output indicates when the traffic generator is ready to accept input from bram_cmd_i bus.
bram_valid_i	Input	For the block RAM interface, the bram_cmd_i bus is accepted when both bram_valid_i and bram_rdy_o are asserted.
clk_i	Input	This signal is the clock input.
cmd_seed_i[31:0]	Input	This bus is the seed for the command PRBS generator.
counts_rst	Input	When counts_rst is asserted, wr_data_counts and rd_data_counts are reset to zero.

Table 1-17: **Traffic Generator Signal Descriptions *(Cont'd)***

Signal Name	Direction	Description
data_mode_i[3:0]	Input	Valid settings for this signal are: 0000: Reserved. 0001: FIXED data mode. Data comes from the fixed_data_i input bus. 0010: DGEN_ADDR (Default). The address is used as the data pattern. 0011: DGEN_HAMMER. All 1s are on the DQ pins during the rising edge of DQS, and all 0s are on the DQ pins during the falling edge of DQS. This option is only valid if parameter DATA_PATTERN = "DGEN_HAMMER" or "DGEN_ALL". 0100: DGEN_NEIGHBOR. All 1s are on the DQ pins during the rising edge of DQS except one pin. The address determines the exception pin location. This option is only valid if parameter DATA_PATTERN = "DGEN_ADDR" or "DGEN_ALL". 0101: DGEN_WALKING1. Walking 1s are on the DQ pins. The starting position of 1 depends on the address value. This option is only valid if parameter DATA_PATTERN = "DGEN_WALKING" or "DGEN_ALL". 0110: DGEN_WALKING0. Walking 0s are on the DQ pins. The starting position of 0 depends on the address value. This option is only valid if parameter DATA_PATTERN = "DGEN_WALKING0" or "DGEN_ALL". 0111: DGEN_PRBS. A 32-stage LFSR generates random data and is seeded by the starting address. This option is only valid if parameter DATA_PATTERN = "DGEN_PRBS" or "DGEN_ALL".
data_seed_i[31:0]	Input	This bus is the seed for the data PRBS generator.
end_addr_i[31:0]	Input	This bus defines the end-address boundary for the port address space. The least-significant bits [3:0] are ignored.
error	Output	This signal is asserted when the readback data is not equal to the expected value.
error_status[n:0]	Output	This signal latches these values when the error signal is asserted: [31:0]: Read start address [37:32]: Read burst length [39:38]: Reserved [40]: mcb_cmd_full [41]: mcb_wr_full [42]: mcb_rd_empty [64 + (DWIDTH - 1):64]: expected_cmp_data [64 + (2*DWIDTH - 1):64 + DWIDTH]: read_data
fixed_addr_i[31:0]	Input	This 32-bit input is the fixed address input bus.
fixed_bl_i[5:0]	Input	This 6-bit input is the fixed burst length input bus.
fixed_data_i[31:0]	Input	This 32-bit input is the fixed data input bus.
fixed_instr_i[2:0]	Input	This 3-bit input is the fixed instruction input bus.

Table 1-17: **Traffic Generator Signal Descriptions *(Cont'd)***

Signal Name	Direction	Description
instr_mode_i[3:0]	Input	Valid settings for this signal are: 0000: Block RAM instruction mode. The instruction comes from the bram_cmd_i input bus. 0001: FIXED instruction mode. The instruction comes from the fixed_instr_i input bus. 0010: W/R instruction mode (Default). This mode generates pseudo-random WRITE and READ instruction sequences. 0011: WP/RP instruction mode. This mode generates pseudo-random WRITE precharge and READ precharge instruction sequences. 0100: W/WP/R/RP. This mode generates pseudo-random WRITE, WRITE precharge, READ, and READ precharge instruction sequences. 0101: W/WP/R/RP/RF. This mode generates pseudo-random WRITE, WRITE precharge, READ, READ precharge, and REFRESH instruction sequences.
mcb_cmd_addr_o[29:0]	Output	Memory controller block MIG DDR2/DDR3 standard user command port interface.
mcb_cmd_bl_o[5:0]	Output	MIG DDR2/DDR3 standard user command port interface.
mcb_cmd_en_o	Output	MIG DDR2/DDR3 standard user command port interface.
mcb_cmd_full_i	Input	MIG DDR2/DDR3 standard user command port interface.
mcb_cmd_instr_[2:0]	Output	MIG DDR2/DDR3 standard user command port interface.
mcb_rd_data_i[DWIDTH-1:0]	Input	MIG DDR2/DDR3 standard user data port interface.
mcb_rd_empty_i	Input	MIG DDR2/DDR3 standard user data port interface.
mcb_rd_en_o	Input	MIG DDR2/DDR3 standard user data port interface.
mcb_wr_data_o[DWIDTH-1:0]	Output	MIG DDR2/DDR3 standard user write data port interface.
mcb_wr_en_o	Output	MIG DDR2/DDR3 standard user write data port interface.
mcb_wr_full_i	Input	MIG DDR2/DDR3 standard user write data port interface.
mode_load_i	Input	When this signal is asserted (High), the values in addr_mode_i, instr_mode_i, bl_mode_i, and data_mode_i are latched and the next traffic pattern is based on the new settings.
rd_data_counts[47:0]	Output	The value of this bus is incremented when data is read from the MIG DDR2/DDR3 standard user read data port.
rst_i	Input	This signal is the reset input.
run_traffic_i	Input	When this signal is asserted (High), the traffic generator starts generating command and data patterns. This signal should be only be asserted when mode_load_i is *not* asserted.
start_addr_i[31:0]	Input	This input defines the start address boundary for the port address space. The least-significant bits [3:0] are ignored.
wr_data_counts[47:0]	Output	The value of this output is incremented when data is sent to the MIG DDR2/DDR3 standard user write data port.

Notes:

1. The block RAM bus interface in the traffic generator is only supported in the Spartan®-6 FPGA environment.

The AXI4 traffic generator signals are the standard signals supported by the AXI4 protocol except for the additional signals mentioned in Verifying the Simulation Using the Example Design (for Designs with the Standard User Interface), page 151l

Memory Initialization and Traffic Test Flow

After power up, the Init Memory Control block directs the traffic generator to initialize the memory with the selected data pattern through the memory initialization procedure.

Memory Initialization

1. The data_mode_i input is set to select the data pattern (for example, data_mode_i[3:0] = `0010` for the address as the data pattern).
2. The start_addr_i input is set to define the lower address boundary.
3. The end_addr_i input is set to define the upper address boundary.
4. bl_mode_i is set to `01` to get the burst length from the fixed_bl_i input.
5. The fixed_bl_i input is set to either 16 or 32.
6. instr_mode_i is set to `0001` to get the instruction from the fixed_instr_i input.
7. The fixed_instr_i input is set to the "WR" command value of the memory device.
8. addr_mode_i is set to `11` for the sequential address mode to fill up the memory space.
9. mode_load_i is asserted for one clock cycle.

When the memory space is initialized with the selected data pattern, the Init Memory Control block instructs the traffic generator to begin running traffic through the traffic test flow procedure (by default, the addr_mode_i, instr_mode_i, and bl_mode_i inputs are set to select PRBS mode).

Traffic Test Flow

1. The addr_mode_i input is set to the desired mode (PRBS is the default).
2. The cmd_seed_i and data_seed_i input values are set for the internal PRBS generator. This step is not required for other patterns.
3. The instr_mode_i input is set to the desired mode (PRBS is the default).
4. The bl_mode_i input is set to the desired mode (PRBS is the default).
5. The data_mode_i input should have the same value as in the memory pattern initialization stage detailed in Memory Initialization.
6. The run_traffic_i input is asserted to start running traffic.
7. If an error occurs during testing (for example, the read data does not match the expected data), the error bit is set until reset is applied.
8. Upon receiving an error, the error_status bus latches the values defined in Table 1-17, page 60.

With some modifications, the example design can be changed to allow addr_mode_i, instr_mode_i, and bl_mode_i to be changed dynamically when run_traffic_i is deasserted. However, after changing the setting, the memory initialization steps need to be repeated to ensure that the proper pattern is loaded into the memory space.

Note: When the data mask option is disabled, the simulation test bench always ties the memory model data mask bit(s) to zero for proper operation.

Setting up for Simulation

The Xilinx UNISIM library must be mapped into the simulator. The test bench provided with the example design supports these pre-implementation simulations:

- The test bench, along with vendor's memory model used in the example design
- The RTL files of the memory controller and the PHY core, created by the MIG tool

To run the simulation, go to this directory:

```
<project_dir>/<component_name>/sim
```

ModelSim is the only supported simulation tool. The simple test bench can be run using ModelSim by executing the `sim.do` script.

Getting Started with EDK

EDK provides an alternative package to the RTL created by the MIG tool in the CORE Generator software. The IP catalog in XPS contains the IP core `axi_v6_ddrx` with the same RTL that is provided by the MIG tool. The difference is that the RTL is packaged as an EDK pcore suitable for use in embedded processor based systems. The `axi_v6_ddrx` pcore only provides an AXI4 slave interface to either DDR2 SDRAM or DDR3 SDRAM in Verilog.

The simplest way to get started with the `axi_v6_ddrx` memory controller is to use the base system builder (BSB) wizard in XPS. The BSB guides the user through a series of options to provide an entire embedded project with an optional `axi_v6_ddrx` memory controller. If the memory controller is selected, an already configured, connected, and tested `axi_v6_ddrx` controller is provided for a particular reference board such as the ML605 board. For more information regarding BSB, refer to chapter 2 of *EDK Concepts, Tools, and Techniques*. [Ref 2]

When starting with a new project, the `axi_v6_ddrx` IP can be added to the design by dragging the memory controller into the project from the IP catalog. The `axi_v6_ddrx` IP is configured using the same MIG tool used in the CORE Generator software and therefore the GUI flow is as described in Getting Started with the CORE Generator Software, page 11. The MIG tool is launched from EDK by double-clicking the axi_v6_ddrx instance, or right-clicking **Configure IP...**. However, instead of generating the RTL top-level wrappers with the parameters already set, the MIG tool sets the parameters for the RTL in the XPS MHS file and in a MIG `.prj` file. From the parameters in the MHS along with the MIG `.prj` file, the pcore can generate the correct constraints and parameter values for itself during the XPS Platform Generator (Platgen) tool. Multiple axi_v6_ddrx cores in a single design can be accomplished by running the MIG GUI for each core, choosing separate I/O banks for each controller.

EDK Clocking

Because the pcore is only a component in the system, the clock/reset structure must also be configured in XPS and is not automatically generated, as in the RTL in the CORE Generator tool. Clocking is described in Clocking Architecture, page 104. These clocking signals are used:

- clk_mem: 1x memory clock
- clk_rd_base: 1x memory clock with fine phase shift enabled. No global clock buffers can be used.

- clk_ref: IDELAYCTRL reference clock, usually 200 MHz (300 MHz can be used in some designs).
- clk: 0.5x memory clock. Also connect to MMCM PSCLK.
- PD_PSEN: Connect to MMCM PSEN port.
- PD_PSINCDEC: Connect to MMCM PSINCDEC port.
- PD_PSDONE: Connect to MMCM PSDONE port.

An example portion of an EDK clock generator instantiation follows, with associated axi_v6_ddrx port connections:

```
BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER C_CLKIN_FREQ = 200000000
 PARAMETER C_CLKOUT0_FREQ = 200000000
 PARAMETER C_CLKOUT0_GROUP = MMCM0
 PARAMETER C_CLKOUT1_FREQ = 400000000
 PARAMETER C_CLKOUT1_GROUP = MMCM0
 PARAMETER C_CLKOUT2_FREQ = 400000000
 PARAMETER C_CLKOUT2_GROUP = MMCM0
 PARAMETER C_CLKOUT2_BUF = FALSE
 PARAMETER C_CLKOUT2_VARIABLE_PHASE = TRUE
 PORT CLKOUT0 = clk_clk_ref_psclk
 PORT CLKOUT1 = clk_mem
 PORT CLKOUT2 = clk_rd_base
 PORT PSCLK = clk_clk_ref_psclk
 PORT PSEN = PD_PSEN
 PORT PSINCDEC = PD_PSINCDEC
 PORT PSDONE = PD_PSDONE
END
```

Similar settings can be used to parameterize a CORE Generator™ tool Clocking Wizard core. For the clk_rd_base clock only, this includes choosing **Dynamic phase shift**, selecting a **No buffer** Drives setting, and checking **Use Fine Ps**.

After the IP is configured and the ports are connected, XPS is used to perform all other aspects of IP management, including generating a bitstream and running simulations. In general, parameters described in this document for the memory controller can be converted to the EDK syntax. This requires all parameters to be upper case and prefixed with "C_." For example, behavioral simulation run time can be improved by adding this MHS parameter to the axi_v6_ddrx instantiation:

```
PARAMETER C_BYPASS_INIT_CAL = FAST
```

AXI4 Interface Connection

After connecting clocks, the slave AXI4 interface is typically connected to an AXI Interconnect core in the **Bus Interfaces** tab of the XPS System Assembly view. If ECC is enabled (C_ECC == "ON"), an additional AXI4-Lite interface is available to access the ECC Control/Status registers. The primary AXI4 interface is called *S_AXI*, and the AXI4-Lite interface is called *S_AXI_CTRL*.

To close timing of the AXI4 interface, it might be necessary to enable register slices from either the MIG tool or the AXI Interconnect GUI.

Note: The native AXI4 interface width is 4x the width of the memory width.

External Ports

The external memory signals can be brought out of the EDK system in the **Ports** tab of the XPS System Assembly view by choosing **Make Ports External** for the (BUS_IF) M_AXI interface collection.

AXI Address

The base and high address range definitions that the memory controller responds to can be set in the **Addresses** tab of the System Assembly view.

For more information about EDK and XPS, refer to *EDK Concepts, Tools, and Techniques* [Ref 2] and the Embedded System Tools Reference manual [Ref 3].

Simulation Considerations

To simulate a design using axi_v6_ddrx, the user must create a test bench that connects a memory model to the axi_v6_ddrx I/O signals. This is generally performed by editing the `system_tb.v/.vhd` test bench template file created by the Simgen tool in XPS to add a memory model. Alternatively, users can transfer the simulator compile commands from Simgen into their own custom simulation/test bench environment.

Note: axi_v6_ddrx does not generally support structural simulation because it is not a supported flow for the underlying MIG PHYs. Thus structural simulation is not recommended.

axi_v6_ddrx simulation should be performed in the behavioral/functional level and requires a simulator capable of mixed-mode Verilog and VHDL language support.

It might be necessary for the test bench to place weak pull-down resistors on all DQ and DQS signals so that the calibration logic can resolve logic values under simulation. Otherwise, "X" propagation of input data might cause simulation of the calibration logic to fail.

For behavioral simulation, the clk and clk_mem ports of axi_v6_ddrx must also be completely phase-aligned.

Core Architecture

This section describes the architecture of the Virtex-6 FPGA memory interface solutions core, providing an overview of the core modules and interfaces.

Overview

The Virtex-6 FPGA memory interface solutions core is shown in Figure 1-45.

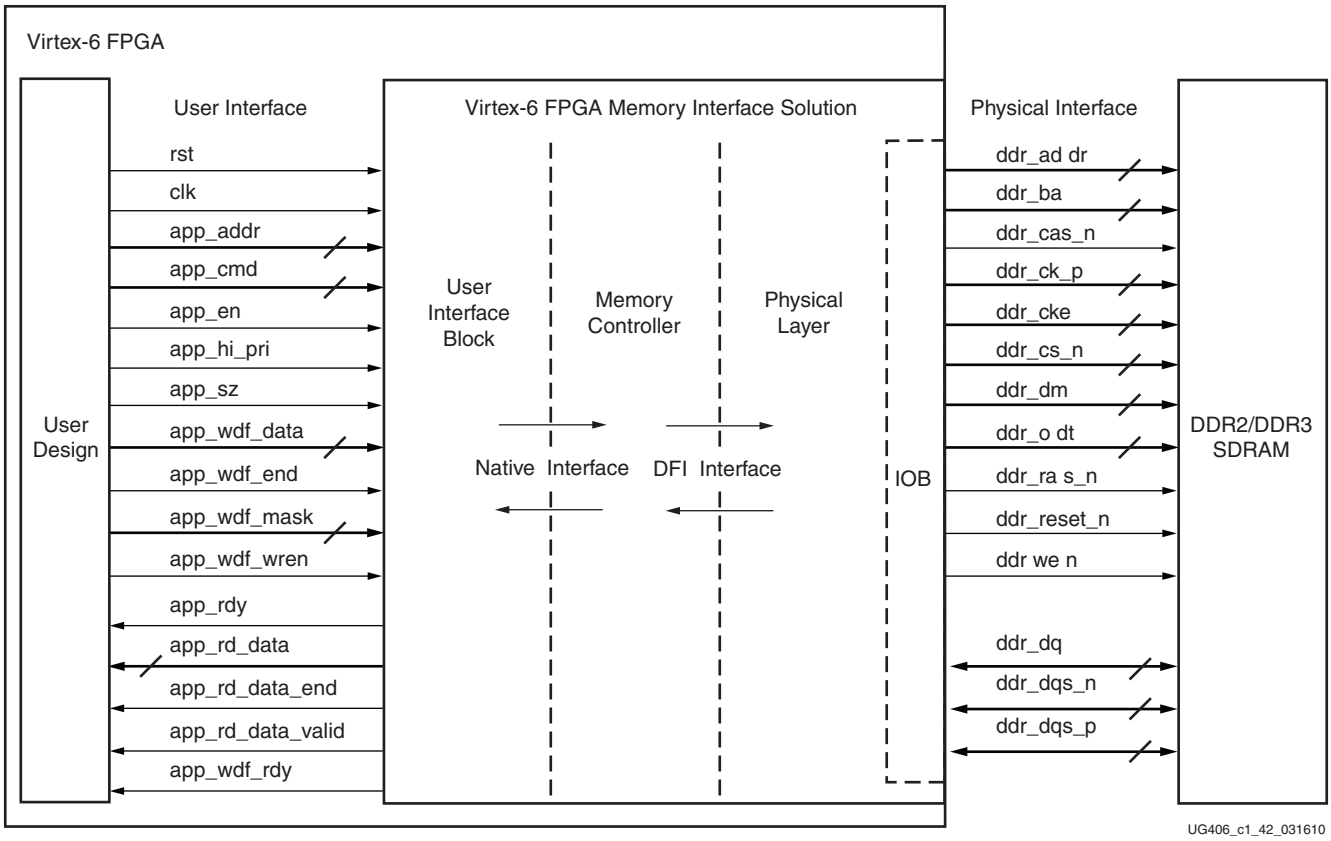


Figure 1-45: **Virtex-6 FPGA Memory Interface Solution**

User Design

The user design block shown in Figure 1-45 is any FPGA design that requires to be connected to an external DDR2 or DDR3 SDRAM. The user design connects to the memory controller via the user interface. An example user design is provided with the core.

AXI4 Slave Interface Block

The AXI4 slave interface maps AXI4 transactions to the UI to provide an industry-standard bus protocol interface to the memory controller.

User Interface Block and User Interface

The UI block presents the UI to the user design block. It provides a simple alternative to the native interface by presenting a flat address space and buffering read and write data.

Memory Controller and Native Interface

The front end of the memory controller (MC) presents the native interface to the UI block. The native interface allows the user design to submit memory read and write requests and provides the mechanism to move data from the user design to the external memory device, and vice versa. The back end of the MC connects to the physical interface and handles all the interface requirements to that module. The MC also provides a reordering option that reorders received requests to optimize data throughput and latency.

PHY and the Physical Interface

The front end of the PHY connects to the MC. The back end of the PHY connects to the external memory device. The PHY handles all memory device signal sequencing and timing.

IDELAYCTRL

An IDELAYCTRL is required in any bank that uses IODELAYs. IODELAYs are associated with the data group, the capture clocks, and the resynchronization (BUFR-rsync) clocks. Any bank/clock region that uses these signals require an IDELAYCTRL.

The MIG tool instantiates one IODELAYCTRL and then uses the IODELAY_GROUP attribute (see the `iodelay_ctrl.v/.vhd` module). Based on this attribute, the ISE tool properly replicates IODELAYCTRLs as needed within the design.

The IDELAYCTRL reference frequency varies based on the selected design frequency. If the selected design frequency is 480 MHz and above, the IDELAYCTRL reference clock frequency is 300 MHz, otherwise it is 200 MHz. If multi-controller designs (for example, DDR3 SDRAM multi-controller designs or combinations of DDR3 SDRAM and QDRII controllers) are generated such that one of the controller frequencies is 480 MHz and above and the other controller frequency is below 480 MHz, then IDELAYCTRLs with reference frequencies of both 200 MHz and 300 MHz are generated. When the MIG tool generates a multi-controller design, the MIG tool only instantiates one IODELAYCTRL with this primitive if only one of the 200 MHz or 300 MHz IODELAYCTRLs is required and allows the tools to replicate. If the design is generated such that both 200 MHz and 300 MHz IODELACYCTRLs are required, the MIG tool instantiates two IODELAYCTRLs with primitives and passes the IODELAY_GROUP parameters accordingly. The MIG tool generates the design such that all 200 MHz IODELAY elements and IODELAYCTRLs use the IODELAY_GROUP parameter value of IODELAY200_MIG, and all 300 MHz IODELAYCTRLs and IODELAYs use the IODELAY_GROUP parameter value of IODELAY300_MIG. Based on the IODELAY_GROUP attribute that is set, the ISE tool replicates the IODELAYCTRLs for each region where the IODELAY blocks exist. When a user creates a multi-controller design on their own (for example, DDR2 SDRAM because it is not supported through the GUI), each MIG output has the component instantiated with the primitive. This violates the rules for IODELAYCTRLs and the usage of the IODELAY_GRP attribute. IODELAYCTRLs need to have only one instantiation of the component with the attribute set properly, and allow the tools to replicate as needed.

User Interface

The UI is shown in Table 1-18 and connects to an FPGA user design to allow access to an external memory device.

Table 1-18: **User Interface**

Signal	Direction	Description
app_addr[ADDR_WIDTH – 1:0]	Input	This input indicates the address for the current request.
app_cmd[2:0]	Input	This input selects the command for the current request.
app_en	Input	This is the active-High strobe for the app_addr[], app_cmd[2:0], app_sz, and app_hi_pri inputs.

Table 1-18: **User Interface *(Cont'd)***

Signal	Direction	Description
app_rdy	Output	This output indicates that the UI is ready to accept commands. If the signal is deasserted when app_en is enabled, the current app_cmd and app_addr must be retried until app_rdy is asserted.
app_hi_pri	Input	This active-High input elevates the priority of the current request.
app_rd_data [APP_DATA_WIDTH – 1:0]	Output	This provides the output data from read commands.
app_rd_data_end	Output	This active-High output indicates that the current clock cycle is the last cycle of output data on app_rd_data[].
app_rd_data_valid	Output	This active-High output indicates that app_rd_data[] is valid.
app_sz (1)	Input	For DDR3 SDRAM, app_sz can be changed dynamically if BURST_MODE is set to OTF.
app_wdf_data [APP_DATA_WIDTH – 1:0]	Input	This provides the data for write commands.
app_wdf_end	Input	This active-High input indicates that the current clock cycle is the last cycle of input data on app_wdf_data[].
app_wdf_mask [APP_MASK_WIDTH – 1:0]	Input	This provides the mask for app_wdf_data[].
app_wdf_rdy	Output	This output indicates that the write data FIFO is ready to receive data. Write data is accepted when app_wdf_rdy = `1'b1` and app_wdf_wren = `1'b1`.
app_wdf_wren	Input	This is the active-High strobe for app_wdf_data[].
app_correct_en	Input	This active-High signal is used to correct the data bits when there are data bit errors. This input is valid when ECC is enabled in the GUI; this corrects single bit errors only. The MIG tool always sets this signal to 1 in the RTL code.
clk(2)	Input	This UI clock must be half of the DRAM clock.
clk_mem(2)	Input	This is a full-frequency memory clock.
clk_rd_base(2)	Input	This clock input from the MMCM goes to BUFR to become the rsync clock and to the BUFIOs to become the capture clock on a per-byte basis. This is a full-frequency clock.

Table 1-18: **User Interface *(Cont'd)***

Signal	Direction	Description
phy_init_done	Output	The PHY asserts phy_init_done when calibration is finished.
app_ecc_multiple_err[3:0] (1)	Output	This signal is applicable when ECC is enabled and is valid along with app_rd_data_valid. The app_ecc_multiple_err[3:0] signal is non-zero if the read data from the external memory has two bit errors per beat of the read burst. The SECDED algorithm does not correct the corresponding read data and puts a non-zero value on this signal to notify the corrupted read data at the UI.
rst	Input	This is the active-High UI reset.

Notes:

1. This signal is brought up to the user design top module for BURST_MODE of OTF. For other configurations, this port resides in the `memc_ui_top` module level only. The value of app_sz is don't care for BC4 and BL8 configurations.
2. Refer to Clocking Architecture, page 104 for more details on clocks. The User Design top module also contains the ui_clk and ui_clk_sync_rst ports. The ui_clk port is a user interface clock that can be used as a reference for driving the user interface signals. Similarly, the ui_clk_sync_rst port is the reset synchronized to ui_clk and can be used to reset the user interface signals. The ui_clk port is generated from the MMCM, which is half of the DRAM clock rate.

app_addr[ADDR_WIDTH – 1:0]

This input indicates the address for the request currently being submitted to the UI. The UI aggregates all the address fields of the external SDRAM and presents a flat address space to the user.

app_cmd[2:0]

This input specifies the command for the request currently being submitted to the UI. The available commands are shown in Table 1-19.

Table 1-19: **Commands for app_cmd[2:0]**

Operation	app_cmd[2:0] Code
Read	001
Write	000

app_en

This input strobes in a request. The user must apply the desired values to app_addr[], app_cmd[2:0], app_sz, and app_hi_pri, and then assert app_en to submit the request to the UI. This initiates a handshake that the UI acknowledges by asserting app_rdy.

app_hi_pri

This input indicates that the current request is a high priority.

app_sz

For DDR3 SDRAM, app_sz can be changed dynamically if BURST_MODE is set to OTF. The app_sz signal specifies the burst length. For BC4, this bit should be set to `0`. For BL8, this bit should be set to `1`.

app_wdf_data[APP_DATA_WIDTH – 1:0]

This bus provides the data currently being written to the external memory.

app_wdf_end

This input indicates that the data on the app_wdf_data[] bus in the current cycle is the last data for the current request.

app_wdf_mask[APP_MASK_WIDTH – 1:0]

This bus indicates which bits of app_wdf_data[] are written to the external memory and which bits remain in their current state.

app_wdf_wren

This input indicates that the data on the app_wdf_data[] bus is valid.

app_rdy

This output indicates to the user whether the request currently being submitted to the UI is accepted. If the UI does not assert this signal after app_en is asserted, the current request must be retried. The app_rdy output is not asserted if:

- PHY/Memory initialization is not yet completed
- All the bank machines are occupied (can be viewed as the command buffer being full)
  - A read is requested and the read buffer is full
  - A write is requested and no write buffer pointers are available
- A periodic read is being inserted

app_rd_data[APP_DATA_WIDTH – 1:0]

This output contains the data read from the external memory.

app_rd_data_end

This output indicates that the data on the app_rd_data[] bus in the current cycle is the last data for the current request.

app_rd_data_valid

This output indicates that the data on the app_rd_data[] bus is valid.

app_wdf_rdy

This output indicates that the write data FIFO is ready to receive data. Write data is accepted when both app_wdf_rdy and app_wdf_wren are asserted.

rst

This is the reset input for the UI.

clk

This is the input clock for the UI. It must be half the frequency of the clock going out to the external SDRAM.

clk_mem

This is a full-frequency clock provided from the MMCM and should only be used as an input to the OSERDES.

clk_rd_base

One copy of full-frequency clk_rd_base is routed to the individual capture clock networks for each DQS group; the phase of each of these individual DQS clocks is then adjusted during read leveling via an IODELAY to position the capture clock in the middle of the read data eye. The phase of clk_rd_base relative to clk_mem (used to drive the forwarded clock to the memory) is phase adjusted via the fine-phase shift feature of the MMCM by the phase detector logic to account for ongoing delay variations in the read capture path due to voltage and temperature changes.

phy_init_done

The PHY asserts phy_init_done when calibration is finished. The application has no need to wait for phy_init_done before sending commands to the memory controller.

AXI4 Slave Interface Block

The AXI4 slave interface block maps AXI4 transactions to the UI interface to provide an industry-standard bus protocol interface to the memory controller. The AXI4 slave interface is optional in designs provided through the MIG tool. The AXI4 slave interface is required with the axi_v6_ddrx memory controller provided in EDK. The RTL is consistent between both tools. For details on the AXI4 signaling protocol, see the ARM AMBA specifications. [Ref 1]

The overall design is composed of separate blocks to handle each AXI channel, which allows for independent read and write transactions. Read and write commands to the UI rely on a read priority arbitration scheme explained in Arbitration in the AXI Shim, page 75 to handle simultaneous requests. The address read/address write modules are responsible for chopping the AXI4 burst/wrap requests into smaller memory size burst lengths of either four or eight, and also conveying the smaller burst lengths to the read/write data modules so they can interact with the user interface.

If ECC is enabled, all write commands are issued as a read-modify-write operation. If an uncorrectable ECC error occurs, the read data is returned with the SLVERR flag. An uncorrectable read error during a read-modify-write operation does not signal an error on the BRESP.

AXI4 Slave Interface Parameters

Table 1-20 lists the AXI4 slave interface parameters.

Table 1-20: **AXI4 Slave Interface Parameters**

Parameter Name	Default Value	Allowable Values	Description
C_S_AXI_ADDR_WIDTH	32	32, 64	This is the width of address read and address write signals. EDK designs are limited to 32.
C_S_AXI_DATA_WIDTH	32	32, 64, 128, 256	This is the width of data signals. The recommended width is 4x the memory data width. The width can be smaller, but not greater than 4x the memory data width.
C_S_AXI_ID_WIDTH	4	1–16	This is the width of ID signals for every channel. This value is automatically computed in EDK designs.
C_S_AXI_SUPPORTS_NARROW_BURST	1	0, 1	This parameter adds logic blocks to support narrow AXI transfers. It is required if any master connected to the memory controller issues narrow bursts. This parameter is automatically set if the AXI data width is smaller than the recommended value.
C_S_AXI_BASEADDR	N/A	Valid address	This parameter specifies the base address for the memory mapped slave interface. Address requests at this address map to rank 1, bank 0, row 0, column 0. The base/high address together define the accessible size of the memory. This accessible size must be a power of 2. Additionally, the base/high address pair must be aligned to a multiple of the accessible size. The minimum accessible size is 4096 bytes.
C_S_AXI_HIGHADDR	N/A	Valid address	This parameter specifies the high address for the memory mapped slave interface. Address requests received above this value wrap back to the base address. The base/high address together define the accessible size of the memory. This accessible size must be a power of 2. Additionally, the base/high address pair must be aligned to a multiple of the accessible size. The minimum accessible size is 4096 bytes.
C_S_AXI_PROTOCOL	AXI4	AXI4	This parameter specifies the AXI protocol (EDK metadata parameter only).

AXI4 Slave Interface Signals

Table 1-21 lists the AXI4 slave interface specific signal. Clock/reset to the interface is provided from the memory controller.

Table 1-21: **AXI4 Slave Interface Signals**

Name	Width	Direction	Active State	Description
clk	1	Input		Input clock to the core.
reset	1	Input	High	Input reset to the core.
s_axi_awid	C_AXI_ID_WIDTH	Input		Write address ID.
s_axi_awaddr	C_AXI_ADDR_WIDTH	Input		Write address.
s_axi_awlen	8	Input		Burst length. The burst length gives the exact number of transfers in a burst.
s_axi_awsize	3	Input		Burst size. This signal indicates the size of each transfer in the burst.
s_axi_awburst	2	Input		Burst type.
s_axi_awlock	1	Input		Lock type. (This is not used in the current implementation.)
s_axi_awcache	4	Input		Cache type. (This is not used in the current implementation.)
s_axi_awprot	3	Input		Protection type. (Not used in the current implementation.)
s_axi_awvalid	1	Input	High	Write address valid. This signal indicates that valid write address and control information are available.
s_axi_awready	1	Output	High	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
s_axi_wdata	C_AXI_DATA_WIDTH	Input		Write data.
s_axi_wstrb	C_AXI_DATA_WIDTH/8	Input		Write strobes.
s_axi_wlast	1	Input	High	Write last. This signal indicates the last transfer in a write burst.
s_axi_wvalid	1	Input	High	Write valid. This signal indicates that write data and strobe are available.
s_axi_wready	1	Output	High	Write ready.
s_axi_bid	C_AXI_ID_WIDTH	Output		Response ID. The identification tag of the write response.
s_axi_bresp	2	Output		Write response. This signal indicates the status of the write response.
s_axi_bvalid	1	Output	High	Write response valid.
s_axi_bready	1	Input	High	Response ready.

Table 1-21: **AXI4 Slave Interface Signals *(Cont'd)***

Name	Width	Direction	Active State	Description
s_axi_arid	C_AXI_ID_WIDTH	Input		Read address ID.
s_axi_araddr	C_AXI_ADDR_WIDTH	Input		Read address.
s_axi_arlen	8	Input		Read burst length.
s_axi_arsize	3	Input		Read burst size.
s_axi_arburst	2	Input		Read burst type.
s_axi_arlock	1	Input		Lock type. (This is not used in the current implementation.)
s_axi_arcache	4	Input		Cache type. (This is not used in the current implementation.)
s_axi_arprot	3	Input		Protection type. (This is not used in the current implementation.)
s_axi_arvalid	1	Input	High	Read address valid.
s_axi_arready	1	Output	High	Read address ready.
s_axi_rid	C_AXI_ID_WIDTH	Output		Read ID tag.
s_axi_rdata	C_AXI_DATA_WIDTH	Output		Read data.
s_axi_rresp	2	Output		Read response.
s_axi_rlast	1	Output		Read last.
s_axi_rvalid	1	Output		Read valid.
s_axi_rready	1	Input		Read ready.

Arbitration in the AXI Shim

The AXI4 protocol calls for independent read and write address channels. The memory controller has one address channel, and the following arbitration options are available for arbitrating between the read and write address channels.

Time Division Multiplexing (TDM)

Equal priority is given to read and write address channels in this mode. The grant to the read and write address channels alternates every clock cycle. The read or write request from the AXI master has no bearing on the grants.

Round Robin

Equal priority is given to read and write address channels in this mode. The grant to the read and write channels depends on the requests from the AXI master. The grant to the read and write address channels alternates every clock cycle provided there is a corresponding request from the AXI master for the address channel. In a given time slot, if the corresponding address channel does not have a request, the grant is given to the other address channel with the pending request.

Read Priority (RD_PRI_REG)

The read address channel is always given priority in this mode. The requests from the write address channel are processed when there are no pending requests from the read address channel.

Read Priority with Starve Limit (RD_PRI_REG_STARVE_LIMIT)

The read address channel is always given priority in this mode. The requests from the write address channel are processed when there are no pending requests from the read address channel or the starve limit for read is reached.

AXI4-Lite Slave Control/Status Register Interface Block

The AXI4-Lite Slave Control Register block provides a processor accessible interface to the ECC memory option. The interface is available when ECC is enabled and the primary slave interface is AXI4. The block provides interrupts, interrupt enable, ECC status, ECC enable/disable, ECC correctable errors counter, first failing correctable/uncorrectable data, ECC and address. Fault injection registers for software testing is provided when the ECC_TEST_FI_XOR (C_ECC_TEST) parameter is "ON". The AXI4-Lite interface is fixed at 32 data bits and signaling follows the standard AMBA AXI4-Lite specifications [Ref 1].

The AXI4-Lite control/status register interface block is implemented in parallel to the AXI4 full memory-mapped interface. The block monitors the output of the native interface to capture correctable (single bit) and uncorrectable (multiple bit) errors. When a correctable and/or uncorrectable error occurs, the interface also captures the byte address of the failure along with the failing data bits and ECC bits. Fault injection is provided by an XOR block placed in the write datapath after the ECC encoding has occurred. Only the first memory beat in a transaction can have errors inserted. For example, in a memory configuration with a data width of 72 and a mode register set to burst length 8, only the first 72 bits are corruptible through the fault injection interface. Interrupt generation based on either a correctable or uncorrectable error can be independently configured with the register interface.

ECC Enable/Disable

The ECC_ON_OFF register enables/disables the ECC decode functionality. However, encoding is always enabled. The default value at start-up can be parameterized with C_ECC_ONOFF_RESET_VALUE. Assigning a value of 1 for the ECC_ON_OFF bit of this register results in the correct_en signal input into the mem_intfc to be asserted. Writing a value of 0 to the ECC_ON_OFF bit of this register results in the correct_en signal to be deasserted. When correct_en is asserted, decoding is enabled, and the opposite is true when this signal is deasserted. ECC_STATUS/ECC_CE_CNT are not updated when ECC_ON_OFF = 0. The FI_D0, FI_D1, FI_D2, and FI_D3 registers are not writable when ECC_ON_OFF = 0.

Single Error and Double Error Reporting

Two vectored signals from the memory controller indicate an ECC error: ecc_single and ecc_multiple. The ecc_single signal indicates if there has been a correctable error, and the ecc_multiple signal indicates if there has been an uncorrectable error. The widths of ecc_multiple and ecc_single are based on the C_NCK_PER_CLK parameter. There can be between 0 and C_NCK_PER_CLK * 2 errors per cycle with each data beat signaled by one of the vector bits. Multiple bits of the vector can be signaled per cycle indicating that multiple correctable errors or multiple uncorrectable errors have been detected. The

ecc_err_addr signal (discussed in Fault Collection) is valid during the assertion of either ecc_single or ecc_multiple.

The ECC_STATUS register sets the CE_STATUS bit and/or UE_STATUS bit for correctable error detection and uncorrectable error detection, respectively.

Interrupt Generation

When interrupts are enabled with the CE_EN_IRQ and/or UE_EN_IRQ bits of the ECC_EN_IRQ register, if a correctable error or uncorrectable error occurs, the interrupt signal is asserted.

Fault Collection

To aid the analysis of ECC errors, there are two banks of storage registers that collect information on the failing ECC decode. One bank of registers is for correctable errors, and another bank is for uncorrectable errors. The failing address, undecoded data, and ECC bits are saved into these register banks as CE_FFA, CE_FFD, and CE_FFE for correctable errors, and UE_FFA, UE_FFD, and UE_FFE for uncorrectable errors. The data in combination with the ECC bits can help determine which bit(s) have failed. CE_FFA stores the address from the ecc_err_addr signal and converts it to a byte address. Upon error detection, the data is latched into the appropriate register. Only the first data beat with an error is stored.

When a correctable error occurs, there is also a counter that counts the number of correctable errors that have occurred. The counter can be read from the CE_CNT register and is fixed as an 8-bit counter; it does not rollover when the maximum value is incremented.

Fault Injection

The ECC fault injection register, FI_D and FI_ECC, facilitates testing of the software drivers. When set, the ECC fault injection register XORs with the MIG DFI datapath to simulate errors in the memory. The DFI interface lies between the Memory Controller and the PHY. It is ideal for injection at this point because this is after the encoding has been completed. There is only support to insert errors on the first data beat, therefore there are two to four FI_D registers to accommodate this. During operation, after the error has been inserted into the datapath, the register clears itself.

AXI4-Lite Slave Control/Status Register Interface Parameters

Table 1-22 lists the AXI4-Lite slave interface parameters.

Table 1-22: **AXI4-Lite Slave Control/Status Register Parameters**

Parameter Name	Default Value	Allowable Values	Description
C_S_AXI_CTRL_ADDR_WIDTH	32	32,64	This is the width of the AXI4-Lite address buses. EDK designs are limited to 32.
C_S_AXI_CTRL_DATA_WIDTH	32	32	This is the width of the AXI4-Lite data buses.
C_ECC_ONOFF_RESET_VALUE	1	0,1	Controls ECC on/off value at start-up/reset.
C_S_AXI_CTRL_BASEADDR	N/A	Valid Address	This parameter specifies the base address for the AXI4-Lite slave interface.

Table 1-22: **AXI4-Lite Slave Control/Status Register Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
C_S_AXI_CTRL_HIGHADDR	N/A	Valid Address	This parameter specifies the high address for the AXI4-Lite slave interface.
C_S_AXI_CTRL_PROTOCOL	AXI4LITE	AXI4LITE	AXI4-Lite protocol (EDK metadata parameter.)

AXI4-Lite Slave Control/Status Register Interface Signals

Table 1-23 lists the AXI4 slave interface specific signals. Clock/reset to the interface is provided from the memory controller.

Table 1-23: **List of New I/O Signals**

Name	Width	Direction	Active State	Description
s_axi_ctrl_awaddr	C_S_AXI_CTRL_ADDR_WIDTH	Input		Write address.
s_axi_ctrl_awvalid	1	Input	High	Write address valid. This signal indicates that valid write address and control information are available.
s_axi_ctrl_awready	1	Output	High	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
s_axi_ctrl_wdata	C_S_AXI_CTRL_DATA_WIDTH	Input		Write data
s_axi_ctrl_wvalid	1	Input	High	Write valid. This signal indicates that write data and strobe are available.
s_axi_ctrl_wready	1	Output	High	Write ready.
s_axi_ctrl_bvalid		Output	High	Write response valid.
s_axi_ctrl_bready	1	Input	High	Response ready.
s_axi_ctrl_araddr	C_S_AXI_CTRL_ADDR_WIDTH	Input		Read address.
s_axi_ctrl_arvalid	1	Input	High	Read address valid.
s_axi_ctrl_arready	1	Output	High	Read address ready.
s_axi_ctrl_rdata	C_S_AXI_CTRL_DATA_WIDTH	Output		Read data.
s_axi_ctrl_rvalid	1	Output		Read valid.

Table 1-23: **List of New I/O Signals *(Cont'd)***

Name	Width	Direction	Active State	Description
s_axi_ctrl_rready	1	Input		Read ready.
interrupt	1	Output	High	IP Global Interrupt signal

AXI4-Lite Slave Control/Status Register Map

ECC register map is shown in Table 1-24. The register map is little endian. Write accesses to read-only or reserved values are ignored. Read accesses to write-only or reserved values return the value `0xDEADDEAD`.

Table 1-24: **ECC Control Register Map**

Address Offset	Register Name	Access Type	Default Value	Description
0x00	ECC_STATUS	R/W	0x0	ECC Status Register
0x04	ECC_EN_IRQ	R/W	0x0	ECC Enable Interrupt Register
0x08	ECC_ON_OFF	R/W	0x0 or 0x1	ECC On/Off Register. If C_ECC_ONOFF_RESET_VALUE = 1, the default value is 0x1.
0x0C	CE_CNT	R/W	0x0	Correctable Error Count Register
(0x10-0x9C) Reserved				
0x100	CE_FFD[31:00]	R	0x0	Correctable Error First Failing Data Register.
0x104	CE_FFD[63:32]	R	0x0	Correctable Error First Failing Data Register.
0x108	CE_FFD[95:64](1)	R	0x0	Correctable Error First Failing Data Register.
0x10C	CE_FFD [127:96](1)	R	0x0	Correctable Error First Failing Data Register.
(0x110 - 0x17C) Reserved				
0x180	CE_FFE	R	0x0	Correctable Error First Failing ECC Register.
(0x184 - 0x1BC) Reserved				
0x1C0	CE_FFA[31:0]	R	0x0	Correctable Error First Failing Address
0x1C4	CE_FFA[63:32](2)	R	0x0	Correctable Error First Failing Address
(0x1C8 - 0x1FC) Reserved				

Table 1-24: **ECC Control Register Map *(Cont'd)***

Address Offset	Register Name	Access Type	Default Value	Description
0x200	UE_FFD [31:00]	R	0x0	Uncorrectable Error First Failing Data Register.
0x204	UE_FFD [63:32]	R	0x0	Uncorrectable Error First Failing Data Register.
0x208	UE_FFD [95:64](1)	R	0x0	Uncorrectable Error First Failing Data Register.
0x20C	UE_FFD [127:96](1)	R	0x0	Uncorrectable Error First Failing Data Register.
(0x210 - 0x27C) Reserved				
0x280	UE_FFE	R	0x0	Uncorrectable Error First Failing ECC Register.
(0x284 - 0x2BC) Reserved				
0x2C0	UE_FFA[31:0]	R	0x0	Uncorrectable Error First Failing Address
0x2C4	UE_FFA[63:32](2)	R	0x0	Uncorrectable Error First Failing Address
(0x2C8 - 0x2FC) Reserved				
0x300	FI_D[31:0](3)	W	0x0	Fault Inject Data Register
0x304	FI_D[63:32](3)	W	0x0	Fault Inject Data Register
0x308	FI_D[95:64](1)(3)	W	0x0	Fault Inject Data Register
0x30C	FI_D[127:96](1)(3)	W	0x0	Fault Inject Data Register
(0x340 - 0x37C) Reserved				
0x380	FI_ECC(3)	W	0x0	Fault Inject ECC Register

Notes:

1. Data bits 64-127 are only enabled if the DQ width is 144 bits.
2. Reporting address bits 63-32 are only available if the address map is > 32 bits.
3. FI_D* and FI_ECC* are only enabled if ECC_TEST parameter has been set to '1'.

AXI4-Lite Slave Control/Status Register Map Detailed Descriptions

ECC_STATUS

This register holds information on the occurrence of correctable and uncorrectable errors. The status bits are independently set to 1 for the first occurrence of each error type. The status bits are cleared by writing a 1 to the corresponding bit position; that is, the status bits can only be cleared to 0 and not set to 1 using a register write. The ECC Status register operates independently of the ECC Enable Interrupt register.

Table 1-25: **ECC Status Register (ECC_STATUS)**

31	2	1	0
Reserved			ECC_STATUS

Table 1-26: **ECC Status Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
1	CE_STATUS	R/W	0	If 1, a correctable error has occurred. This bit is cleared when a 1 is written to this bit position.
0	UE_STATUS	R/W	0	If 1, an uncorrectable error has occurred. This bit is cleared when a 1 is written to this bit position

ECC_EN_IRQ

This register determines if the values of the CE_STATUS and UE_STATUS bits in the ECC Status Register assert the Interrupt output signal (ECC_Interrupt). If both CE_EN_IRQ and UE_EN_IRQ are set to 1 (enabled), the value of the Interrupt signal is the logical OR between the CE_STATUS and UE_STATUS bits.

Table 1-27: **ECC Interrupt Enable Register (ECC_EN_IRQ)**

31	2	1	0
Reserved			ECC_EN_IRQ

Table 1-28: **ECC Interrupt Enable Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
1	CE_EN_IRQ	R/W	0	If 1, the value of the CE_STATUS bit of ECC Status Register is propagated to the Interrupt signal. If 0, the value of the CE_STATUS bit of ECC Status Register is not propagated to the Interrupt signal.
0	UE_EN_IRQ	R/W	0	If 1, the value of the UE_STATUS bit of ECC Status Register is propagated to the Interrupt signal. If 0, the value of the UE_STATUS bit of ECC Status Register is not propagated to the Interrupt signal.

ECC_ON_OFF

The ECC On/Off Control Register allows the application to enable or disable ECC checking. The design parameter, C_ECC_ONOFF_RESET_VALUE (default on) determines the reset value for the enable/disable setting of ECC. This facilitates start-up operations when ECC might or might not be initialized in the external memory. When disabled, ECC checking is disabled for read but ECC generation is active for write operations.

Table 1-29: **ECC On/Off Control Register (ECC_ON_OFF)**

31	1	0
Reserved		ECC_ON_OFF

Table 1-30: **ECC On/Off Control Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
0	ECC_ON_OFF	R/W	Specified by design parameter, C_ECC_ONOFF_RESET_VALUE	If 0, ECC checking is disabled on read operations. (ECC generation is enabled on write operations when C_ECC = 1). If 1, ECC checking is enabled on read operations. All correctable and uncorrectable error conditions are captured and status is updated.

CE_CNT

This register counts the number of occurrences of correctable errors. It can be cleared or preset to any value using a register write. When the counter reaches its maximum value, it does not wrap around, but instead it stops incrementing and remains at the maximum value. The width of the counter is defined by the value of the C_CE_COUNTER_WIDTH parameter. The value of the CE counter width is fixed to 8 bits.

Table 1-31: **Correctable Error Counter Register (CE_CNT)**

31	8	7	0
Reserved		CE_FFA[31:0]	

Table 1-32: **Correctable Error Counter Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[7:0]	CE_CNT	R/W	0	Holds the number of correctable errors encountered.

CE_FFA[31:0]

This register stores the address (bits [31:0]) of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the address of the next correctable error. Storing of the failing address is enabled after reset.

Table 1-33: **Correctable Error First Failing Address Register (CE_FFA[31:0])**

31	0
CE_FFA[31:0]	

Table 1-34: **Correctable Error First Failing Address [31:0] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	CE_FFA[31:0]	R	0	Address (bits [31:0]) of the first occurrence of a correctable error

CE_FFA[63:32]

Note: This register is unused if C_S_AXI_ADDR_WIDTH < 33.

This register stores the address (bits [63:32]) of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the address of the next correctable error. Storing of the failing address is enabled after reset.

Table 1-35: **Correctable Error First Failing Address Register (CE_FFA[63:32])**

31	0
CE_FFA[63:32]	

Table 1-36: **Correctable Error First Failing Address [63:32] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	CE_FFA[63:32]	R	0	Address (bits [63:32]) of the first occurrence of a correctable error.

CE_FFD[31:0]

This register stores the (uncorrected) failing data (bits [31:0]) of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next correctable error. Storing of the failing data is enabled after reset.

Table 1-37: **Correctable Error First Failing Data Register (CE_FFD[31:0])**

31	0
CE_FFD[31:0]	

Table 1-38: **Correctable Error First Failing Data [31:0] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	CE_FFD[31:0]	R	0	Data (bits [31:0]) of the first occurrence of a correctable error.

CE_FFD[63:32]

This register stores the (uncorrected) failing data (bits [63:32]) of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next correctable error. Storing of the failing data is enabled after reset.

Table 1-39: **Correctable Error First Failing Data Register (CE_FFD[63:32])**

31	0
CE_FFD[63:32]	

Table 1-40: **Correctable Error First Failing Data [63:32] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	CE_FFD[63:32]	R	0	Data (bits [63:32]) of the first occurrence of a correctable error.

CE_FFD[95:64]

Note: This register is only used when DQ_WIDTH == 144.

This register stores the (uncorrected) failing data (bits [95:64]) of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next correctable error. Storing of the failing data is enabled after reset.

Table 1-41: **Correctable Error First Failing Data Register (CE_FFD[95:64])**

31	0
CE_FFD[95:64]	

Table 1-42: **Correctable Error First Failing Data [95:64] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	CE_FFD[95:64]	R	0	Data (bits [95:64]) of the first occurrence of a correctable error.

CE_FFD[127:96]

Note: This register is only used when DQ_WIDTH == 144.

This register stores the (uncorrected) failing data (bits [127:96]) of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next correctable error. Storing of the failing data is enabled after reset.

Table 1-43: **Correctable Error First Failing Data Register (CE_FFD[127:96])**

31	0
CE_FFD[127:96]	

Table 1-44: **Correctable Error First Failing Data [127:96] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	CE_FFD [127:96]	R	0	Data (bits [127:96]) of the first occurrence of a correctable error.

CE_FFE

This register stores the ECC bits of the first occurrence of an access with a correctable error. When the CE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the ECC of the next correctable error. Storing of the failing ECC is enabled after reset.

Table 1-45 and Table 1-46 describe the register bit usage when DQ_WIDTH = 72.

Table 1-45: **Correctable Error First Failing ECC Register (CE_FFE) for 72-Bit External Memory Width**

31	8	7	0
Reserved		CE_FFE	

Table 1-46: **Correctable Error First Failing ECC Register Bit Definitions for 72-Bit External Memory Width**

Bit(s)	Name	Core Access	Reset Value	Description
[7:0]	CE_FFE	R	0	ECC (bits [7:0]) of the first occurrence of a correctable error.

Table 1-47 and Table 1-48 describe the register bit usage when DQ_WIDTH = 144.

Table 1-47: **Correctable Error First Failing ECC Register (CE_FFE) for 144-Bit External Memory Width**

31	9	8	0
Reserved		CE_FFE	

Table 1-48: **Correctable Error First Failing ECC Register Bit Definitions for 144-Bit External Memory Width**

Bit(s)	Name	Core Access	Reset Value	Description
[8:0]	CE_FFE	R	0	ECC (bits [8:0]) of the first occurrence of a correctable error.

UE_FFA[31:0]

This register stores the address (bits [31:0]) of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the address of the next uncorrectable error. Storing of the failing address is enabled after reset.

Table 1-49: **Uncorrectable Error First Failing Data Register (UE_FFA[31:0])**

31	0
UE_FFA[31:0]	

Table 1-50: **Uncorrectable Error First Failing Address [31:0] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	UE_FFA [31:0]	R	0	Address (bits [31:0]) of the first occurrence of an uncorrectable error.

UE_FFA[63:32]

Note: This register is unused if C_S_AXI_ADDR_WIDTH < 33.

This register stores the address (bits [63:32]) of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the address of the next uncorrectable error. Storing of the failing address is enabled after reset.

Table 1-51: **Uncorrectable Error First Failing Data Register (UE_FFA[63:32])**

31	0
UE_FFA[63:32]	

Table 1-52: **Uncorrectable Error First Failing Address [31:0] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	UE_FFA[63:32]	R	0	Address (bits [63:32]) of the first occurrence of an uncorrectable error

UE_FFD[31:0]

This register stores the (uncorrected) failing data (bits [31:0]) of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next uncorrectable error. Storing of the failing data is enabled after reset.

Table 1-53: **Uncorrectable Error First Failing Data Register (UE_FFD[31:0])**

31	0
UE_FFD[31:0]	

Table 1-54: **Uncorrectable Error First Failing Data [31:0] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	UE_FFD[31:0]	R	0	Data (bits [31:0]) of the first occurrence of an uncorrectable error.

UE_FFD[63:32]

This register stores the (uncorrected) failing data (bits [63:32]) of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next uncorrectable error. Storing of the failing data is enabled after reset.

Table 1-55: **Uncorrectable Error First Failing Data Register (UE_FFD[63:32])**

31	0
UE_FFD[63:32]	

Table 1-56: **Uncorrectable Error First Failing Data [63:32] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	UE_FFD [63:32]	R	0	Data (bits [63:32]) of the first occurrence of an uncorrectable error.

UE_FFD[95:64]

Note: This register is only used when the DQ_WIDTH == 144.

This register stores the (uncorrected) failing data (bits [95:64]) of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next uncorrectable error. Storing of the failing data is enabled after reset.

Table 1-57: **Uncorrectable Error First Failing Data Register (UE_FFD[95:64])**

31	0
UE_FFD[95:64]	

Table 1-58: **Uncorrectable Error First Failing Data [95:64] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	UE_FFD[95:64]	R	0	Data (bits [95:64]) of the first occurrence of an uncorrectable error.

UE_FFD[127:96]

Note: This register is only used when the DQ_WIDTH == 144.

This register stores the (uncorrected) failing data (bits [127:96]) of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the data of the next uncorrectable error. Storing of the failing data is enabled after reset.

Table 1-59: **Uncorrectable Error First Failing Data Register (UE_FFD[127:96])**

31	0
UE_FFD[127:96]	

Table 1-60: **Uncorrectable Error First Failing Data [127:96] Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	UE_FFD[127:96]	R	0	Data (bits [127:96]) of the first occurrence of an uncorrectable error.

UE_FFE

This register stores the ECC bits of the first occurrence of an access with an uncorrectable error. When the UE_STATUS bit in the ECC Status Register is cleared, this register is re-enabled to store the ECC of the next uncorrectable error. Storing of the failing ECC is enabled after reset.

Table 1-61 and Table 1-62 describe the register bit usage when DQ_WIDTH = 72.

Table 1-61: **Uncorrectable Error First Failing ECC Register (UE_FFE) for 72-Bit External Memory Width**

31	8	7	0
Reserved		UE_FFE	

Table 1-62: **Uncorrectable Error First Failing ECC Register Bit Definitions for 72-Bit External Memory Width**

Bit(s)	Name	Core Access	Reset Value	Description
[7:0]	UE_FFE	R	0	ECC (bits [7:0]) of the first occurrence of an uncorrectable error.

Table 1-63 and Table 1-64 describe the register bit usage when DQ_WIDTH = 144.

Table 1-63: **Uncorrectable Error First Failing ECC Register (UE_FFE) for 144-Bit External Memory Width**

31	9	8	0
Reserved		UE_FFE	

Table 1-64: **Uncorrectable Error First Failing ECC Register Bit Definitions for 144-Bit External Memory Width**

Bit(s)	Name	Core Access	Reset Value	Description
[8:0]	UE_FFE	R	0	ECC (bits [8:0]) of the first occurrence of an uncorrectable error.

FI_D0

This register is used to inject errors in data (bits [31:0]) written to memory and can be used to test the error correction and error signaling. The bits set in the register toggle the corresponding data bits (word 0 or bits [31:0]) of the subsequent data written to the memory without affecting the ECC bits written. After the fault has been injected, the Fault Injection Data Register is cleared automatically.

The register is only implemented if C_ECC_TEST = "ON" and C_ECC = "ON" in EDK or ECC_TEST_FI_XOR = "ON" and ECC = "ON" in a MIG design in the CORE Generator tool.

Injecting faults should be performed in a critical region in software; that is, writing this register and the subsequent write to the memory must not be interrupted.

Table 1-65: **Fault Injection Data Register (FI_D0)**

31	0
FI_D0	

Table 1-66: **Fault Injection Data (Word 0) Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	FI_D0	W	0	Bit positions set to 1 toggle the corresponding bits [31:0] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

Special consideration must be given across FI_D0, FI_D1, FI_D2, and FI_D3 such that only a single error condition is introduced.

FI_D1

This register is used to inject errors in data (bits [63:32]) written to memory and can be used to test the error correction and error signaling. The bits set in the register toggle the corresponding data bits (word 1 or bits [63:32]) of the subsequent data written to the memory without affecting the ECC bits written. After the fault has been injected, the Fault Injection Data Register is cleared automatically.

This register is only implemented if C_ECC_TEST = "ON" and C_ECC = "ON" in EDK or ECC_TEST_FI_XOR = "ON" and ECC = "ON" in a MIG design in the CORE Generator tool.

Injecting faults should be performed in a critical region in software; that is, writing this register and the subsequent write to the memory must not be interrupted.

Table 1-67: **Fault Injection Data Register (FI_D1)**

31	0
FI_D1	

Table 1-68: **Fault Injection Data (Word 1) Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	FI_D1	W	0	Bit positions set to 1 toggle the corresponding bits [63:32] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

FI_D2

Note: This register is only used when DQ_WIDTH =144.

This register is used to inject errors in data (bits [95:64]) written to memory and can be used to test the error correction and error signaling. The bits set in the register toggle the corresponding data bits (word 2 or bits [95:64]) of the subsequent data written to the memory without affecting the ECC bits written. After the fault has been injected, the Fault Injection Data Register is cleared automatically.

This register is only implemented if C_ECC_TEST = “ON” and C_ECC = “ON” in EDK or ECC_TEST_FI_XOR = “ON” and ECC = “ON” in a MIG design in the CORE Generator tool.

Injecting faults should be performed in a critical region in software; that is, writing this register and the subsequent write to the memory must not be interrupted.

Table 1-69: **Fault Injection Data Register (FI_D2)**

31	0
FI_D2	

Table 1-70: **Fault Injection Data (Word 2) Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	FI_D2	W	0	Bit positions set to 1 toggle the corresponding bits [95:64] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

Special consideration must be given across FI_D0, FI_D1, FI_D2, and FI_D3 such that only a single error condition is introduced.

FI_D3

Note: This register is only used when DQ_WIDTH =144.

This register is used to inject errors in data (bits [127:96]) written to memory and can be used to test the error correction and error signaling. The bits set in the register toggle the corresponding data bits (word 3 or bits [127:96]) of the subsequent data written to the

memory without affecting the ECC bits written. After the fault has been injected, the Fault Injection Data Register is cleared automatically.

The register is only implemented if C_ECC_TEST = "ON" and C_ECC = "ON" in EDK or ECC_TEST_FI_XOR = "ON" and ECC = "ON" in a MIG design in the CORE Generator tool.

Injecting faults should be performed in a critical region in software; that is, writing this register and the subsequent write to the memory must not be interrupted.

Table 1-71: **Fault Injection Data Register (FI_D3)**

31	0
FI_D3	

Table 1-72: **Fault Injection Data (Word 3) Register Bit Definitions**

Bit(s)	Name	Core Access	Reset Value	Description
[31:0]	FI_D3	W	0	Bit positions set to 1 toggle the corresponding bits [127:96] of the next data word written to the memory. The register is automatically cleared after the fault has been injected.

FI_ECC

This register is used to inject errors in the generated ECC written to the memory and can be used to test the error correction and error signaling. The bits set in the register toggle the corresponding ECC bits of the next data written to memory. After the fault has been injected, the Fault Injection ECC Register is cleared automatically.

The register is only implemented if C_ECC_TEST = "ON" and C_ECC = "ON" in EDK or ECC_TEST_FI_XOR = "ON" and ECC = "ON" in a MIG design in the CORE Generator tool.

Injecting faults should be performed in a critical region in software; that is, writing this register and the subsequent write to memory must not be interrupted.

Table 1-73 and Table 1-74 describe the register bit usage when DQ_WIDTH = 72.

Table 1-73: **Fault Injection ECC Register (FI_ECC) for 72-Bit External Memory Width**

31	8	7	0
Reserved		FI_ECC	

Table 1-74: **Fault Injection ECC Register Bit Definitions for 72-Bit External Memory Width**

Bit(s)	Name	Core Access	Reset Value	Description
[7:0]	FI_ECC	R	0	Bit positions set to 1 toggle the corresponding bit of the next ECC written to the memory. The register is automatically cleared after the fault has been injected.

Table 1-75 and Table 1-76 describe the register bit usage when DQ_WIDTH = 144.

Table 1-75: **Fault Injection ECC Register (FI_ECC) for 144-Bit External Memory Width**

31 … 9	8 … 0
Reserved	FI_ECC

Table 1-76: **Fault Injection ECC Register Bit Definitions for 144-Bit External Memory Width**

Bit(s)	Name	Core Access	Reset Value	Description
8:0	FI_ECC	R	0	Bit positions set to 1 toggle the corresponding bit of the next ECC written to the memory. The register is automatically cleared after the fault has been injected.

User Interface Block

The UI block presents the UI to a user design. It provides a simple alternative to the native interface. The UI block:

- Buffers read and write data
- Reorders read return data to match the request order
- Presents a flat address space and translates it to the addressing required by the SDRAM

Native Interface

The native interface connects to an FPGA user design to allow access to an external memory device.

Command Request Signals

The native interface provides a set of signals that request a read or write command from the memory controller to the memory device. These signals are summarized in Table 1-77.

Table 1-77: **Native Interface Command Signals**

Signal	Direction	Description
accept	Output	This output indicates that the memory interface accepts the request driven on the last cycle.
bank[2:0]	Input	This input selects the bank for the current request.
bank_mach_next[]	Output	This output is reserved and should be left unconnected.
cmd[2:0]	Input	This input selects the command for the current request.
col[COL_WIDTH – 1:0]	Input	This input selects the column address for the current request.

Table 1-77: **Native Interface Command Signals *(Cont'd)***

Signal	Direction	Description
data_buf_addr[7:0]	Input	This input indicates the data buffer address where the memory controller: • Locates data while processing write commands. • Places data while processing read commands.
hi_priority	Input	This input is reserved and should be connected to logic 0.
rank[]	Input	This input is reserved and should be connected to logic 0.
row[ROW_WIDTH – 1:0]	Input	This input selects the row address for the current request.
size	Input	This input is reserved and should be connected to logic 0.
use_addr	Input	The user design strobes this input to indicate that the request information driven on the previous state is valid.

The bank, row, and column comprise a target address on the memory device for read and write operations. Commands are specified using the cmd[2:0] input to the core. The available read and write commands are shown in Table 1-78.

Table 1-78: **Memory Interface Commands**

Operation	cmd[2:0] Code
Memory read	000
Memory write	001
Reserved	All other codes

accept

This signal indicates to the user design whether or not a request is accepted by the core. When the accept signal is asserted, the request submitted on the last cycle is accepted, and the user design can either continue to submit more requests or go idle. When the accept signal is deasserted, the request submitted on the last cycle was not accepted and must be retried.

use_addr

The user design asserts the use_addr signal to strobe the request that was submitted to the native interface on the previous cycle.

data_buf_addr

The user design must contain a buffer for data used during read and write commands. When a request is submitted to the native interface, the user design must designate a location in the buffer for when the request is processed. For write commands, data_buf_addr is an address in the buffer containing the source data to be written to the external memory. For read commands, data_buf_addr is an address in the buffer that

receives read data from the external memory. The core echoes this address back when the requests are processed.

Write Command Signals

The native interface has signals that are used when the memory controller is processing a write command (Table 1-79). These signals connect to the control, address, and data signals of a buffer in the user design.

Table 1-79: **Native Interface Write Command Signals**

Signal	Direction	Description
wr_data[(4 ´ DQ_WIDTH) – 1:0]	Input	This is the input data for write commands.
wr_data_addr[7:0]	Output	This output provides the base address for the source data buffer for write commands.
wr_data_be[(4 ´ DQ_WIDTH/8) – 1:0]	Input	This input provides the byte enable for the write data.
wr_data_en	Output	This output indicates that the memory interface is reading data from a data buffer for a write command.
wr_data_offset[0:0]	Output	This output provides the offset for the source data buffer for write commands.

wr_data

This bus is the data that needs to be written to the external memory. This bus can be connected to the data output of a buffer in the user design.

wr_data_addr

This bus is an echo of data_buf_addr when the current write request is submitted. The wr_data_addr bus can be combined with the wr_data_offset signal and applied to the address input of a buffer in the user design.

wr_data_be

This bus is the byte enable (data mask) for the data currently being written to the external memory. The byte to the memory is written when the corresponding wr_data_be signal is deasserted.

wr_data_en

When asserted, this signal indicates that the core is reading data from the user design for a write command. This signal can be tied to the chip select of a buffer in the user design.

wr_data_offset

This bus is used to step through the data buffer when the burst length requires more than a single cycle to complete. This bus, in combination with rd_data_addr, can be applied to the address input of a buffer in the user design.

Read Command Signals

The native interface provides a set of signals used when the memory controller is processing a read command (Table 1-80). These signals are similar to those for processing write commands, except that they transfer data from the memory device to a buffer in the user design.

Table 1-80: **Native Interface Read Command Signals**

Signal	Direction	Description
rd_data[(4 ´ DQ_WIDTH) – 1:0]	Output	This is the output data from read commands.
rd_data_addr[7:0]	Output	This output provides the base address of the destination buffer for read commands.
rd_data_en	Output	This output indicates that valid read data is available on the rd_data bus.
rd_data_offset[0:0]	Output	This output provides the offset for the destination buffer for read commands.

rd_data

This bus is the data that was read from the external memory. It can be connected to the data input of a buffer in the user design.

rd_data_addr

This bus is an echo of data_buf_addr when the current read request is submitted. This bus can be combined with the rd_data_offset signal and applied to the address input of a buffer in the user design.

rd_data_en

This signal indicates when valid read data is available on rd_data for a read request. It can be tied to the chip select and write enable of a buffer in the user design.

rd_data_offset

This bus is used to step through the data buffer when the burst length requires more than a single cycle to complete. This bus can be combined with rd_data_addr and applied to the address input of a buffer in the user design.

Memory Controller

In the core's default configuration, the memory controller (MC) resides between the UI block and the physical layer. This is depicted in Figure 1-46.

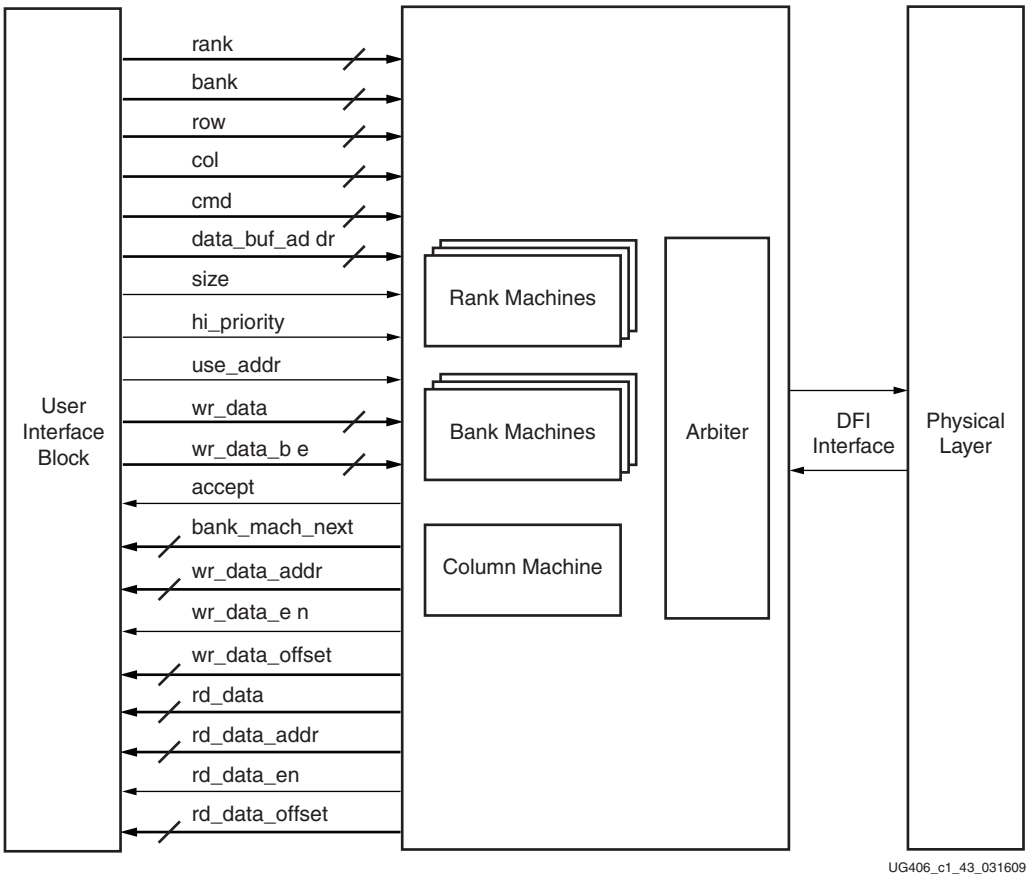


Figure 1-46: **Memory Controller**

The memory controller is the primary logic block of the memory interface. The memory controller receives requests from the UI and stores them in a logical queue. Requests are optionally reordered to optimize system throughput and latency.

The MC block is organized as four main pieces:

- A configurable number of "bank machines"
- A configurable number of "rank machines"
- A column machine
- An arbitration block

Bank Machines

Most of the MC logic resides in the bank machines. Bank machines correspond to DRAM banks. A given bank machine manages a single DRAM bank at any given time. However, bank machine assignment is dynamic, so it is not necessary to have a bank machine for each physical bank. The number of banks can be configured to trade off between area and performance.

When a request is accepted, it is assigned to a bank machine. When a request is complete, the bank machine is released and is made available for assignment to another request. Bank machines issue all the commands necessary to complete the request.

On behalf of the current request, a bank machine must generate row commands and column commands to complete the request. Row and column commands are independent but must adhere to DRAM timing requirements. Column commands can be reordered for the purpose of optimizing memory interface throughput. The ordering algorithm nominally ensures data coherence.

Rank Machines

The rank machines correspond to DRAM ranks. Rank machines monitor the activity of the bank machines and track rank or device-specific timing parameters. For example, a rank machine monitors the number of activate commands sent to a rank within a time window. After the allowed number of activates have been sent, the rank machine generates an inhibit signal that prevents the bank machines from sending any further activates to the rank until the time window has shifted enough to allow more activates. Rank machines are statically assigned to a physical DRAM rank.

Column Machine

The single column machine generates the timing information necessary to manage the DQ data bus. Although there can be multiple DRAM ranks, because there is a single DQ bus, all the columns in all DRAM ranks are managed as a single unit. The column machine monitors commands issued by the bank machines and generates inhibit signals back to the bank machines so that the DQ bus is utilized in an orderly manner.

Arbitration Block

The arbitration block receives requests to send commands to the DRAM array from the bank machines. Row commands and column commands are arbitrated independently. For each command opportunity, the arbiter block selects a row and a column command to forward to the physical layer. The arbitration block implements a round-robin protocol to ensure forward progress.

Reordering

DRAM accesses are broken into two quasi-independent parts, row commands and column commands. Each request occupies a logical queue entry, and each queue entry has an associated bank machine. These bank machines track the state of the DRAM rank or bank it is currently bound to, if any.

If necessary, the bank machine attempts to activate the proper rank, bank, or row on behalf of the current request. In the process of doing so, the bank machine looks at the current state of the DRAM to decide if various timing parameters are met. Eventually, all timing parameters are met and the bank machine arbitrates to send the activate. The arbitration is done in a simple round-robin manner. Arbitration is necessary because several bank machines might request to send row commands (activate and precharge) at the same time.

Not all requests require an activate. If a preceding request has activated the same rank, bank, or row, a subsequent request might inherit the bank machine state and avoid the precharge/activate penalties.

After the necessary rank, bank, or row is activated and the RAS to CAS delay timing is met, the bank machine tries to issue the CAS-READ or CAS-WRITE command. Unlike the row command, all requests issue a CAS command. Before arbitrating to send a CAS command,

the bank machine must look at the state of the DRAM, the state of the DQ bus, priority, and ordering. Eventually, all these factors assume their favorable states and the bank machine arbitrates to send a CAS command. In a manner similar to row commands, a round-robin arbiter uses a priority scheme and selects the next column command.

The round-robin arbiter itself is a source of reordering. Assume for example that an otherwise idle MC receives a burst of new requests while processing a refresh. These requests queue up and wait for the refresh to complete. After the DRAM is ready to receive a new activate, all waiting requests assert their arbitration requests simultaneously. The arbiter selects the next activate to send based solely on its round-robin algorithm, independent of request order. Similar behavior can be observed for column commands.

Error Correcting Code (ECC)

The memory controller optionally implements an *error correcting code*. This code protects the contents of the DRAM array from corruption. A Single Error Correct Double Error Detect (SECDED) code is used. All single errors are detected and corrected. All two-bit errors are detected. Errors of more than two bits might or might not be detected.

The block diagram of ECC is as shown in Figure 1-47. These blocks are instantiated in the memory controller (`mc.v`) module.

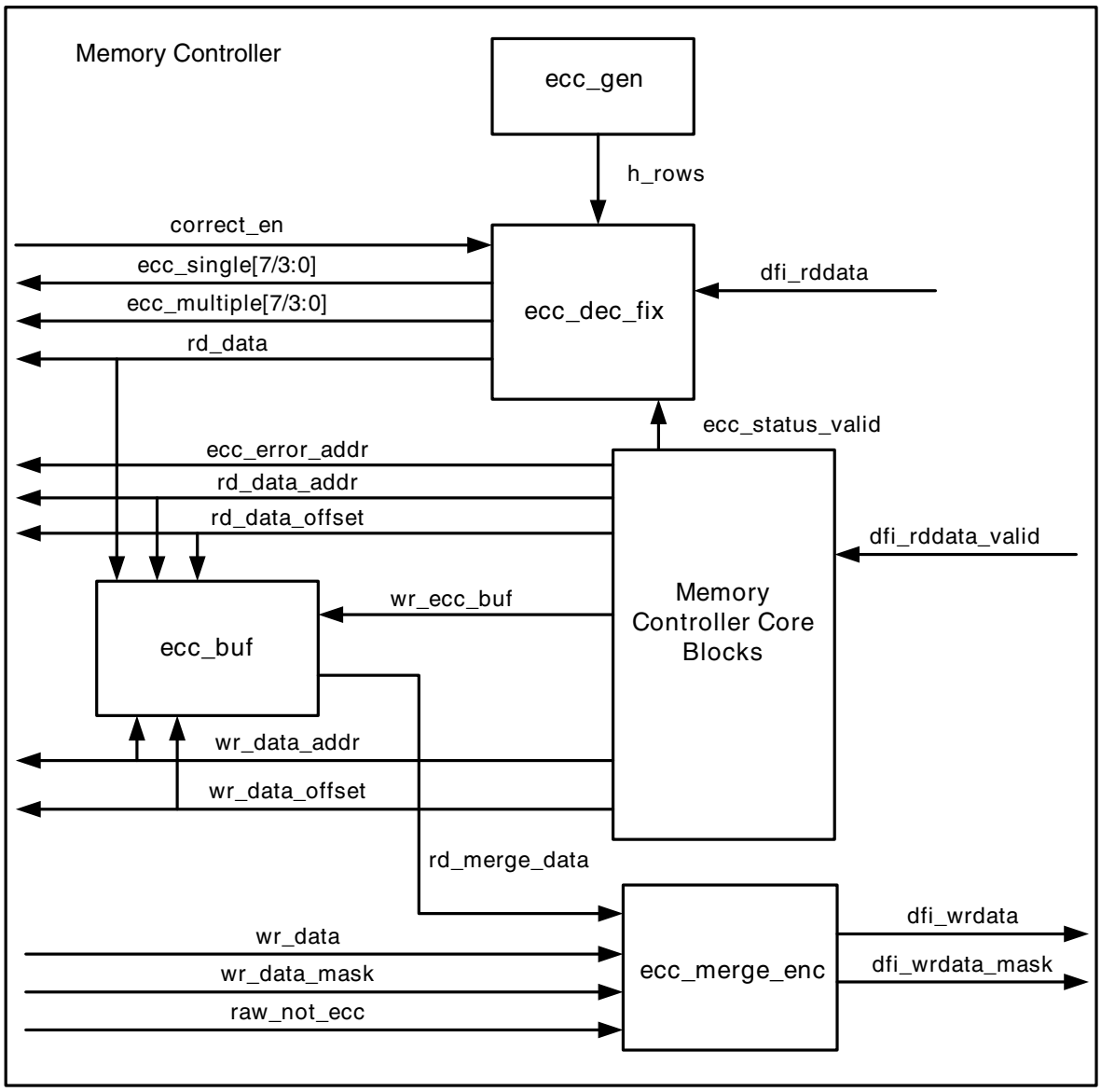


Figure 1-47: **ECC Block Diagram**

The ECC mode is optional and supported only for a data width of 72 bits. The data mask feature is disabled when ECC mode is enabled. When ECC mode is enabled, the entire DQ width is always written. The DRAM DM bits cannot be used because the ECC operates over the entire DQ data width.

The ECC functionality is implemented as three functional blocks: write data merge and ECC generate block, a read data ECC decode and correct block, and a data buffer block for temporarily holding the read data for read-modify-write cycles. A fourth block generates the ECC H matrix and passes these matrixes to the ECC generate and correct blocks.

For full burst write commands, data fetched from the write data buffer traverses the ECC merge and generate block. This block computes the ECC bits and appends them to the data. The ECC generate step is given one CLK state. Thus the data must be fetched from the write data buffer one state earlier relative to the write command compared to when

ECC is not enabled. At the user interface level, data must be written into the write data buffer no later than one state after the command is written into the command buffer. Other than the earlier data requirement, ECC imposes no other performance loss for writes.

For read cycles, all data traverses the ECC decode fix (ecc_dec_fix) block. This process starts when the PHY indicates read data availability on the dfi_rddata_valid signal. The decode fix process is divided into two CLK states. In the first state, the syndromes are computed. In the second state, the syndromes are decoded and any indicated bit flips (corrections) are performed. Also in the second state, the ecc_single and ecc_multiple indications are computed based on the syndrome bits and the ecc_status_valid timing signal generated by the memory controller core logic. The core logic also provides an ecc_err_addr bus. This bus contains the address of the current read command. Error locations can be logged by looking at the ecc_single, ecc_multiple, and ecc_err_addr buses. ECC imposes a two-state latency penalty for read requests.

Read-Modify-Write

Any writes of less than the full DRAM burst must be performed as a read-modify-write cycle. The specified location must be read, corrections (if any) performed, merged with the write data, ECC computed, and then written back to the DRAM array. The *wr_bytes* command is defined for ECC operation. When the wr_bytes command is issued, the memory controller always performs a read-modify-write cycle instead of a simple write cycle. The byte enables must always be valid, even for simple commands. Specifically, all byte enables must be asserted for all wr commands when ECC mode is enabled. Table 1-81 shows the available commands when ECC mode is enabled.

Table 1-81: **Commands for app_cmd[2:0]**

Operation	app_cmd[2:0] code
Write	000
Read	001
Write Bytes	011

When the wr_bytes command is issued, the memory controller performs a read-modify-write (RMW) cycle. When a wr_bytes command is at the head of the queue, it first issues a read. But unlike a normal read command, the request remains in the queue. A bit is set in the read response queue indicating this is a RMW cycle. When the read data is returned for this read command, app_rd_data_valid is not asserted. Instead, the ECC is decoded, corrections (if any) are made, and the data is written into the ECC data buffer. Meanwhile, the original wr_bytes command is examining all read returns. Based on the data_buf_addr stored in the read return queue, the wr_bytes request can determine when its read data is available in the ECC data buffer. At this point, the wr_bytes request starts arbitrating to send the write command. When the command is granted, data is fetched from the write data buffer and the ECC data buffer, merged as directed by the byte enables, ECC is computed, and data is written to the DRAM. The wr_bytes command has significantly lower performance than normal write commands. In the best case, each wr_bytes command requires a DRAM read cycle and a DRAM write cycle instead of simple DRAM write cycle. Read-to-write and write-to-read turnaround penalties further degrade throughput.

The memory controller can buffer up to nBANK_MACHS wr_bytes commands. As long as these commands do not conflict on a rank-bank, the memory controller strings together the reads and then the writes, thereby avoiding much of the read-to-write and write-to-read turnaround penalties. However, if the stream of wr_bytes commands is to a single

rank-bank, each RMW cycle is completely serialized and throughput is significantly degraded. If performance is important, it is best to avoid the wr_bytes command.

Table 1-82 provides the details of ECC ports at the user interface.

Table 1-82: **User Interface for ECC Operation**

Signal	Direction	Description
app_correct_en_i	Input	When asserted, this active-High signal corrects single bit data errors. This input is valid only when ECC mode is enabled.
app_ecc_multiple_err[7:0]	Output	This signal is applicable when ECC is enabled. It is valid along with app_rd_data_valid. The app_ecc_multiple_err signal is non-zero if the read data from the external memory has two bit errors per beat of the read burst. The SECDED algorithm does not correct the corresponding read data and puts a non-zero value on this signal to notify the corrupted read data at the UI. This signal is 4 bits wide when 2:1 mode is selected.
app_raw_not_ecc_i[7:0]	Input	This signal is applicable when ECC_TEST is enabled ("ON"). It is valid along with app_rd_data_valid. This signal is asserted to control the individual blocks to be written with raw data in the ECC bits. This signal is 4 bits wide in 2:1 mode.

ECC Self-Test Functionality

Under normal operating conditions, the ECC part of the data written to the DRAM array is not visible at the user interface. This can be problematic for the purposes of system self-test because there is no way to test the bits in the DRAM array corresponding to the ECC bits. Also errors cannot be sent to test the ECC generation and correction logic.

Controlled by the top-level parameter ECC_TEST, a DRAM array test mode can be generated. When the ECC_TEST parameter is "ON", the entire width of the DQ data bus is extended through the read and write buffers in the user interface. When ECC_TEST is "ON", the ECC correct enable is deasserted.

To write arbitrary data into both the data and ECC parts of the DRAM array, write the desired data into the extended width write data FIFO, and assert the corresponding app_raw_not_ecc_i bit with the data. The width of app_raw_not_ecc_i is 7 bits (4 bits in 2:1 mode), allowing individual ECC blocks to be written with raw data in the ECC bits, or the normal computed ECC bits. In this way, any arbitrary pattern can be written into the DRAM array.

In the read interface, the extended data simply appears with the normal data. However, the corrector might be trying to "correct" the read data. This is probably not desired during array pattern test and hence app_correct_en_i should be set to zero to disable correction.

With the above two features, an array pattern test can be achieved. ECC generation logic can be tested by writing data patterns but not asserting app_raw_not_ecc_i and

deasserting app_correct_en_i. The data along with the computed ECC bits can be read out and compared. ECC decode correct logic can be tested by asserting app_correct_en_i and writing the desired raw pattern as described above. When the data is read back, the operation of decode correct can be observed.

PHY

The PHY provides a physical interface to an external DDR2 or DDR3 SDRAM. The PHY generates the signal timing and sequencing required to interface to the memory device. It contains the clock-, address-, and control-generation logic, write and read datapaths, and state logic for initializing the SDRAM after power-up. In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

The PHY is provided as a single HDL codebase for both DDR2 and DDR3 SDRAMs. The MIG tool customizes the SDRAM type and numerous other design-specific parameters through top-level HDL parameters and constraints contained in a user constraints file (UCF).

Overall PHY Architecture

A block diagram of the PHY design is shown in Figure 1-48.

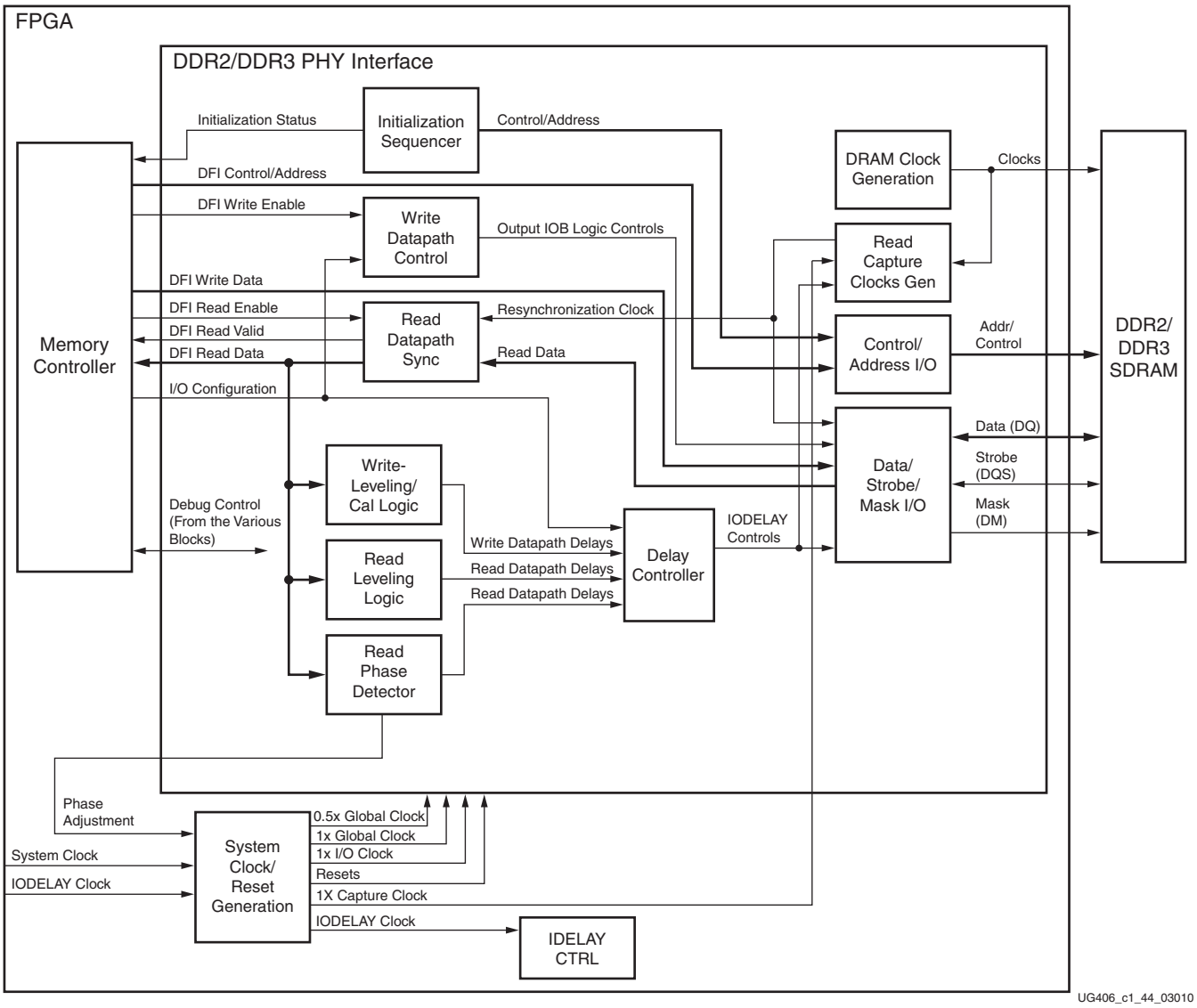


Figure 1-48: **PHY Block Diagram**

Initialization Sequence

After deassertion of system reset, the PHY performs the required power-on initialization sequence for the memory. This is followed by several stages of timing calibration for both the write and read datapaths. After calibration is complete, the PHY indicates that initialization is finished, and the controller can begin issuing commands to the memory. Figure 1-49 shows the initialization sequence.

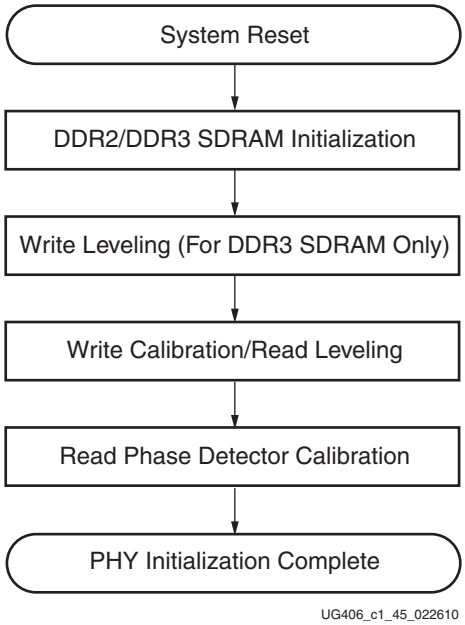


Figure 1-49: **PHY Overall Initialization Sequence**

Clocking Architecture

The PHY design requires that MMCM modules are used to generate various clocks, and both global and local clock networks are used to distribute the clock throughout the design.

The clock generation and distribution circuitry and networks drive blocks within the PHY that can be divided roughly into four separate, general functions:

- Internal (FPGA) logic
- Write-path (output) I/O logic
- Read-path (input) and delay I/O logic, and read data resynchronization logic in the FPGA logic
- IODELAY reference clock (either 200 MHz or 300 MHz)

One MMCM is required for the PHY. In terms of general functionality, the MMCM is used to generate the clocks for most of the internal logic, write-path I/O logic, and the clocks required for read data capture and resynchronization. The clocking architecture, with the exception of the IODELAY reference clock, is shown in Figure 1-50.

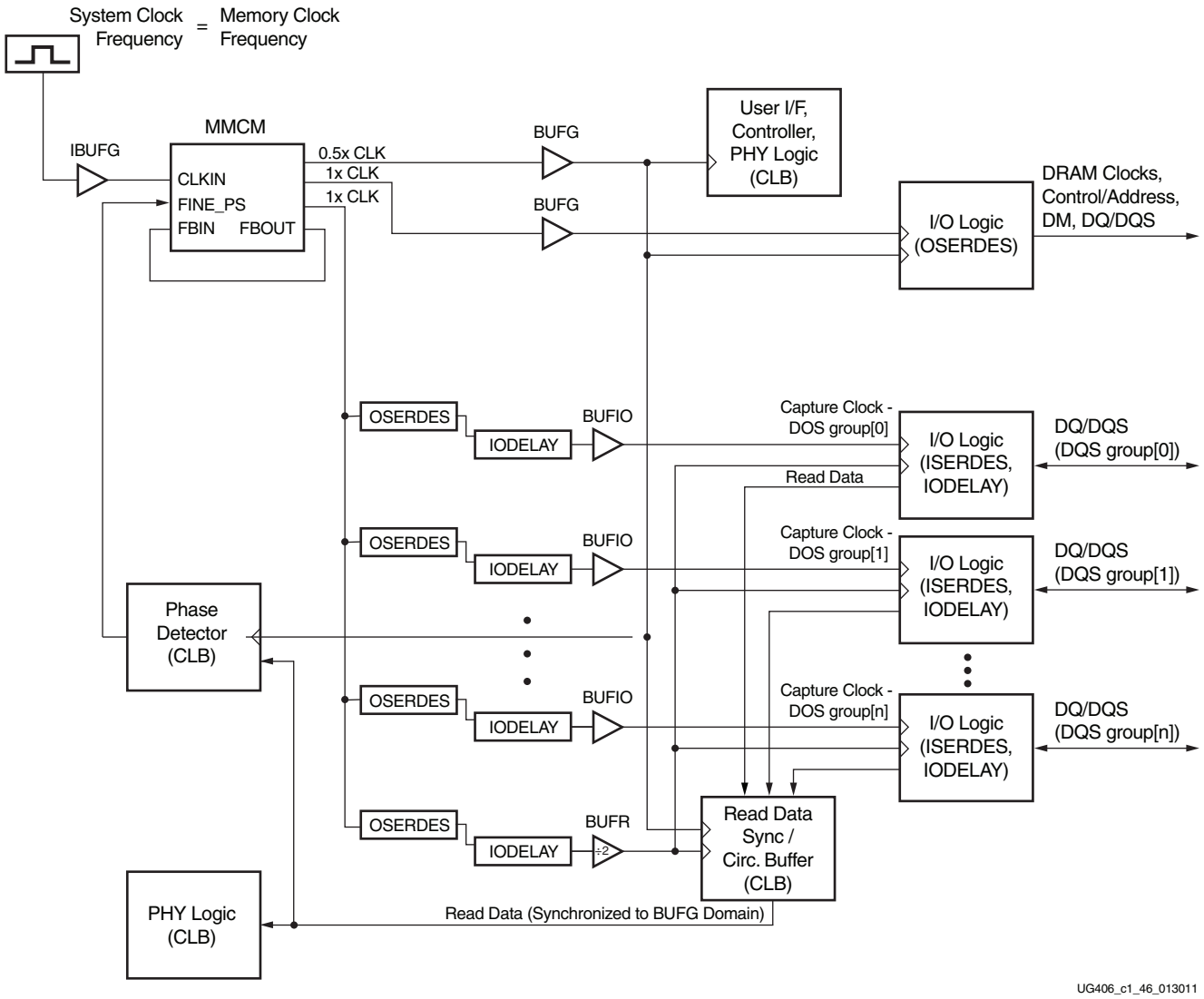


Figure 1-50: **PHY Clocking Architecture**

The MIG tool generates a design with the appropriate MMCM parameters based on the MMCM input clock operating at the same frequency as the memory interface. Users who are using a different ratio of MMCM input clock to memory interface clock must modify these parameters appropriately in the `example_top.v/.vhd` module. These parameters are in turn passed to the MMCM instantiated in `infrastructure.v/.vhd`:

- CLKFBOUT_MULT_F
- DIVCLK_DIVIDE
- CLKOUT_DIVIDE

When modifying these parameters, the user should refer to both the MMCM Switching Characteristics section of the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* (DS152) and the Mixed-Mode Clock Manager section of the *Virtex-6 FPGA Clock Resources User Guide* (UG362) to determine the appropriate parameter values.

A 200 MHz or 300 MHz IODELAY clock (depending on the desired IODELAY tap resolution) must be supplied to the IDELAYCTRL module. The IDELAYCTRL module

continuously calibrates the IODELAY elements in the I/O region to account for varying environmental conditions. The current core assumes an external clock signal is driving the IDELAYCTRL module. If a PLL clock drives the IDELAYCTRL input clock, the PLL lock signal needs to be incorporated in the rst_tmp_idelay signal inside the `IODELAY_CTRL.v/vhd` module. This ensures that the clock is stable before being used.

I/O Architecture

Circuitry to support both write and read leveling requirements for DDR2 and DDR3 SDRAM exist within the logic for each I/O block, and new features have been added to each of the three elements within each I/O block (ISERDES, OSERDES, and IODELAY). The block diagram for the I/O logic and dedicated routing associated with a bidirectional data (DQ) pin is shown in Figure 1-51.

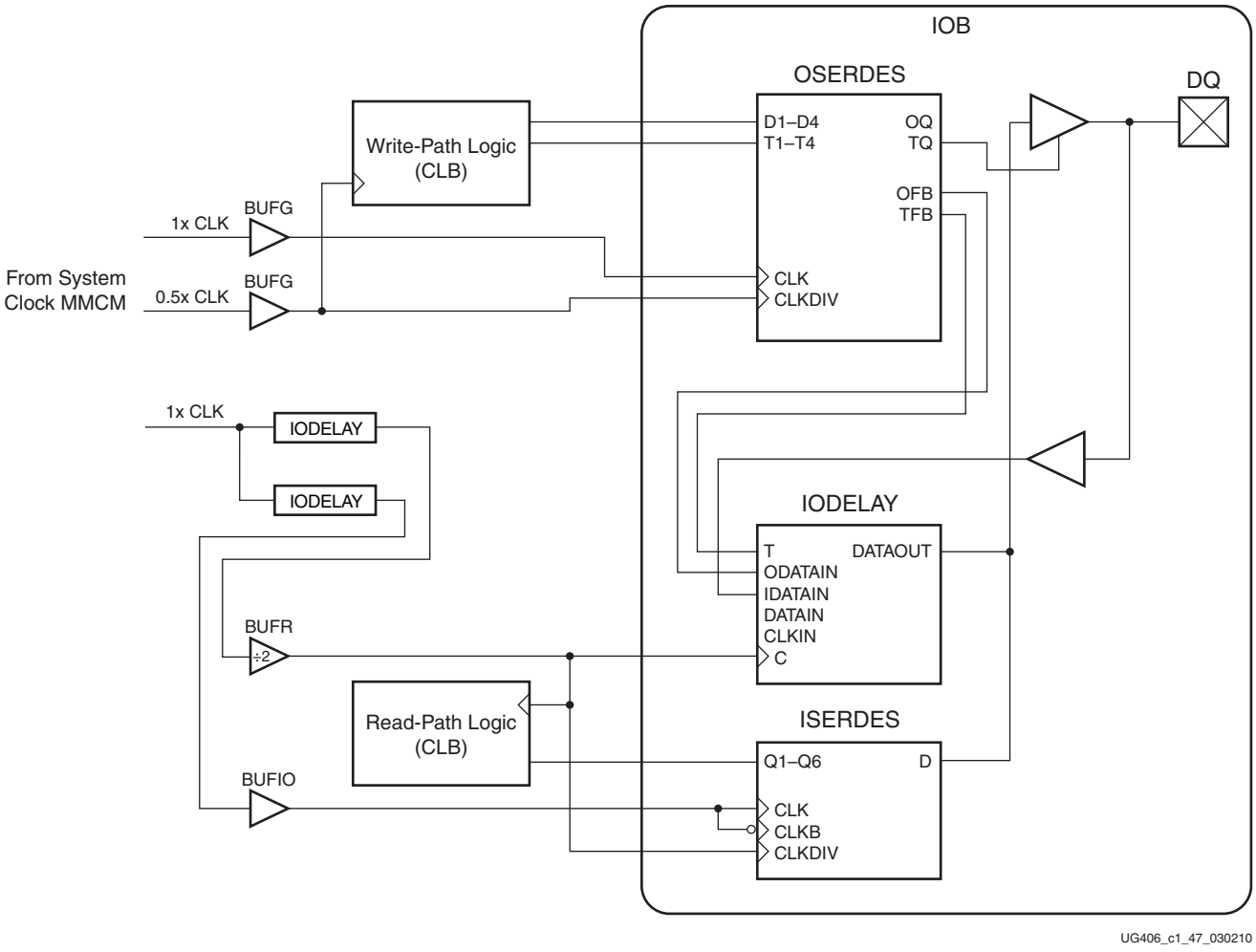


Figure 1-51: **DQ I/O Block Diagram**

Memory Initialization

The PHY executes a JEDEC-compliant DDR2 or DDR3 initialization sequence for memory following deassertion of system reset. Each DDR2 or DDR3 SDRAM has a series of mode registers, accessed via mode register set (MRS) commands. These mode registers determine various SDRAM behaviors, such as burst length, read and write CAS latency,

and additive latency. The particular bit values programmed into these registers are configurable in the PHY and determined by the values of top-level HDL parameters.

Write Datapath

Write Leveling

DDR3 SDRAM modules have adopted fly-by topology on clocks, address, commands, and control signals to improve signal integrity. Specifically, the clocks, address, and control signals are all routed in a daisy-chained fashion, and termination is located at the end of each trace. However, this causes a skew between the strobe (DQS) and the clock (CK) at each memory device on the module. A new feature in DDR3 SDRAMs, write leveling allows the controller to adjust each write DQS phase independently with respect to the CK forwarded to the DDR3 SDRAM device. This compensates for the skew between DQS and CK and meets the tDQSS specification. During write leveling, DQS is driven by the FPGA memory interface and DQ is driven by the DDR3 SDRAM device to provide feedback. The FPGA memory interface should have the capability to delay DQS until a 0-to-1 transition is detected on DQ. Write leveling is performed once after power-up. The DQS delay can be achieved with IODELAY in the Virtex-6 FPGA.

The write leveling block diagram is shown in Figure 1-52. The DQS driven by the FPGA memory interface is used to clock a flip-flop with CK as its input. The output of this flip-flop is provided as a feedback to the FPGA memory interface via DQ. If the level on DQ is 0, the DQS is delayed until a 1 is detected on DQ.

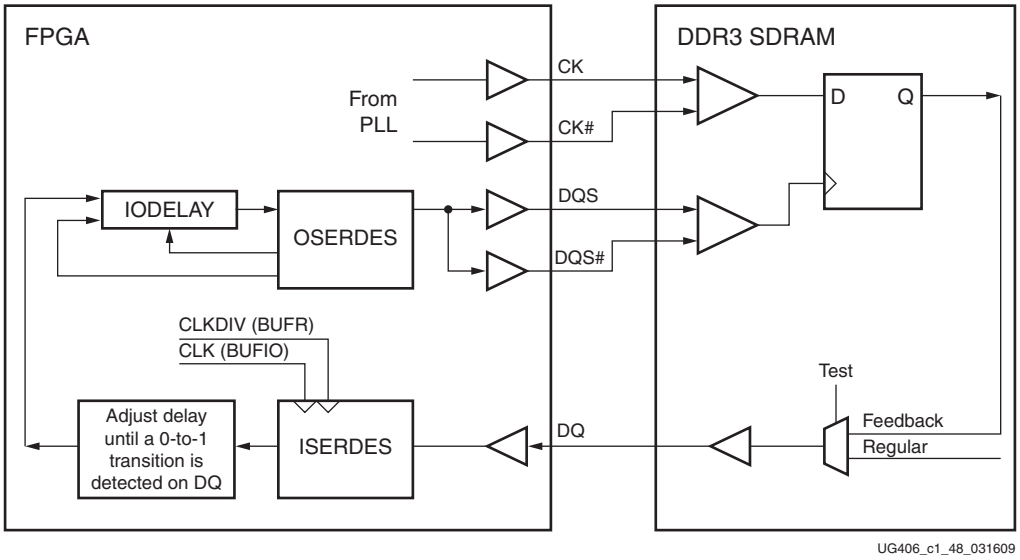


Figure 1-52: **Write Leveling Block Diagram**

The write leveling timing diagram is shown in Figure 1-53. A DQS pulse is output by the FPGA memory interface to detect the level of CK at the DDR3 SDRAM. The interval between DQS pulses is specified as a minimum of 16 clock cycles. DQS is delayed using the IODELAY in unit tap increments until a 0-to-1 transition is detected on the feedback DQ input. The DQS delay established by write leveling ensures the tDQSS specification. The write leveling algorithm used in this memory interface design limits the number of IODELAY taps required to delay DQS to half a CK period by inverting DQS when required, as shown in Figure 1-54.

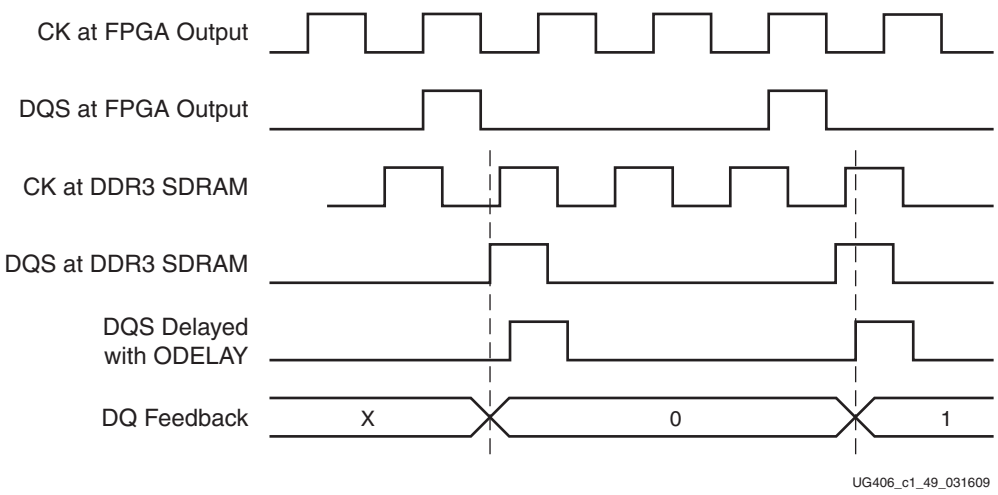


Figure 1-53: **Write Leveling Timing Diagram**

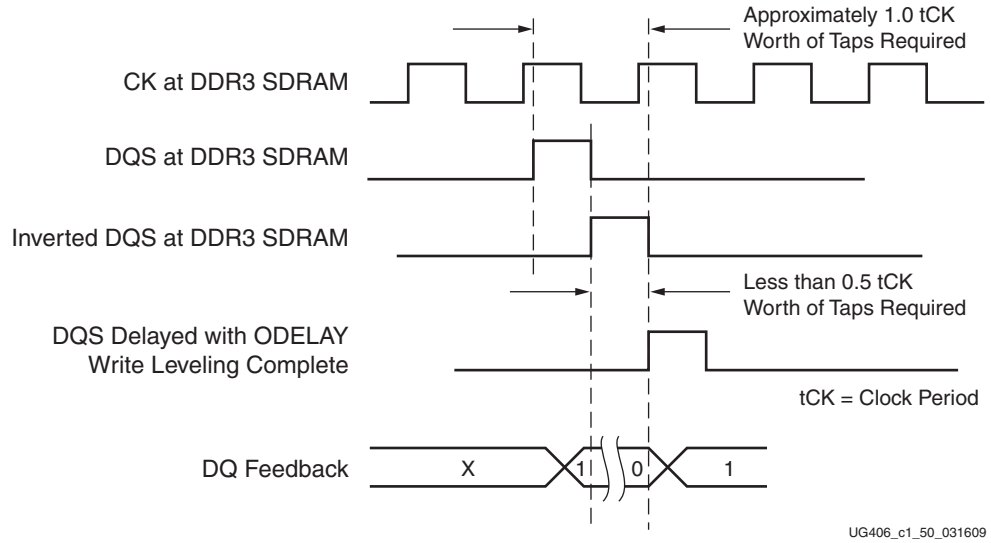


Figure 1-54: **Write Leveling Algorithm**

Write Calibration

Write calibration is required to align DQS and its associated DQ bits to the correct CK cycle to compensate for PCB trace delays and I/O buffer delays that exceed a CK cycle. This calibration requires a data pattern to be written and then read back to verify that the desired pattern was written to memory correctly. During write calibration, the appropriate clock cycles of delay are added until the desired data pattern is read back. This calibration is performed on a per-byte basis. As in the case of write leveling, this sequence is only executed for DDR3 SDRAM when fly-by routing topologies are used.

Read Datapath

The read datapath ensures that both DQ data and the DQS strobe during a read from the DDR2 or DDR3 SDRAM are reliably captured and transferred to the system clock domain. Data transfer takes place across several clock domains as it is moved through the ISERDES to the FPGA system clock domain.

The various clocks required for each stage of capture are generated by a single MMCM. The MMCM generates both the clocks used to capture the data directly and the clock used to move the data from the ISERDES into the CLB fabric. These clocks are routed to IODELAY elements, which then drive BUFIO and BUFR local clock buffers. The IODELAY elements of the capture clocks allow each of these clocks to be adjusted individually to provide for reliable capture of the read data from memory.

The DQS from the memory is not directly used to capture the corresponding read data. Read data is captured using an internally generated capture clock. However, the phase of DQS is monitored during reads and compared to the capture clock. As their phases vary with changing environmental conditions, the capture clock phase is adjusted.

The various stages (many of which are internal to the ISERDES block) are shown in Figure 1-55. Data transfer takes four stages:

- Stage 1: Data is initially captured in the ISERDES block of each DQ/DQS input/output block (IOB) using a local capture clock running at the same clock frequency as the memory clock. A separate capture clock is generated for each DQS group. The phase of this capture clock is adjusted using both the fine phase shift feature of the sourcing MMCM and an IODELAY to constantly position the rising and falling edges of the capture clock within the DRAM read data window.
- Stage 2: Data is then transferred within the ISERDES block to a divided-by-2 (half rate) version of the capture clock. Each ISERDES block generates its own version of this half-rate clock.
- Stage 3: Data is transferred from the half-rate capture clock domain to a half-rate clock driven by a BUFR regional clock buffer. A single BUFR is used for all the ISERDES in up to three I/O banks. Because a single BUFR is used to synchronize multiple capture clock domains, each of which can have a different phase, the DYNCLKDIVSEL input of the ISERDES block is used to selectively invert the BUFR clock within each ISERDES block to maximize the data transfer timing margin during this synchronization stage. The read leveling calibration logic determines the DYNCLKDIVSEL values on a per DQS-group basis, and ISERDES blocks from different DQS groups can have different DYNCLKDIVSEL values. When the data is on the BUFR clock domain, it can be transferred to the CLB fabric.
- Stage 4: The data is then sent through a multiplexing and pipelining stage within the fabric to align the captured data on a system clock boundary (bitslip), and to deskew DQS groups across the entire data word, by adding pipeline delay to DQS groups that arrive faster than others. This circuit is implemented within the CLB fabric and clocked by the BUFR. The output of this circuit is then synchronized to the system clock (BUFG) domain through a circular buffer implemented in the CLB fabric.

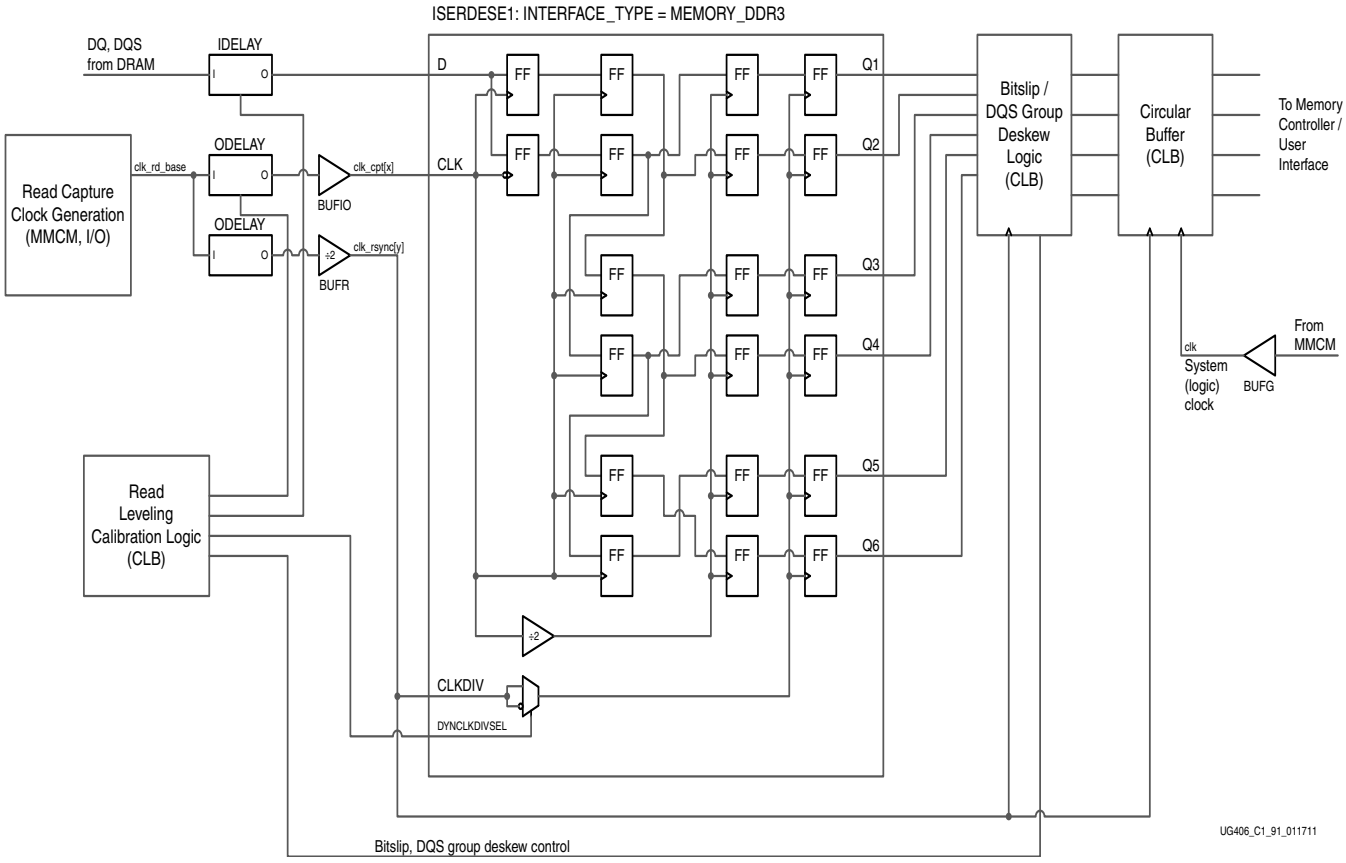


Figure 1-55: **Read Data Synchronization (Logic for Single DQS/DQ Bit Shown)**

Read Leveling

After system reset, memory initialization, and write leveling, the PHY executes a read timing training calibration sequence to ensure reliable synchronization of read data from the memory to the FPGA core clock domain. This calibration procedure adjusts the timing of each of the read data synchronization stages using the IODELAY elements to factor out static timing uncertainties, such as PCB-related trace delays, and process-dependent (static) propagation delays from the memory and the FPGA. Read-leveling consists of two stages during which different timing delays are adjusted. During both stages, a fixed pattern is written to memory and continuously read back.

During the first stage, the capture clock is adjusted individually for each DQS group to find the edges of the read data valid window (read data eye). After one or both edges have been found, the capture clock phase is adjusted so that the data capture at the ISERDES for bits in that DQS group takes place in the middle of the data eye.

Following the first stage of calibration and prior to the second, an adjustment takes place to increase the capture to resynchronzation clock data transfer timing margin. The PHY, on a per-DQS group basis, varies both the phase of the resynchronization clock and the polarity of the DYNCLKDIVSEL input of the DQ and DQS ISERDES to determine which polarity of DYNCLKDIVSEL results in the greater timing margin on the capture clock-to-resynchronization data transfer. At the conclusion of this adjustment, the resynchronization clock is restored to its original phase, and the optimal DYNCLKDIVSEL settings determined during this adjustment are retained.

During the second stage, word alignment logic associated with each bit and DQS group is adjusted to align the entire captured data word in the resynchronization clock domain. This accounts for such factors as skew between different DQS groups. This also dynamically determines the round-trip time between when a read command is issued to the PHY, and when the corresponding read data is returned to the memory controller.

Phase Detector

In the Virtex-6 FPGA memory interface design, read DQ is not sampled by the corresponding DQS signal. Instead, read DQ is sampled by a free-running clock operating at the same frequency as the differential SDRAM CK/CK# signals. The free-running clock has a single source for all DQ bits, but the phase of each byte capture clock output can be separately adjusted using IODELAY elements. The phase detector initially locks the phase of each byte-capture clock such that it is in phase with the corresponding DQS signal (Figure 1-56).

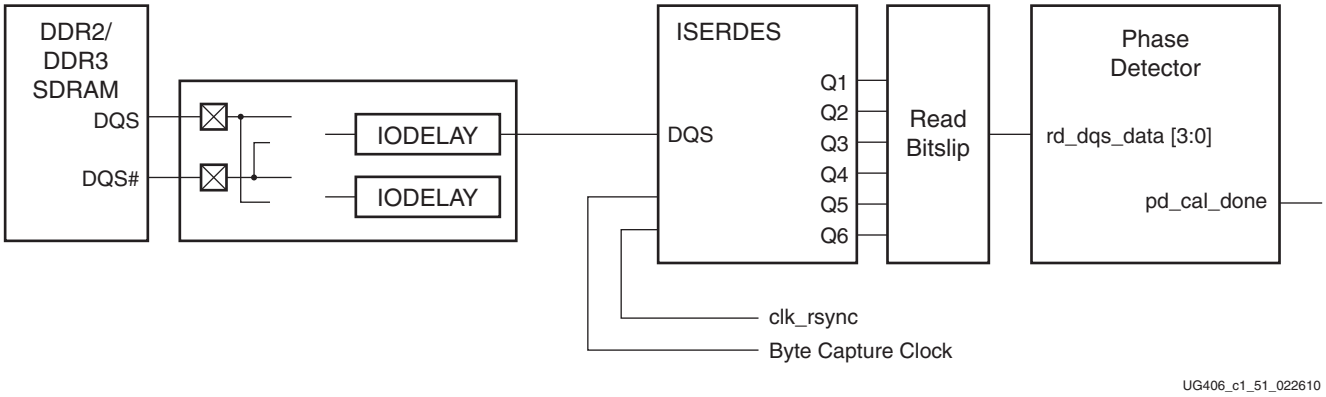


Figure 1-56: **Phase Detector Block Diagram**

Subsequent changes in capture timing delays after initial calibration due to voltage and temperature changes can be compensated for by maintaining the phase relationship between the byte-capture clock and the corresponding DQS. Periodic dummy reads are required from the memory controller to dynamically maintain phase lock between the byte-capture clock and DQS (Figure 1-57).

Periodic compensation can be accomplished by adjusting the phase of the MMCM-generated source capture clock using the fine-phase shift capability of the MMCM.

This method allows fine adjustment of the capture clocks of all bytes simultaneously but does not allow control over individual byte clock phase adjustment.

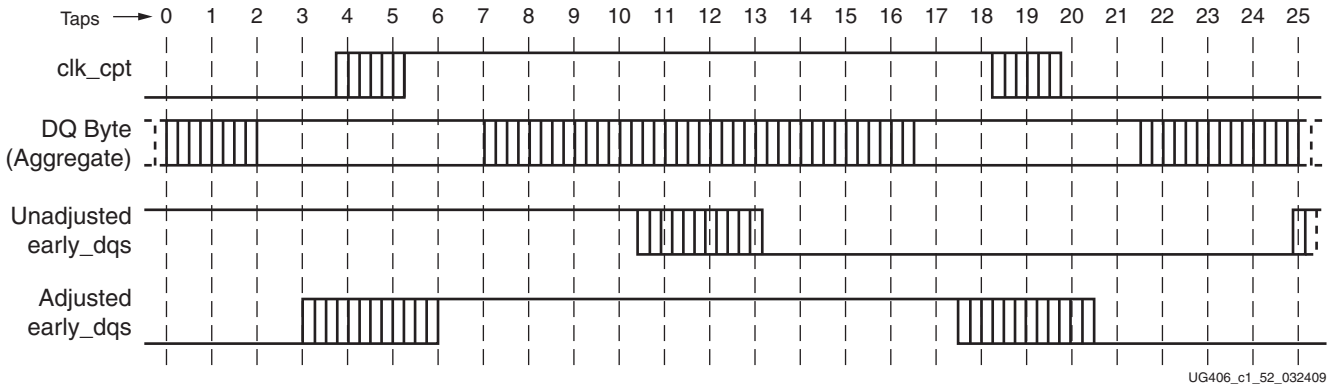


Figure 1-57: **Phase Detector Timing Diagram**

Physical Interface

The physical interface connects the FPGA to an external DDR2 or DDR3 SDRAM device. The I/O signals for this interface are shown in Table 1-83.

Table 1-83: **Physical Interface Signals**

Signal	Direction	Description
ddr_addr[ROW_WIDTH – 1:0]	Output	Address
ddr_ba[BANK_WIDTH – 1:0]	Output	Bank address
ddr_cas_n	Output	Command
ddr_ck_n[0:0]	Output	Inverted clock
ddr_ck_p[0:0]	Output	Clock
ddr_cke[0:0]	Output	Clock enable
ddr_cs_n[0:0]	Output	Chip select
ddr_dm[(DQ_WIDTH/8) – 1:0]	Output	Data mask
ddr_dq[DQ_WIDTH – 1:0]	Input/Output	Data
ddr_dqs_n[DQS_WIDTH – 1:0]	Input/Output	Data strobe
ddr_dqs_p[DQS_WIDTH – 1:0]	Input/Output	Data strobe
ddr_odt[0:0]	Output	On-die termination (ODT)
ddr_ras_n	Output	Command
ddr_reset_n	Output	Reset
ddr_we_n	Output	Command
SDA (1)	Input	Serial data for I^2C interface to SPD EEPROM
SCL (1)	Output	Serial clock for I^2C interface to SPD EEPROM

Notes:

1. These pins are not used in the current memory interface designs. Refer to Configuration, page 139 for more information.

These signals can be connected to the corresponding signals on the memory device.

Designing with the Core

The core is bundled with an example design that can be simulated. The example design can be used as a starting point for the user design or as a reference for debugging purposes.

Only supported modifications should be made to the configuration of the core. See Customizing the Core, page 128 for supported configuration parameters.

Interfacing to the Core

The memory controller can be connected using either the AXI4 slave interface, the UI, or the native interface. The AXI4 slave interface provides a full AXI4 memory-mapped compliant slave ideal for connecting to processor subsystems. The AXI4 slave interface converts its transactions to pass them over the UI. The UI resembles a simple FIFO interface and always returns the data in order. The native interface offers higher performance in some situations, but is more challenging to use.

The native interface contains no buffers and returns data as soon as possible, but the return data might be out of order. The application must reorder the received data internally if the native interface is used and reordering is enabled. The following sections describe timing protocols of each interface and how they should be controlled.

AXI4 Slave Interface

The AXI4 slave interface follows the AXI4 memory-mapped slave protocol specification as described in the ARM AMBA open specifications. Refer to this specification [Ref 1] for the signaling details of the AXI4 slave interface.

AXI Addressing

The AXI address from the AXI master is a true byte address. The address at the user interface of the memory controller is normalized to the data width of the external memory. The AXI shim normalizes the address from the AXI master to the memory data width based on the AXSIZE parameter. The address at the input of the memory controller user interface from the AXI shim can be configured in two modes. See the User Interface section for more details. The Bank-Row-Column or the Row-Bank-Column addressing mode explained in User Interface is applied to the normalized address from the AXI shim.

The normalization process remaps the address based on the data width in the AXI shim. Table 1-84 shows the first step in normalization done based on user interface data widths. The value of the user interface data width is equal to 2 * nCK_PER_CLK * PAYLOAD_WIDTH. PAYLOAD_WIDTH is a parameter in the user design top which specifies the width of DQ bus used for the user data. See Table 1-85 for more information on parameters. The shift operation is performed to define the ratio between DRAM width and the User Interface (UI) width.

Table 1-84: **Shift Operation for AXI Address Based on Data Width**

UI Data Width	Right Shift (Applied to Address Bits)
32	No shift
64	Shift by 1
128	Shift by 2

Table 1-84: **Shift Operation for AXI Address Based on Data Width *(Cont'd)***

UI Data Width	Right Shift (Applied to Address Bits)
256	Shift by 3
512	Shift by 4

The resultant address is then masked using Equation 1-1 for generating the mask bits.

```
BURST_MASK = {ADDR_WIDTH {1'b1}} ^ {(BURST_LEN + nCK_PER_CLK/2) {1'b1}};
```
Equation 1-1

Where:

1. ADDR_WIDTH is the memory controller address width.
2. BURST_LEN is equal to "1" for BC4 (DDR3) or BL4 (DDR2) and equal to "2" for BL8.
3. nCK_PER_CLK is the memory controller clock to DRAM clock ratio (always equal to "2").

For example, if the AXI data width is 64 bits and the address applied for the given transaction on the AXI interface is `0092_4920` (hex):

1. After the first shift operation, the resulting address is `0049_2490` (hex).
2. Assuming the address width is 28 and burst mode is BL8, the mask value is `0FFF_FFF8` (hex).
3. The resulting address is `32'h0049_2490`.
4. This address is driven to the UI address and if the address mode is "ROW_BANK_COLUMN", the corresponding addresses are ROW = `0249` (hex), BANK = `1` (hex), and COLUMN = `090` (hex).

The address increments are performed as defined by Table 1-85 for AXI burst transactions.

Table 1-85: **Address Increments for Multiple AXI Bursts**

UI Data Width	BURST_LEN	Address Increment Value
32	1	4
32	2	8
64	1	8
64	2	16
128	1	16
128	2	32
256	1	32
256	2	64
512	1	64
512	2	128

User Interface

The mapping between the User Interface address bus and the physical memory row, bank, and column is configurable. Depending on how the application data is organized, the

Bank-Row- Column or Row-Bank-Column addressing scheme can be chosen to optimize controller efficiency. These addressing schemes are shown in Figure 1-58 and Figure 1-59.

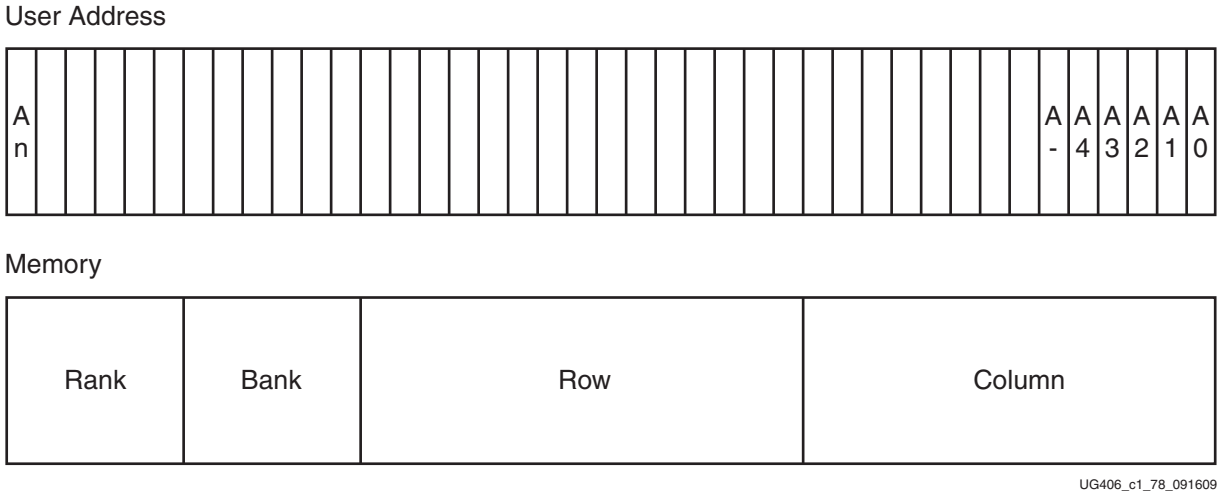


Figure 1-58: **Memory Address Mapping for Bank-Row-Column Mode in the UI Module**

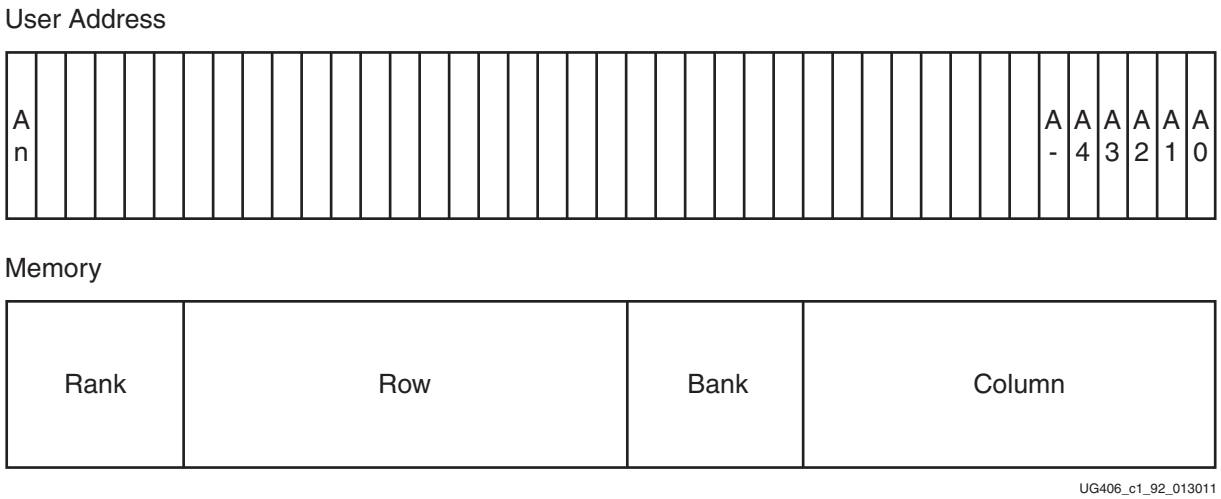


Figure 1-59: **Memory Address Mapping for Row-Bank-Column Mode in the UI Module**

Command Path

When the user logic app_en signal is asserted and the app_rdy signal is asserted from the UI, a command is accepted and written to the FIFO by the UI. The command is ignored by the UI whenever app_rdy is deasserted. The user logic needs to hold app_en High along with the valid command and address values until app_rdy is asserted as shown in Figure 1-60.

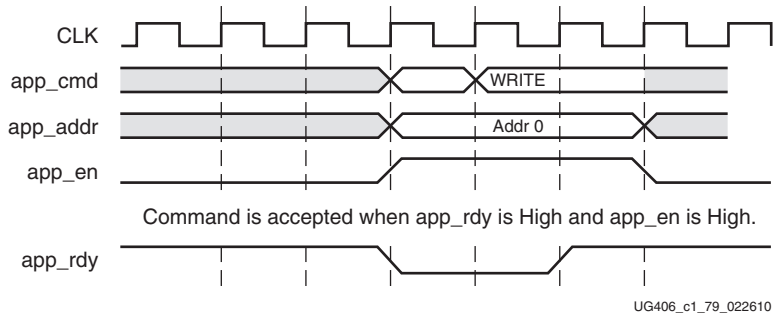


Figure 1-60: **UI Command Timing Diagram with app_rdy Asserted**

A non back-to-back write command can be issued as shown in Figure 1-61.

This figure depicts three scenarios for the app_wdf_data, app_wdf_wren, and app_wdf_end signals, as per the following:

1. Write data is presented along with the corresponding write command (second half of BL8).
2. Write data is presented before the corresponding write command.
3. Write data is presented after the corresponding write command, but should not exceed the limitation of two clock cycles.

For write data that is output after the write command has been registered, as shown in Note 3, the maximum delay is two clock cycles.

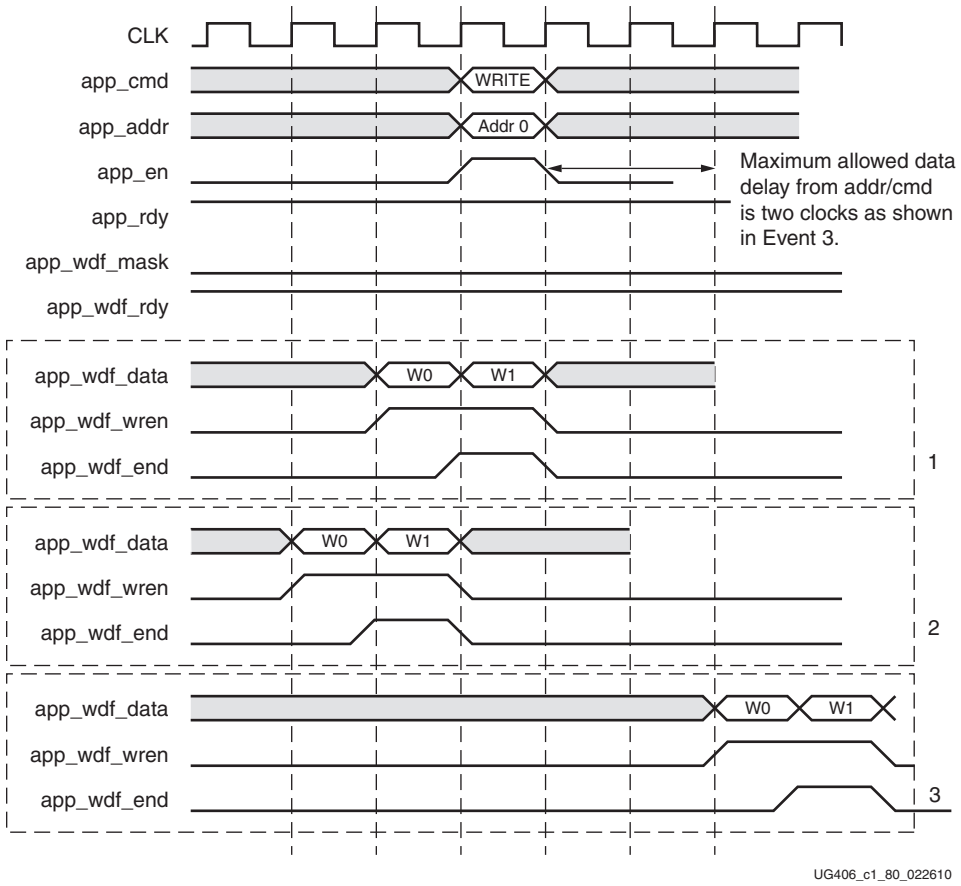


Figure 1-61: **UI Write Timing Diagram (Memory Burst Type = BL8)**

Write Path

The write data is registered in the write FIFO when app_wdf_wren is asserted and app_wdf_rdy is High (Figure 1-62). If app_wdf_rdy is deasserted, the user logic needs to hold app_wdf_wren and app_wdf_end High along with the valid app_wdf_data value until app_wdf_rdy is asserted. The app_wdf_mask signal can be used to mask out the bytes to write to external memory.

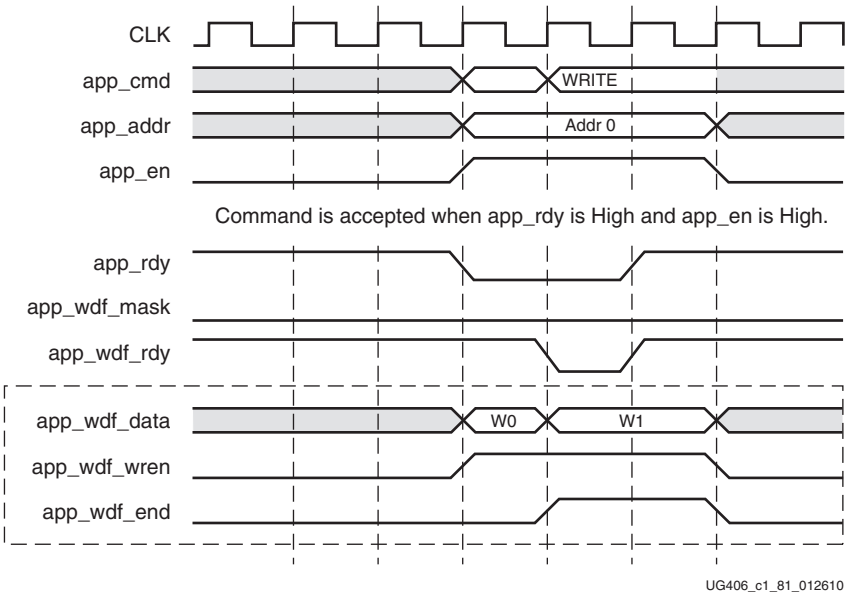


Figure 1-62: **UI Interface Write Timing Diagram with app_wdf_rdy Asserted (DDR2 and DDR3 SDRAM: Memory Burst Type = BL8)**

As shown in Figure 1-60, page 116, the maximum delay for a single write between the write data and the associated write command is two clock cycles. When issuing back-to-back write commands, there is no maximum delay between the write data and the associated back-to-back write command, as shown in Figure 1-63.

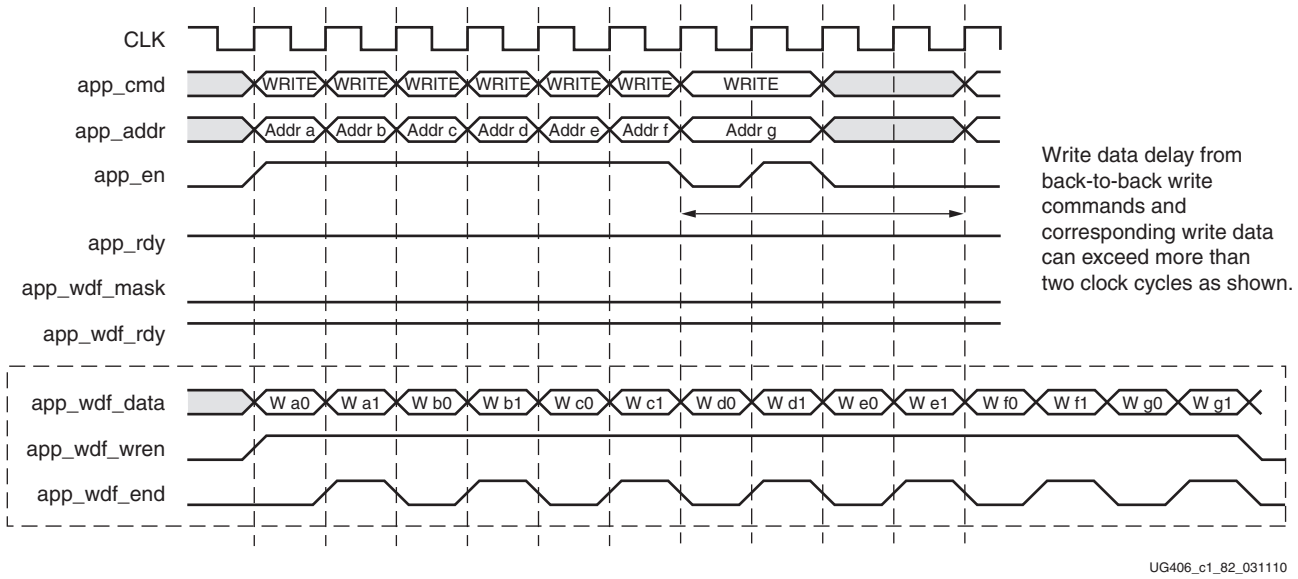


Figure 1-63: **UI Back-to-Back Write Commands Timing Diagram (Memory Burst Type = BL8)**

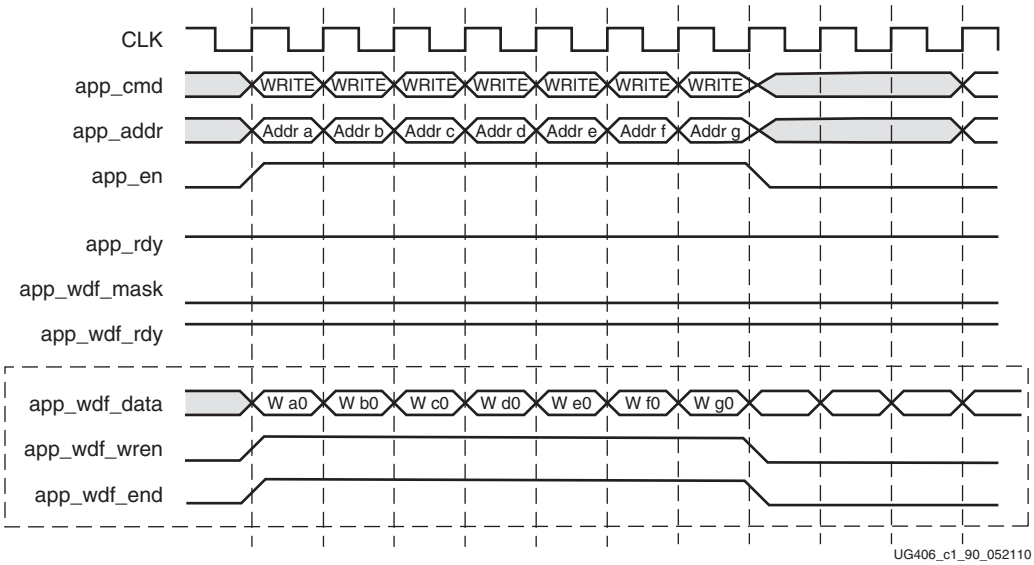


Figure 1-64: **UI Back-to-Back Write Commands Timing Diagram (DDR2 SDRAM: Memory Burst Type = BL4)**

The app_wdf_end signal must be used to indicate the end of a memory write burst. Figure 1-64 shows the write timing for a design that has the memory burst type set to four. For memory burst types of eight, the app_wdf_end signal must be asserted on the second write data word, as shown in Figure 1-60, page 116.

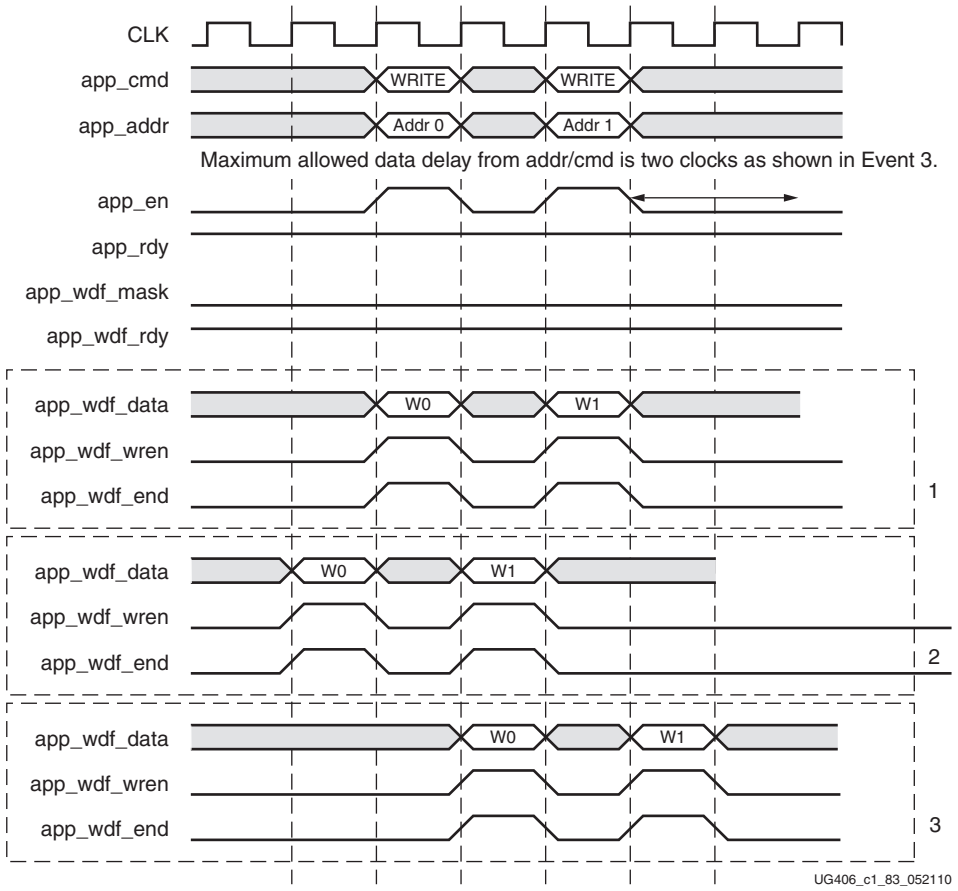


Figure 1-65: **UI Scattered Write Timing Diagram (DDR3 SDRAM: Memory Burst Type = BC4)**

The notes relevant to Figure 1-65 are:

1. The add/write command is issued at the same cycle of the last user burst word.
2. The add/write command is issued one clock delay from the last user burst word.
3. The user data is allowed to have at most a two-clock latency after the corresponding write command has been issued.

When the memory burst type is set to BC4, last four bits of burst are ignored by the SDRAM. The DDR3 SDRAM provides the on the fly (OTF) mode and allows the user logic to change the memory burst type via the A12 address bit. The user can end a write transaction earlier for four write bits by asserting the app_wdf_end signal, as shown in Figure 1-66.

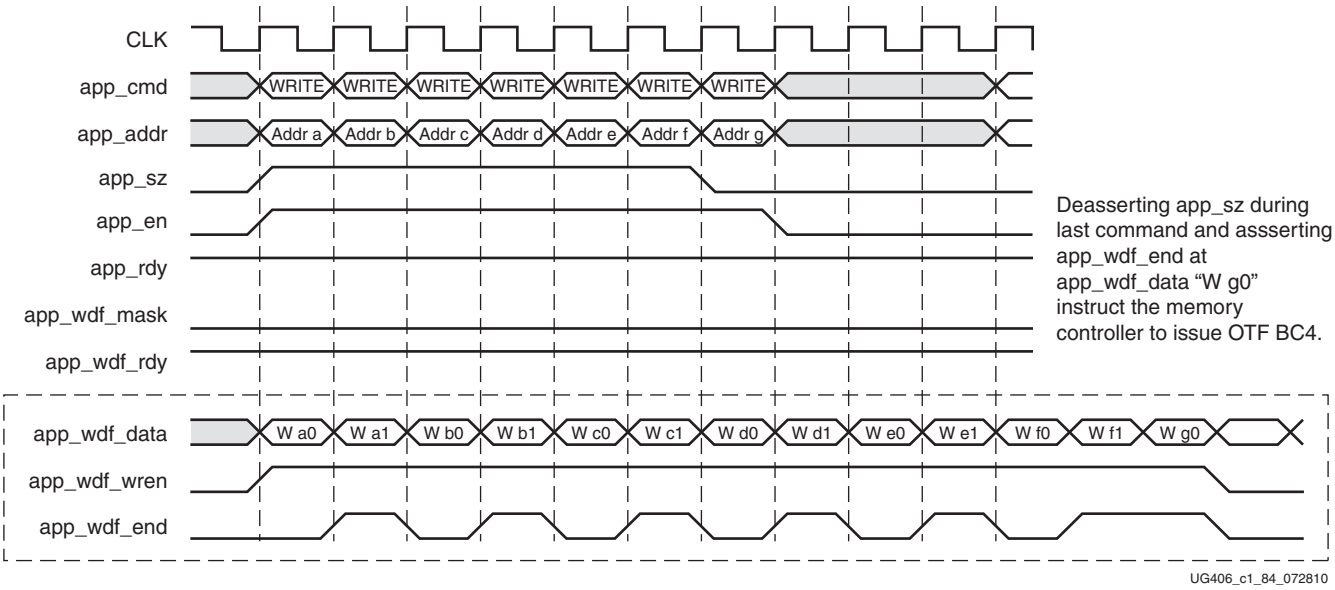


Figure 1-66: **UI Back-to-Back Write Commands Timing Diagram (Memory Burst Type = OTF)**

The map of the application interface data to the DRAM output data can be explained with an example. For a 2:1 memory controller to DRAM clock ratio, the application data width is 32 bits. Hence for a BL8 transaction, the data at the application interface must be provided in two clock cycles. The app_wdf_end signal is asserted for the second data as shown in Figure 1-67. In this case, the application data provided in the first cycle is `0000_0405` (hex) and the data provided in the last cycle is `0000_080A` (hex). This is for a BL8 transaction.

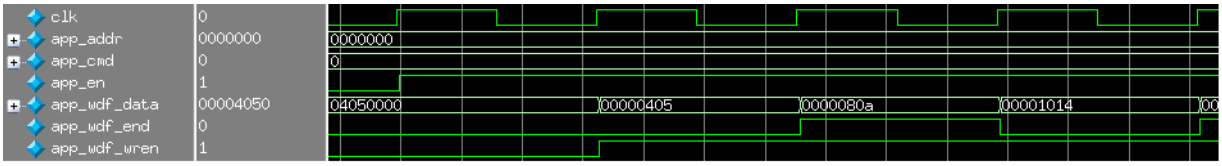


Figure 1-67: **Data at the Application Interface for 2:1 Mode**

The corresponding data at the DRAM interface is as shown in Figure 1-68.

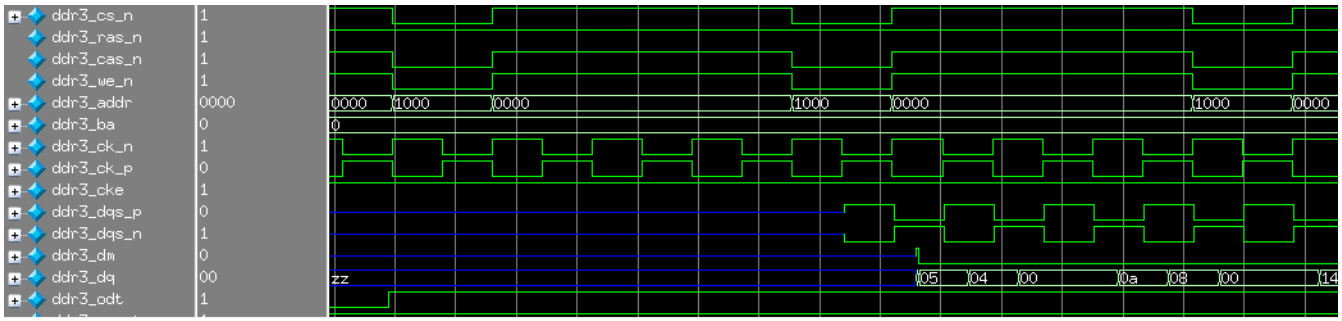


Figure 1-68: **Data at the DRAM Interface for 2:1 Mode**

The data values at different clock edges are as shown in Table 1-86.

Table 1-86: **Data Values at Different Clock Edges**

Rise0	Fall0	Rise1	Fall1	Rise2	Fall2	Rise3	Fall3
05	04	00	00	0a	08	00	00

Read Path

The read data is returned by the UI in the requested order and is valid when app_rd_data_valid is asserted (Figure 1-69). The app_rd_data_end signal indicates the end of each read command burst and is not needed in user logic.

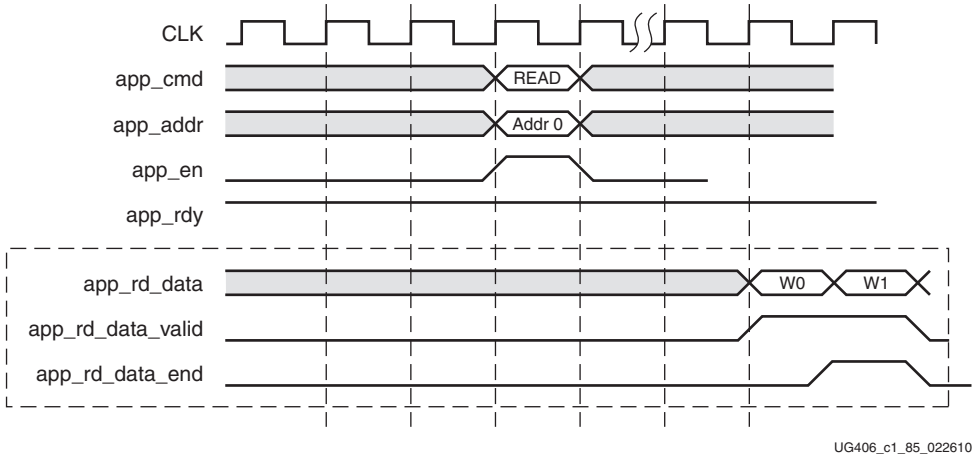


Figure 1-69: **UI Read Timing Diagram**

Native Interface

The native interface protocol is shown in Figure 1-70.

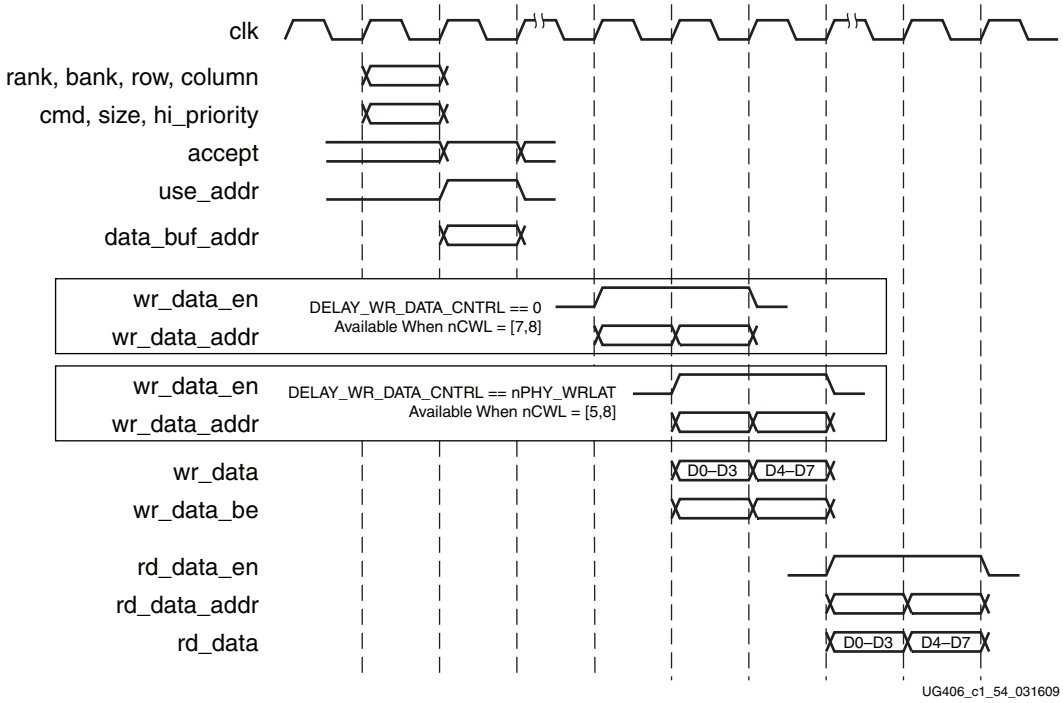


Figure 1-70: **Native Interface Protocol**

Requests are presented to the native interface as an address and a command. The address is composed of the bank, row, and column inputs. The command is encoded on the cmd input.

The address and command are presented to the native interface one state before they are validated with the use_addr signal. The memory interface indicates that it can accept the request by asserting the accept signal. Requests are confirmed as accepted when use_addr and accept are both asserted in the same clock cycle. If use_addr is asserted but accept is not, the request is not accepted and must be repeated. This behavior is shown in Figure 1-71.

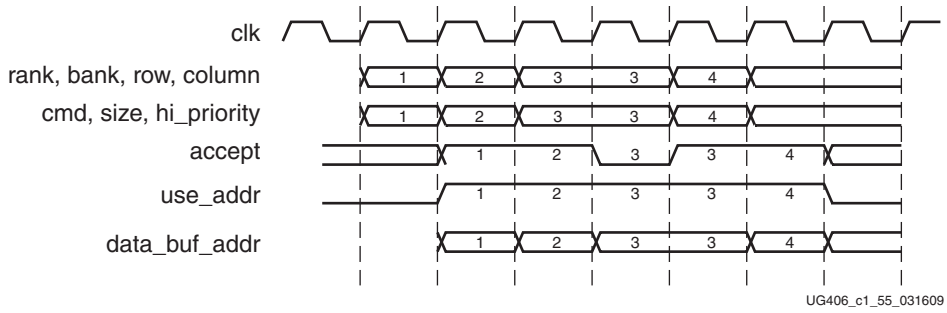


Figure 1-71: **Native Interface Flow Control**

In Figure 1-71, requests 1 and 2 are accepted normally. The first time request 3 is presented, accept is driven Low, and the request is not accepted. The user design retries request 3,

which is accepted on the next attempt. Request 4 is subsequently accepted on the first attempt.

The data_buf_addr bus must be supplied with requests. This bus is an address pointer into a buffer that exists in the user design. It tells the core where to locate data when processing write commands and where to place data when processing read commands. When the core processes a command, the core echoes data_buf_addr back to the user design via wr_data_addr for write commands and rd_data_addr for read commands. This behavior is shown in Figure 1-72. Write data must be supplied in the same clock cycle that wr_data_en is asserted.

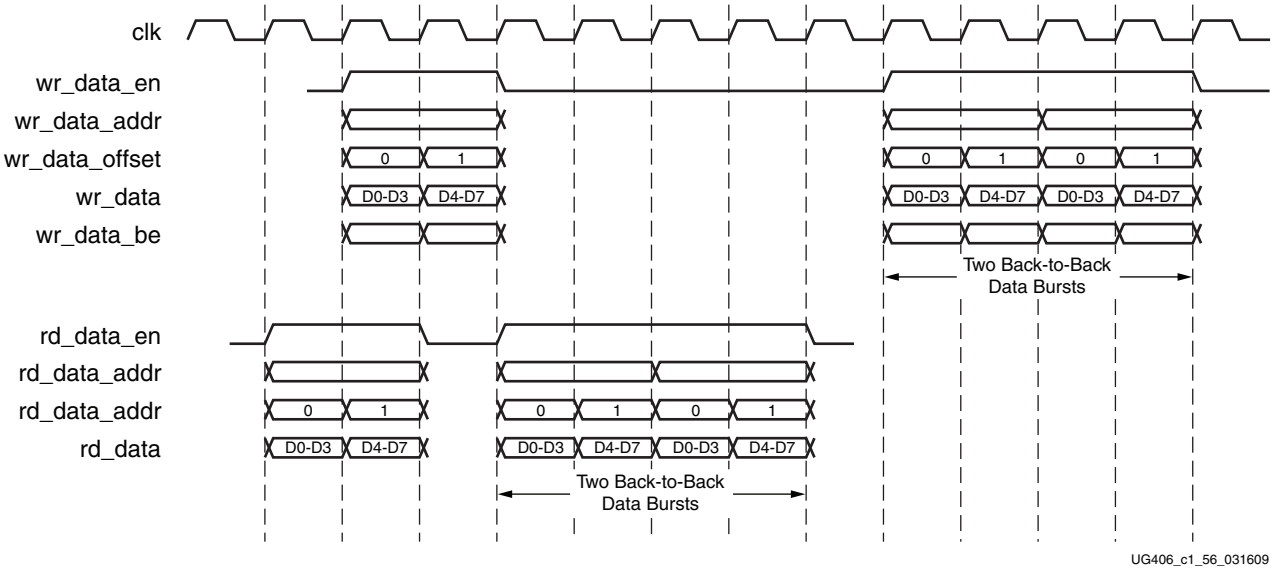


Figure 1-72: **Command Processing**

Transfers can be isolated with gaps of non-activity, or there can be long bursts with no gaps. The user design can identify when a request is being processed and when it finishes by monitoring the rd_data_en and wr_data_en signals. When the rd_data_en signal is asserted, the memory controller has completed processing a read command request. Similarly, when the wr_data_en signal is asserted, the memory controller is processing a write command request.

When NORM ordering mode is enabled, the memory controller reorders received requests to optimize throughput between the FPGA and memory device. The data is returned to the user design in the order processed, not the order received. The user design can identify the specific request being processed by monitoring rd_data_addr and wr_data_addr. These fields correspond to the data_buf_addr supplied when the user design submits the request to the native interface. Both of these scenarios are depicted in Figure 1-72.

The native interface is implemented such that the user design must submit one request at a time and, thus, multiple requests must be submitted in a serial fashion. Similarly, the core must execute multiple commands to the memory device one at a time. However, due to pipelining in the core implementation, read and write requests can be processed in parallel at the native interface.

Read Latency

Read latency is measured from the point where the read command is accepted by the UI or native interface. In general, read latency varies based on several parameters:

- The number of commands already in the 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
- Commands can be interrupted, and banks/rows can forcibly be closed when the periodic AUTO REFRESH command is issued
- CAS latency

Table 1-87 shows the read latency in memory clock cycles for two cases of the Virtex-6 FPGA DDR2 and DDR3 memory interfaces. Both cases have the refresh, zqcalib, and periodic reads disabled.

Table 1-87: **Read Latency for Virtex-6 FPGA DDR2 and DDR3 Memory**

Scenario	Read Latency in Memory Clock Cycles	
	UI-to-UI Interface	MC Native to MC Native Interface
Read to an unopened bank	40	34
Read to an opened bank	38	32

Core Constraints

The Virtex-6 FPGA DDR2/DDR3 memory interface solutions require a number of UCF constraints in addition to the system clock period, and pinout-related constraints to meet specified performance requirements.

Constraints provided with the integrated block solution have been tested in hardware and provide consistent results. Constraints can be modified, but modifications should only be made with a thorough understanding of the effect of each constraint. Support is not provided for designs that deviate from the provided constraints.

Timing Constraints

This section defines clock frequency and critical datapath requirements for the PHY core.

BUFR Resynchronization Clock Period Constraint

BUFR local clock buffer(s) are used in the design to synchronize read data captured at the ISERDES into the CLB logic. These clocks run at half the memory clock rate. They must be specified separately because the timing tools cannot infer their clock frequency as these clocks are forwarded through OSERDES blocks. The period constraint must be set to twice the memory clock period. For example, for a DDR3 design with the memory running at 500 MHz (1,000 Mb/s), this constraint should be set to 4.0 ns.

```
NET "u_memc_ui_top/u_mem_intfc/phy_top0/clk_rsync[?]" TNM_NET =
TNM_clk_rsync;
TIMESPEC "TS_clk_rsync" = PERIOD "TNM_clk_rsync" 4.0 ns;
```

BUFR Resynchronization Full-Cycle Path

The Q outputs of the DQ and DQS capture ISERDES are clocked by a BUFR local buffer. The polarity of this clock within the ISERDES is determined by the DYNCLKDIVSEL input of the ISERDES, the setting of which is determined by the PHY during read leveling. The Q outputs in turn drive two groups of CLB flip-flops, one clocked by the BUFR rising edge, and the other by the BUFR falling edge. Only one of the two groups of CLB flip-flops is used based on the setting of DYNCLKDIVSEL. For example, if DYNCLKDIVSEL is set to 1, the ISERDES Q outputs are clocked by the falling edge of BUFR, and only the BUFR falling-edge CLB flops are used. A constraint in the UCF is added to prevent the half-cycle path between the rising edge of BUFR (driving the ISERDES Q outputs) and the CLB flip-flops clocked by the falling edge of BUFR from being analyzed—this is a false path. This constraint forces full-cycle timing to be applied globally for all rising to falling edge BUFR paths. The only such paths in the design are between the ISERDES Q outputs and the CLB flip-flops. If the user modifies the logic in the BUFR domain such that other rising to falling edge paths exist, the constraint below must be modified to affect only the DQ and DQS ISERDES Q output paths.

```
TIMEGRP 'TG_clk_rsync_rise' = RISING 'TNM_clk_rsync';
TIMEGRP 'TG_clk_rsync_fall' = FALLING 'TNM_clk_rsync';
TIMESPEC 'TS_clk_rsync_rise_to_fall' =
FROM 'TG_clk_rsync_rise' TO 'TG_clk_rsync_fall' 'TS_sys_clk' * 2;
```

PHY_INIT_DATA_SEL Multi-Cycle Path

The PHY_INIT_DATA_SEL net selects whether to output to the DDR2/DDR3 memory data from the controller portion of the MIG design, or data internally generated within the PHY to the memory. A multi-cycle path constraint is added in the UCF for PHY_INIT_DATA_SEL to relax the timing constraint on this net. The user should not increase the timing constraint beyond what is specified in the MIG-generated output.

```
INST
"u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_init/u_ff_phy_init_data_sel"
TNM = "TNM_PHY_INIT_SEL";
TIMESPEC "TS_MC_PHY_INIT_SEL" = FROM "TNM_PHY_INIT_SEL" TO FFS =
"TS_sys_clk"*4;
```

Location and I/O Constraints

The MIG tool generates the pin locations and I/O standard constraints according to the selected memory type and the defined physical layer rules, as listed in Design Guidelines, page 135. A number of internal elements (OSERDES, IODELAY, MMCM) are explicitly located via UCF LOC constraints in a MIG-generated design. Changing the location of these constraints to accommodate post-MIG pinout changes is possible, but requires that the user be familiar with the X-Y coordinate system used to specify internal FPGA locations. The user should also be able to determine the correct X-Y location for a particular constraint in a given device and package. X-Y locations can be determined through use of the PARTGen utility or fpga_editor tools.

Resynchronization Clock Forwarding and Distribution Elements

The resynchronization clock is generated and distributed through an OSERDES, an IODELAY (configured as an ODELAY), and a BUFR local clock buffer network. The resynchronization clock is used to synchronize read data captured at the ISERDES into the FPGA logic. There is one distinct resynchronization clock per I/O column used on the part.

The OSERDES and IODELAY for a particular resynchronization clock are, in turn, associated with a particular I/O site. The requirements of this I/O site are:

- The P-side of a clock-capable I/O pin pair must be used. This can either be an SRCC or MRCC I/O pin. The N-side of a clock-capable I/O pair cannot be used for this purpose.
- All other DDR2/DDR3 data (DQ) pins in the same I/O column must either be in the same bank or in the bank immediately above or below.
- The I/O site must not be connected on the user's PCB. Although no signal is actively driven out of the FPGA, the I/O site corresponding to this pin is unavailable to other logic.

After an appropriate clock-capable I/O pin is located, PARTGen or fpga_editor can be used to determine the corresponding X-Y coordinates for that particular I/O site. For example, for an XC6VLX240T-FF1156 device, if the I/O site corresponds to pin D34 (Bank 16), the LOC constraints for the OSERDES and IODELAY are:

```
INST
"u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_read/u_phy_rdclk_gen/gen_loo
p_col0.u_oserdes_rsync" LOC = "OLOGIC_X0Y183";
INST
"u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_read/u_phy_rdclk_gen/gen_loo
p_col0.u_odelay_rsync" LOC = "IODELAY_X0Y183";
```

One of the two BUFRs accessible by the MRCC I/O site must be specified. The X-Y coordinate of this BUFR can be determined through the fpga_editor. In the previous example, for an MRCC used in Bank 16, either BUFR_X0Y9 or BUFR_X0Y10 can be used:

```
INST
"u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_read/u_phy_rdclk_gen/gen_loo
p_col0.u_bufr_rsync" LOC = "BUFR_X0Y9";
```

CONFIG_PROHIBIT constraints for each of the I/O sites used in this manner are inserted by the MIG tool to prevent the use of these pins for other logic.

Capture Clock Forwarding and Distribution Elements

The capture clock is generated and distributed through an OSERDES, an IODELAY (configured as an ODELAY), and a BUFIO local clock buffer network. The capture clock is used to direct capture read data from the DDR2/DDR3 memory at the ISERDES. There is one distinct capture clock per DQS group; for example, for a 72-bit data bus with an 8:1 DQ:DQS ratio, there are nine different capture clocks, one for each byte. The OSERDES and IODELAY for a particular capture clock are, in turn, associated with a particular I/O site. The requirements of this I/O site are:

- The P-side of a clock-capable I/O pin pair must be used. This can either be an SRCC or MRCC I/O pin. The N-side of a clock-capable I/O pair cannot be used for this purpose.
- All DQ, DQS, and DM pins belonging to a particular DQS group must reside in the same bank.
- The I/O site must not be connected on the user's PCB. Although no signal is actively driven out of the FPGA, the I/O site corresponding to this pin is unavailable to other logic.

After an appropriate clock-capable I/O pin is located, PARTGen or fpga_editor can be used to determine the corresponding X-Y coordinates for that particular I/O site. For

example, for a XC6VLX240T-FF1156 device, if the I/O site corresponds to pin C28 (Bank 25), the LOC constraints for the OSERDES and IODELAY are:

```
INST
"u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_read/u_phy_rdclk_gen/gen_ck_
cpt[0].u_oserdes_cpt" LOC = "OLOGIC_X1Y141";
INST
"u_memc_ui_top/u_mem_intfc/phy_top0/u_phy_read/u_phy_rdclk_gen/gen_ck_
cpt[0].u_odelay_cpt" LOC = "IODELAY_X1Y141";
```

Unlike the case for the resynchronization clock constraints, the BUFIO location does not need to be specified. The tools automatically infer the use of an appropriate BUFIO. CONFIG_PROHIBIT constraints for each of the I/O sites used in this manner are also inserted by the MIG tool to prevent the use of these pins for other logic.

MMCM Locations

The Virtex-6 FPGA DDR2/DDR3 design requires the use of an MMCM. There are two MMCMs per horizontal clock row (HROW). The MIG tool places this MMCM in the same HROW as the bank in which the DDR3 control and address outputs are located. For example, for a XC6VLX240T-FF1156 device, if the control and address is placed in either Banks 26 or 36 (these banks reside on the same HROW), the corresponding MMCM location is:

```
INST "u_infrastructure/u_mmcm_adv"      LOC = "MMCM_ADV_X0Y9";
```

Customizing the Core

The Virtex-6 FPGA memory interface solution supports several configurations for both DDR2 and DDR3 SDRAM devices. The specific configuration is defined by Verilog parameters in the top level of the core. The MIG tool should be used to regenerate a design when parameters need to be changed. The parameters set by the MIG tool are summarized in Table 1-88 and Table 1-89. The parameter name in EDK differs slightly and is listed as the name in parentheses. If the parameter name does not have an equivalent EDK parameter name, no name is listed in parentheses. The value of the parameter is either auto-computed or fixed and cannot be modified.

Table 1-88: **Virtex-6 FPGA Memory Solution Configuration Parameters**

Parameter	Description	Options
REFCLK_FREQ(1) (C_REFCLK_FREQ)	This is the reference clock frequency for IODELAYCTRLs. This can be set to 200.0 for any speed grade device or 300.0 for a -2 or -3 device. For more information, refer to the IODELAYE1 Attribute Summary table in the *Virtex-6 FPGA SelectIO Resources User Guide* [Ref 4].	200.0, 300.0
SIM_BYPASS_INIT_CAL(2) (C_BYPASS_INIT_CAL)	This parameter is used to set the individual calibration and initialization procedure used for simulation. It can be used to significantly shorten the simulation time during initial power-up, initialization, and calibration.	“OFF” “SKIP” “FAST”
SIM_CAL_OPTION(3)	This is the calibration procedure for simulation. The value of this parameter is ignored when SIM_BYPASS_INIT_CAL is set to a value other than “NONE”.	“NONE” “FAST_CAL”

Table 1-88: **Virtex-6 FPGA Memory Solution Configuration Parameters *(Cont'd)***

Parameter	Description	Options
SIM_INIT_OPTION(4)	This is the initialization procedure for simulation. The value of this parameter is ignored when SIM_BYPASS_INIT_CAL is set to a value other than "NONE".	"NONE" "SKIP_PU_DLY"
nCK_PER_CLK	This is the number of memory clocks per clock.	2
nCS_PER_RANK (C_NCS_PER_RANK)	This is the number of unique CS outputs per rank for the PHY.	1, 2
DQS_CNT_WIDTH	This is the number of bits required to index the DQS bus and is given by ceil($\log_2$(DQS_WIDTH)).	
ADDR_WIDTH	This is the memory address bus width. It is equal to RANK_WIDTH + BANK_WIDTH + ROW_WIDTH + COL_WIDTH.	
BANK_WIDTH (C_BANK_WIDTH)	This is the number of memory bank address bits.	This option is based on the selected memory device.
CS_WIDTH (C_CS_WIDTH)	This is the number of unique CS outputs to memory.	This option is based on the selected MIG tool configuration.
CK_WIDTH (C_CK_WIDTH)	This is the number of CK/CK# outputs to memory.	This option is based on the selected MIG tool configuration.
CKE_WIDTH (C_CKE_WIDTH)	This is the number of CKE outputs to memory.	This option is based on the selected MIG tool configuration.
COL_WIDTH (C_COL_WIDTH)	This is the number of memory column address bits.	This option is based on the selected memory device.
RANK_WIDTH	This is the number of bits required to index the RANK bus and is given by ceil($\log_2$(RANKS)).	This option is based on the selected memory device.
ROW_WIDTH (C_ROW_WIDTH)	This is the DRAM component address bus width.	This option is based on the selected memory device.
DM_WIDTH (C_DM_WIDTH)	This is the number of data mask bits.	DQ_WIDTH/8
DQ_WIDTH (C_DQ_WIDTH)	This is the memory DQ bus width.	This parameter supports DQ widths from 8 to a maximum of 144 in increments of 8. The available maximum DQ width is frequency dependent on the selected memory device.
DQS_WIDTH (C_DQS_WIDTH)	This is the memory DQS bus width.	DQ_WIDTH/8
BURST_MODE (C_BURST_MODE)	This is the memory data burst length.	DDR2: "8", "4" DDR3: "8", "4", "OTF"
BM_CNT_WIDTH	This is the number of bits required to index a bank machine and is given by ceil($\log_2$(nBANK_MACHS)).	

Table 1-88: **Virtex-6 FPGA Memory Solution Configuration Parameters *(Cont'd)***

Parameter	Description	Options
ADDR_CMD_MODE (C_ADDR_CMD_MODE)	This parameter is used by the controller to calculate timing on the memory addr/cmd bus. The 2T option has memory command and address signals asserted for two cycles to provide better signal integrity. 1T can allow better controller efficiency because it provides more time slots for commands to the memory. 1T is recommended for DDR3 SDRAM, and 2T is recommended for wider DDR2 SDRAM interfaces such as UDIMM. Board simulations are recommended to determine if 2T is required.	"2T" "1T"
ORDERING(5) (C_ORDERING)	This option reorders received requests to optimize data throughput and latency.	"NORM": Allows the memory controller to reorder commands to memory to obtain the highest possible efficiency. "STRICT": Forces the memory controller to execute commands in the exact order received.
STARVE_LIMIT	This sets the number of times a read request can lose arbitration before the request declares itself high priority. The actual number of lost arbitrations is STARVE_LIMIT × nBANK_MACHS.	1, 2, 3, ... 10
WRLVL	This option enables write leveling calibration in DDR3 designs. For DIMM designs, this is required to be ON. For DDR3 component designs that use fly-by routing, this option should be turned ON. The value of this parameter is ignored when SIM_BYPASS_INIT_CAL is set to "SKIP".	DDR3: "ON", "OFF" DDR2: "OFF"
PHASE_DETECT	This is the phase detector. It adjusts capture for voltage and temperature compensation. The value of this parameter is ignored when SIM_BYPASS_INIT_CAL is set to "SKIP".	"ON": For interfaces above 250 MHz "OFF": For interfaces below 250 MHz
RTT_NOM (C_RTT_NOM)	This is the nominal ODT value.	DDR3_SDRAM: "120": RZQ/2 "60": RZQ/4 "40": RZ/6 DDR2_SDRAM: "DISABLED": RTT_NOM disabled. "150": 150 Ω "75": 75 Ω "50": 50 Ω
RTT_WR (C_RTT_WR)	This is the dynamic ODT write termination used in multiple-RANK designs. For single-component designs, RTT_WR should be disabled.	DDR3_SDRAM: "OFF": RTT_WR disabled. "120": RZQ/2 "60": RZQ/4

Table 1-88: **Virtex-6 FPGA Memory Solution Configuration Parameters *(Cont'd)***

Parameter	Description	Options
OUTPUT_DRV (C_OUTPUT_DRV)	This is the DRAM reduced output drive option.	"HIGH" "LOW"
REG_CTRL (C_REG_CTRL)	This is the option for DIMM or unbuffered DIMM selection.	"ON": Registered DIMM "OFF": Components, SODIMMs, UDIMMs.
IODELAY_GRP (C_IODELAY_GRP)	This is an ASCII character string to define an IODELAY group used in a memory design. This is used by the ISE tools to group all instantiated IODELAYs into the same bank. Unique names must be assigned when multiple IP cores are implemented on the same FPGA.	Default: "IODELAY_MIG"
nDQS_COL0 (C_NDQS_COL0)	This parameter defines how many DQS groups are implemented in inner I/O column 0.	
nDQS_COL1 (C_NDQS_COL1)	This parameter defines how many DQS groups are implemented in inner I/O column 1.	
nDQS_COL2 (C_NDQS_COL2)	This parameter defines how many DQS groups are implemented in outer I/O column 2. The performance is lower if DQS groups are implemented in the outer I/O column.	
nDQS_COL3 (C_NDQS_COL3)	This parameter defines how many DQS groups are implemented in outer I/O column 3. The performance is lower if DQS groups are implemented in the outer I/O column.	
DQS_LOC_COL0 (C_DQS_LOC_COL0)	This parameter defines which DQ bytes are mapped to inner I/O column 0.	Bytes 0, 1, and 3 are mapped to the column in a 16-bit DQ design, for example, `24'h020100`
DQS_LOC_COL1 (C_DQS_LOC_COL1)	This parameter defines which DQ bytes are mapped to inner I/O column 1.	Bytes 3, 4, 5, 6, 7, and 8 are mapped to the column, for example, `48'h080706050403`
DQS_LOC_COL2 (C_DQS_LOC_COL2)	This parameter defines which DQ bytes are mapped to outer I/O column 2.	
DQS_LOC_COL3 (C_DQS_LOC_COL3)	This parameter defines which DQ bytes mapped to outer I/O column 3.	
ECC_TEST (C_ECC_TEST)	This option, when set to "ON," allows the entire DRAM bus width to be accessible though the UI. For example, if DATA_WIDTH == 64, the app_rd_data width is 288.	"ON" "OFF"
PAYLOAD_WIDTH	This is the actual DQ bus used for user data.	ECC_TEST = OFF: PAYLOAD_WIDTH = DATA_WIDTH ECC_TEST = ON: PAYLOAD_WIDTH = DQ_WIDTH
DEBUG_PORT	This option enables debug signals/control.	"ON" "OFF"

Table 1-88: **Virtex-6 FPGA Memory Solution Configuration Parameters *(Cont'd)***

Parameter	Description	Options
TCQ	This is the clock-to-Q delay for simulation purposes.	(The value is in picoseconds.)
tCK (C_TCK)	This is the memory tCK clock period (ps).	The value, in picoseconds, is based on the selected frequency in the MIG tool.
MEM_ADDR_ORDER (C_MEM_ADDR_ORDER)	This option selects the address mapping scheme between the User Interface address bus and physical memory row, bank, and column.	"BANK_ROW_COLUMN" "ROW_BANK_COLUMN"

Notes:

1. The lower limit (maximum frequency) is pending characterization.
2. SIM_BYPASS_INIT_CAL is used to reduce simulation time by bypassing and/or abbreviating the initial power-up and calibration sequence. Setting this parameter to either "SKIP" or "FAST" overrides the user-specified value of four other parameters, SIM_INIT_OPTION, SIM_CAL_OPTION, WRLVL, and PHASE_DETECT, and causes the initialization and calibration sequence to be shortened. The user can also achieve the same result by setting SIM_BYPASS_INIT_CAL to "OFF", and individually setting the value of these four parameters.

   Setting SIM_BYPASS_INIT_CAL to "FAST" bypasses the memory power-up initialization and performs an abbreviated calibration sequence. Setting SIM_BYPASS_INIT_CAL to "FAST" causes these parameters to be overridden to these values:

   - SIM_INIT_OPTION = "SKIP_PU_DLY"
   - SIM_CAL_OPTION = "FAST_CAL"
   - WRLVL and PHASE_DETECT retain their user-specified settings

   Setting SIM_BYPASS_INIT_CAL to "SKIP" bypasses the memory power-up initialization, skips the calibration sequence, and disables blocks within the PHY. The "SKIP" setting causes calibration to be skipped and fixes various timing relationships between the FPGA and memory. As a result, the "SKIP" setting should only be used in a behavioral simulation environment that does not assert additional propagation delays (for example, to model PCB trace delays between the FPGA and memory). Setting SIM_BYPASS_INIT_CAL to "SKIP" causes these parameters to be overridden to these values:

   - SIM_INIT_OPTION = "SKIP_PU_DLY"
   - SIM_CAL_OPTION = "SKIP_CAL"
   - WRLVL = "OFF"
   - PHASE_DETECT = "OFF"

   Setting SIM_BYPASS_INIT_CAL to "OFF" causes the above four parameters to retain their user-specified values. SIM_BYPASS_INIT_CAL should be set to "OFF" for implementation, or the core does not function properly. See notes (3) and (4) for descriptions of the SIM_CAL_OPTION and SIM_INIT_OPTION and parameters.
3. Core initialization during simulation can be greatly reduced by using SIM_CAL_OPTION. Two simulation modes are supported. Setting SIM_CAL_OPTION to FAST_CAL causes read calibration to occur on only one bit per memory device. This is then used across the remaining data bits. When SIM_CAL_OPTION is set to SKIP_CAL, no read calibration occurs, and the incoming clocks and data are assumed to be aligned. SIM_CAL_OPTION should be set to NONE for implementation, or the core does not function properly.
4. SIM_INIT_OPTION can be used to reduce simulation time by bypassing some or all of the memory initialization procedure. SKIP_PU_DLY is the preferred setting for both DDR2 and DDR3 SDRAMs.
5. When set to NORM, ORDERING enables the reordering algorithm in the memory controller. When set to STRICT, request reordering is disabled, which greatly limits throughput to the external memory device. However, it can be helpful during initial core integration because requests are processed in the order received; the user design does not need to keep track of which requests are pending and which requests have been processed.

The parameters listed in Table 1-89 depend on the selected memory clock frequency, memory device, memory configuration, and FPGA speed grade. The parameter name in EDK differs slightly and is listed as the name in parentheses. If the parameter name does not have an equivalent EDK parameter name, no name is listed in parentheses. The values for these parameters are embedded in the `memc_ui_top` IP core and should not be modified from the user's top level. Xilinx strongly recommends that users rerun the MIG tool for different configurations.

Table 1-89: **Embedded Virtex-6 FPGA Memory Solution Configuration Parameters**

Parameter	Description	Options
tFAW (C_TFAW)	This is the minimum interval of four active commands.	This value, in picoseconds, is based on the device selection in the MIG tool.
tPRDI (C_PRDI)	This is the periodic read interval for the read phase detector.	This value, in picoseconds, is based on the selected frequency in the MIG tool.
tRRD (C_TRRD)	This is the ACTIVE-to-ACTIVE minimum command period.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRAS (C_TRAS)	This is the minimum ACTIVE-to-PRECHARGE period for memory.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRCD (C_TRCD)	This is the ACTIVE-to-READ or -WRITE command delay.	This value, in picoseconds, is based on the device selection in the MIG tool.
tREFI (C_TREFI)	This is the average periodic refresh interval for memory.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRFC (C_RFC)	This is the REFRESH-to-ACTIVE or REFRESH-to-REFRESH command interval.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRP (C_TRP)	This is the PRECHARGE command period.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRTP (C_TRTP)	This is the READ-to-PRECHARGE command delay.	This value, in picoseconds, is based on the device selection in the MIG tool.
tWTR (C_TWTR)	This is the WRITE-to-READ command delay.	This value, in picoseconds, is based on the device selection in the MIG tool.
tZQI (C_TZQI)	This is the timing window to perform the ZQCL command in DDR3 SDRAM.	This value, in CK, is based on the device selection in the MIG tool.
tZQCS (C_TZQCS)	This is the timing window to perform the ZQCS command in DDR3 SDRAM.	This value, in CK, is based on the device selection in the MIG tool.
nAL	This is the additive latency in memory clock cycles.	0
CL (C_CL)	This is the read CAS latency. The available option is frequency dependent in the MIG tool.	DDR3: 5, 6, 7, 8, 9
CWL (C_CWL)	This is the write CAS latency. The available option is frequency dependent in the MIG tool.	DDR3: 5, 6, 7, 8
BURST_TYPE (C_BURST_TYPE)	This is an option for the ordering of accesses within a burst.	“Sequential” “Interleaved”
IBUF_LPWR_MODE	This option enables or disables the low-power mode for the input buffers.	“ON” “OFF”
IODELAY_HP_MODE (C_IODELAY_HP_MODE)	This option enables or disables the IODELAY high-performance mode.	“ON” ”OFF”
USE_DM_PORT (C_USE_DM_PORT)	This is the enable data mask option used during memory write operations.	1: Enable 0: Disable
CAL_WIDTH (C_CAL_WIDTH)	This parameter defines how many DRAM ranks are calibrated. For the current design, only single-rank is supported.	“HALF”

Table 1-89: **Embedded Virtex-6 FPGA Memory Solution Configuration Parameters *(Cont'd)***

Parameter	Description	Options
CK_WIDTH (C_CK_WIDTH)	This is the number of CK/CK# outputs to memory.	
DQ_CNT_WIDTH	This is ceil(log2(DQ_WIDTH)).	
DRAM_TYPE (C_DRAM_TYPE)	This is the supported memory standard for the memory controller.	"DDR2" "DDR3"
DRAM_WIDTH	This is the DQ bus width per DRAM component.	
AL	This is the additive latency.	0
nBANK_MACHS (C_NBANK_MACHS)	This is the number of bank machines. A given bank machine manages a single DRAM bank at any given time.	2, 3, 4, 5, 6, 7, 8
DATA_BUF_ADDR_WIDTH	This is the bus width of the request tag passed to the memory controller.	4
DATA_BUF_OFFSET_WIDTH	This is the bus width of the data offset that depends on nCK_PER_CLK and the maximum burst length.	0, 1
SLOT_0_CONFIG (C_SLOT_0_CONFIG)	This is the rank mapping.	Single-rank setting: `8'b0000_0001` Dual-rank setting: `8'b0000_0011`
ECC (C_ECC)	This is the error correction code, available in 72-bit and 144-bit data width configurations.	72, 144
RANKS (C_RANKS)	This is the number of ranks.	
DATA_WIDTH	This parameter determines the write data mask width and depends on whether or not ECC is enabled.	ECC = ON: DATA_WIDTH = DQ_WIDTH + ECC_WIDTH ECC = OFF: DATA_WIDTH = DQ_WIDTH
APP_DATA_WIDTH	This UI_INTFC parameter specifies the payload data width in the UI.	APP_DATA_WIDTH = PAYLOAD_WIDTH × 4
APP_MASK_WIDTH	This UI_INTFC parameter specifies the payload mask width in the UI.	

Design Guidelines

Guidelines for DDR3 and DDR2 SDRAM designs are covered in this section.

DDR3 SDRAM

This section describes guidelines for DDR3 SDRAM designs, including bank selection, pin allocation, pin assignments, termination, I/O standards, and trace lengths.

Design Rules

Memory types, memory parts, and data widths are restricted based on the selected FPGA, FPGA speed grade, and the design frequency. The maximum frequency of the design for commercial grade parts is:

- 400 MHz (-1 speed grade devices).
- 533 MHz (-2 and -3 speed grade devices).
- 333 MHz (-1 speed grade CXT devices).
- 400 MHz (-2 speed grade CXT devices).
- 400 MHz (low-power Virtex-6 devices).

Note: The final maximum frequency will be determined after characterization.

For frequencies above 333 MHz, only data widths up to 72 bits are allowed. For frequencies of 333 MHz and below, data widths up to 144 bits are allowed. The listed frequency ranges are preliminary values. The final frequency ranges are subject to characterization results.

DDR3 Component PCB Routing

Fly-by routing topology is required for the clock, address, and control lines. Fly-by means that this group of lines is routed in a daisy-chain fashion and terminated appropriately at the end of the line. The trace length of each signal within this group to a given component must be matched. The controller uses write leveling to account for the different skews between components. This technique uses fewer FPGA pins because signals do not have to be replicated.

Pin Assignments

The MIG tool generates pin assignments for a memory interface based on physical layer rules.

Bank and Pin Selection Guides for DDR3 Designs

The MIG tool selects address/control banks, data banks, and pin allocations that are restricted to the rules outlined below.

Bank Selection Rules

The bank selection rules are:

- Address/control groups can be selected only in the inner column banks.
- The first bank selected for the address/control group has CK[0] and CK#[0] pins.
- Banks consisting of CK[0] and CK#[0] have MMCMs utilized in the H-row.

- For design frequencies higher than 400 MHz, only inner column banks are allowed for data group selection. For design frequencies 400 MHz and below, both inner and outer column banks are allowed for data group selection.
- Only inner or outer column banks are allowed for selection.
- Inner and outer column banks that reside one row above, one row below, and on the same row of banks consisting of the CK[0] and CK#[0] pins are enabled for data group pin selection. This restriction is represented by a boundary box called the vicinity box.
- The system clock group can only be selected in the banks consisting of GC pins or in the inner column banks that are in the same H-row of allocated MMCMs.
- The system clock group and the rest of the design group pins (address/control group, data group, and system control group) cannot coexist in the same bank due to different voltage standards.
- A master bank must be selected for each column if DCI cascade is to be used.
- The system clock group bank cannot be selected as a master bank.
- One MMCM is always used for the design.

Pin Allocation Rules

The pin allocation rules are:

- Address/Control Group:
  - This group consists of A, BA, CK, CK#, CKE, CS#, RAS#, CAS#, WE#, ODT, and RESET# memory signals.
  - Only inner column banks are allowed for selection.
  - The memory clock signals (CK and CK#) are allocated to the differential pair pins (P-N pair).
  - The VRN/VRP pins are utilized for pin allocation. In this case, DCI cascading is applied to support the DCI standard on address/control group signals.
  - The VREF pins are utilized for pin allocation.
  - For DIMM designs, this group also includes sda and scl pins.
- Data Group:
  - The DQS group consists of the DQ and DM memory pins and their corresponding DQS and DQS# pins.
  - The DQS and DQS# pins are allocated to a P-N pair.
  - Each DQS group in a bank is associated with a CC-P pin reserved for BUFIO.
  - In a column of banks allocated with data group pins, at least one bank has a CC-P pin reserved for BUFR.
  - If VRN/VRP pins are utilized for pin allocation in a bank, the DCI cascading feature must be applied to support DCI. In this case, a master bank must be selected.
- System Clock Group:
  - This group consists of:
    - Design clocks: sys_clk_p, sys_clk_n (differential), or sys_clk (single-ended).
    - Reference clocks: clk_ref_p, clk_ref_n (differential), or clk_ref (single-ended).
    - Pins: sys_rst, error, and phy_init_done.

    - For ECC enabled designs, this group also includes the ECC app_ecc_multiple_err error pin.
  - Only inner column banks are allowed for selection.
  - The CC pins are allocated for the system clock group if the system clock bank is in the address/control bank H-row of allocated MMCMs.
  - The GC pins are allocated for the system clock group if the system clock bank is any one of banks 24, 25, 34, or 35.
  - This group is associated with the 2.5V I/O standard.
- Master Bank:
  - Except for the system clock bank, only banks within the vicinity of a given column of banks can be selected as a master bank.
  - A master bank always has unused VRN/VRP pins. Thus, in a given bank, VRN/VRP pins are reserved if they are selected in a master bank.
  - A master bank should always have at least one SSTL15_II_DCI I/O standard input pin. If not, a dummy input pin is allocated with the SSTL15_II_DCI I/O standard.
  - All data group banks in a given column act as slave banks if a master bank is selected in that column. The same is listed in the generated UCF.

Data/Strobe/Mask Span Allocation Rules

When generating or verifying the data/strobe pinout of a high-performance Virtex-6 FPGA DDR3 design, the MIG tool requires that the data, strobe, and mask pins in a given DQS group be kept not only in the same I/O bank, but also within a given number of "IOB clock delay intervals" within that bank to minimize the effect of clock skew on I/O timing. Virtex-6 FPGA DDR3 designs operating at 400 MHz or below do not need to adhere to these IOB clock delay interval limit requirements; however, they are still subject to the rule that DQ/DQS/DM for a given DQS group be kept in the same I/O bank.

An IOB clock delay interval is defined as the difference in clock arrival time at two IOBs driven by the same clock network. BUFIOs are routed to the 40 I/Os in a bank such that the clock tree enters the bank and connects to the two middle IOBs in that bank first. The clock tree then travels up and down from that point. Using the nomenclature used in both a PARTGEN-generated package file and fpga_editor, the first two I/Os to receive the clock are IO_L9N_MRCC_X and IO_L10P_MRCC_X. The last two I/Os to receive a clock are IO_L0P_X and IO_L19N_X—these are at the two extreme ends of the clock tree. The arrival time of the clock to IO_9N_MRCC_X and IO_L10P_MRCC_X is roughly the same, while the arrival time to IO_L0P_X and IO_L19N_X is roughly the same.

Data/Strobe/Mask span allocation rules for Virtex-6 FPGA designs operating above 400 MHz are:

- DQS/DQS# can be no more than 7 IOB clock delay intervals from their corresponding DQ bits. This requirement is also extended to apply to the corresponding DM output.
- All data/strobe/masks in a single DQS group can span no more than 12 IOB clock delay intervals.

BUFR Allocation Rules

The BUFR allocation rules are:

- If data group pins are allocated in a column of banks, at least one bank must have a CC-P pin reserved for BUFR.
- In a column of banks, if only one bank is allocated with data group pins, the same bank has a CC-P pin reserved for BUFR.
  - The above rule is valid for x8 and x16 part designs.
  - For x4 part designs, to accommodate more data width in a single bank, BUFR is always allocated in the bank below or above the selected data group bank.
- In a column of banks, if two consecutive banks are allocated with data group pins, the data group bank (address/control bank row) has a CC-P pin reserved for BUFR.
- In a column of banks, if three consecutive banks are allocated with data group pins, the middle bank allocated for the data group pins has a CC-P pin reserved for BUFR.
- In a column of banks, if two banks are allocated with data group pins and the intervening bank (address/control bank row) is left idle, a CC-P pin is reserved in the same idle bank.
- When the data group placement uses two rows of banks, if one bank row is allocated with data group pins and the other bank row is allocated with non-data group pins (address/control group, system control group, or system clock group) both in one column, the non-data group bank row has a CC-P pin reserved for BUFR.
- When the data group placement uses three rows of banks, if at least one bank row is allocated with data group pins and the middle bank row (address/control bank row) is allocated with non-data group pins (address/control group, system control group, or system clock group) all in one column, the non-data group bank row has a CC-P pin reserved for BUFR.

For any group, adhere to this priority order for pin allocation in banks:

- In a given column of banks, the preference of pin allocation in banks is in descending order. A top-down order of banks is followed.
- In a given bank, pins are allocated in top-down order.
- The priority order of columns is:
  - Inner-left column
  - Inner-right column
  - Outer-left column
  - Outer-right column

To optimize routing for the PCB during layout (to avoid crossing of nets or buses), it might be necessary to swap pin locations depending on the number of layers available and the interface topology.

Any changes to the pin assignments require modifications to the UCF provided by the MIG tool and might also require changes to the source code. These rules apply when changing pin assignments after the MIG tool has generated a design:

- The address and control pin assignments can be swapped with each other as needed.
- DQ and DM pin assignments within the same byte can be swapped with each other. The affected bits require a change to the pin assignment LOC constraints in the UCF.

Configuration

The UCF contains timing, pin, and I/O standard information. The sys_clk constraint sets the operating frequency of the interface and is set through the MIG GUI. The MIG GUI must be rerun if this needs to be altered, because other internal parameters are affected. For example:

```
NET "sys_clk_p" TNM_NET = TNM_sys_clk;
TIMESPEC "TS_sys_clk" = PERIOD "TNM_sys_clk" 1.875 ns;
```

The clk_ref constraint sets the frequency for the IODELAY reference clock, which is typically 200 MHz. For example:

```
NET "clk_ref_p" TNM_NET = TNM_clk_ref;
TIMESPEC "TS_clk_ref" = PERIOD "TNM_clk_ref" 5 ns;
```

The I/O standards are set appropriately for the DDR3 interface with SSTL15, SSTL15_T_DCI, or DIFF_SSTL15_T_DCI, as appropriate. LVDS_25 is used for the system clock (sys_clk) and I/O delay reference clock (clk_ref). These standards can be changed, as required, for the system configuration. These signals are brought out to the top level for system connection:

- sys_rst: This is the main system reset.
- phy_init_done: This signal indicates when the internal calibration is done and that the interface is ready for use.
- error: This signal is generated by the example design's traffic generator if read data does not match the write data.

These signals are all set to LVCMOS25 and can be altered as needed for the system design. They can be generated and used internally instead of being brought out to pins.

The SCL and SDA pins are used to access the Serial Presence Detect (SPD) EEPROM located on the DIMM. These pins are not used in the current memory interface designs and are reserved for future use. The SCL and SDA pins can be used for other purposes if the user determines that access to the SPD is not required. The DDR3 RDIMM interface uses the default values for the register on the RDIMM. This is sufficient for the current set of RDIMM parts that this interface supports. If an RDIMM is used that requires specific register programming information to be extracted from the SPD, and this register programming information is not available statically on the data sheet, then the SCL and SDA pins are required. This is not expected to occur frequently.

This interface contains several pins that must be reserved for internal use and cannot be used externally for any other purpose. These pins are used for the internal BUFR, BUFIO, and clock phase monitor and are identified in the UCF as CONFIG PROHIBIT and LOC constraints. Additional information is listed in the UCF.

Termination

These rules apply to termination for DDR3 SDRAM:

- Simulation (IBIS or other) is highly recommended. The loading of address (A, BA), command (RAS_N, CAS_N, WE_N), and control (CS_N, ODT) signals depends on various factors, such as speed requirements, termination topology, use of unbuffered DIMMs, and multiple rank DIMMs, and can be a limiting factor in reaching a performance target.
- Unidirectional signals are to be terminated with the memory device's internal termination or a pull-up of 50Ω to V_{TT} at the load (Figure 1-73). A split 100Ω termination to V_{CCO} and a 100Ω termination to GND can be used (Figure 1-74), but

takes more power. For bidirectional signals, the termination is needed at both ends of the signal (DCI/ODT or external termination).

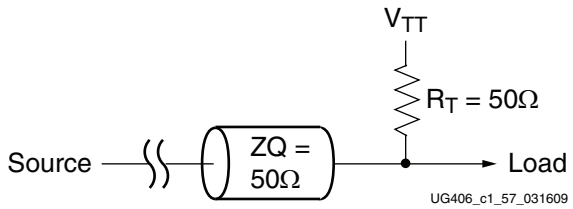


Figure 1-73: **50Ω Termination to V_{TT}**

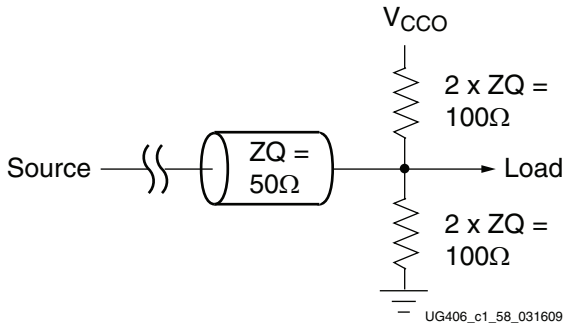


Figure 1-74: **100Ω Split Termination to V_{CCO} and GND**

- Differential signals should be terminated with the memory device's internal termination or a 100Ω differential termination at the load (Figure 1-75). For bidirectional signals, termination is needed at both ends of the signal (DCI/ODT or external termination).

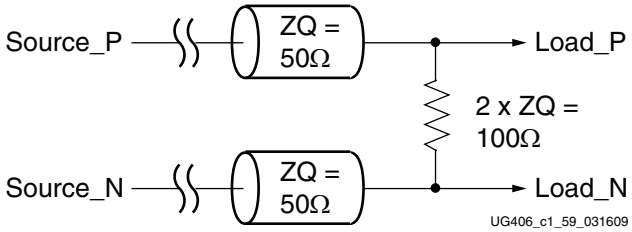


Figure 1-75: **100Ω Differential Termination**

- All termination must be placed as close to the load as possible. The termination can be placed before or after the load provided that the termination is placed within a small distance of the load pin. The allowable distance can be determined by simulation.
- DCI can be used at the FPGA as long as the DCI rules such as VRN/VRP are followed.
- The RESET and CKE signals are not terminated. These signals should be pulled down during memory initialization with a 4.7 kΩ resistor connected to GND.
- ODT, which terminates a signal at the memory, and DCI, which terminates a signal at the FPGA, are required. The MIG tool should be used to specify the configuration of the memory system for setting the mode register properly. Refer to Micron technical note TN-47-01 [Ref 5] for additional details on ODT.
- ODT applies to the DQ, DQS, and DM signals only. If ODT is used, the mode register must be set appropriately to enable ODT at the memory.

- DM should be pulled to GND if ODT is used but DM is not driven by the FPGA (for scenarios where the Data Mask not used or disabled).

I/O Standards

These rules apply to the I/O standard selection for DDR3 SDRAMs:

- Designs generated by the MIG tool use the SSTL15_T_DCI and DIFF_SSTL15_T_DCI standards for all bidirectional I/O (DQ, DQS).
- The SSTL15 and DIFF_SSTL15 standards are used for unidirectional outputs, such as control/address, and forward memory clocks.

The MIG tool creates the UCF using the appropriate standard based on input from the GUI.

Trace Lengths

The trace lengths described here are for high-speed operation and can be relaxed depending on the target bandwidth requirements of the application. The package delay should be included when determining the effective trace length. The most accurate and recommended method for determining the delay is to use the L and C values for each pin from the IBIS models. The delay value is determined as the square root of (L × C). Alternatively, a less accurate method is to use the PARTGen utility.

The PARTGen utility generates a `PKG` file that contains the package trace length in microns (µm) for every pin of the device under consideration. For example, to obtain the package delay information for a Virtex-6 FPGA LX240T-FF1156, this command should be issued:

```
partgen -v xc6vlx240tff1156
```

This generates a file named `xc6vlx240tff1156.pkg` in the current directory with package trace length information for each pin in microns. A typical 6.5 fs/micron (6.5 ps/mm) conversion formula should be used to obtain the corresponding electrical propagation delay. While applying specific trace-matching guidelines for the DDR2 SDRAM interface, this additional package delay term should be considered for the overall electrical propagation delay. Different die in the same package might have different delays for the same package pin. If this is expected, the values should be averaged appropriately. This decreases the maximum possible performance for the target device. These rules indicate the maximum skew between DDR3 SDRAM signals:

- The maximum skew between any DQ and its associated DQS/DQS# should be ±5 ps.
- The maximum skew between any address and control signals and the corresponding CK/CK# should be ±25 ps.
- DQ to DQS matching can be relaxed by the change in clock period as the frequency is lowered from the maximum. For example, the maximum supported frequency for the -2 speed grade part is 533 MHz for the center columns. The bit time at this frequency is 937.5 ps. The DQ to DQS PCB skew is allowed to be ±5 ps. If this design operated at 400 MHz, the bit time is 1250 ps. The change in period is 1250 minus 937.5 or 312.5 ps. Half of this value is 156 ps. Thus the new skew allowed is ±(156 + 5) or ±161 ps.

Write leveling is required for both DIMMs and components. Designs using multiple components should arrange the components in a fly-by routing topology similar to a DIMM where the address, control, and clocks are shared between the components and the signal arrives at each component at a different time. The data bus routing for each component should be as short as possible. Each signal should be routed on a single PCB layer to minimize discontinuities caused by additional vias.

Noise and DLL Lock

System noise should be kept to a minimum while the DDR3 SDRAM is locking its delay-locked loop (DLL). The FPGA system should be kept as quiet as possible until the phy_init_done signal is asserted, indicating that the DLL lock and interface calibration are completed. Failure to do so might result in improper calibration and data corruption.

DDR2 SDRAM

This section describes guidelines for DDR2 SDRAM designs, including bank selection, pin allocation, pin assignments, termination, I/O standards, and trace lengths.

Design Rules

Memory types, memory parts, and data widths are restricted based on the selected FPGA, FPGA speed grade, and the design frequency. The maximum frequency of the design for commercial grade parts is:

- 400 MHz (-1, -2, and -3 speed grade devices).
- 333 MHz (-1 and -2 speed grade CXT devices).
- 300 MHz (low-power Virtex-6 devices).

Note: The final maximum frequency will be determined after characterization.

For frequencies above 333 MHz, only data widths up to 72 bits are allowed. For frequencies of 333 MHz and below, data widths up to 144 bits are allowed. The listed frequency ranges are preliminary values. The final frequency ranges are subject to characterization results.

Pin Assignments

The MIG tool generates pin assignments for a memory interface based on physical layer rules.

Bank and Pin Selection Guides for DDR2 Designs

The MIG tool selects address/control banks, data banks, and pin allocations that are restricted to the rules outlined below.

Bank Selection Rules

The bank selection rules are:

- Address/control groups can be selected only in the inner column banks.
- The first bank selected for the address/control group has CK[0] and CK#[0] pins.
- Banks consisting of CK[0] and CK#[0] have MMCMs utilized in the H-row.
- For design frequencies higher than 333 MHz, only inner column banks are allowed for data group selection. For design frequencies of 333 MHz and below, both inner and outer column banks are allowed for data group selection.
- Only inner or outer column banks are allowed for selection.
- Inner and outer column banks that reside one row above, one row below, and on the same row of banks consisting of the CK[0] and CK#[0] pins are enabled for data group pin selection. This restriction is represented by a boundary box called the vicinity box.

- The system clock group can only be selected in the banks consisting of GC pins or in the inner column banks that are in the same H-row of allocated MMCMs.
- The system clock group and the rest of the design group pins (address/control group, data group, and system control group) cannot coexist in the same bank due to different voltage standards.
- A master bank must be selected for each column if DCI cascade is to be used.
- The system clock group bank cannot be selected as a master bank.
- One MMCM is always used for the design.

Pin Allocation Rules

The pin allocation rules are:

- Address/Control Group:
  - This group consists of A, BA, CK, CK#, CKE, CS#, RAS#, CAS#, WE#, ODT, and RESET# memory signals.
  - Only inner column banks are allowed for selection.
  - The memory clock signals (CK and CK#) are allocated to the differential pair pins (P-N pair).
  - The VRN/VRP pins are utilized for pin allocation. In this case, DCI cascading is applied to support the DCI standard on address/control group signals.
  - The VREF pins are utilized for pin allocation.
  - For DIMM designs, this group also includes sda and scl pins.
- Data Group:
  - The DQS group consists of the DQ and DM memory pins and their corresponding DQS and DQS# pins.
  - The DQS and DQS# pins are allocated to a P-N pair.
  - Each DQS group in a bank is associated with a CC-P pin reserved for BUFIO.
  - In a column of banks allocated with data group pins, at least one bank has a CC-P pin reserved for BUFR.
  - If VRN/VRP pins are utilized for pin allocation in a bank, the DCI cascading feature must be applied to support DCI. In this case, a master bank must be selected.
- System Clock Group:
  - This group consists of:
    - Design clocks: sys_clk_p, sys_clk_n (differential), or sys_clk (single-ended).
    - Reference clocks: clk_ref_p, clk_ref_n (differential), or clk_ref (single-ended).
    - Pins: sys_rst, error, and phy_init_done.
    - For ECC enabled designs, this group also includes the ECC app_ecc_multiple_err error pin.
  - Only inner column banks are allowed for selection.
  - The CC pins are allocated for the system clock group if the system clock bank is in the address/control bank H-row of allocated MMCMs.
  - The GC pins are allocated for the system clock group if the system clock bank is any one of banks 24, 25, 34, or 35.
  - This group is associated with the 2.5V I/O standard.

- Master Bank:
  - Except for the system clock bank, only banks within the vicinity of a given column of banks can be selected as a master bank.
  - A master bank always has unused VRN/VRP pins. Thus, in a given bank, VRN/VRP pins are reserved if they are selected in a master bank.
  - A master bank should always have at least one SSTL18_II_DCI I/O standard input pin. If not, a dummy input pin is allocated with the SSTL18_II_DCI I/O standard.
  - All data group banks in a given column act as slave banks if a master bank is selected in that column. The same is listed in the generated UCF.

BUFR Allocation Rules

The BUFR allocation rules are:

- If data group pins are allocated in a column of banks, at least one bank must have a CC-P pin reserved for BUFR.
- In a column of banks, if only one bank is allocated with data group pins, the same bank has a CC-P pin reserved for BUFR.
  - The above rule is valid for x8 and x16 part designs.
  - For x4 part designs, to accommodate more data width in a single bank, BUFR is always allocated in the bank below or above the selected data group bank.
- In a column of banks, if two consecutive banks are allocated with data group pins, the data group bank (address/control bank row) has a CC-P pin reserved for BUFR.
- In a column of banks, if three consecutive banks are allocated with data group pins, the middle bank allocated for the data group pins has a CC-P pin reserved for BUFR.
- In a column of banks, if two banks are allocated with data group pins and the intervening bank (address/control bank row) is left idle, a CC-P pin is reserved in the same idle bank.
- When the data group placement uses two rows of banks, if one bank row is allocated with data group pins and the other bank row is allocated with non-data group pins (address/control group, system control group, or system clock group) both in one column, the non-data group bank row has a CC-P pin reserved for BUFR.
- When the data group placement uses three rows of banks, if at least one bank row is allocated with data group pins and the middle bank row (address/control bank row) is allocated with non-data group pins (address/control group, system control group, or system clock group) all in one column, the non-data group bank row has a CC-P pin reserved for BUFR.

For any group, adhere to this priority order for pin allocation in banks:

- In a given column of banks, the preference of pin allocation in banks is in descending order. A top-down order of banks is followed.
- In a given bank, pins are allocated in top-down order.
- The priority order of columns is:
  - Inner-left column
  - Inner-right column
  - Outer-left column
  - Outer-right column

Selecting RTT (Nominal) Value of ODT

The user should select internal termination resistance of the memory module for DQ, DQS/DQS#, LDQS/LDQS#, UDQS/UDQS# and UDM/LDM signals on the DIMM. This improves the signal integrity of the memory channel.

To optimize routing for the PCB during layout (to avoid crossing of nets or buses), it might be necessary to swap pin locations depending on the number of layers available and the interface topology.

Any changes to the pin assignments require modifications to the UCF provided by the MIG tool and might also require changes to the source code. These rules apply when changing pin assignments after the MIG tool has generated a design:

- The address and control pin assignments can be swapped with each other as needed.
- DQ and DM pin assignments within the same byte can be swapped with each other. The affected bits require a change to the pin assignment LOC constraints in the UCF.

The MIG tool supports the Single Rank and Dual Rank parts with single slot scenarios only. Table 1-90 and Table 1-91 refer to the simulation ODT matrix of MIG supported scenarios for writes and reads, respectively.

Table 1-90: **2-Slot Simulation ODT Matrix for Writes**

Slot 1	Slot 2	Write to	Controller ODT	Slot 1		Slot 2	
				Rank 1	Rank 2	Rank 1	Rank 2
DR(1)	Empty	Slot 1	OFF	120Ω	ODT OFF	na(2)	na
Empty	DR	Slot 2	OFF	na	na	120Ω	ODT OFF
SR(3)	Empty	Slot 1	OFF	120Ω	na	na	na
Empty	SR	Slot 2	OFF	na	na	120Ω	na

Notes:

1. DR: Dual Rank.
2. na: not applicable.
3. SR: Single Rank.

Table 1-91: **2-Slot Simulation ODT Matrix for Reads**

Slot 1	Slot 2	Write to	Controller ODT	Slot 1		Slot 2	
				Rank 1	Rank 2	Rank 1	Rank 2
DR(1)	Empty	Slot 1	60Ω	ODT OFF	ODT OFF	na(2)	na
Empty	DR	Slot 2	60Ω	na	na	ODT OFF	ODT OFF
SR(3)	Empty	Slot 1	60Ω	ODT OFF	na	na	na
Empty	SR	Slot 2	60Ω	na	na	ODT OFF	na

Notes:

1. DR: Dual Rank.
2. na: not applicable.
3. SR: Single Rank.

Configuration

The UCF contains timing, pin, and I/O standard information. The sys_clk constraint sets the operating frequency of the interface and is set through the MIG GUI. The MIG GUI

must be rerun if this needs to be altered, as other internal parameters are affected. For example:

```
NET "sys_clk_p" TNM_NET = TNM_sys_clk;
TIMESPEC "TS_sys_clk" = PERIOD "TNM_sys_clk" 1.875 ns;
```

The clk_ref constraint sets the frequency for the IODELAY reference clock which is typically 200 MHz. For example:

```
NET "clk_ref_p" TNM_NET = TNM_clk_ref;
TIMESPEC "TS_clk_ref" = PERIOD "TNM_clk_ref" 5 ns;
```

The I/O standards are set appropriately for the DDR2 interface with the SSTL18 variants as determined through the MIG GUI. LVDS_25 is used for the system clock (sys_clk) and I/O delay reference clock (clk_ref). These standards can be changed, as required, for the system configuration. These signals are brought out to the top level for system connection:

- sys_rst: This is the main system reset.
- phy_init_done: This signal indicates when the internal calibration is done and that the interface is ready for use.
- error: This signal is generated by the example design's traffic generator if read data does not match the write data.

These signals are all set to LVCMOS25 and can be altered as needed for the system design. They can be generated and used internally instead of being brought out to pins.

The SCL and SDA pins are used to access the SPD EEPROM located on the DIMM. These pins are not used in the current memory interface designs and are reserved for future use. The SCL and SDA pins can be used for other purposes if the user determines that access to the SPD is not required. The DDR3 RDIMM interface uses the default values for the register on the RDIMM. This is sufficient for the current set of RDIMM parts that this interface supports. If an RDIMM is used that requires specific register programming information to be extracted from the SPD, and this register programming information is not available statically on the data sheet, then the SCL and SDA pins are required. This is not expected to occur frequently.

This interface contains several pins that must be reserved for internal use and cannot be used externally for any other purpose. These pins are used for the internal BUFR, BUFIO, and clock phase monitor and are identified in the UCF as CONFIG PROHIBIT and LOC constraints. Additional information is listed in the UCF.

Termination

These rules apply to termination for DDR2 SDRAM:

- Simulation (using IBIS or other) is highly recommended. The loading of address (A, BA), command (RAS_N, CAS_N, WE_N), and control (CS_N, ODT) signals depend on various factors, such as speed requirements, termination topology, use of unbuffered DIMMs, and multiple rank DIMMs, and can be a limiting factor in reaching a performance target.
- Unidirectional signals should be terminated with the memory device's internal termination or a pull-up of 50Ω to V_{TT} at the load (Figure 1-73, page 140). A split 100Ω termination to V_{CCO} and a 100Ω termination to GND can be used (Figure 1-74, page 140), but takes more power. For bidirectional signals, the termination is needed at both ends of the signal (DCI/ODT or external termination).
- Differential signals should be terminated with the memory device's internal termination or a 100Ω differential termination at the load (Figure 1-75, page 140). For

bidirectional signals, termination is needed at both ends of the signal (DCI/ODT or external termination).

- All termination must be placed as close to the load as possible. The termination can be placed before or after the load provided that the termination is placed within a small distance of the load pin. The allowable distance can be determined by simulation.
- DCI can be used at the FPGA as long as the DCI rules such as VRN/VRP are followed.
- For DIMMs, the CK signals should be terminated by a 5 pF capacitor between the two legs of the differential signal instead of the 100Ω resistor termination (Figure 1-76). This is because the CK signals are already terminated on each DIMM.

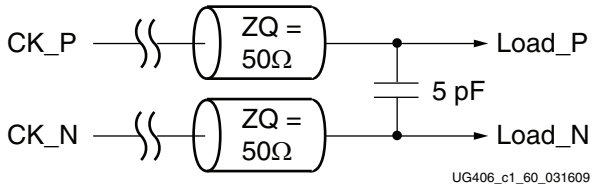


Figure 1-76: **5 pF Differential Termination on Clocks**

- The ODT and CKE signals are not terminated. These signals should be pulled down during memory initialization with a 4.7 kΩ resistor connected to GND.
- ODT applies to the DQ/DQS/DM signals only. If ODT is used, the Mode register must be set appropriately in the RTL design. DM should be pulled to GND if ODT is used but DM is not driven by the FPGA (for scenarios where the Data Mask not used or disabled). To save board space, DCI, which terminates a signal at the FPGA, and ODT, which terminates a signal at the memory can be used to minimize the number of external resistors on the board.

I/O Standards

These rules apply to the I/O standard selection for DDR2 SDRAMs:

- Designs generated by the MIG tool use the SSTL18_II I/O standard by default for all memory interface signals. When DCI is selected in the MIG tool, DCI for SSTL18_II is applied on input, output, and in-out memory interface signals.
- The SSTL18_I/II standards can be selected from the MIG tool. When SSTL18_I is selected, the I/O standard for bidirectional signals remains SSTL18_II.
- When DCI is selected in the MIG tool for SSTL18_I, the DCI I/O standard is applied only to memory interface signals that are inputs or in-outs to the FPGA.

Trace Lengths

The trace lengths described here are for high-speed operation and can be relaxed depending on the target bandwidth requirements of the application. The package delay should be included when determining the effective trace length. The most accurate and recommended method for determining the delay is to use the L and C values for each pin from the IBIS models. The delay value is determined as the square root of (L × C). Alternatively, a less accurate method is to use the PARTGen utility.

The PARTGen utility generates a `PKG` file that contains the package trace length in microns for every pin of the device under consideration. For example, to obtain the package delay information for a Virtex-6 FPGA LX240T-FF1156, this command should be issued:

```
partgen -v xc6vlx240tff1156
```

This generates a file named `xc6vlx240tff1156.pkg` in the current directory with package trace length information for each pin in microns. A typical 6.5 fs/micron (6.5 ps/mm) conversion formula should be used to obtain the corresponding electrical propagation delay. While applying specific trace-matching guidelines for the DDR2 SDRAM interface, this additional package delay term should be considered for the overall electrical propagation delay. Different die in the same package might have different delays for the same package pin. If this is expected, the values should be averaged appropriately. This decreases the maximum possible performance for the target device.

These rules indicate the maximum skew between DDR2 SDRAM signals at 400 MHz:

- The maximum skew between any DQ and its associated DQS/DQS# should be ±25 ps.
- The maximum skew between any address and control signals and the corresponding CK/CK# should be ±50 ps.
- The maximum skew between any DQS/DQS# and CK/CK# should be ±100 ps.
- DQ to DQS matching can be relaxed by the change in clock period as the frequency is lowered from the maximum. For example, the maximum supported frequency for the -2 speed grade part is 400 MHz for the center columns. The bit time at this frequency is 1250 ps. The DQ to DQS PCB skew is allowed to be ±25 ps. If this design operated at 333 MHz, the bit time is 1500 ps. The change in period is 1500 minus 1250, or 250 ps. Half of this value is 125 ps. Thus the new skew allowed is ±(125 + 25) or ±150 ps.

Noise and DLL Lock

System noise should be kept to a minimum while the DDR3 SDRAM is locking its DLL. The FPGA system should be kept as quiet as possible until the phy_init_done signal is asserted, indicating that the DLL lock and interface calibration are completed. Failure to do so might result in improper calibration and data corruption.

Pin Mapping for x4 RDIMMs

Table 1-92 shows an example pin mapping for x4 DDR2 and DDR3 registered DIMMs between the memory data sheet and the user constraints file (UCF).

Table 1-92: **Pin Mapping for x4 DDR2 DIMMs**

Memory Data Sheet	MIG UCF
DQ[63:0]	DQ[63:0]
CB3 - CB0	DQ[67:64]
CB7 - CB4	DQ[71:68]
DQS0, DQS0	DQS[0], DQS_N[0]
DQS1, DQS1	DQS[2], DQS_N[2]
DQS2, DQS2	DQS[4], DQS_N[4]
DQS3, DQS3	DQS[6], DQS_N[6]
DQS4, DQS4	DQS[8], DQS_N[8]
DQS5, DQS5	DQS[10], DQS_N[10]
DQS6, DQS6	DQS[12], DQS_N[12]
DQS7, DQS7	DQS[14], DQS_N[14]

Table 1-92: **Pin Mapping for x4 DDR2 DIMMs *(Cont'd)***

Memory Data Sheet	MIG UCF
DQS8, DQS8	DQS[16], DQS_N[16]
DQS9, DQS9	DQS[1], DQS_N[1]
DQS10, DQS10	DQS[3], DQS_N[3]
DQS11, DQS11	DQS[5], DQS_N[5]
DQS12, DQS12	DQS[7], DQS_N[7]
DQS13, DQS13	DQS[9], DQS_N[9]
DQS14, DQS14	DQS[11], DQS_N[11]
DQS15, DQS15	DQS[13], DQS_N[13]
DQS16, DQS16	DQS[15], DQS_N[15]
DQS17, DQS17	DQS[17], DQS_N[17]

Debugging Virtex-6 FPGA DDR2/DDR3 SDRAM Designs

This section defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the memory interface design process.

Introduction

The DDR2 and DDR3 SDRAM memory interfaces in the Virtex-6 FPGA simplify the challenges associated with memory interface design. However, every application environment is unique and proper due diligence is required to ensure a robust design. Careful attention must be given to functional testing through simulation, proper synthesis and implementation, adherence to PCB layout guidelines, and board verification through IBIS simulation and signal integrity analysis.

This section defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the design process. Details are provided on:

- Functional verification using the UNISIM simulation models
- Design implementation verification
- Board layout verification
- Using the DDR2/DDR3 PHY to debug board-level issues
- General board-level debug techniques
- PHY layer debug port signal descriptions

The two primary issues encountered during verification of a memory interface are:

- Calibration not completing properly
- Data corruption during normal operation

Problems might be seen in simulation, hardware, or both due to various root causes. Figure 1-77 shows the overall flow for debugging problems associated with these two general types of issues.

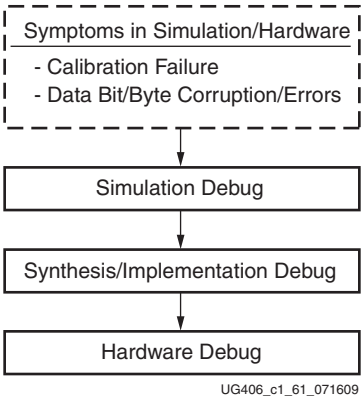


Figure 1-77: **Virtex-6 FPGA DDR2/DDR3 SDRAM Debug Flowchart**

Debug Tools

Many tools are available to debug memory interface design issues. This section indicates which resources are useful for debugging a given situation.

Example Design

Generation of a DDR2 or DDR3 design through the MIG tool produces an example design and a user design. The example design includes a synthesizable test bench with a traffic generator that is fully verified in simulation and hardware. This example design can be used to observe the behavior of the MIG design and can also aid in identifying board-related problems. See Quick Start Example Design, page 52 for complete details on the example design. This debug section further describes using the example design to verify setup of a proper simulation environment and to perform hardware validation.

Debug Signals

The MIG tool includes a Debug Signals Control option on the FPGA Options screen. Enabling this feature allows calibration, tap delay, and read data signals to be monitored using the ChipScope analyzer. Selecting this option port maps the debug signals to VIO modules of the ChipScope analyzer in the design top module. See Getting Started with the CORE Generator Software, page 11 for details on enabling this debug feature. The debug port is disabled for functional simulation and can only be enabled if the signals are actively driven by the user design.

Reference Boards

The ML605 evaluation kit is a Xilinx development board that interfaces to a DDR3 SODIMM. This board can be used to test user designs and analyze board layout.

ChipScope Pro Tool

The ChipScope Pro tool inserts logic analyzer, bus analyzer, and VIO software cores directly into the design. The ChipScope Pro tool allows the user to set trigger conditions to capture application and MIG signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool [Ref 6].

Simulation Debug

Figure 1-78 shows the debug flow for simulation.

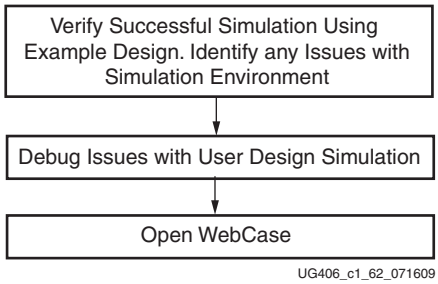


Figure 1-78: **Simulation Debug Flowchart**

Verifying the Simulation Using the Example Design (for Designs with the Standard User Interface)

The example design generated by the MIG tool includes a simulation test bench, appropriately set up memory model and parameter file based on memory selection in the MIG tool, and a ModelSim DO script file. See Quick Start Example Design, page 52 for detailed steps on running the example design simulation. Successful completion of this example design simulation verifies a proper simulation environment. This shows that the simulation tool and Xilinx libraries are set up correctly. For detailed information on setting up Xilinx libraries, refer to COMPXLIB in the *Command Line Tools User Guide* [Ref 7] and the *Synthesis and Simulation Design Guide* [Ref 8]. For simulator tool support, refer to the *Virtex-6 FPGA Memory Interface Solutions Data Sheet* [Ref 9].

A working example design simulation completes memory initialization and runs traffic in response to the traffic generator stimulus. Successful completion of memory initialization and calibration results in the assertion of the phy_init_done signal. When this signal is asserted, the traffic generator takes control and begins executing writes and reads according to its parameterization. Refer to Quick Start Example Design, page 52 for details on the available traffic generator data patterns and corresponding top-level parameters.

Table 1-93 and Table 1-94 show the signals and parameters of interest, respectively, during simulation.

Table 1-93: **Signals of Interest During Simulation**

Signal Name	Usage
phy_init_done	This signal indicates completion of calibration.
error	This signal indicates a mismatch between the data written from the UI and data received during a read on the UI. This signal is a part of the example design.
app_en	This signal should be enabled for the app_addr, app_cmd, app_sz, and app_hi_pri inputs.
app_cmd	This is the command for the current request.
app_addr	This is the address for the current request.
app_wdf_data	This is the data provided from the UI for write commands.
app_wdf_en	This is the enable for data available on app_wdf_data.

Table 1-93: **Signals of Interest During Simulation *(Cont'd)***

Signal Name	Usage
app_wdf_end	This signal indicates the last cycle of data on app_wdf_data for the current write.
app_rd_data	This is the read data returned to the UI.
app_rd_data_end	This signal indicates the last cycle of data on app_rd_data for the current read.
app_rd_data_valid	This signal indicates that the data on app_rd_data is valid.

Table 1-94: **Parameters of Interest During Simulation**

Parameter Name	Usage
SIM_BYPASS_INIT_CAL	This parameter sets the simulation initialization and calibration procedures.
END_ADDRESS	This is the end address for the traffic generator example design.
PRBS_EADDR_MASK_POS	This is the end address for PRBS data.
ORDERING	This parameter turns the reordering controller logic on and off.

The top-level simulation test bench (`sim_tb_top.v/vhd`) provided with the traffic generator sets certain parameters to significantly reduce simulation time. When SIM_BYPASS_INIT_CAL is set to "FAST", the MIG design skips the initial 200 µs initialization delay and executes an abbreviated calibration sequence.

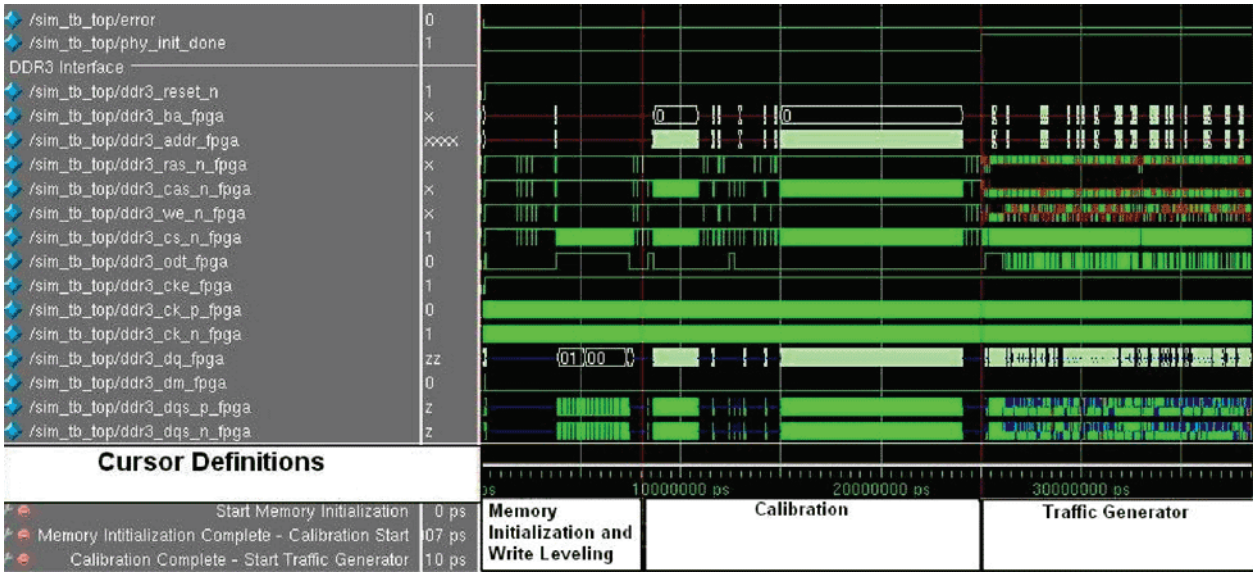
For the entire calibration to complete, set the SIM_BYPASS_INIT_CAL parameter to "OFF." For this setting, read leveling performs multiple iterations and applies averaged results. This situation is impractical to achieve in simulations and the "OFF" value is intended for hardware implementations only. To see the complete calibration sequence in simulation, except for the multiple read leveling iterations, set these parameters as indicated:

`SIM_BYPASS_INIT_CAL = "OFF"` in sim_tb_top

`SIM_CAL_OPTION = "FAST_WIN_DETECT"` in phy_top module

When END_ADDRESS is set to `32'h000003FF` and PRBS_EADDR_MASK_POS is set to `32'hFFFFFC00`, the address range accessed in the memory model is shortened. If the full address space is to be accessed, the MEM_BITS memory model parameter must be increased to avoid memory overflow errors. These parameters should only be set to the above values in the simulation test bench. The top-level MIG design file (`example_top.v/vhd`) cannot use any abbreviated values for these parameters in order for the design to properly initialize, calibrate, and access the full memory array in hardware. The MIG output properly sets the abbreviated values in the test bench and the full range of values in the top-level design module.

Figure 1-79 shows a high-level view of a successful simulation using the provided example design with the abbreviated simulation parameter settings from Table 1-88.



UG406_c1_63_071709

Figure 1-79: **Successful Simulation of Example Design**

The simulation can be divided into these main sections: clock phase calibration, memory initialization, write leveling, calibration, and execution of the traffic generator.

Memory Initialization

Memory initialization skips the initial 200 µs delay but completes all remaining steps as defined by the DDR2/DDR3 JEDEC specification. Successful completion of memory initialization generates a message similar to the following in the simulator transcript:

```
# PHY_INIT: Memory Initialization completed at x ps
```

Write Leveling (DDR3 SDRAM Only)

To satisfy the tDQSS requirement, write leveling is performed to compensate for skew between DQS and CK. Write leveling starts after memory initialization completes. Successfully completion of write leveling generates a message similar to the following in the simulator transcript:

```
# PHY_INIT: Write Leveling completed at x ps
```

Calibration

Calibration completes read leveling, write calibration, and read enable calibration. This is completed over two stages. This sequence successfully completes when the phy_init_done signals asserts. For more details, refer to PHY, page 102.

The first stage performs per-byte read leveling calibration. The data pattern used during this stage is `FF00FF00FF00`. The data pattern is first written to the memory (Figure 1-80).

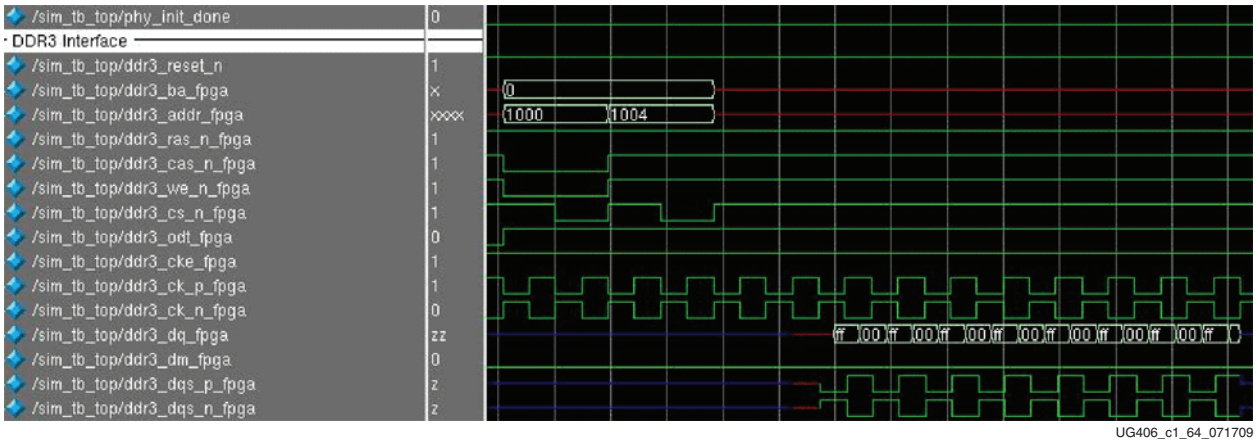


Figure 1-80: **First Stage of Calibration: Data Write to Memory**

This pattern is then continuously read back while the per-byte calibration completes (Figure 1-81).

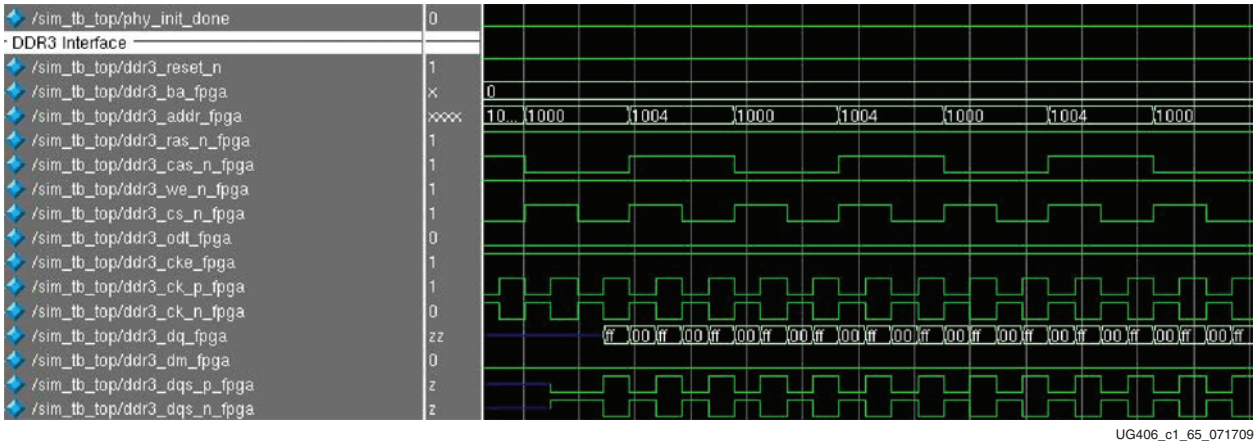


Figure 1-81: **First Stage of Calibration: Data Read from Memory**

Successful completion of this first stage of calibration generates a message similar to the following in the simulator transcript:

```
# PHY_INIT: Read Leveling Stage 1 completed at x ps
```

The second stage performs write calibration and read enable calibration. The data pattern used during this stage is `FF00AA55AA9966`. The data pattern is first written to the memory (Figure 1-82).

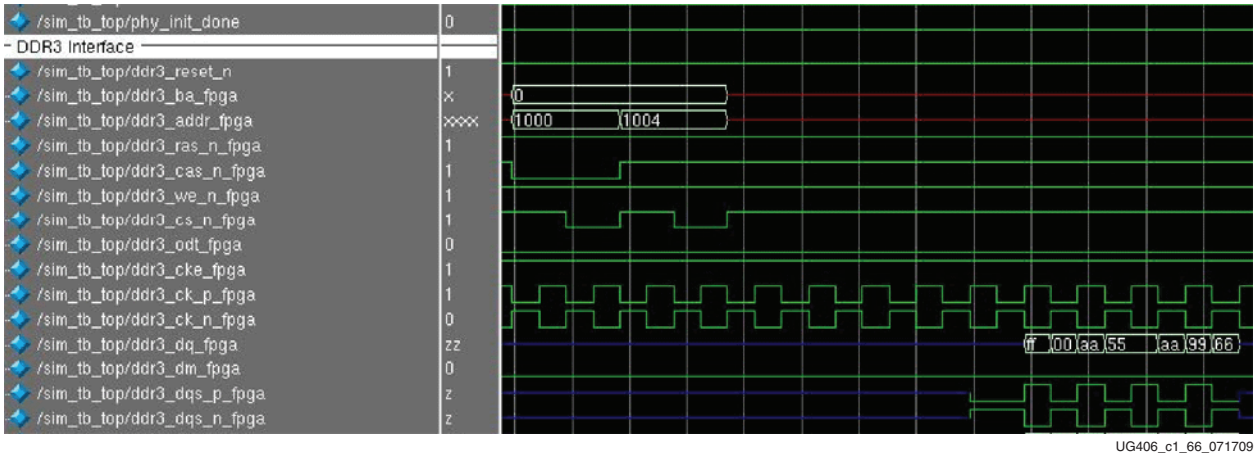


Figure 1-82: **Second Stage of Calibration: Data Write to Memory**

The data pattern is then continuously read back while the write calibration and read enable calibration completes (Figure 1-83).

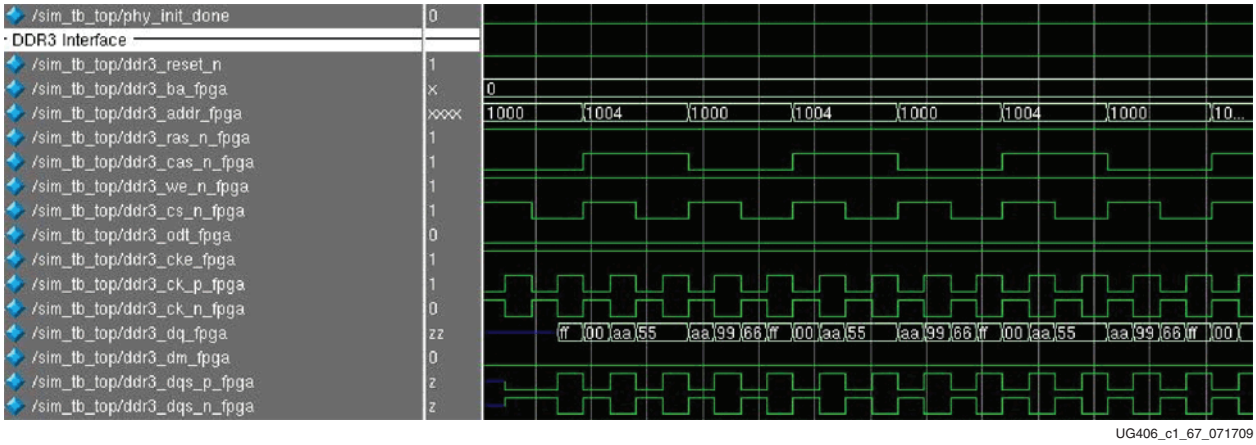


Figure 1-83: **Second Stage of Calibration: Data Read from Memory**

Successful completion of this second stage of calibration generates a message similar to the following in the simulator transcript:

```
# PHY_INIT: Read Leveling Stage 2 completed at x ps
```

The phy_init_done signal is asserted, signifying successful completion of the entire calibration process. A read command is issued as part of the phase detector for continuous calibration. See Phase Detector, page 111 for more details. Successful completion of this initial phase detector calibration generates a message similar to the following in the simulator transcript:

```
# PHY_INIT: Phase Detector Initial Cal completed at x ps
```

Traffic Generator

After phy_init_done is asserted and the initial phase detector read is performed, the traffic generator takes control of writing to and reading from the memory. The data written is compared to the data read back, and any mismatches trigger the error signal to be asserted. Figure 1-84 shows successful implementation of the traffic generator with no assertions on the error signal.

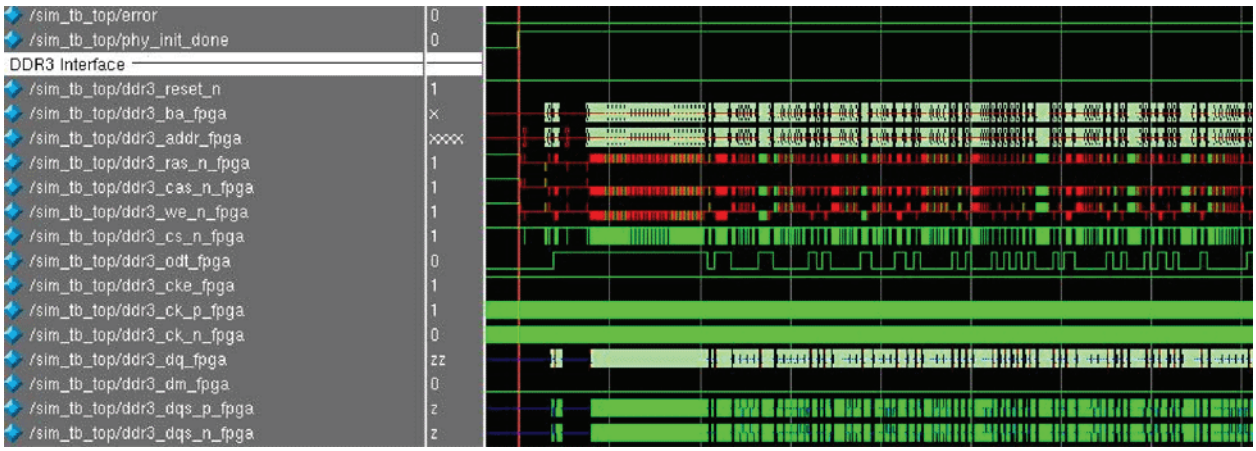


Figure 1-84: **Successful Implementation of the Traffic Generator**

Debug Issues with User Design Simulation

After the simulation environment and parameter settings are verified by successful simulation of the example design, issues with the user design simulation can be investigated. Because the environment and parameters are verified to work properly, calibration in the user design completes without error as long as no RTL changes exist.

Data Errors

Issues that might be seen with user design simulation exist within the generation of user writes and reads. Thus, it is crucial to understand how to drive the UI to properly send writes and reads. For more information, refer to User Interface, page 68 and Interfacing to the Core, page 113.

When providing addresses to the UI on app_addr, the controller is expecting a flat address. The address must be provided linearly on app_addr as rank address (when used), bank address, row address, and column address. Bit 10 needs to be provided as an address location. The controller accounts for setting the A10 auto-precharge bit when appropriate.

When sending write and read commands, the corresponding UI inputs should be properly asserted and deasserted. For more information, refer to User Interface, page 68 and Interfacing to the Core, page 113. The traffic generator design provided within the example design can be used as a further source of proper behavior on the UI.

To debug data errors seen on the DDR2 or DDR3 interface, UI signals must be pulled into the simulation waveform. In the ModelSim Instance window, highlight u_ip_top (Figure 1-85). The necessary UI signals are then shown in the Objects window. Highlight the UI signals noted in Table 1-93, page 151 and Table 1-94, page 152, right-click, and select **Add → To Wave → Selected Signals**.

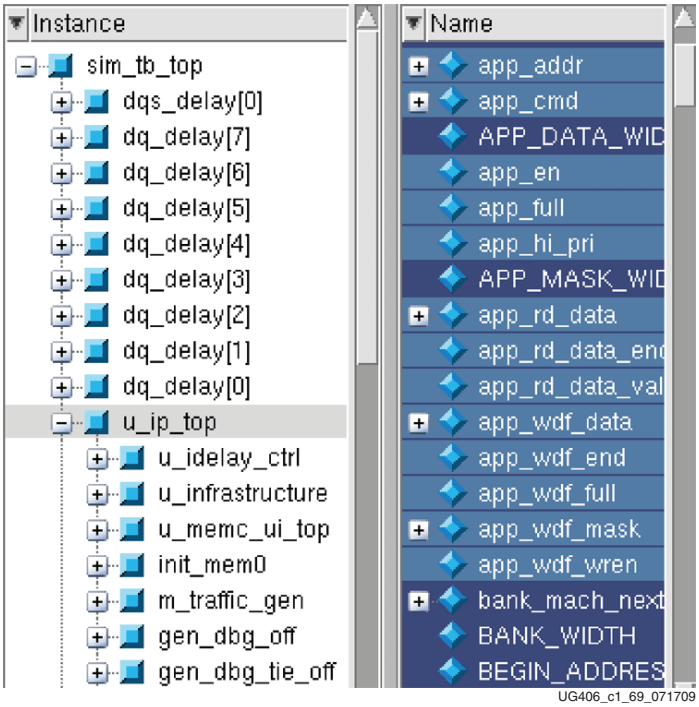


Figure 1-85: **ModelSim Instance Window**

Figure 1-86 provides an example of a write on the UI.

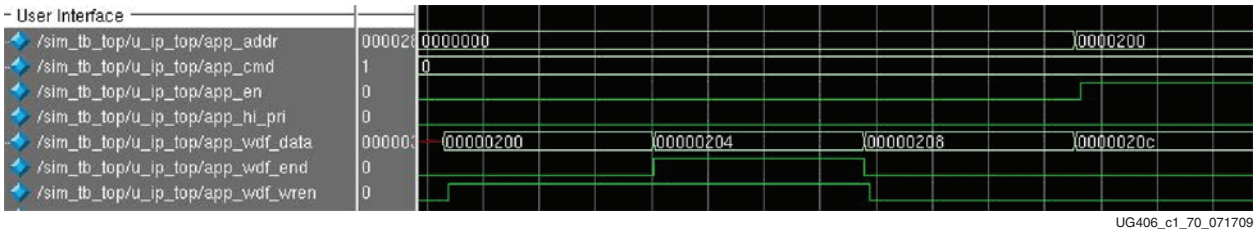


Figure 1-86: **User Interface Write**

Figure 1-87 provides an example of a write on the DDR interface.

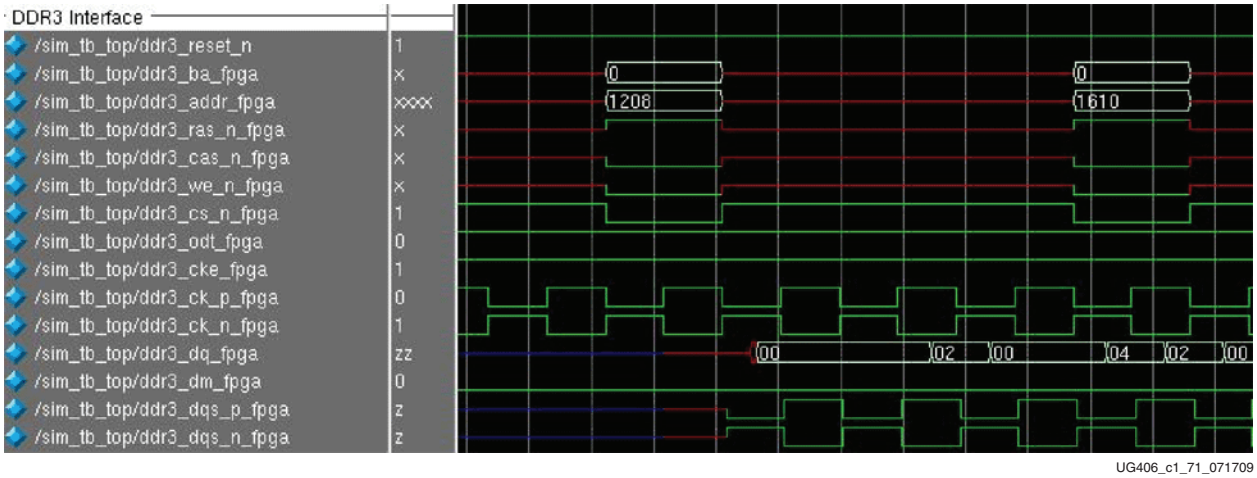


Figure 1-87: **DDR Interface Write**

Figure 1-88 provides an example of a read on the UI.

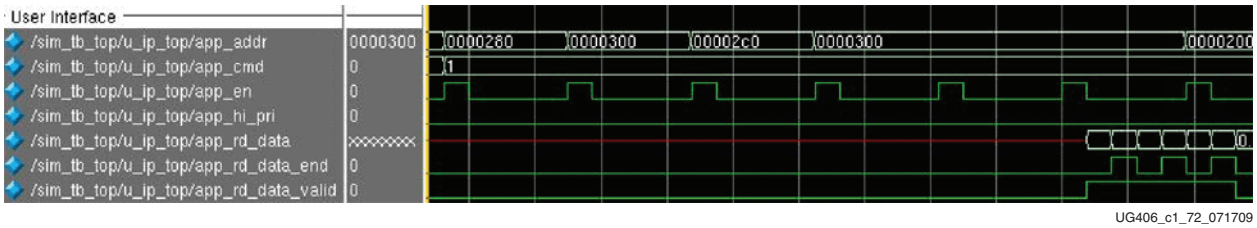


Figure 1-88: **User Interface Read**

Figure 1-89 provides an example of a read on the DDR interface.

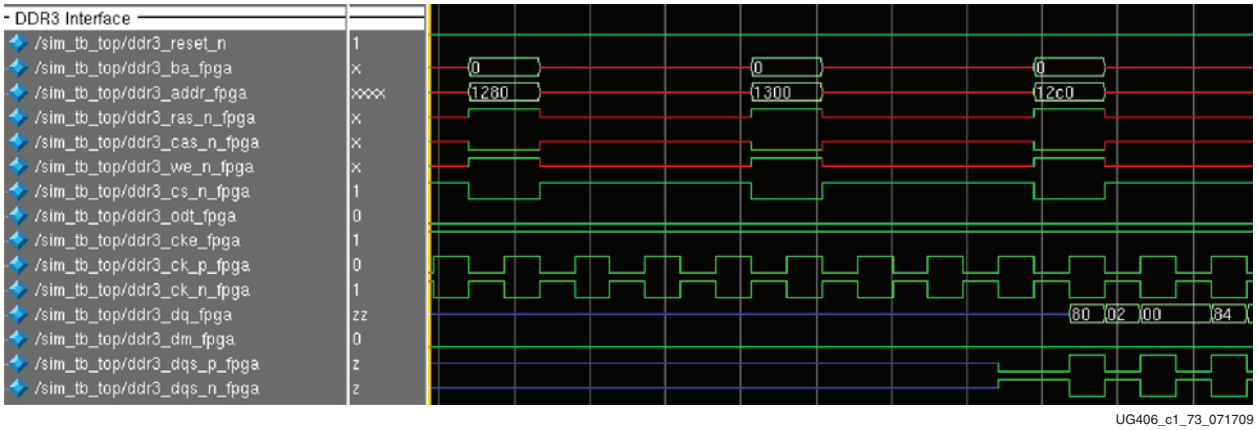


Figure 1-89: **Read on DDR Interface**

The controller uses reordering logic by default. See Reordering, page 97 for more details. Because of this, the order of commands sent on the UI might or might not match the order of commands seen on the DDR interface. The reordering algorithm reorders commands on the DDR interface to improve efficiency. However, the data seen on the UI is reordered

back to the order requested by the user. Because of this, the DDR and UI might not match. It might be useful to turn reordering on and off for debugging purposes or to study efficiency improvements for specific traffic patterns. This is controlled with the top-level parameter ORDERING. By default, this is set to NORM, which uses the full capability of the reordering algorithms. To turn reordering off in simulation, the ORDERING parameter in the simulation test bench (`sim_tb_top.v`) should be set to STRICT.

Verifying the Simulation Using the Example Design (for Designs with the AXI4 User Interface)

The example design generated when the AXI4 interface is selected as the user interface is different compared to the traffic generator standard user interface. The intent of this synthesizable test bench is to verify the AXI4 transactions as well as the memory controller transactions. However, this test bench does not verify all memory controller features and is aimed at verifying the AXI4 SHIM features. Table 1-95 shows the signals of interest during verification of the AXI4 test bench. These signals can be found in the example_top module.

Table 1-95: **Signals of Interest During Simulation for AXI4 Test Bench**

Signal	Description
test_cmptd	When asserted, this signal indicates that the current round of tests with random reads and writes was completed. This signal is deasserted when a new test starts.
write_cmptd	This signal is asserted for one clock, indicating that the current write transaction was completed.
cmd_err	When asserted, this signal indicates that the command phase of the AXI4 transaction (read or write) had an error.
write_err	When asserted, this signal indicates that the write transaction to memory resulted in error.
dbg_wr_sts_vld	When asserted, this signal indicates a valid status for the write transaction on the dbg_wr_sts bus. This signal is asserted even if the write transaction does not complete.
dbg_wr_sts	This signal indicates the status of the write transaction. The details of the status are given in Table 1-96.
read_cmptd	This signal is asserted for one clock, indicating that the current read transaction was completed.
read_err	When asserted, this signal indicates that the read transaction to the memory resulted in error.
dbg_rd_sts_vld	When asserted, this signal indicates a valid status for the read transaction on the dbg_rd_sts bus. This signal is asserted even if the read transaction does not complete.
dbg_rd_sts	This signal indicates the status of the read transaction. The details of the status are given in Table 1-97.

The initialization and the calibration sequence remain the same as that indicated in Verifying the Simulation Using the Example Design (for Designs with the Standard User Interface), page 151. Figure 1-90 shows the status generated for a write transaction.

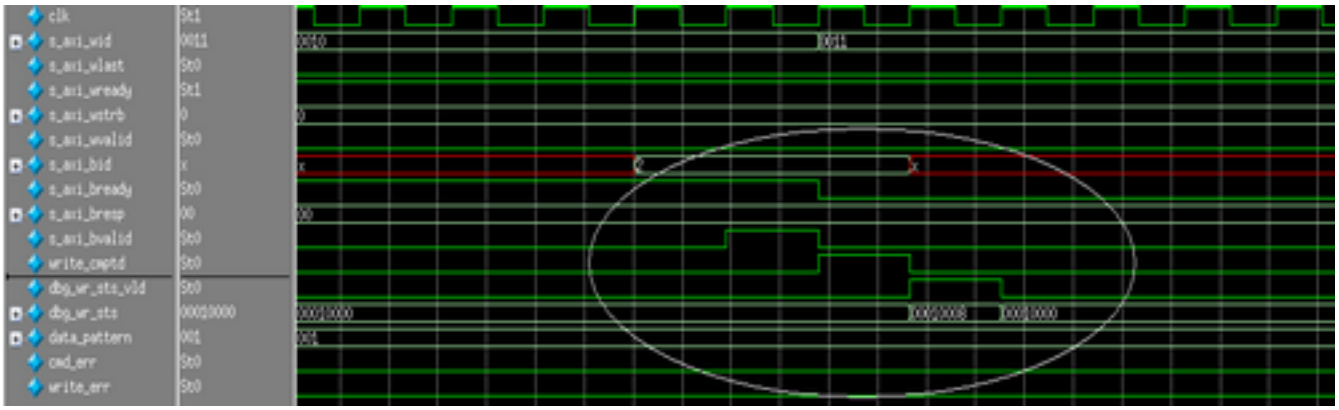


Figure 1-90: **Status for the Write Transaction**

Table 1-96: **Debug Status for the Write Transaction**

Bits	Status Description
1:0	The write response received for AXI
5:2	Response ID for the write response
8:6	AXI wrapper write FSM state when time-out (watchdog timer should be enabled) occurred: • `3'b001` - Data write transaction • `3'b010` - Waiting for acknowledgment for written data • `3'b011` - Dummy data write transaction • `3'b100` - Waiting for response from the response channel
15:9	Reserved
16	Command error occurred during write transaction
17	Write error occurred, the write transaction could not be completed
20:18	Data pattern used for the current transaction • `3'b000` - `5A` and `A5` • `3'b001` - PRBS pattern • `3'b010` - Walking zeros • `3'b011` - Walking ones • `3'b100` - All ones • `3'b101` - All zeros
31:21	Reserved

Figure 1-91 shows the status generated for a read transaction.

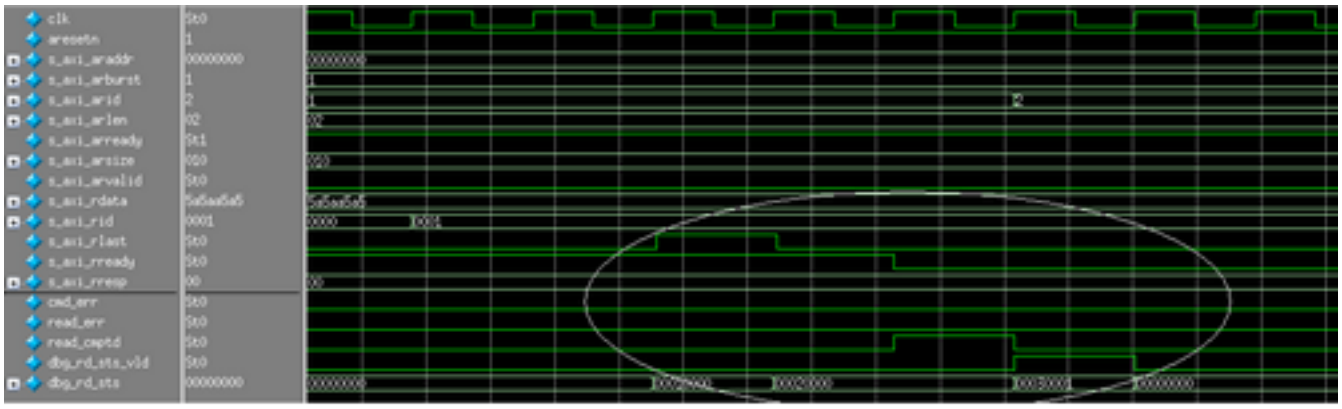


Figure 1-91: **Status for the Read Transaction**

Table 1-97: **Debug Status for the Read Transaction**

Bits	Status Description
0	Read error response on AXI
1	Incorrect response ID presented by the AXI slave
3:2	AXI wrapper read FSM state when time-out (watchdog timer should be enabled) occurred: • `2'b01` - Read command transaction • `2'b10` - Data read transaction
15:4	Reserved
16	Command error occurred during read transaction
17	Read error occurred, read transaction could not be completed
18	Data mismatch occurred between the written data and read data
26:19	Pointer value for which the mismatch occurred
29:27	Data pattern used for the current check: • `3'b000` - `5A` and `A5` • `3'b001` - PRBS pattern • `3'b010` - Walking zeros • `3'b011` - Walking ones • `3'b100` - All ones • `3'b101` - All zeros
31:30	Reserved

The calibration and other DDR data read and write transactions are similar to what is described in the previous sections. The AXI4 write and read transactions are started only after the phy_init_done signal is asserted.

Synthesis and Implementation Debug

Figure 1-92 shows the debug flow for synthesis and implementation.

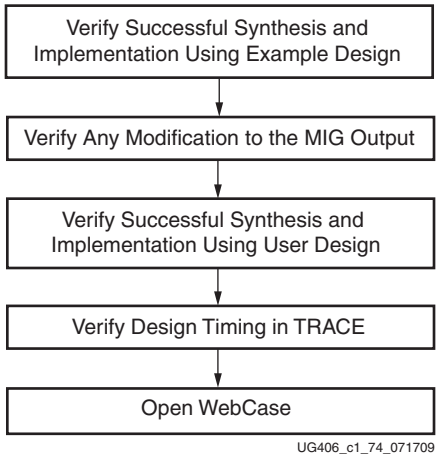


Figure 1-92: **Synthesis and Implementation Debug Flowchart**

Verify Successful Synthesis and Implementation

The example design and user design generated by the MIG tool include synthesis/implementation script files and user constraint files (`.ucf`). These files should be used to properly synthesize and implement the targeted design and generate a working bitstream. The synthesis/implementation script file, called `ise_flow.bat`, is located in both the `example_design/par` and `user_design/par` directories. Execution of this script runs either the example design or the user design through synthesis, translate, MAP, PAR, TRACE, and BITGEN. The options set for each of these processes are the only ones that have been tested with the DDR2 and DDR3 MIG designs. A successfully implemented design completes all processes with no errors (including zero timing errors).

Verify Modifications to the MIG Output

The MIG tool allows the user to select the FPGA banks for the memory interface signals. Based on the banks selected, the MIG tool outputs a UCF with all required location constraints. This file is located in both the `example_design/par` and `user_design/par` directories and should not be modified.

The MIG tool outputs open source RTL code parameterized by top-level HDL parameters. These parameters are set by the MIG tool and should not be modified manually. If changes are required, such as increasing or decreasing the frequency, the MIG tool should be rerun to create an updated design. Manual modifications are not supported and should be verified independently in behavioral simulation, synthesis, and implementation.

Identifying and Analyzing Timing Failures

The MIG DDR2 and DDR3 designs have been verified to meet timing using the example design across a wide range of configurations. However, timing violations might occur, for example, when integrating the MIG design with the user's specific application logic. Any timing violations that are encountered must be isolated. The timing report output by TRACE (`.twx/.twr`) should be analyzed to determine if the failing paths exist in the MIG DDR2/DDR3 design or the UI (backend application) to the MIG design. If failures are

encountered, the user must ensure that the build options (that is, XST, MAP, PAR) specified in the `ise_flow.bat` file are used.

If failures still exist, Xilinx has many resources available to aid in closing timing. The PlanAhead™ tool [Ref 10] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 11] provides valuable information on all available Xilinx constraints.

Hardware Debug

Figure 1-93 shows the debug flow for hardware.

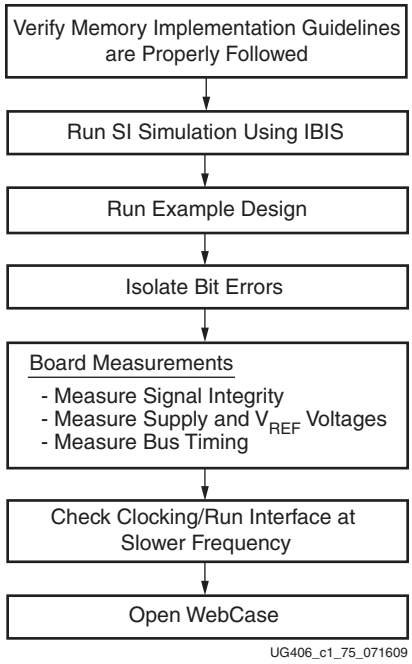


Figure 1-93: **Hardware Debug Flowchart**

Verifying Design Guidelines

See Design Guidelines, page 135 for specifications on termination, I/O standards, and trace matching. The guidelines provided therein are specific to both the DDR2 and DDR3 designs. It is important to verify that these guidelines have been referred to during board layout. Failure to follow these guidelines or modifications to a MIG tool provided pinout, or both, can result in problematic behavior in hardware as discussed in this debugging section.

Clocking

The external clock source should be measured to ensure frequency, stability (jitter), and usage of the expected FPGA pin. The designer must ensure that the design follows all clocking guidelines. If clocking guidelines have been followed, the interface should be run at a slower speed. Not all designs or boards can accommodate slower speeds. Lowering the frequency increases the marginal setup or hold time, or both, due to PCB trace mismatch, poor signal integrity, or excessive loading. When lowering the frequency, the MIG tool should be rerun to regenerate the design with the lower clock frequency. Portions

of the calibration logic are sensitive to the CLK_PERIOD parameter; thus, manual modification of the parameter is discouraged.

Verifying Board Pinout

The user should ensure that the pinout provided by the MIG tool is used without modification. There are a few pin swaps that are supported. These pin modifications are detailed in Pin Assignments, page 135. Then, the board schematic should be compared to the `<design_name>.pad` report generated by PAR. This step ensures that the board pinout matches the pins assigned in the implemented design.

Running Signal Integrity Simulation with IBIS Models

To verify that board layout guidelines have been followed, signal integrity simulations must be run using the I/O buffer information specification (IBIS). These simulations should always be run for both pre-board and post-board layouts. The purpose of running these simulations is to confirm the signal integrity on the board.

The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 12] can be used as a guideline. This chapter provides a detailed look at signal integrity correlation results for the ML561 board. It can be used as an example for signal integrity analysis. It also provides steps to create a design-specific IBIS model to aid in setting up the simulations. While this guide is specific to Virtex-5 devices and the ML561 development board, the principles therein can be applied to Virtex-6 FPGA MIG designs.

Running the Example Design

The MIG tool provided example design is a fully verified design that can be used to test the memory interface on the board. It rules out any issues with the backend logic interfacing with the MIG core. In addition, the traffic generator provided by the MIG tool can be parameterized to send out different data patterns that test different board-level concerns. For example, a Hammer pattern stresses the memory interface for simultaneous switching outputs (SSOs), while a “Walking 1s” or “Walking 0s” pattern tests if each memory DQ bit can be set to 1 and 0, independent of other bits. See Quick Start Example Design, page 52 for full details on the available data patterns.

Debugging Common Hardware Issues

When calibration failures and data errors are encountered in hardware, the ChipScope analyzer should be used to analyze the behavior of MIG core signals. For detailed information about using the ChipScope analyzer, refer to the *ChipScope Pro 11.1 Software and Cores User Guide* [Ref 14].

A good starting point in hardware debug is to load the provided example_design onto the board in question. This is a known working solution with a traffic generator design that checks for data errors. This design should complete successfully with the assertion of phy_init_done and no assertions of error. Assertion of phy_init_done signifies successful completion of calibration while no assertions of error signifies that the data is written to and read from the memory compare with no data errors.

PHY Layer Debug Port

The top-level wrapper, `memc_ui_top.v`, provides both input and output signals. These signals can be used to debug the various sections of the PHY layer logic if the debug option is checked when generating the design through the MIG tool. These signals can also be

used to monitor PHY outputs, such as the results of the various calibration stages, disable certain features of the PHY, and to adjust portions of the PHY I/O timing.

All debug signals are prefixed with "dbg_" and are synchronous to the clk clock. These signals are listed in Table 1-98, page 165 along with descriptions of the data they profile.

When the debug option is checked when generating a design with the MIG tool, an example ChipScope analyzer debug configuration is instantiated that consists of an Integrated Controller (ICON), Integrated Logic Analyzer (ILA), and several VIO modules.

Table 1-98: **PHY Layer Debug Signals**

Bus Name	I/O	Width	Description
Write Path Calibration Debug			
dbg_wl_dqs_inverted	Output	DQS_WIDTH	This is a 1-bit value indicating whether the DQS output is inverted as a result of write leveling: 1: Inverted 0: Not inverted
dbg_wr_calib_clk_delay	Output	2 × DQS_WIDTH	This is a 2-bit value indicating the number of memory clock cycles of delay for each DQS group.
dbg_wl_odelay_dqs_tap_cnt	Output	5 × DQS_WIDTH	This is a 5-bit IODELAY output tap count for each DQS.
dbg_wl_odelay_dq_tap_cnt	Output	5 × DQS_WIDTH	This is a 5-bit IODELAY output tap count for all DQ and DM bits in each DQS group.
dbg_wr_tap_set_en	Input	1	If this input is driven High, then dbg_wr_dqs_tap_set and dbg_wr_dq_tap_set are used to set the output ODELAY delays for DQS and DQ, respectively, during write transactions to the DDRx memory. In the case of DDR3 SDRAM, when write leveling is enabled, the delay values set by the debug port take effect only after completion of write leveling.
dbg_wr_dqs_tap_set	Input	5 x DQS_WIDTH	This is a 5-bit IODELAY tap count for each DQS used to override the output delay values used during writes. This bus is used only if dbg_wr_tap_set_en is set High.
dbg_wr_dq_tap_set	Input	5 x DQS_WDITH	This is a 5-bit IODELAY tap count for all DQ and DM in the corresponding DQS group used to override the output delay values used during writes. This bus is used only if dbg_wr_tap_set_en is set High.
dbg_wrlvl_start	Output	1	This is used only for DDR3 designs. It is pulsed to a 1 at the start of write leveling.
dbg_wrlvl_done	Output	1	This is used only for DDR3 designs. It is driven to a static 1 after the completion of write leveling.
dbg_wrlvl_err	Output	1	This is used only for DDR3 designs. It is driven to a static 1 if an error occurred during write leveling.
Read Path Calibration Debug			

Table 1-98: **PHY Layer Debug Signals *(Cont'd)***

Bus Name	I/O	Width	Description
dbg_rdlvl_start	Output	2	Each bit is driven to a static 1 as each stage of read leveling is started. The dbg_rdlvl_start[0] signal corresponds to stage 1.
dbg_rdlvl_done	Output	2	Each bit is driven to a static 1 as each stage of read leveling is completed. The dbg_rdlvl_done[0] signal corresponds to stage 1.
dbg_rdlvl_err	Output	2	This output indicates if an error occurred during each stage of read leveling.
dbg_cpt_tap_cnt	Output	5 × DQS_WIDTH	This is a 5-bit ODELAY tap count for the capture clock for each DQS group. This value is updated if dbg_pd_inc_cpt and dbg_pd_dec_cpt are used to change the capture clock tap count dynamically.
dbg_cpt_first_edge_cnt	Output	5 × DQS_WIDTH	This is a 5-bit ODELAY tap count for each DQS group indicating the capture clock tap count reached during read leveling when the first edge of the read data window is found.
dbg_cpt_second_edge_cnt	Output	5 × DQS_WIDTH	This is a 5-bit ODELAY tap count for each DQS group indicating the capture clock tap count reached during read leveling when the second inner edge of the read data window is found. Set to 0 if second edge was not found.
dbg_rd_bitslip_cnt	Output	2 × DQS_WIDTH	This is a 2-bit value indicating the number of bit times by which read data for each DQS group is additionally delayed. Along with dbg_cpt_tap_cnt and dbg_rd_clkdly_cnt, this value is used to for read data deskew.
dbg_rd_clkdly_cnt	Output	2 × DQS_WIDTH	This is a 2-bit value indicating the number of clk cycles by which read data for each DQS group is additionally delayed. Along with dbg_cpt_tap_cnt and dbg_rd_bitslip_cnt, this value is used for read data deskew.
dbg_rd_active_dly	Output	5	This is a 5-bit value indicating the delay in clk cycles between when a read command is issued to the PHY, and when valid read data is available at the PHY interface.
Phase Detector Debug			
dbg_pd_off	Input	1	This input should be driven High to disable initial calibration of the read phase detector. The value of this signal must not be changed after the DDR3 interface logic has come out of reset.
dbg_pd_maintain_off	Input	1	This input should be driven High to disable periodic calibration of the read phase detector.

Table 1-98: **PHY Layer Debug Signals *(Cont'd)***

Bus Name	I/O	Width	Description
dbg_pd_maintain_0_only	Input	1	This input should be driven High to disable the read phase detector periodic IODELAY adjustment for bytes 1 and higher. When this signal is High, byte 0 maintenance continues to adjust the read clock MMCM phase, which affects the sampling time for all bytes.
Miscellaneous Tap Count Monitoring			
dbg_dqs_p_tap_cnt	Output	5 × DQS_WIDTH	This is a 5-bit value indicating the instantaneous delay value of the IODELAY for the corresponding DQS. This value fluctuates between write and read values depending on the current bus traffic.
dbg_dq_tap_cnt	Output	5 × DQS_WIDTH	This is a 5-bit tap count for the capture clock of the DQ I/O in each DQS group. This value fluctuates between write and read values depending on the current bus traffic.
Read Data Capture Clock Adjustment			
dbg_inc_cpt	Input	1	This input increments the tap count for the capture clock IODELAY specified by dbg_inc_dec_sel only when the read phase detector is disabled by driving dbg_pd_maintain_off High. The tap value is incremented by one for every clk cycle when this signal is held High. This signal and dbg_dec_cpt must not be driven High on the same clock cycle.
dbg_dec_cpt	Input	1	This input decrements the tap count for the capture clock IODELAY specified by dbg_inc_dec_sel only when the read phase detector is disabled by driving dbg_pd_maintain_off High. The tap value is incremented by one for every clk cycle when this signal is held High. This signal and dbg_inc_cpt must not be driven High on the same clock cycle.
dbg_inc_rd_dqs	Input	1	This input increments the input tap count for the DQS IODELAY specified by dbg_inc_dec_sel. The tap value is incremented by one for every clk cycle when this signal is held High. Only the DQS input delay (that is, during reads) is varied; the value of the DQS delay when it is an output (during writes) is not varied. This signal and dbg_dec_rd_dqs must not be driven High on the same clock cycle.
dbg_dec_rd_dqs	Input	1	This input decrements the input tap count for the DQS IODELAY specified by dbg_inc_dec_sel. The tap value is decremented by one for every clk cycle when this signal is held High. Only the DQS input delay (that is, during reads) is varied; the value of the DQS delay when it is an output (during writes) is not varied. This signal and dbg_inc_rd_dqs must not be driven High on the same clock cycle.

Table 1-98: **PHY Layer Debug Signals *(Cont'd)***

Bus Name	I/O	Width	Description
dbg_inc_dec_sel	Input	DQS_CNT_WIDTH	This input determines the specific capture clock or DQS IODELAY to vary using dbg_inc_cpt, dbg_dec_cpt, dbg_dec_rd_dqs, and dbg_dec_rd_dqs. When neither of these four signals is asserted, the value of this bus is a don't care.
dbg_inc_rd_fps	Input	1	This input increments the phase of the MMCM output (CLKOUT2) used to drive the capture clock for all DQS groups via the fine phase shift (FPS) feature of the MMCM. For every clk cycle that this input is held High, the phase of this clock is incremented by 1/56 of the MMCM VCO period. This input can therefore be used to make a much finer-grained adjustment to the capture clock phase in comparison to varying IODELAY taps via the dbg_inc/dec_cpt inputs; however, any changes to the phase made via this signal affect the capture clocks for all DQS groups. This input only has an effect if read phase detector updates are disabled, either by driving dbg_pd_off or dbg_pd_maintain_off High; otherwise, the read phase detector simply readjusts the MMCM phase to match the incoming DQS phase on future reads and "undoes" any phase changes made via this signal. This signal and dbg_dec_rd_fps must not be driven High on the same clock cycle.
dbg_dec_rd_fps	Input	1	This input decrements the phase of the MMCM output used to drive the capture clock for all DQS groups via the fine phase shift feature of the MMCM. See the description for dbg_inc_rd_fps for more information on this input.
Synchronized Read Data			
dbg_rddata	Output	4 × DQ_WIDTH	This is the capture read data synchronized to the clk clock domain.

Isolating Bit Errors

An important hardware debug step is to try to isolate when and where the bit errors occur. Looking at the bit errors, these should be identified:

- Are errors seen on data bits belonging to certain DQS groups?
- Are errors seen on accesses to certain addresses, banks, or ranks of memory?

  For example, on designs that can support multiple varieties of DIMM modules, all possible address and bank bit combinations should be supported.
- Do the errors only occur for certain data patterns or sequences?

  This can indicate a shorted or open connection on the PCB. It can also indicate an SSO or crosstalk issue.

It might be necessary to isolate whether the data corruption is due to writes or reads. This case can be difficult to determine because if writes are the cause, read back of the data is bad as well. In addition, issues with control or address timing affect both writes and reads. Some experiments that can be tried to isolate the issue are:

- If the errors are intermittent, have the controller issue a small initial number of writes, followed by continuous reads from those locations. If the reads intermittently yield bad data, there is a potential read problem.
- Check/vary only write timing:
  - If on-die termination is used, check that the correct value is enabled in the DDR2/DDR3 device and that the timing on the ODT signal relative to the write burst is correct.
  - Use ODELAY to vary the phase of DQ relative to DQS.
- Vary only read timing:
  - Check the IDELAY values after calibration. Look for variations between IDELAY values. IDELAY values should be very similar for DQs in the same DQS group.
  - Vary the IDELAY taps after calibration for the bits that are returning bad data. This affects only the read capture timing.

Board Measurements

The signal integrity of the board and bus timing must be analyzed. The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 12] describes expected bus signal integrity. While this guide is specific to Virtex-5 devices and the ML561 development board, the principles therein can be applied to Virtex-6 FPGA MIG designs.

Another important board measurement is the reference voltage levels. It is important that these voltage levels are measured when the bus is active. These levels can be correct when the bus is idle, but might drop when the bus is active.

Supported Devices for Virtex-6 FPGAs

Designs generated using the MIG tool are independent of memory package. Thus, the package part of the memory component part number is replaced with XX, where XX indicates a don't care condition.

Table 1-99 through Table 1-102 list memory devices supported by the tool for Virtex-6 FPGA DDR2 and DDR3 designs. Table 1-99 lists the supported components for DDR2 SDRAM.

Table 1-99: **Supported Components for DDR2 SDRAM**

Components	Packages
MT47H64M16XX-25	HR, HQ, CF, HW, HV
MT47H128M8XX-25	HR, HQ, CF, HW, HV
MT47H256M4XX-25E	HR, HQ, CF, HW, HV
MT47H128M16XX-3	HG
MT47H256M8XX-3	HG
MT47H256M8XX-37E	HG

Table 1-100 lists the supported DIMMs for DDR2 SDRAM.

Table 1-100: **Supported DIMMs for DDR2 SDRAM**

DIMM Type	Part Number
RDIMM	MT9HTF12872PY-80E
RDIMM	MT18HTF25672PY-80E
RDIMM	MT18HTF6472PY-40E
SODIMM	MT8HTF6464HY-667
SODIMM	MT9HTF12872CHY-667
UDIMM	MT8HTF6464HY-667
UDIMM	MT9HTF12872CHY-667

Table 1-101 lists the supported components for DDR3 SDRAM.

Table 1-101: **Supported Components for DDR3 SDRAM**

Components	Packages
MT41J128M8XX-15E	HY, BY, LA
MT41J64M16XX-15E	HY, BY, LA
MT41J256M4XX-15E	HY, BY, LA

Table 1-102 lists the supported DIMMs for DDR3 SDRAM.

Table 1-102: **Supported DIMMs for DDR3 SDRAM**

DIMM Type	Part Number
RDIMM	MT9JSF12872PY-1G1
RDIMM	MT18JSF25672PY-1G1
SODIMM	MT4JSF6464HY-1G1
UDIMM	MT8JTF12864AY-1G4
UDIMM	MT9JSF12872AY-1G4
UDIMM	MT4JTF6464AY1G1

XILINX®

Chapter 2

QDRII+ SRAM Memory Interface Solution

Introduction

The QDRII+ SRAM memory interface solution is a physical layer for interfacing Virtex®-6 FPGA user designs to QDRII+ SRAM devices. QDRII+ SRAMs are the latest generation of QDR SRAM devices that offer high-speed data transfers on separate read and write buses on the rising and falling edges of the clock. These memory devices are used in high-performance systems as temporary data storage, such as:

- Look-up tables in networking systems
- Packet buffers in network switches
- Cache memory in high-speed computing
- Data buffers in high-performance testers

The QDRII+ SRAM memory solutions core is a PHY that takes simple user commands, converts them to the QDRII+ protocol, and provides the converted commands to the memory. The PHY's half-frequency design enables the user to provide one read and one write request per cycle eliminating the need for a memory controller and the associated overhead, thereby reducing the latency through the core. Unique capabilities of the Virtex-6 family allow the PHY to maximize performance and simplify read data capture within the FPGA. The full solution is complete with a synthesizable reference design.

This chapter describes the core architecture and information about using, customizing, and simulating a LogiCORE™ IP QDRII+ SRAM memory interface core for the Virtex-6 FPGA. Although this soft memory controller core is a fully verified solution with guaranteed performance, termination and trace routing rules for PCB design need to be followed to have the best possible design. For detailed board design guidelines, see Design Guidelines, page 222.

For detailed information and updates about the Virtex-6 FPGA QDRII+ SRAM memory interface core, refer to the Virtex-6 FPGA data sheets [Ref 9], [Ref 13] on the Virtex-6 FPGA memory interface product page.

Getting Started

This section is a step-by-step guide to run the design through implementation with the Xilinx tools, and simulate the example design using the provided synthesizable test bench.

System Requirements

These are needed to implement the QDRII+ SRAM memory interface core:

- Microsoft Windows XP Professional (32-/64-bit) or Linux operating systems.
- ISE® Design Suite, version 13.4.

Quick Start Example Design

After the core is successfully generated, the example design HDL can be processed through the Xilinx implementation toolset. The MIG tool provides a simple synthesizable test bench to generate traffic to test the core. An architectural overview of the test bench is shown in Figure 2-1. The top level of the test bench (`sim_tb_top`) is located in `<project_dir>/sim` and contains the memory model to simulate against the top level of the example design (`ip_top`). The `ip_top` folder contains the infrastructure module, the iodelay controller, and the simple test bench. The infrastructure module generates all the clocking signals needed by the core. In `tb_top` are the modules used to generate commands, data, and addresses, as well as a comparator module that checks the responses to verify whether or not the correct data was returned.

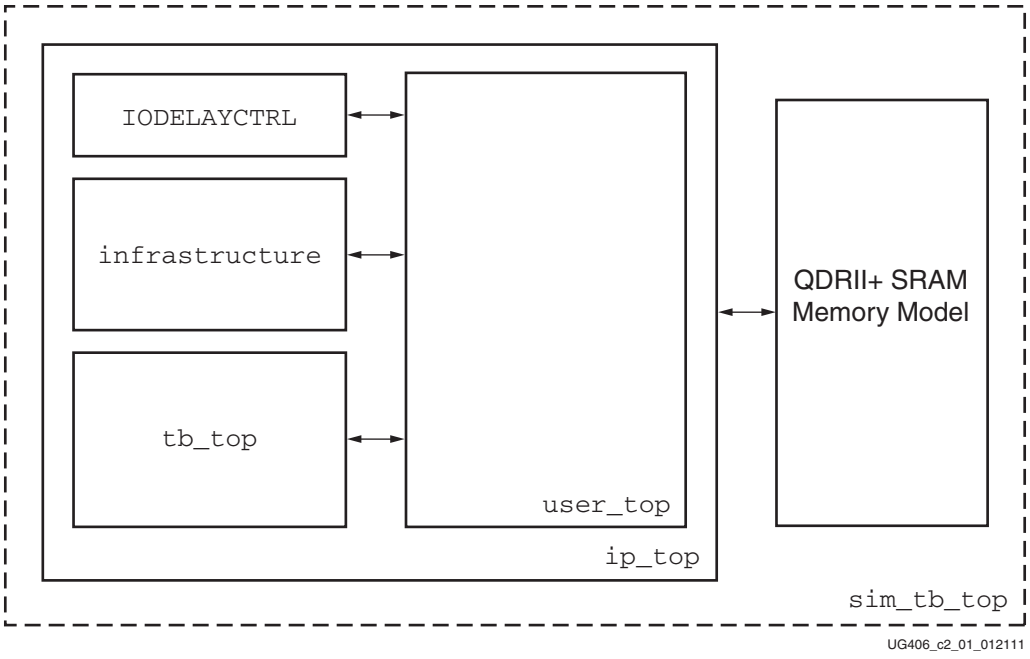


Figure 2-1: **Top Level of Test Bench**

Simulating the Example Design

The Xilinx® UNISIM library must be mapped into the simulator. The test bench provided with the example design supports these pre-implementation simulations:

- The test bench along with vendor's memory model used in the example design

- The RTL files of the memory controller and the PHY core, created by the MIG tool

The simulation can be run from this directory:

```
<component_name>/example_design/sim
```

ModelSim is the only supported simulation tool. The simple test bench can be run using ModelSim by executing the `sim.do` script.

Implementing the Example Design

The `ise_flow.bat` script file runs the design through synthesis, translate, map, and par. This script file sets all the required options and should be referred to for the recommended build options for the design.

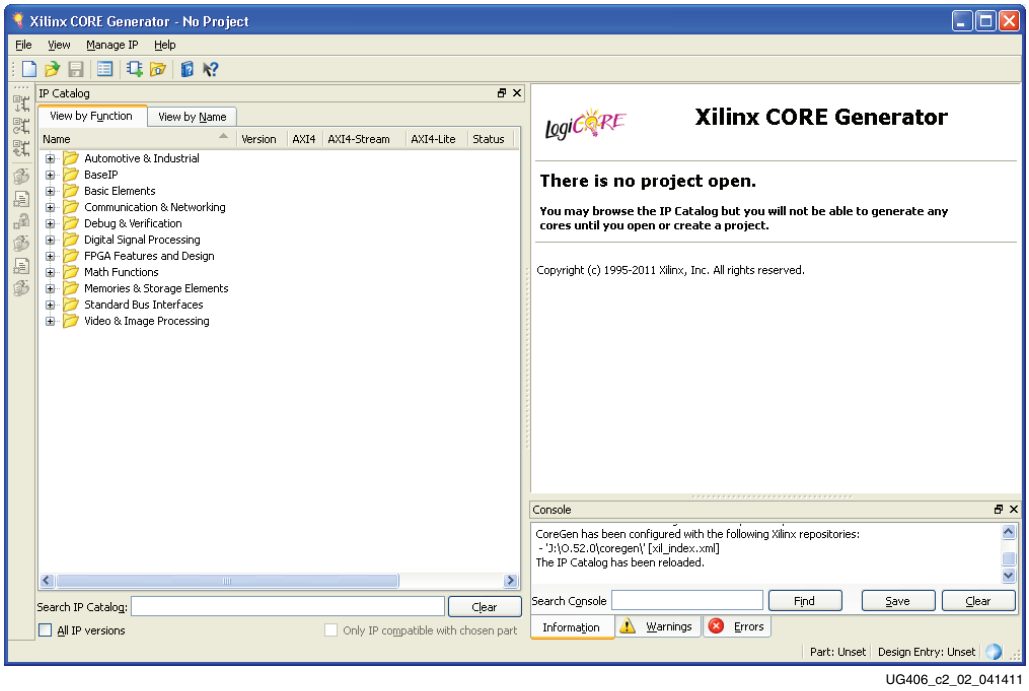
Customizing and Generating the Core

Generation through the Graphical User Interface

The Memory Interface Generator is a self-explanatory wizard tool that can be invoked under the CORE Generator™ software. This section is intended to help in understanding the various steps involved in using the MIG tool.

These steps should be followed to generate a Virtex-6 FPGA QDRII+ SRAM design:

1. Launch the CORE Generator software by selecting **Start → Xilinx ISE Design Suite 13.4 → ISE → Accessories → CORE Generator** (Figure 2-2).



UG406_c2_02_041411

Figure 2-2: **Xilinx CORE Generator Software**

2. Choose **File** → **New project** to open the New Project dialog box. Create a new project named Virtex6_MIG_Example_Design (Figure 2-3).

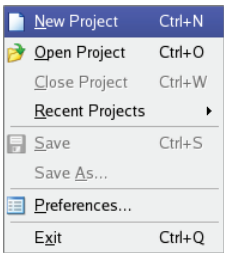


Figure 2-3: **New CORE Generator Software Project**

3. Enter a project name and location. Click **Save** (Figure 2-4).

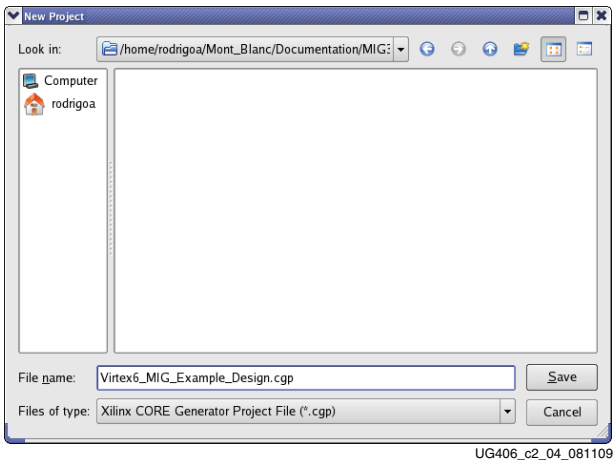


Figure 2-4: **New Project Menu**

4. Select these project options for the part (Figure 2-5).
   a. Family: **Virtex-6**
   b. Device: **xc6vlx240t**
   c. Package: **ff1156**
   d. Speed Grade: **-2**

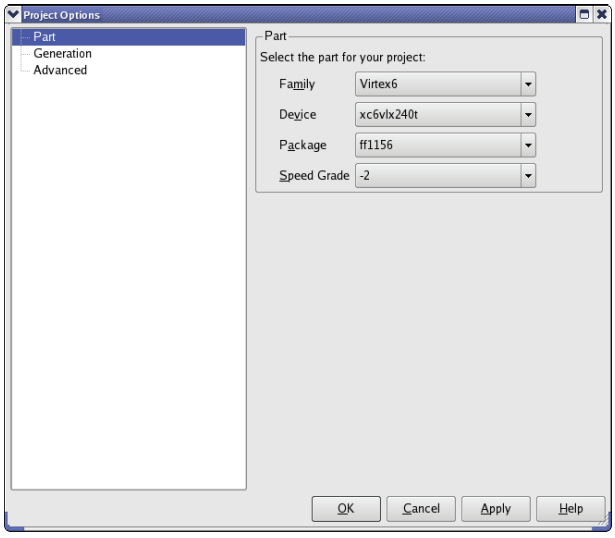


Figure 2-5: **CORE Generator Software Project Options**

5. Select **Generation** from the menu on the left. From this menu, select **Verilog** or **VHDL** as the Design Entry and **ISE** for the Vendor Flow Setting. Click **OK** to finish the Project Options setup (Figure 2-6).

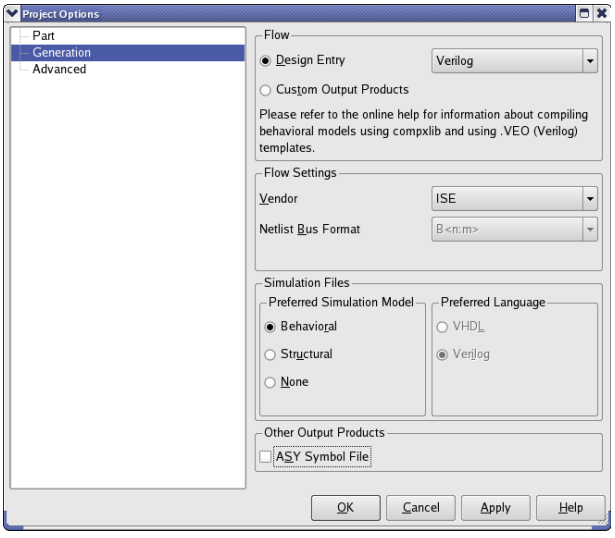
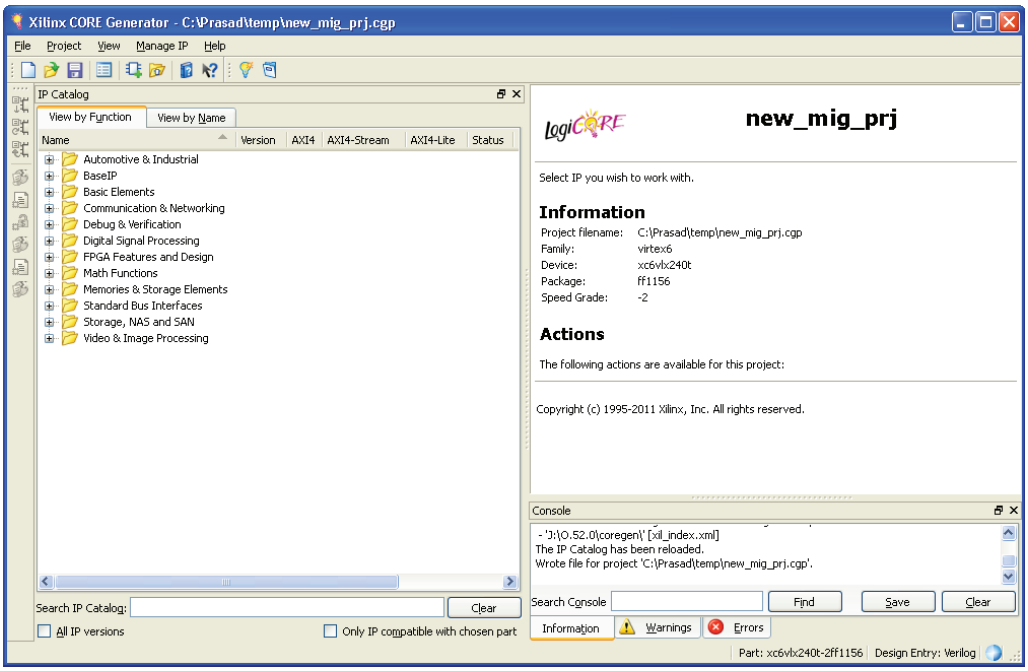


Figure 2-6: **CORE Generator Software Design Flow Settings**

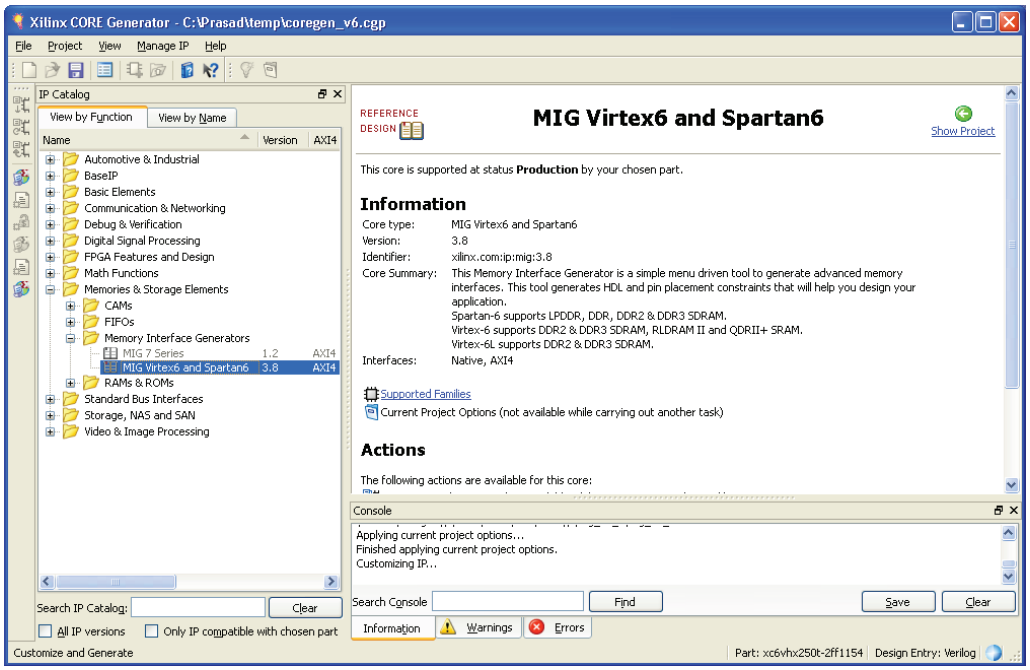
6. Select **Memory Interface Generator (MIG)** by expanding **Memories & Storage Elements** (Figure 2-7).



UG406_c2_07_041411

Figure 2-7: **Virtex-6_MIG_Example_Design Project Page**

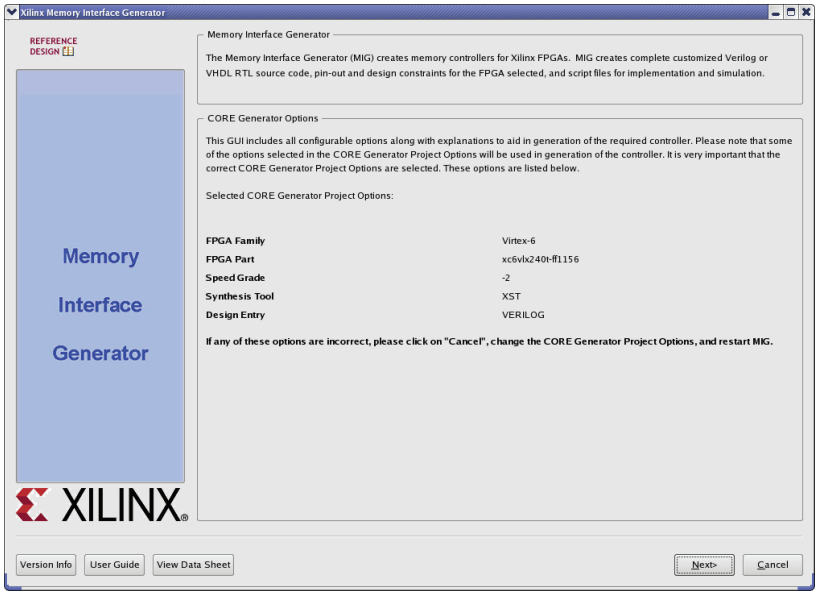
7. Launch the MIG tool wizard by selecting **Memories & Storage Elements** → **Memory Interface Generators** → **MIG** (Figure 2-8).



UG406_c2_08_041411

Figure 2-8: **Starting the MIG Tool Wizard**

8. The options screen in the CORE Generator software displays the details of the selected CORE Generator software options that are selected before invoking the MIG tool (Figure 2-9).



UG406_c2_09_081109

Figure 2-9: **Virtex-6 FPGA Memory Interface Generator Front Page**

9. Click **Next** to display the Output Options window.

MIG Tool Output Options

1. Select the **Create Design** radio button to create a new memory core. Enter a component name in the Component Name field (Figure 2-10).

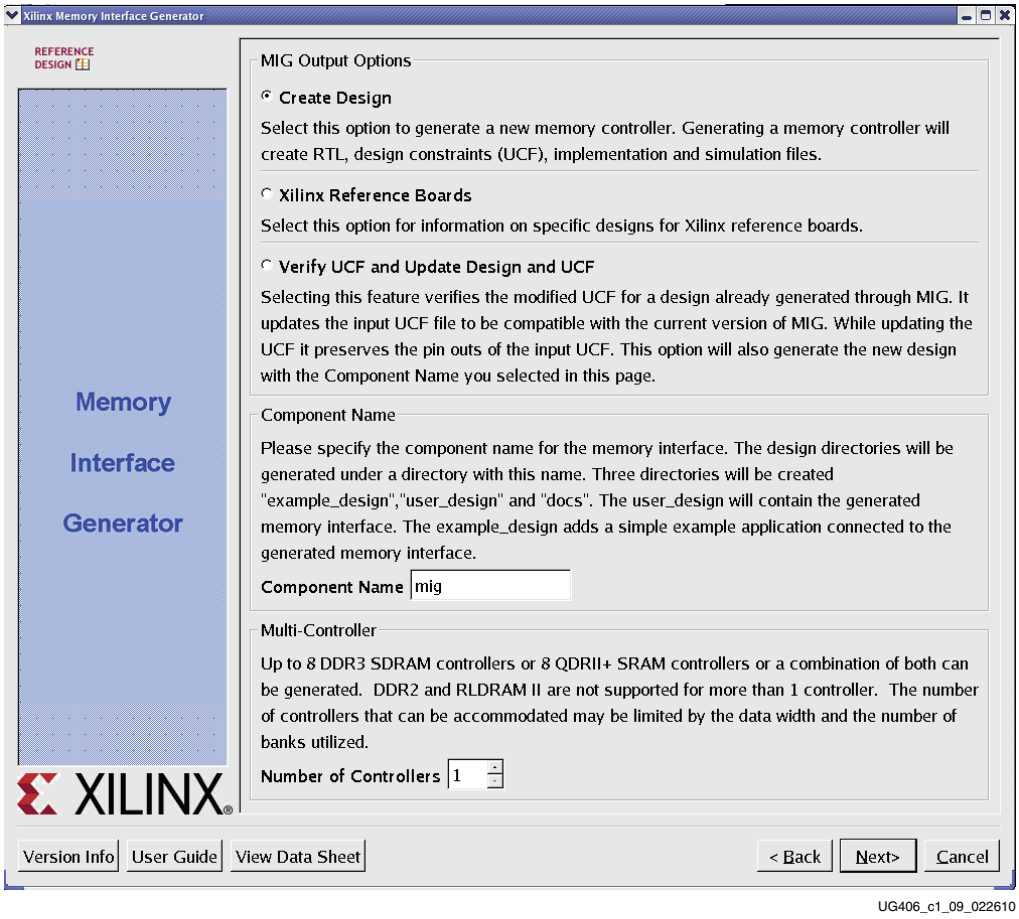


Figure 2-10: **MIG Tool Output Options**

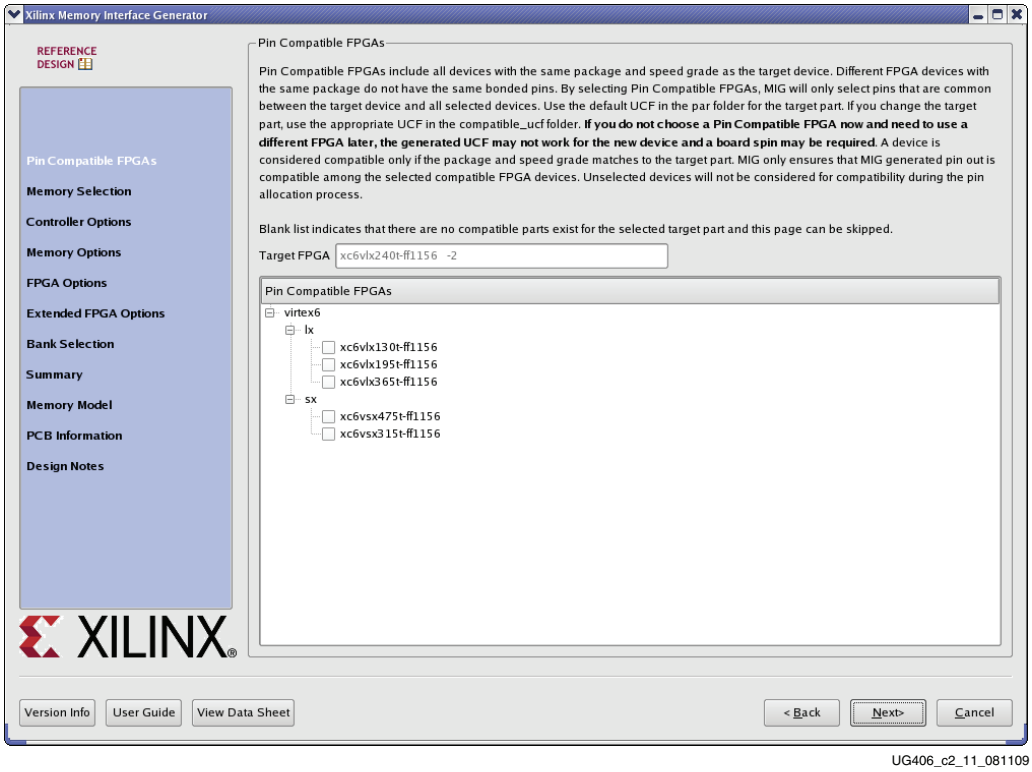
MIG tool outputs are generated with the folder name `<component name>`.

Note: Only alphanumeric characters can be used for `<component name>`. Special characters cannot be used. This name should always start with an alphabetical character and can end with an alphanumeric character.

2. Click **Next** to display the Pin Compatible FPGAs window.

Pin Compatible FPGAs

The Pin Compatible FPGAs window lists FPGAs in the selected family having the same package. If the generated pinout from the MIG tool needs to be compatible with any of these other FPGAs, this option should be used to select the FPGAs with which the pinout has to be compatible (Figure 2-11).



UG406_c2_11_081109

Figure 2-11: **Pin-Compatible Virtex-6 FPGAs**

1. Select any of the compatible FPGAs in the list. Only the common pins between the target and selected FPGAs are used by the MIG tool. The name in the text box signifies the target FPGA selected.
2. Click **Next** to display the Memory Selection window.

Creating the Virtex-6 FPGA QDRII+ SRAM Memory Design

Memory Selection

This page displays all memory types that are supported by the selected FPGA family.

1. Select the **QDRII+ SRAM** controller type.
2. Click **Next** to display the Controller Options window (Figure 2-12).

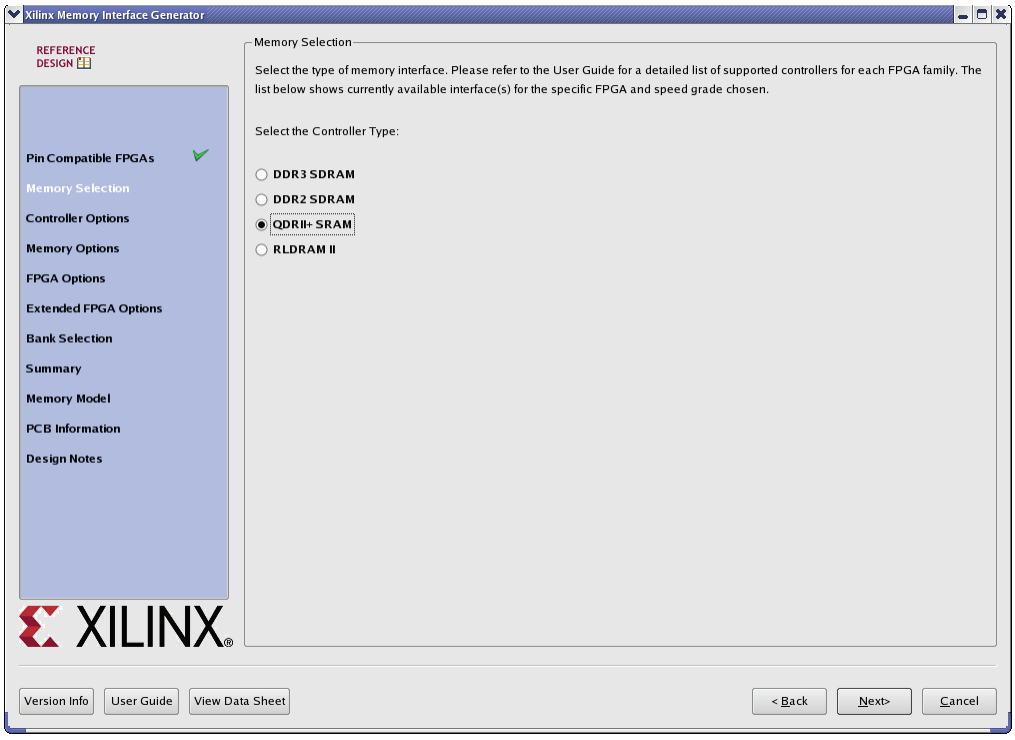
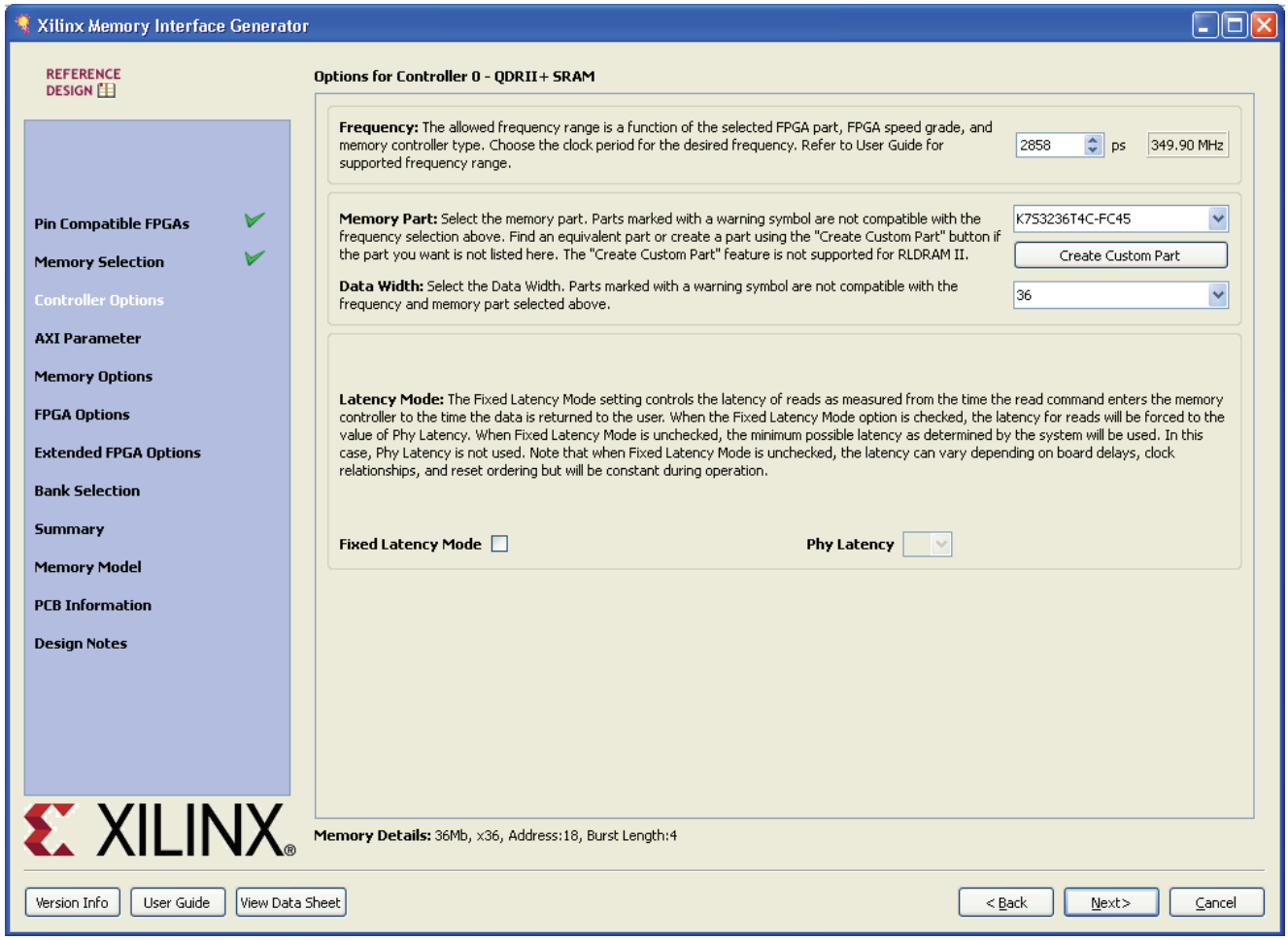


Figure 2-12: **Memory Type and Controller Selection**

Controller Options

This page shows the various controller options that can be selected (Figure 2-13).

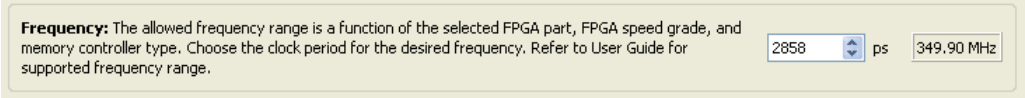


UG406_c2_13_041411

Figure 2-13: **Controller Options**

The controller options page also contains these pull-down menus to modify different features of the design.

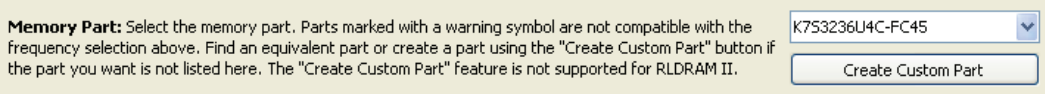
- **Frequency**: This feature indicates the operating frequency for all the controllers (Figure 2-14). The frequency block is limited by factors such as the selected FPGA and device speed grade.



UG406_c2_14_041411

Figure 2-14: **Frequency Selection**

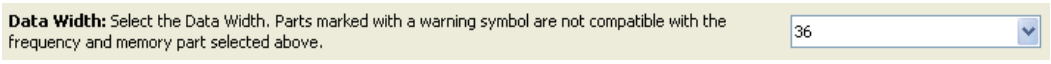
- **Memory Part**: This option selects the memory part for the design. Selections can be made from the list, or if the part is not listed, a new part can be created (Figure 2-15). QDRII+ SRAM devices of read latency 2.0 and 2.5 clock cycles are supported by the design. If a desired part is not available in the list, the user can generate or create an equivalent device and then modify the output to support the desired memory device.



UG406_c2_15_052010

Figure 2-15: **Memory Part Selection**

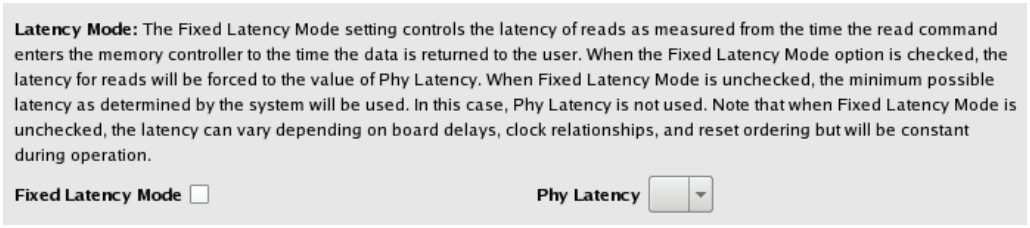
- **Data Width**: The data width value can be selected here based on the memory part selected. The MIG tool supports values in multiples of the individual device data widths (Figure 2-16).



UG406_c2_16_052010

Figure 2-16: **Data Width Selection**

- **Latency Mode**: If fixed latency through the core is needed, the Fixed Latency Mode option allows the user to select the desired latency. This option can be used if the user design needs a read response returned in a predictable number of clock cycles. To use this mode, select the **Fixed Latency Mode** box. After enabling fixed latency, the pull-down box allows the user to select the number of cycles until the read response is returned to the user. This value ranges from 19 to 30 cycles (Figure 2-17). If Fixed Latency Mode is not used, the core uses the minimum number of cycles through the system.



UG406_C2_17_081109

Figure 2-17: **Latency Selection**

- **System Clock**: The MIG tool supports the use of differential system and reference clocks or single-ended clocks. This should be set to the type of input clock that is used (Figure 2-18).

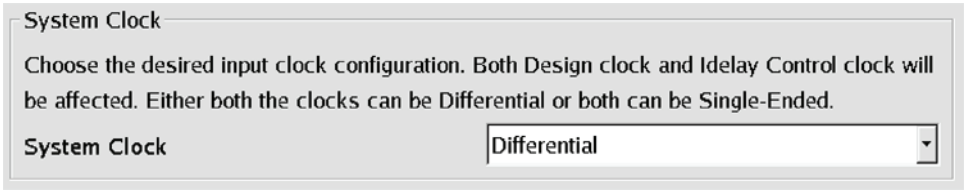


Figure 2-18: **System Clock Type**

- **Debug Signals Control**: This option indicates whether the ChipScope™ analyzer should be incorporated into the generated design (Figure 2-19). See Debugging Virtex-6 FPGA QDRII+ SRAM Designs, page 224 for more details on the signals that are provided when this option is turned on.

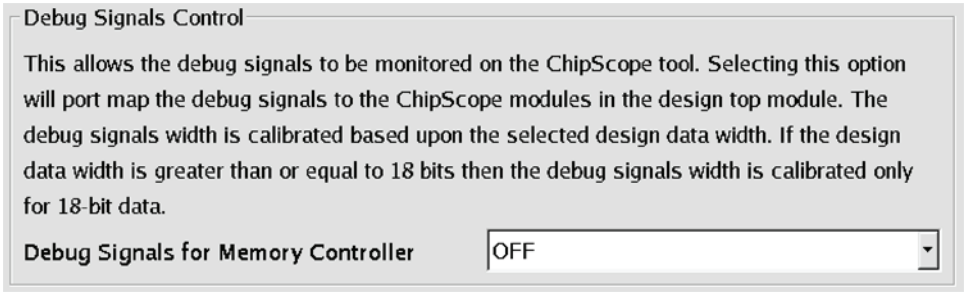


Figure 2-19: **Debug Enable**

- **Internal Vref Selection.** Internal Vref can be used for data read banks to allow the use of the VREF pins for normal I/O usage (Figure 2-20).

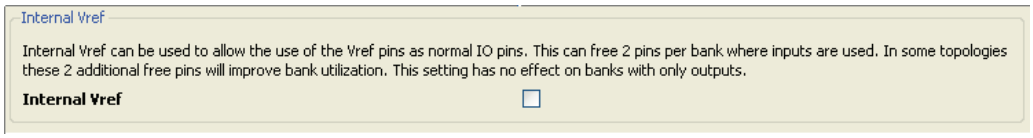


Figure 2-20: **Internal VREF Selection**

- **Digitally Controlled Impedance (DCI)**: When selected, this option internally terminates the signals from the QDR II+ SRAM read path (Figure 2-21).

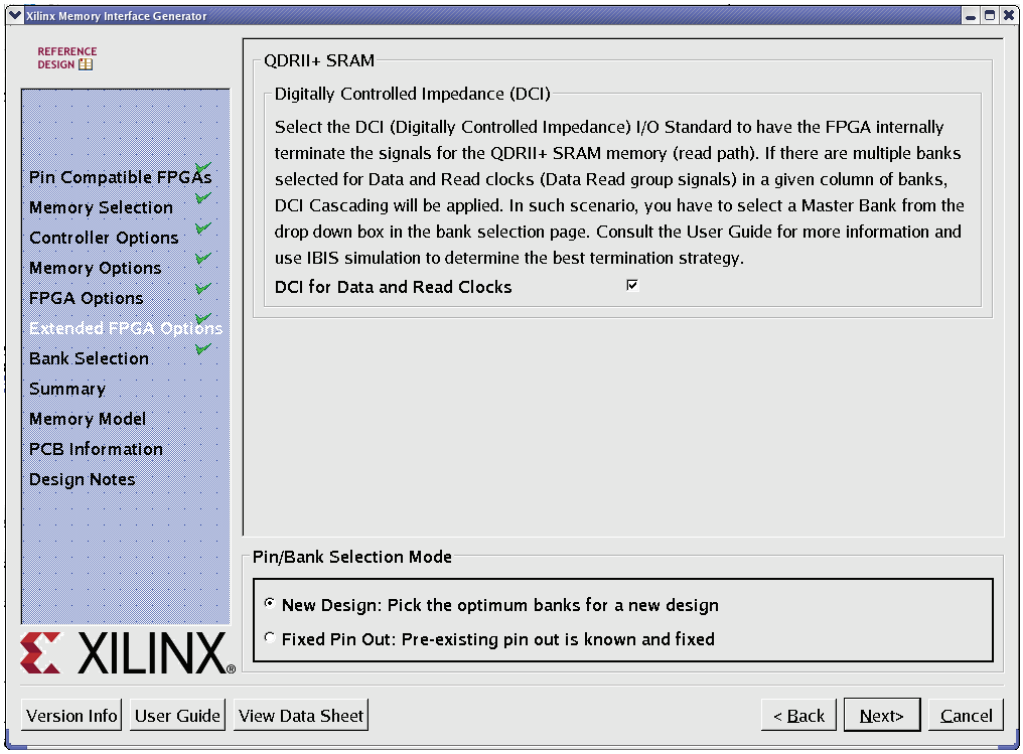


Figure 2-21: **DCI Selection**

- **Pin Selection.** The Pin Selection mode allows the user to specify an existing pinout and generate a new RTL core for this pinout, or pick banks for a new design. Figure 2-22 shows the option for using an existing pinout. For each pin, a bank should be chosen, then within that bank, a pin must be assigned. Click the **Validate** button to check a pinout against the MIG pinout rules.

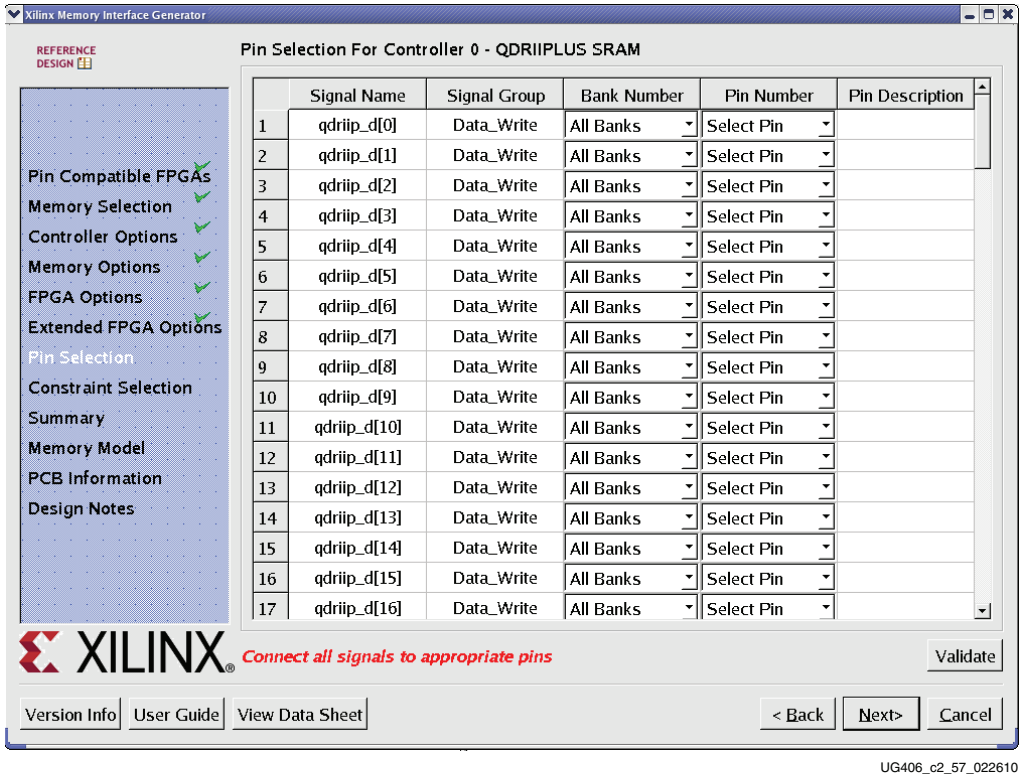
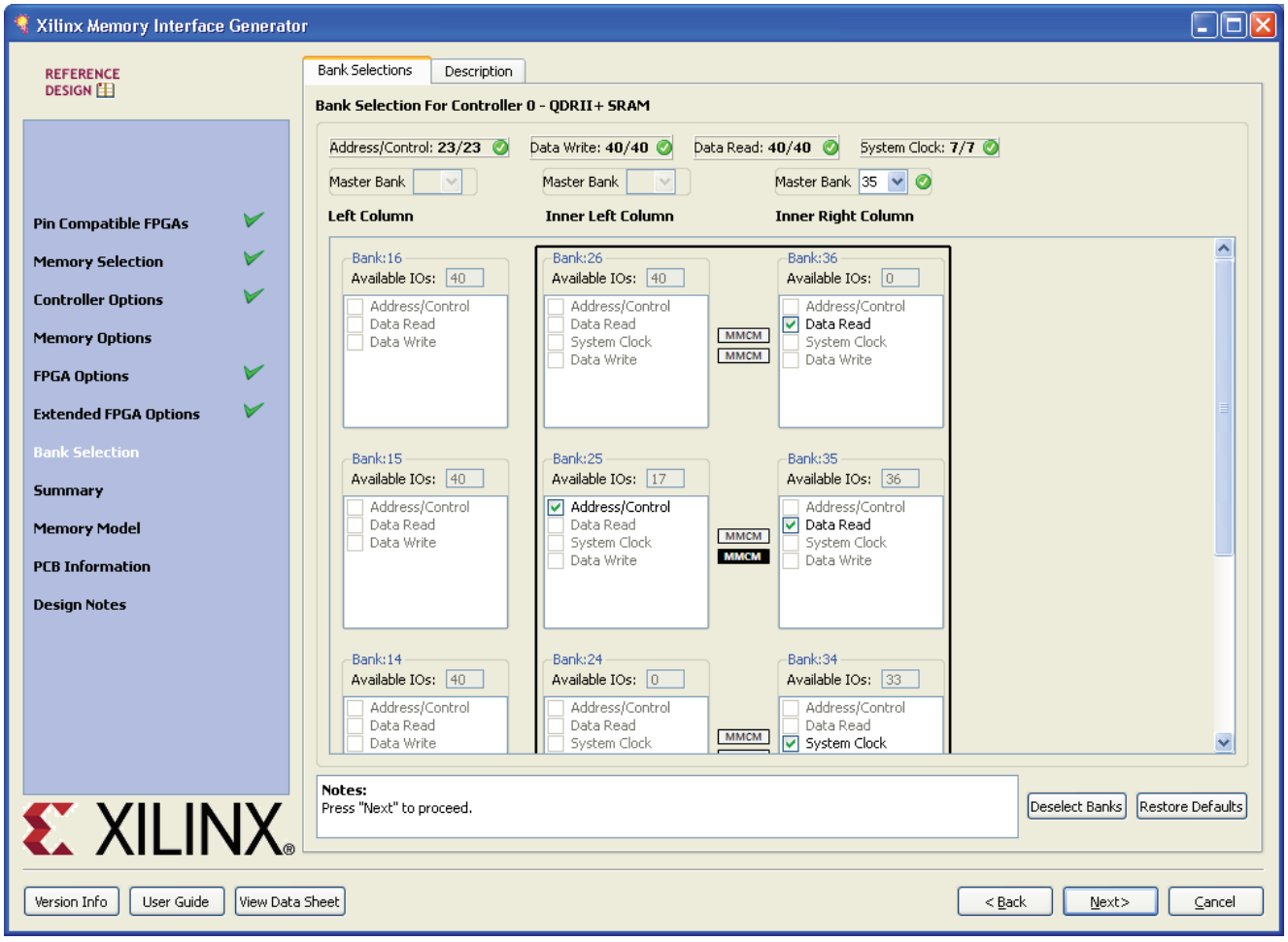


Figure 2-22: **Pin Selection**

Bank Selections

This page allows the selection of banks for the memory interface (Figure 2-23). Banks can be selected for different classes of memory signals, such as:

- Address and control signals
- Data write signals
- Data read signals
- System control signals
- System clocks



UG406_c2_22_052010

Figure 2-23: **Bank Selections Page**

For customized settings, click **Deselect Banks** and select the appropriate bank and memory signals. When selecting banks, read the Description tab for a list of bank selection and pin allocation rules that must be followed.

Click **Next** to select the default settings and move to the next page.

- **Address Group Selection**: Select the address/control group from one of the white banks. Only the inner columns are allowed (Figure 2-24).

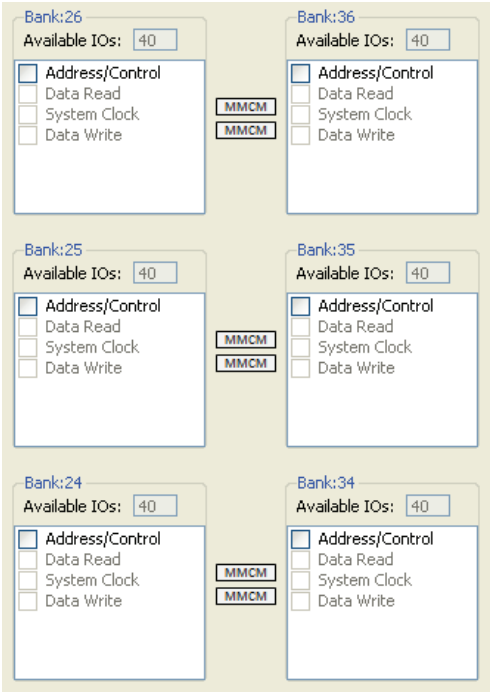


Figure 2-24: **Address Group Selection**

- **Data Write Selection**: After the address group is assigned, the data write group must be selected. The data write group is grayed in banks that are invalid based on the selected address group. To complete the data write selection, enough banks must be selected to hold the data and byte write signals (Figure 2-25).

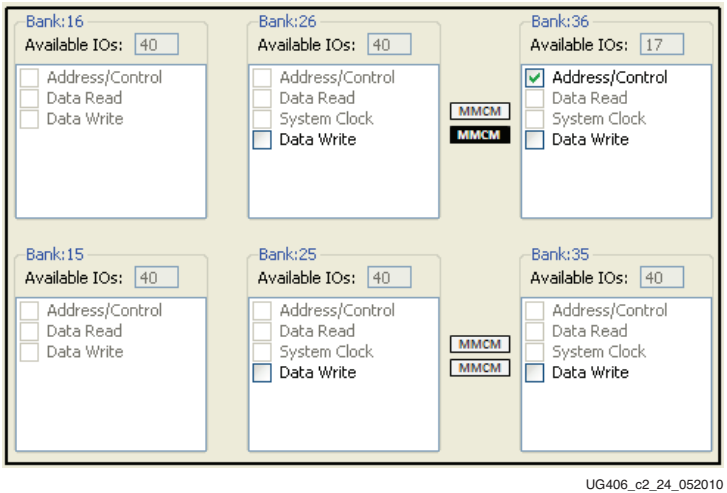


Figure 2-25: **Data Write Group Selection**

- **Data Read Selection**: After the data write group is assigned, the data read group needs to be selected. To complete the data read selection, enough banks must be selected to hold the data and byte write signals (Figure 2-26).

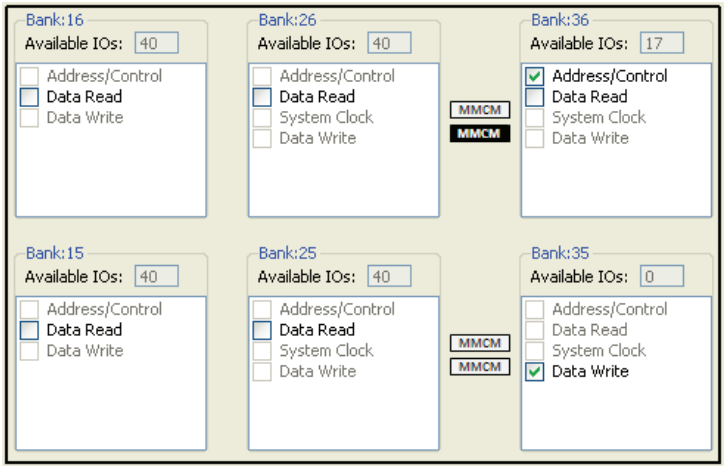


Figure 2-26: **Data Read Group Selection**

- **System Clocks Selection**: After the data read group is assigned, the system clocks group must be selected. Select this group from any of the enabled banks (Figure 2-27).

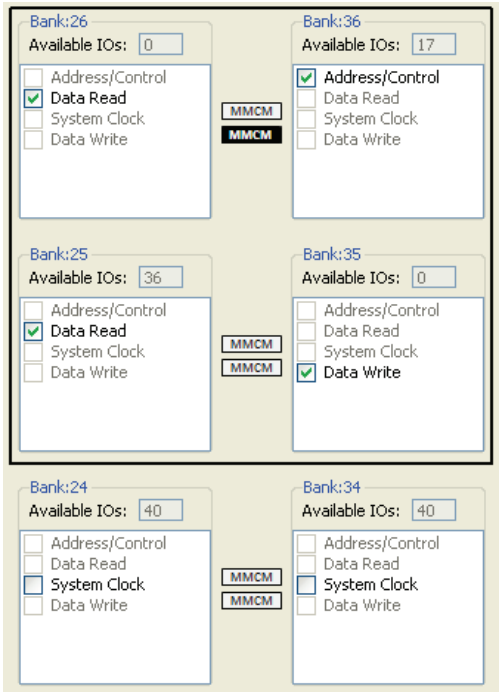


Figure 2-27: **System Clocks Group Selection**

- **Master Bank Selection**: Two extra pins are required to set up a DCI reference that provides better signal integrity. Select the master bank from the pull-down menu (Figure 2-28).

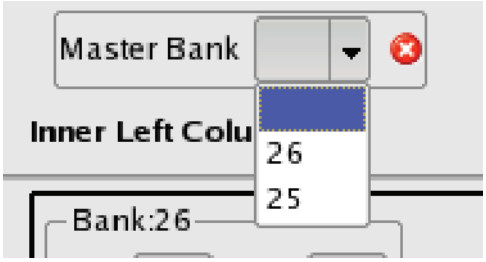


Figure 2-28: **Master Bank Selection**

After the banks have been assigned, click **Next** to view the summary.

Summary

The summary window provides the complete details about the Virtex-6 FPGA memory core selection, interface parameters, CORE Generator software options, and FPGA options of the active project (Figure 2-29).

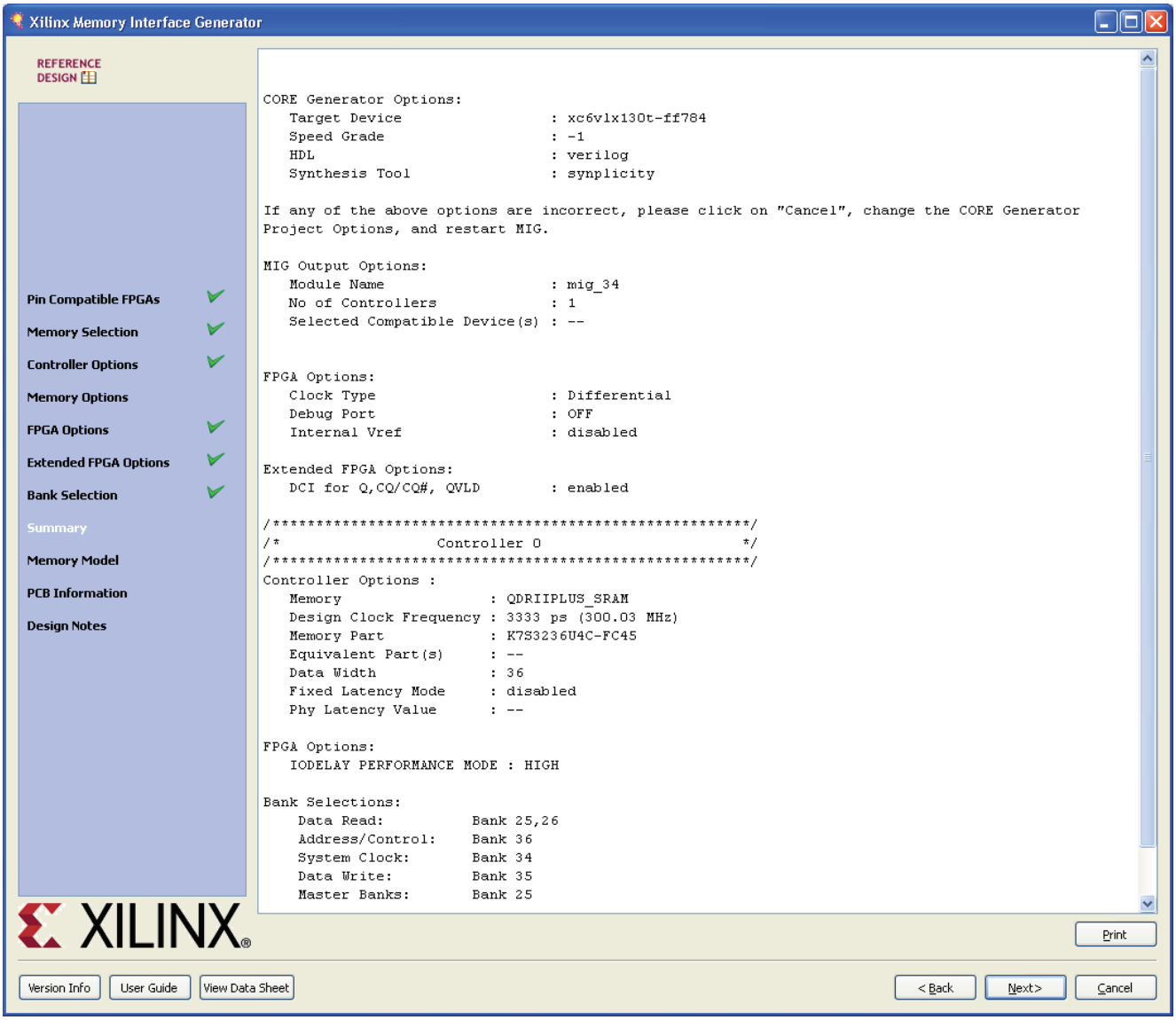


Figure 2-29: **Summary**

Memory Models

The MIG tool does not output the Samsung memory models needed for simulation as noted on the Memory Models page (Figure 2-30). The required models can be downloaded from the Samsung website at http://www.samsung.com/global/business/semiconductor/products/sram/Products_HighSpeedSRAM.html.

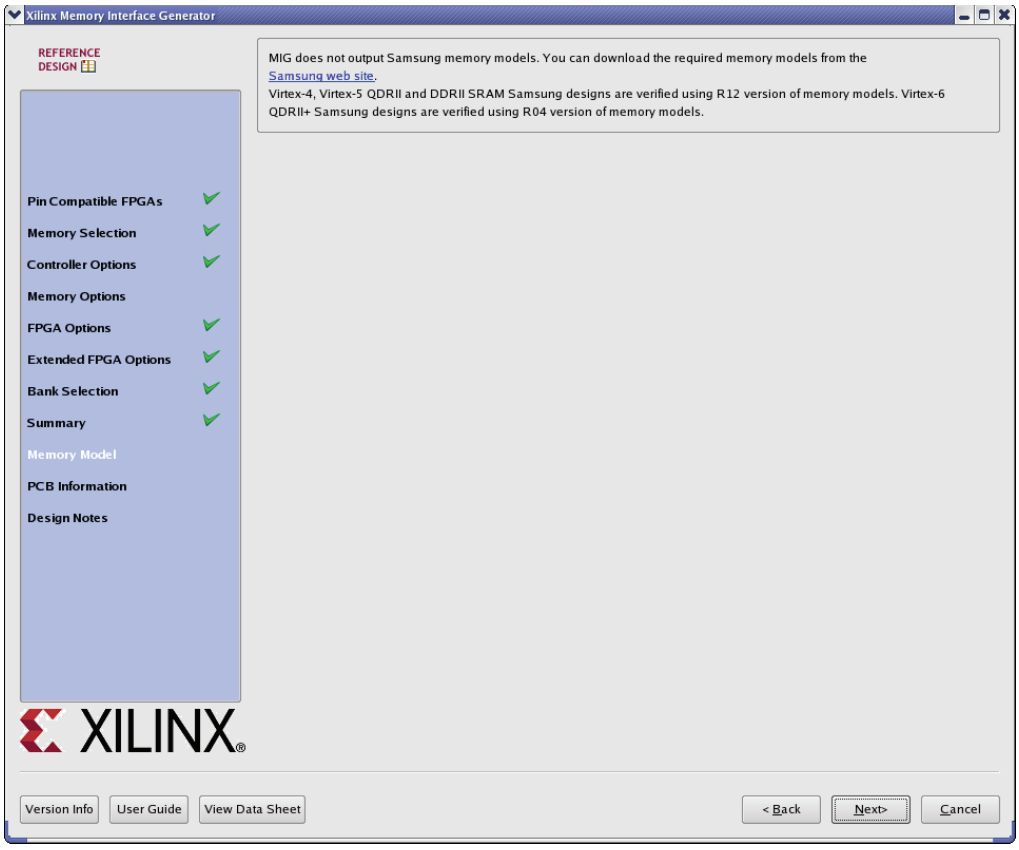


Figure 2-30: **Memory Models Page**

Click **Next** to move to the PCB Information page.

PCB Information

This page displays the PCB-related information to be considered while designing the board that uses the MIG tool generated designs (Figure 2-31).

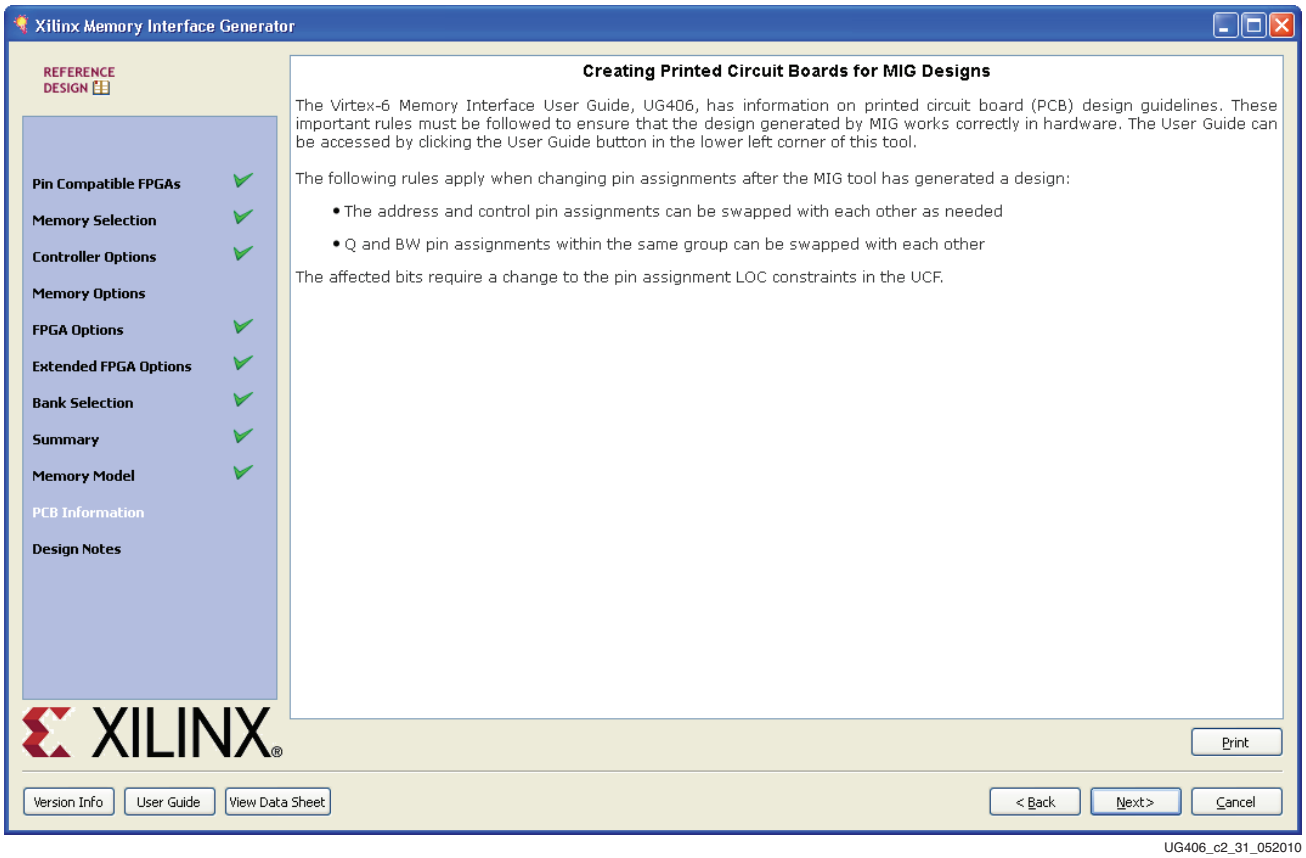


Figure 2-31: **PCB Information Page**

Design Notes

Click the **Generate** button to generate the design files. The MIG tool generates three output directories: `docs`, `example_design`, and `user_design`. See Directory Structure and File Descriptions, page 197 for more details on the contents of these directories.

The MIG tool outputs some useful design notes that should be considered before proceeding (Figure 2-32).

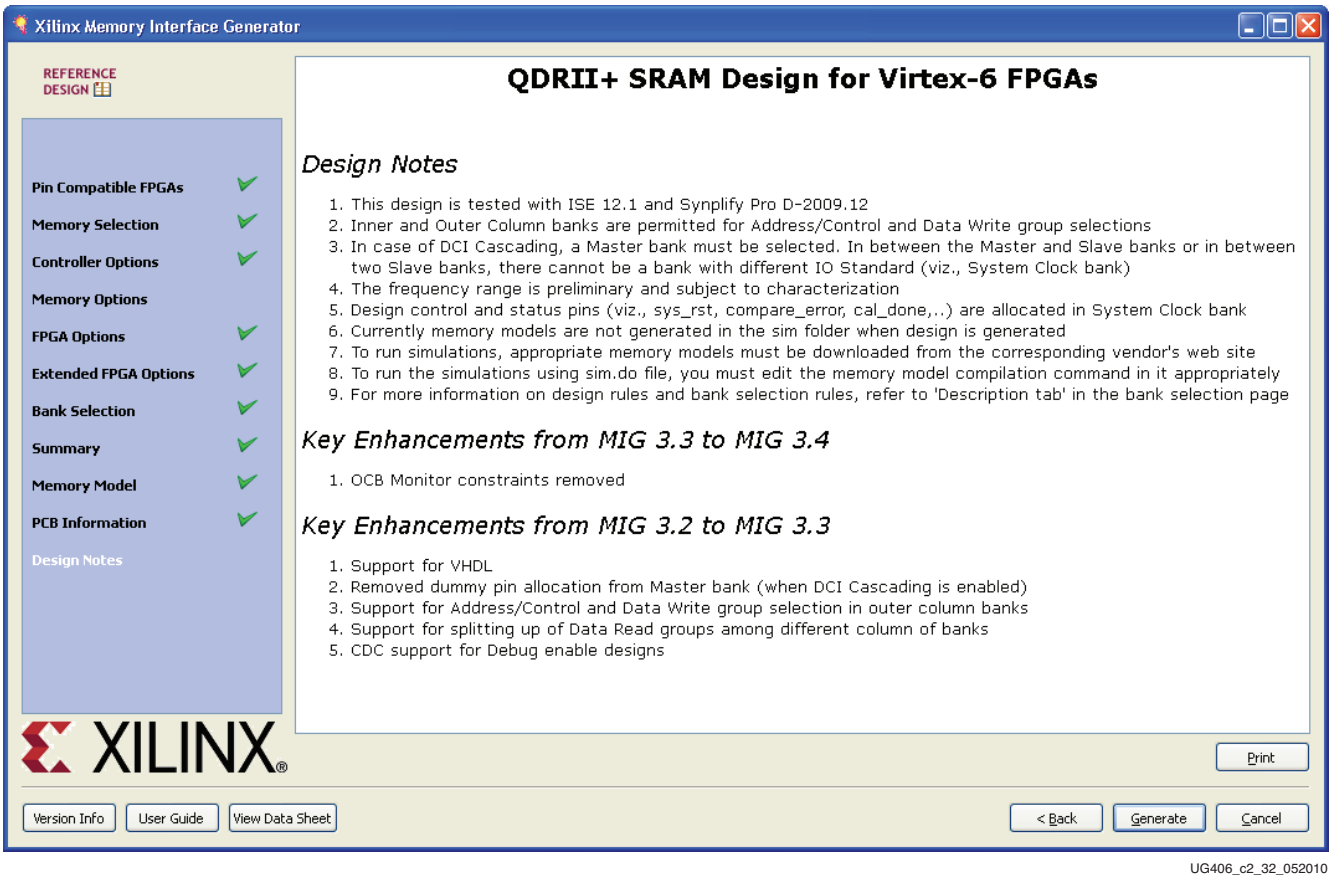


Figure 2-32: **Design Notes**

After generating the design, a Readme page is displayed with the CORE Generator software output descriptions (Figure 2-33).

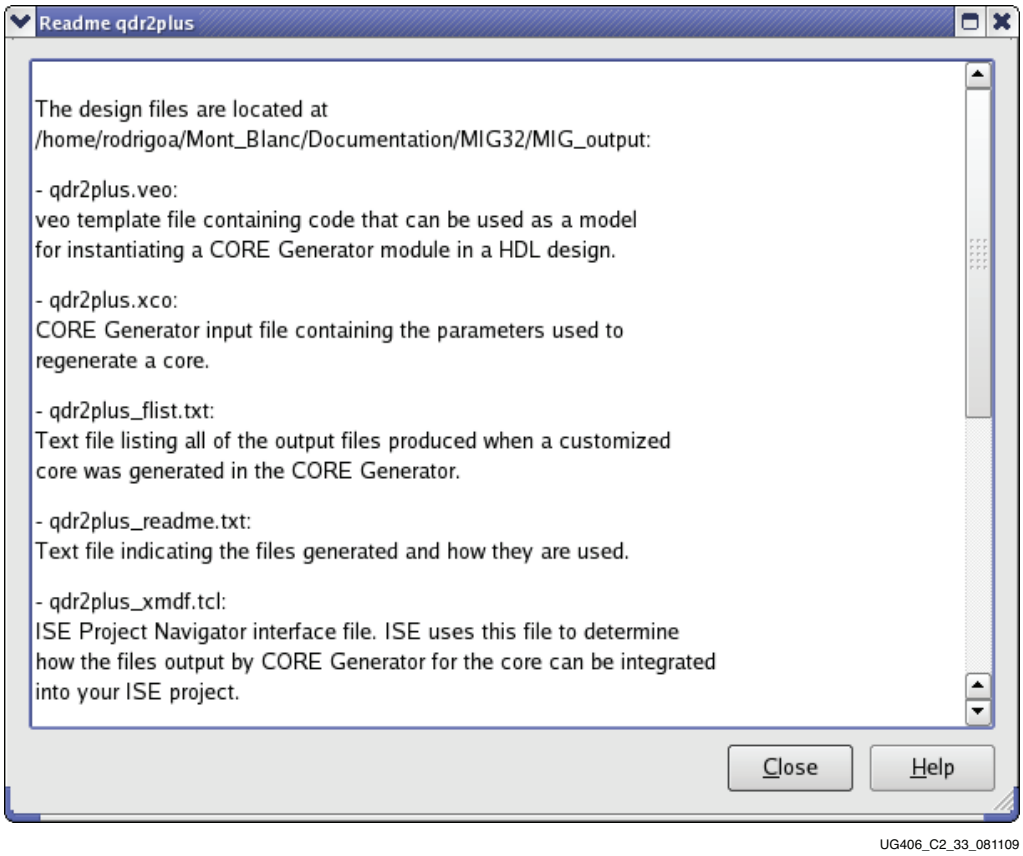


Figure 2-33: **Readme Page**

Click **Close** to return to the CORE Generator software.

Directory Structure and File Descriptions

Overview

Output Directory Structure

The MIG tool outputs are generated with folder name <component name>.

Figure 2-34 shows the output directory structure of the selected design from the MIG tool. In the <component name> directory, three folders are created:

- docs
- example_design
- user_design

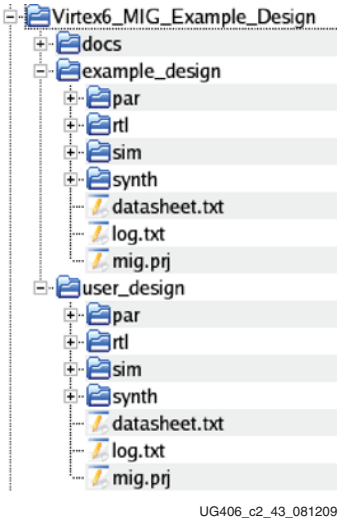


Figure 2-34: **The MIG Tool Output Directory Structure**

The example_design and user_design directories contain most of the same files. However, they are provided separately for easy simulation and integration into the user's design. The example_design directory provides an example user application that sends traffic through the core. This example design is used for simulation and contains the complete synthesizable test bench. The user_design directory provides only those files needed to integrate the core into the user's logic and does not include the simulation or test bench files.

Directory and File Contents

The Virtex-6 device core directories and their associated files are listed in this section.

<component name>/docs

The docs folder contains the PDF documentation.

<component name>/example_design

The example_design folder contains four folders, namely, par, rtl, sim, and synth. Table 2-1 to Table 2-8 list the contents of these directories along with file descriptions.

<component name>/example_design/par

Table 2-1 lists the files in the example_design/par directory.

Table 2-1: **Files in example_design/par Directory**

Name	Description
example_top.ucf	This is the user constraints file generated from the bank selections.

<component name>/example_design/rtl

Table 2-2 lists the files in the example_design/rtl directory.

Table 2-2: **Files in example_design/rtl Directory**

Name	Description
example_top.v/vhd	This top-level module serves as an example for connecting the user design to the Virtex-6 FPGA memory interface core.
clk_ibuf.v/vhd	This module instantiates the system clock input buffers.
qdr_rld_infrastructure.v/vhd	This modules helps in clock generation and distribution.
iodelay_ctrl.v/vhd	This module instantiates the IDELAYCTRL primitive needed for IODELAY use.
phy_d_q_io.v/vhd	This module is the I/O module for the entire D and Q bus for a single memory.
phy_iob.v/vhd	This module instantiates the modules that use IOBs.
qdr_rld_phy_ocb_mon.v/vhd	This module contains the logic for aligning two clock signals for phase detection.
phy_oserdes_io.v/vhd	This module is the I/O module for a single bit of data going to the memory.
phy_read_clk_io.v/vhd	This module is the I/O module for the incoming CQ/CQ# echo clocks from the memory.
phy_read_data_align.v/vhd	This module realigns the incoming data.
phy_read_dcb.v/vhd	This module transfers the data from the clk_rd domain into the clk domain.
phy_read_dly_ctrl.v/vhd	This module drives the IODELAY control for each clock and data I/O based on the control from the calibration logic.
phy_read_stage1_cal.v/vhd	This module contains the logic for stage 1 calibration.
phy_read_stage2_cal.v/vhd	This module contains the logic for stage 2 calibration.
phy_read_sync.v/vhd	This module synchronizes control signals from the clk domain to the clk_rd domain.
phy_read_top.v/vhd	This is the top-level of the read path.

Table 2-2: **Files in `example_design/rtl` Directory *(Cont'd)***

Name	Description
phy_read_vld_gen.v/vhd	This module contains the logic to generate the valid signal for the read data returned on the user interface.
phy_reset_sync.v/vhd	This module contains the reset synchronization logic for all resets used in the core.
qdr_phy_top.v/vhd	This is the top-level module for the physical layer.
phy_v6_d_q_io.v/vhd	This is the I/O module for a single Q bit coming from the memory.
phy_write_control_io.v/vhd	This module contains the logic for the control signals going to the memory.
phy_write_data_io.v/vhd	This module contains the logic for the data and byte writes going to the memory.
phy_write_init_sm.v/vhd	This module contains the logic for the initialization state machine.
phy_write_top.v/vhd	This is the top-level wrapper for the write path.
tb_addr_gen.v/vhd	This module generates the addresses used in the example test bench for simulation.
tb_cmp_data.v/vhd	This module is the comparison module for the data returning from the PHY in a simulation.
tb_cmp_data_bits.v/vhd	This is the comparison module for a single bit of data.
tb_data_gen.v/vhd	This module generates the data to use for write requests in the example test bench.
tb_top.v/vhd	This is the top-level of the synthesizable test bench.
tb_wr_rd_sm.v/vhd	This is the test bench write/read state machine that issues commands during simulation.
user_top.v/vhd	This is the top-level wrapper for the PHY.
qdr_rld_phy_pd.v/vhd	This module contains the phase-detection logic.

`<component name>/example_design/sim`

Table 2-3 lists the files in the `example_design/sim` directory.

Table 2-3: **Files in `example_design/sim` Directory**

Name	Description
glbl.v	This file is used for initializing the simulation environment.
sim.do	This is the script used for running a ModelSim simulation.
sim_tb_top.v/vhd	This is the top-level simulation file.

<component name>/example_design/synth

Table 2-4 lists the files in the example_design/synth directory.

Table 2-4: **Files in example_design/synth Directory**

Name	Description
example_top.lso	This is a library search order file provided for XST.
example_top.prj	This is the ISE software project file used for synthesis.

<component name>/user_design/par

Table 2-5 lists the files in the user_design/par directory.

Table 2-5: **Files in user_design/par Directory**

Name	Description
<component name>.ucf	This is the user constraints file generated from the bank selections.

<component name>/user_design/rtl

Table 2-6 lists the files in the user_design/rtl directory.

Table 2-6: **Files in user_design/rtl Directory**

Name	Description
<component name>.v/vhd	This top-level module serves as an example for connecting the user design to the Virtex-6 FPGA memory interface core.
clk_ibuf.v/vhd	This module instantiates the system clock input buffers.
qdr_rld_infrastructure.v/vhd	This module helps in clock generation and distribution.
iodelay_ctrl.v/vhd	This module instantiates the IDELAYCTRL primitive needed for IODELAY use.
phy_d_q_io.v/vhd	This is the I/O module for the entire D and Q bus for a single memory.
phy_iob.v/vhd	This module instantiates the modules that use IOBs.
qdr_rld_phy_ocb_mon.v/vhd	This module contains the logic for aligning two clock signals for phase detection.
phy_oserdes_io.v/vhd	This is the I/O module for a single bit of data going to the memory.
phy_read_clk_io.v/vhd	This is the I/O module for the incoming CQ/CQ# echo clocks from the memory.
phy_read_data_align.v/vhd	This module realigns the incoming data.
phy_read_dcb.v/vhd	This module transfers the data from the clk_rd domain into the clk domain.

Table 2-6: **Files in `user_design/rtl` Directory *(Cont'd)***

Name	Description
`phy_read_dly_ctrl.v/vhd`	This module drives the IODELAY control for each clock and data I/O based on the control from the calibration logic.
`phy_read_stage1_cal.v/vhd`	This module contains the logic for stage 1 calibration.
`phy_read_stage2_cal.v/vhd`	This module contains the logic for stage 2 calibration.
`phy_read_sync.v/vhd`	This module synchronizes control signals from the clk domain to the clk_rd domain.
`phy_read_top.v/vhd`	This is the top-level of the read path.
`phy_read_vld_gen.v/vhd`	This module contains the logic to generate the valid signal for the read data returned on the user interface.
`phy_reset_sync.v/vhd`	This module contains the reset synchronization logic for all resets used in the core.
`qdr_phy_top.v/vhd`	This is the top-level module for the physical layer.
`phy_v6_d_q_io.v/vhd`	This is the I/O module for a single Q bit coming from the memory.
`phy_write_control_io.v/vhd`	This module contains the logic for the control signals going to the memory.
`phy_write_data_io.v/vhd`	This module contains the logic for the data and byte writes going to the memory.
`phy_write_init_sm.v/vhd`	This module contains the logic for the initialization state machine.
`phy_write_top.v/vhd`	This is the top-level wrapper for the write path.
`user_top.v/vhd`	This is the top-level wrapper for the PHY.
`qdr_rld_phy_pd.v/vhd`	This module contains the phase-detection logic.

`<component name>/user_design/sim`

Table 2-7 lists the files in the `user_design/sim` directory.

Table 2-7: **Files in `user_design/sim` Directory**

Name	Description
`glbl.v`	This file is used for initializing the simulation environment.
`sim.do`	This is the script used for running a ModelSim simulation.
`sim_tb_top.v/vhd`	This is the top-level simulation file.
`tb_addr_gen.v/vhd`	This module generates the addresses used in the example test bench for simulation.
`tb_cmp_data.v/vhd`	This is the comparison module for the data returning from the PHY in a simulation.

Table 2-7: **Files in `user_design/sim` Directory *(Cont'd)***

Name	Description
`tb_cmp_data_bits.v/vhd`	This is the comparison module for a single bit of data.
`tb_data_gen.v/vhd`	This module generates the data to use for write requests in the example test bench.
`tb_top.v/vhd`	This is the top-level of the synthesizable test bench.
`tb_wr_rd_sm.v/vhd`	This is the test bench write/read state machine that issues commands during simulation.

`<component name>/user_design/synth`

Table 2-8 lists the files in the `user_design/synth` directory.

Table 2-8: **Files in `user_design/synth` Directory**

Name	Description
`<component name>.lso`	This is a library search order file provided for XST.
`<component name>.prj`	This is the ISE project file used for synthesis.

Designing with the Core

The core is bundled with an example design that can be simulated. The example design can be used as a starting point for the user design or as a reference for debugging purposes. Only supported modifications should be made to the configuration of the core. See Customizing the Core, page 220 for supported configuration parameters.

Verify UCF and Update Design and UCF Rules

Verify UCF and Update Design and UCF verifies the input UCF for bank selection, pin allocation, and constraint allocation rules, and generates warnings or error reports for any issue. It does not verify the input `.prj` file. This feature is useful to verify any UCF pinout changes after the design is generated from the MIG tool. The user must load the MIG generated `.prj` file (the original `.prj` file) without any modifications. The verification report is not correct if any of the parameters in the original `.prj` file are altered. In the CORE Generator tool, the recustomization option should be selected to reload the project. The design can be generated only when Verify UCF does not report an error in the verification report. Warnings can be ignored while generating a design.

These rules are verified from the input UCF:

- If a pin is allocated to more than one signal, the tool reports an error.
  - The tool stops further verification if the UCF does not adhere to the uniqueness property.
- The associative property is verified:
  - If the Read Clock pin is allocated to the multi-region clock-capable (MRCC) P pin, all its associated signals should be allocated within the banks above and below.
  - If the Read Clock pin is allocated to the MRCC P pin, all its associated signals should be allocated within the banks above and below.
- Banks should be allocated for the group's address, data read, and data write within the vicinity arena.

  - An error occurs if a bank is allocated outside the vicinity arena.
- The system clock should be selected either to the GC bank (24, 25, 34, and 35) or to the bank adjacent to the capture clock bank.
  - The system clock bank can be selected adjacent to the capture clock bank only when the frequency of this controller is not repeated to any other controllers. If the frequency of this controller is repeated to another controller, the system clock group must be allocated to any one of the GC banks (24, 25, 34, and 35).
  - The signal pairs sys_clk and clk_ref should be allocated to the CC pair or GC pair pins (for banks adjacent to the capture clock bank) or to the GC pair pins (for GC banks).
- The memory clock pairs should be allocated to the differential signals.
- In the DCI CASCADE syntax, the selected configuration should require the master bank.
  - The slave banks provided should be valid.
- A valid MMCM constraint value should be provided, otherwise a warning is generated.

If the UCF satisfies the above rules, the updated design can be generated. The design:

- Provides the latest HDL.
- Updates the UCF with the latest clock constraints or any TIGs provided by keeping the same pinout.
- Generates even the compatible UCFs if the project loaded contains the compatible FPGA selection.

Error Messages

This section describes the error messages that are generated when verifying the UCF. The reference UCF must follow the MIG naming conventions (refer to the UCF generated by the MIG tool, or names used for the ML605 board).

- **Uniqueness**: If two or more signals are allocated to the same pins in the reference UCF, an error message is listed in the directed file with a user-assigned name.

  The error message format is "<*signalname1*> and <*signalname2*> are allocated to the same pin."

  For example, if qdriip_q[0] and qdriip_cq[0] are allocated to the same pin, such as:

  ```
  NET "qdriip_q[0]" LOC = "D12";
  NET "qdriip_cq[0]" LOC = "D12";
  ```

  Then this error message is displayed:

  ```
  ERROR: qdriip_q[0]and qdriip_cq[0] are allocated to the same pin.
  Pins are not unique.
  ```

- **Association**: If the read clock pins are allocated to the SRCC pins, these error messages are displayed:

  ```
  ERROR: Pin Names (qdriip_cq_n[1]) and (qdriip_q[32]) should be
  allocated in the same bank as the strobe pins are allocated to 'SRCC
  P' pin.
  ERROR: Pin Names (qdriip_cq_n[1]) and (qdriip_q[34]) should be
  allocated in the same bank as the strobe pins are allocated to 'SRCC
  P' pin.
  ```

```
ERROR: Pin Names (qdriip_cq_n[1]) and (qdriip_q[33]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_n[1]) and (qdriip_q[35]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[68]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[69]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[65]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[64]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[67]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[66]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[70]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.

ERROR: Pin Names (qdriip_cq_p[3]) and (qdriip_q[71]) should be
allocated in the same bank as the strobe pins are allocated to 'SRCC
P' pin.
```

- **Vicinity Verification**: Error messages are displayed when the pins are allocated out of the vicinity arena.
  - If the data write bank selected is out of the vicinity arena, these error messages are displayed:

```
ERROR: c0_qdriip_d[0](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_d[1](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_d[2](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_d[3](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_d[4](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_d[5](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.
```

```
ERROR: c0_qdriip_d[6](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_d[7](DataWrite) should not be allocated to bank 42.
The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_qdriip_bw[0](ByteWrite) should not be allocated to bank
42. The rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.
```

- **Differential Pair Verification**: If the system clock pins are not allocated to the differential pairs, these error messages are displayed:

```
ERROR: "sys_clk_p" Should be allocated to either CC P pin or GC P
pin.

ERROR: "sys_clk_n" Should be allocated to either CC N pin or GC N
pin.

ERROR: "sys_clk_p" and "sys_clk_n" Should be allocated to either CC
or GC P/N pair.

ERROR: "clk_ref_p" Should be allocated to either CC P pin or GC P
pin.

ERROR: "clk_ref_n" Should be allocated to either CC N pin or GC N
pin.

ERROR: "clk_ref_p" and "clk_ref_n" Should be allocated to either CC
or GC P/N pair.
```

- **Absence of Signals**: If one or more signal pin pairs is missing and/or commented in the given UCF against the selected inputs, the verification result indicates the absence of these signal pin pairs as a warning.

  The warning message format is "Signal <*signal_name*> is expected, but not present in the UCF." For example:

```
WARNING: Signal "qdriip_q[15]" expected, but not present in the UCF.

WARNING: Signal "qdriip_q [16]" expected, but not present in the
UCF.

WARNING: Signal "qdriip_q [17]" expected, but not present in the
UCF.
```

- **Master Bank Verification**: This verifies whether the provided master bank is valid for the selected DCI banks in the column. This error message is displayed when the valid master bank is not provided for the column:

```
ERROR: the master bank "23" provided is not valid master bank.
Following are the valid master bank "24, 25" for the column "1".
```

Core Architecture

This section describes the design implementation of the PHY.

Overview

Figure 2-35 shows a high-level block diagram of the Virtex-6 FPGA QDRII+ SRAM memory interface solution. This figure shows both the internal FPGA connections to the client interface for initiating read and write commands, and the external interface to the memory device.

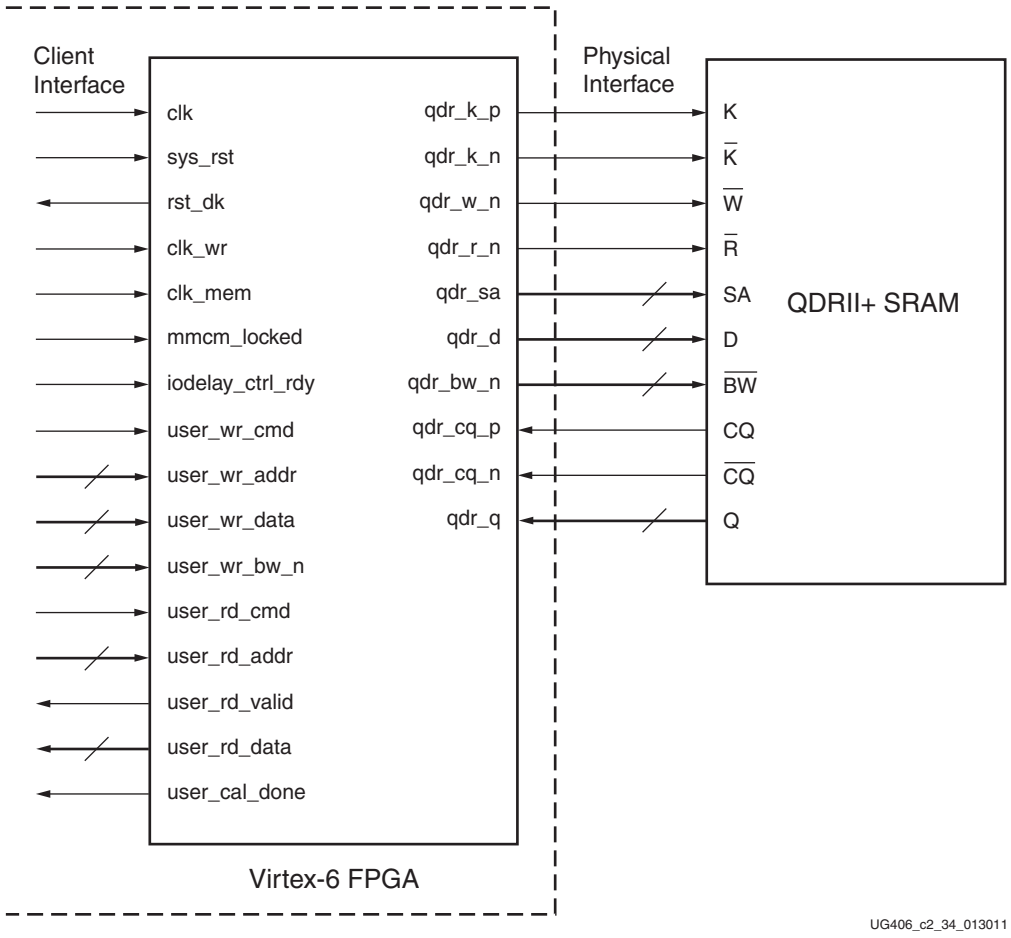


Figure 2-35: **High-Level Block Diagram of the Memory Interface Solution**

The PHY is composed of these elements, as shown in Figure 2-36:

- Client interface
- Physical interface
- Read path
- Write path

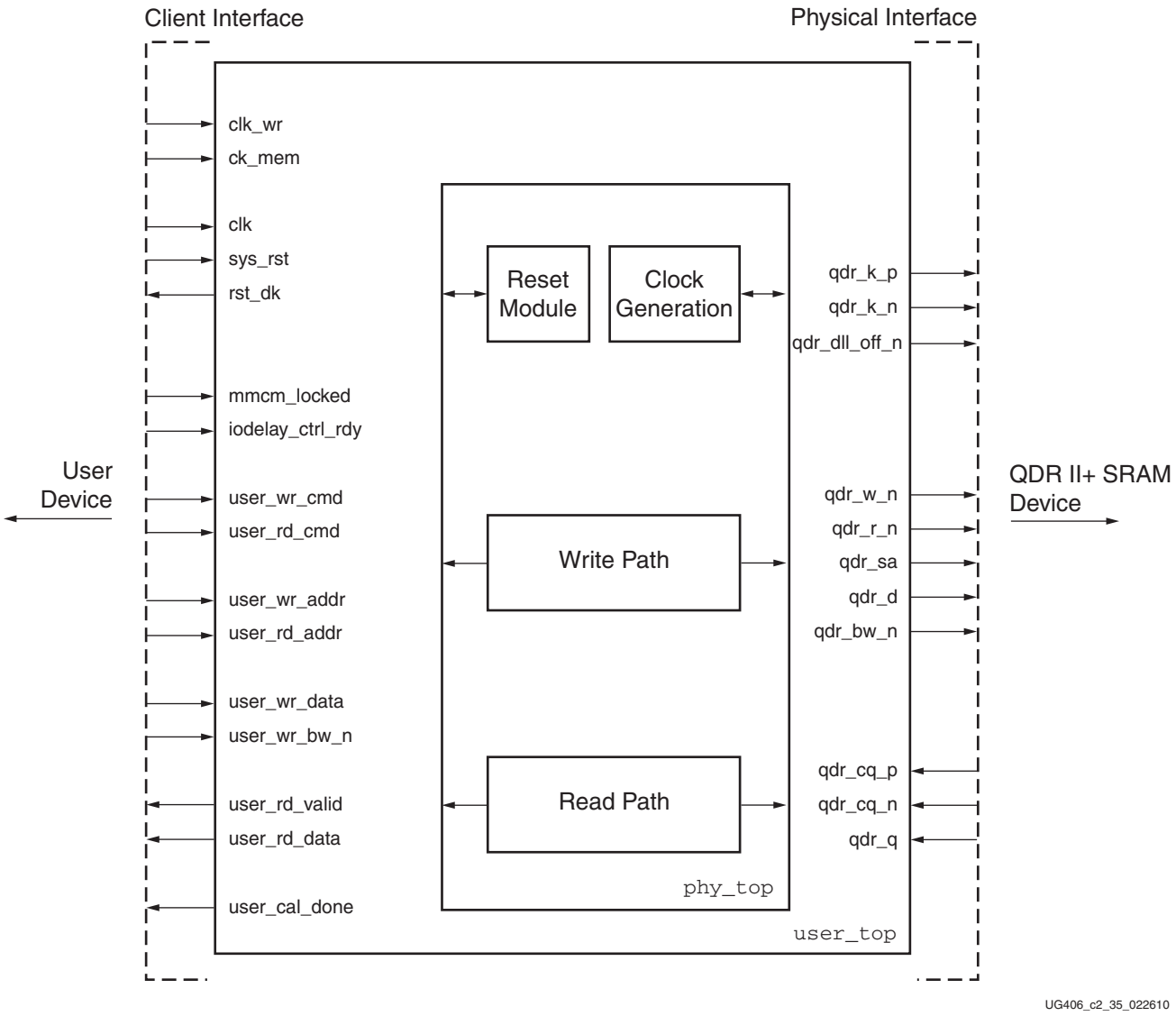


Figure 2-36: **Components of the QDR II+ SRAM Memory Interface Solution**

The client interface (also known as the user interface) uses a simple protocol based entirely on single data rate (SDR) signals to make read and write requests. Refer to Client Interface for more details describing this protocol.

The physical interface generating the proper timing relationships and DDR signaling to communicate with the external memory device, while conforming to QDRII+ protocol and timing requirements. Refer to Physical Interface, page 211 for more details.

Within the PHY, logic is broken up into read and write paths. The write path generates the QDRII+ signaling for generating read and write requests. This includes clocking, control signals, address, data, and byte writes. The read path is responsible for calibration and providing read responses back to the user with a corresponding valid signal. Refer to Calibration, page 216 for more details describing this process.

Client Interface

The client interface connects the Virtex-6 FPGA user design to the QDRII+ SRAM memory solutions core to simplify interactions between the user and the external memory device.

Command Request Signals

The client interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in Table 2-9. To accommodate for burst length 2 devices, the client interface contains ports for two read and two write transactions. When using burst length 4, only the ports ending in 0 should be used. Although the top level contains debug signals, these are left out of Table 2-9 and are described further in Debugging Virtex-6 FPGA QDRII+ SRAM Designs, page 224.

Table 2-9: **Client Interface Request Signals**

Signal	Direction	Description
user_cal_done	Output	**Calibration Done**. This signal indicates to the user design that read calibration is complete and the user can now initiate read and write requests from the client interface.
user_rd_addr0[ADDR_WIDTH – 1:0]	Input	**Read Address**. This bus provides the address to use for a read request. It is valid when user_rd_cmd0 is asserted.
user_rd_cmd0	Input	**Read Command**. This signal is used to issue a read request and indicates that the address on port 0 is valid.
user_rd_data0[DATA_WIDTH × BURST_LEN – 1:0]	Output	**Read Data**. This bus carries the data read back from the read command issued on user_rd_cmd0.
user_rd_valid0	Output	**Read Valid**. This signal indicates that data read back from memory is now available on user_rd_data0 and should be sampled.
user_rd_addr1[ADDR_WIDTH – 1:0]	Input	**Read Address**. This bus provides the address to use for a read request. It is valid when user_rd_cmd1 is asserted.
user_rd_cmd1	Input	**Read Command**. This signal is used to issue a read request and indicates that the address on port 1 is valid.
user_rd_data1[DATA_WIDTH × 2 – 1:0]	Output	**Read Data**. This bus carries the data read back from the read command issued on user_rd_cmd1.

Table 2-9: **Client Interface Request Signals *(Cont'd)***

Signal	Direction	Description
user_rd_valid1	Output	**Read Valid**. This signal indicates that data read back from memory is now available on user_rd_data1 and should be sampled.
user_wr_addr0[ADDR_WIDTH – 1:0]	Input	**Write Address**. This bus provides the address for a write request. It is valid when user_wr_cmd0 is asserted.
user_wr_bw_n0[BW_WIDTH × BURST_LEN – 1:0]	Input	**Write Byte Writes**. This bus provides the byte writes to use for a write request. It is valid when user_wr_cmd0 is asserted. These enables are active Low.
user_wr_cmd0	Input	**Write Command**. This signal is used to issue a write request and indicates that the corresponding sideband signals on write port 0 are valid.
user_wr_data0[DATA_WIDTH × BURST_LEN – 1:0]	Input	**Write Data**. This bus provides the data to use for a write request. It is valid when user_wr_cmd0 is asserted.
user_wr_addr1[ADDR_WIDTH – 1:0]	Input	**Write Address**. This bus provides the address for a write request. It is valid when user_wr_cmd1 is asserted.
user_wr_bw_n1[BW_WIDTH × 2 – 1:0]	Input	**Write Byte Writes**. This bus provides the byte writes to use for a write request. It is valid when user_wr_cmd1 is asserted. These enables are active Low.
user_wr_cmd1	Input	**Write Command**. This signal is used to issue a write request and indicates that the corresponding sideband signals on write port 1 are valid.
user_wr_data1[DATA_WIDTH × 2 – 1:0]	Input	**Write Data**. This bus provides the data to use for a write request. It is valid when user_wr_cmd1 is asserted.

Interfacing with the Core through the Client Interface

The client interface protocol is the same for using the port 0 or port 1 interface signals and is shown in Figure 2-37.

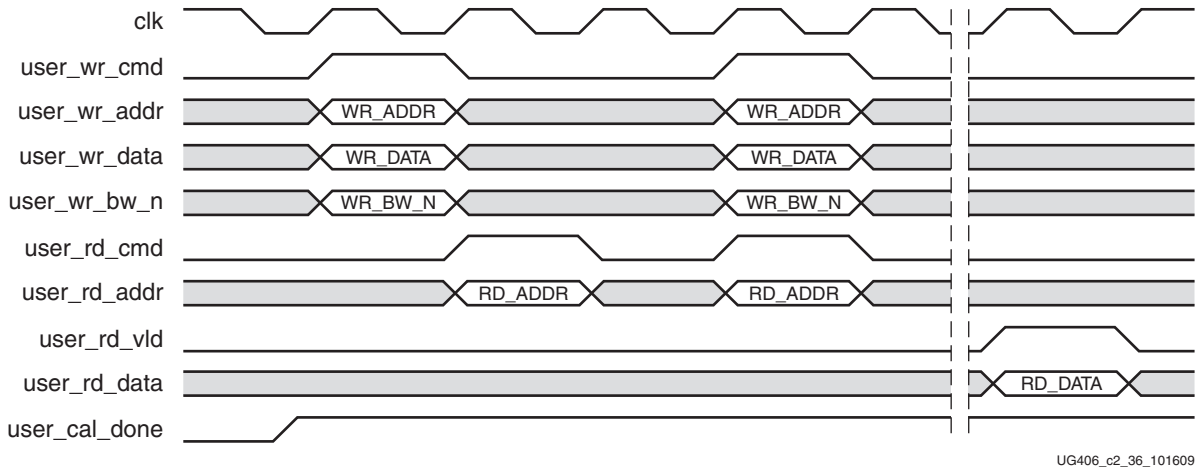


Figure 2-37: **Client Interface Protocol**

Before any requests can be made, the user_cal_done signal must be asserted High, as shown in Figure 2-37, no read or write requests can take place, and the assertion of user_wr_cmd or user_rd_cmd on the client interface is ignored. A write request is issued by asserting user_wr_cmd as a single cycle pulse. At this time, the user_wr_addr, user_wr_data, and user_wr_bw_n signals must be valid. On the following cycle, a read request is issued by asserting user_rd_cmd for a single cycle pulse. At this time, user_rd_addr must be valid. After one cycle of idle time, a read and write request are both asserted on the same clock cycle. In this case, the read to the memory occurs first, followed by the write.

Figure 2-37 also shows data returning from the memory device to the user design. The user_rd_vld signal is asserted, indicating that user_rd_data is now valid. This should be sampled on the same cycle that user_rd_vld is asserted because the core does not buffer returning data. This functionality can be added in by the user, if desired. The data returned is not necessarily from the read commands shown in Figure 2-37 and is solely to demonstrate protocol.

Core Clocking and Reset Requirements

The PHY requires several clocks to function properly. These clocks are described in Table 2-10. As part of calibration, the reset signals used by the core must be tightly controlled, and it is highly recommended that these signals are not altered. To control the reset signals, a reset module exists within the PHY that synchronizes all reset signals and then provides sends them back for use through the client interface. These signals are also shown in Table 2-10.

Table 2-10: **Client Interface Clocking and Reset Signals**

Signal	Direction	Description
clk	Input	**Divided Clock**. This clock is half the frequency of the memory clock and is used as the main system clock.
clk_mem	Input	**Full Frequency Memory Clock**. This is a full-frequency clock provided from the MMCM and should only be used as an input to the OSERDES.
clk_wr	Input	**Write Clock**. This is a full-frequency clock provided from the MMCM.
iodelay_ctrl_rdy	Input	**IODELAY Controller Ready**. This is a signal from the IODELAY controller indicating that the IODELAYs are ready to be used. The PHY is held in reset until the controller is ready.
mmcm_locked	Input	**MMCM Locked**. This signal indicates that the MMCM is locked.
rst_clk	Output	**Divided Clock Reset**. This is the synchronized reset provided from the PHY back to the user's client interface.
sys_rst	Input	**System Reset**. This is the asynchronous reset to be synchronized in the reset module within the PHY. This signal must be held for at least three clk cycles.

Physical Interface

The physical interface is the connection from the FPGA memory interface solution to an external QDRII+ SRAM device. The I/O signals for this interface are shown in Table 2-11. These signals can be directly connected to the corresponding signals on the QDRII+ SRAM device.

Table 2-11: **Physical Interface Signals**

Signal	Direction	Description
qdr_cq_n	Input	**QDR CQ#**. This is the echo clock returned from the memory derived from qdr_k_n. This clock is used by the phase detector circuitry.
qdr_cq_p	Input	**QDR CQ**. This is the echo clock returned from the memory derived from qdr_k_p. This clock is used by the PHY to sample the rising edge data and qdr_q valid signals.
qdr_d	Output	**QDR Data**. This is the write data from the PHY to the QDR II+ memory device.
qdr_dll_off_n	Output	**QDR DLL Off**. This signal turns off the DLL in the memory device.
qdr_bw_n	Output	**QDR Byte Write**. This is the byte write signal from the PHY to the QDRII+ SRAM device.
qdr_k_n	Output	**QDR Clock K#**. This is the inverted input clock to the memory device.
qdr_k_p	Output	**QDR Clock K**. This is the input clock to the memory device.

Table 2-11: **Physical Interface Signals *(Cont'd)***

Signal	Direction	Description
qdr_q	Input	**QDR Data Q**. This is the data returned from reads to memory.
qdr_sa	Output	**QDR Address**. This is the address supplied for memory operations.
qdr_w_n	Output	**QDR Write**. This is the write command to memory.
qdr_r_n	Output	**QDR Read**. This is the read command to memory.

Interfacing with the Memory Device

Figure 2-38 shows the physical interface protocol for a four-word memory device.

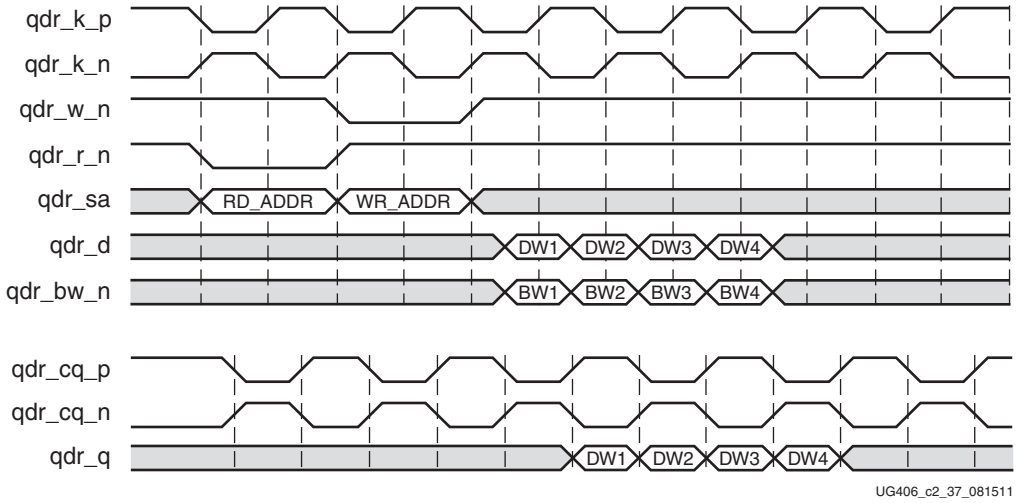


Figure 2-38: **Four-Word Burst Length Memory Device Protocol**

In four-word burst mode:

- The address is in SDR format
- All signals as input to the memory are center aligned with respect to qdr_k_p
- The data for a write request follows on the next rising edge of qdr_k_p after an assertion of qdr_w_n
- Byte writes are sampled along with data
- The qdr_qvld signal is asserted half a cycle before the return of data edge aligned to the qdr_cq_n clock
- The qdr_q signal is edge aligned to qdr_cq_p and qdr_cq_n

Write Path

The write path to the QDRII+ SRAM includes the address, data, and control signals necessary to execute a write operation. The address signals in four-word burst length mode and control signals to the memory all use SDR formatting. The write data values qdr_d and qdr_bw_n also utilize DDR formatting to achieve the required four-word burst within the given clock periods. Figure 2-39 shows a high-level block diagram of the write path and its submodules.

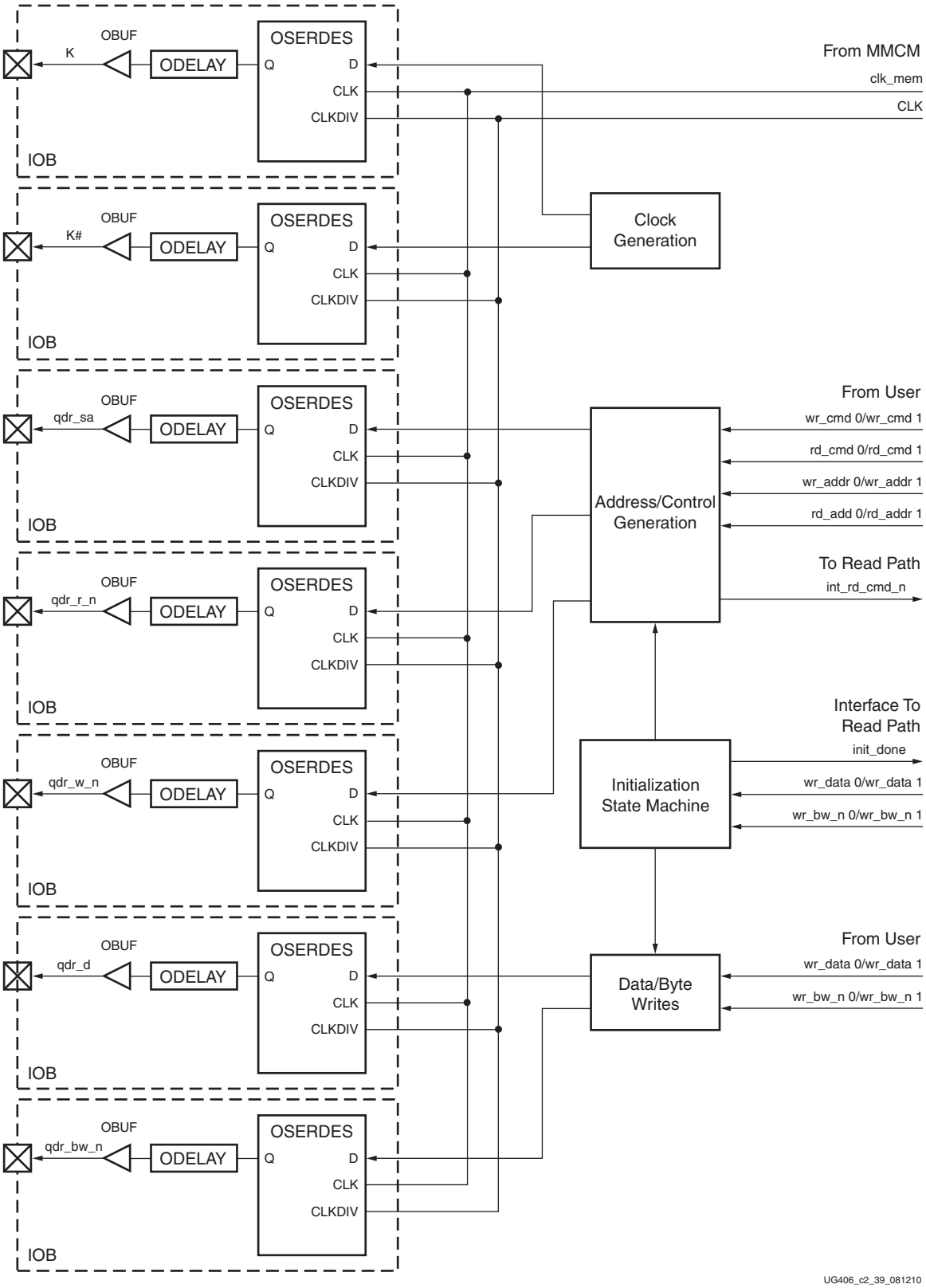


Figure 2-39: **Write Path**

I/O Architecture

The QDRII+ SRAM memory interface solution uses OSERDES primitives found in the Virtex-6 FPGA I/O blocks for clocking all outputs of the PHY to the memory device. Built-in Virtex-6 FPGA OSERDES functions simplify the task of generating the proper clock, address, data, and control signaling for communication with the memory device. The flow through the OSERDES uses two different input clocks to achieve the required functionality. Data input ports D1/D2 or D3/D4 are clocked in using the clock provided on the CLKDIV input port (clk in this case), and then passed through a parallel-to-serial conversion block.

Figure 2-40 shows a high-level block diagram of this flow. The OSERDES is used to clock all outputs from the PHY to the memory device.

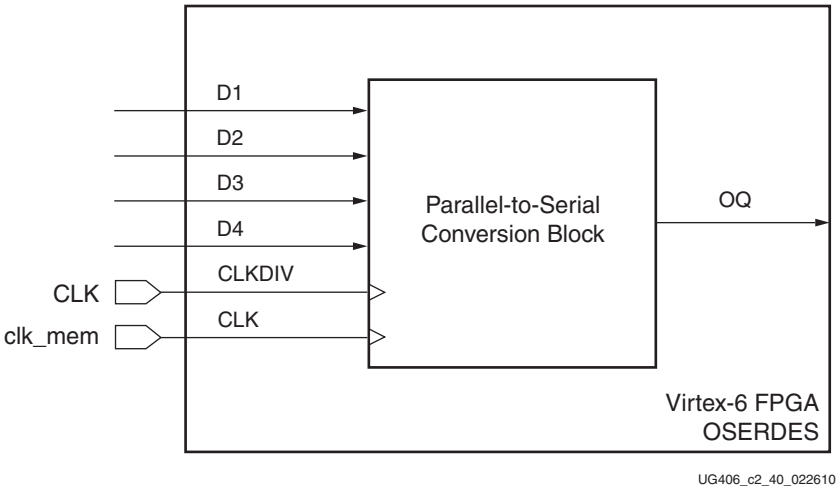


Figure 2-40: **OSERDES Flow**

Upon exiting the OSERDES, all the output signals must be presented center aligned with respect to the generated clocks K/K#. For this reason, the IODELAY blocks within the I/O blocks are also used in conjunction with the OSERDES to achieve center alignment. To avoid inflicting too much jitter on the output signals, the data should not be moved. Instead, the clock, address, and controls are shifted based on the burst length such that the appropriate signals are center aligned.

In addition to generating the output signals to memory, the write path also assists the read path with calibration. This logic performs reads and writes based on signals provided from the read path indicating which stage of calibration the PHY is in. The state machine begins by asserting the init_done signal indicating that calibration can begin. This signal is asserted only after the OSERDES are ready to accept data. Stage one calibration is ready to begin when cal_stage1_start is asserted. During this stage, the pattern `0x00FF_0F0F` is written to the memory device and then read back continuously until signaled to begin stage two. During stage one, CQ is calibrated along with data Q. Stage two calibration requires one write of the pattern `0xAAAA` followed by one read to calibrate the valid signal for read responses. To understand the full calibration process, see Calibration, page 216.

Read Path

The CQ-based data capture scheme enables capture of read data from the memory at very high clock rates. The read path captures the returning data and provides a valid strobe back to the user indicating that the return data is on the client interface. Before any read can take place, calibration must occur. Calibration is the main function of the read path and occurs once on reset followed by continuous dynamic calibration. After the initial settings are in place, dynamic calibration takes over to account for any voltage and temperature changes that might affect the system's once ideal settings.

Data Capture

Figure 2-41 shows a high-level block diagram of the path the data takes from entering the FPGA until given to the user. To capture the data, the read path utilizes an IODELAY and ISERDES that exist in every I/O block on the Virtex-6 FPGA. The IODELAY is used to shift the clocks or data entering the FPGA to adjust its alignment relative to other signals. Following this shift, data then passes through the ISERDES where data is captured using CQ. After the data is retrieved, it enters a data alignment module and optionally realigns, as seen in Figure 2-41. More details about this alignment are covered in Calibration. Data is transferred from the clk_rd domain to the clk domain through a circular buffer built using distributed RAM. In the clk domain, the valid signal is generated and provided with data back to the user.

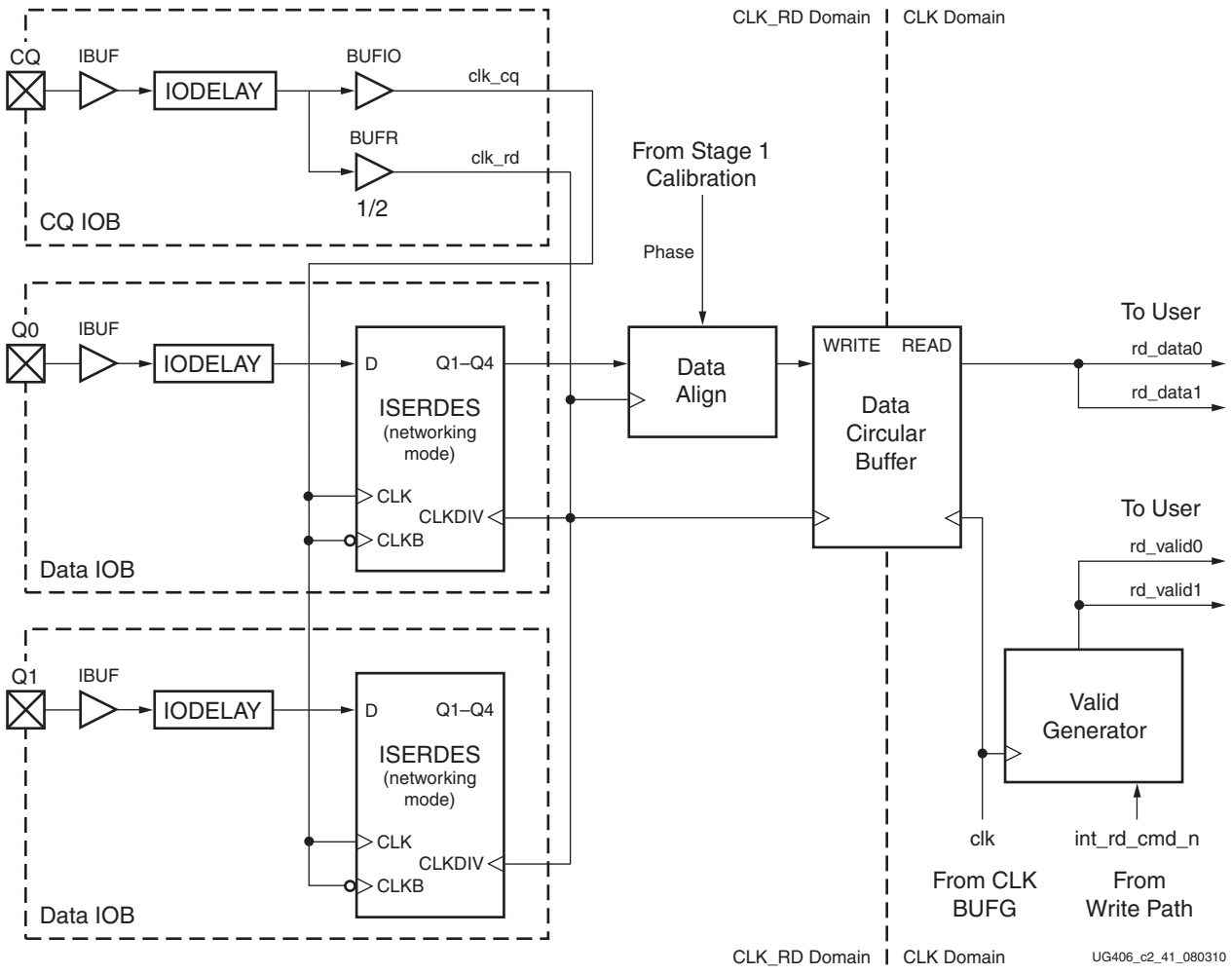


Figure 2-41: **Read Capture**

Calibration

The calibration logic consists of a one-time initial calibration followed by continuous real-time calibration. This logic provides the requested amount of delay on read data (Q) and the read clocks (CQ/CQ#) to align the clock in the data valid window. This is done using the IODELAY elements within the I/O block of the Virtex-6 FPGA. The IODELAY elements delay the input in increments of 75 ps up to a maximum delay of 2.4 ns when using a reference clock frequency of 200 MHz (-1 devices). For a reference clock frequency of 300 MHz (in -3 devices), the delay increment is 52 ps for a maximum delay of 1.6 ns. An IODELAYCTRL module is needed in conjunction with the IODELAYs to maintain the resolution of the IODELAY elements.

Calibration begins after the echo clocks CQ/CQ# are stable from the memory device. The amount of time required to wait for the echo clocks to become stable is based upon the memory vendor and should be specified using the CLK_STABLE parameter to the core. Prior to this point, all read path logic is held in reset. Calibration is performed in two stages:

1. Calibration of CQ with respect to Q, followed by data realignment
2. Resolving latency and valid generation

Calibration of CQ/CQ# and Q and Data Realignment

When the data is returned from memory, it is initially edge aligned to the CQ/CQ# clocks. To safely capture the data, a sample must be taken from the center of the data. Center aligning the CQ clocks to Q provides the greatest possible margin for a successful capture. To assist this stage of calibration, the write path performs an initial write of `0x0FF0_0F0F` followed by continuous reads from this location so that the calibration logic has predefined data to calibrate against.

During calibration, delay adjustments are made from either delaying the clock or data through the use of IODELAYs. The basic flow through this stage of calibration is:

1. Find the best taps setting to center align CQ and Q[0]'s rising and falling edge data for each memory.
2. Perform a fine phase alignment of the ISERDES outputs and find the best tap setting for CQ and Q[0].
3. Determine which phase alignment of the ISERDES outputs provides the best results. (The best result is determined by what delays the data the least and what is most accurately found in the center of Q).
4. Set the selected phase alignment.
5. Each subsequent bit is now calibrated to remove any skew differences relative to Q[0].
6. This process repeats for each memory device on the interface.

Resolving Latency and Valid Generation

This phase of calibration:

- Sets the latency for fixed-latency mode. See Customizing the Core, page 220 for more details describing fixed latency mode.
- Matches the latency for each memory when wider memories are derived from small memories.
- Sends the determined latency to the read valid generation logic.

This stage is required to generate the valid signal associated with the data on the client interface. During this stage of calibration, a single write of `0xAAAA` is written to memory and read back. Doing this allows the read logic to count how many cycles elapse before the expected data returns. The basic flow through this phase is:

1. Count cycles until the read data arrives for each memory device.
2. Determine what value to use as the fixed latency. This value can either be the set value indicated by the user from the PHY_LATENCY parameter or the maximum latency across all memory devices.
3. Calibrate the generation of the read valid signal. Using the value determined in the previous step, delay the read valid signal to align with the read data for user.
4. Assert cal_done.

Dynamic Calibration

After calibration is complete, an ideal tap setting is found to center align the CQ clock to the data. However, due to changes in voltage or temperature, this relationship can shift over time and no longer be ideal. To compensate for this, real-time calibration is performed to add or decrement taps to CQ when necessary.

Reset Module

The reset module synchronizes all the reset signals across all clock domains. This includes the resets from the client interface for the IODELAYCTRL reference clock, the system reset, write path reset, and read path reset. This logic is contained within the PHY because strict timing must be met on the read path reset with respect to the system reset due to the synchronization logic between these two domains. In addition, the read path must remain in reset until echo clocks CQ/CQ# are stable from the memory.

The system reset is provided asynchronously to the reset module and then gated with the appropriate signals to use for synchronization across four different clock domains. All reset signals used by the PHY are asynchronously asserted and synchronously deasserted. Figure 2-42 shows the reset scheme used within the reset module.

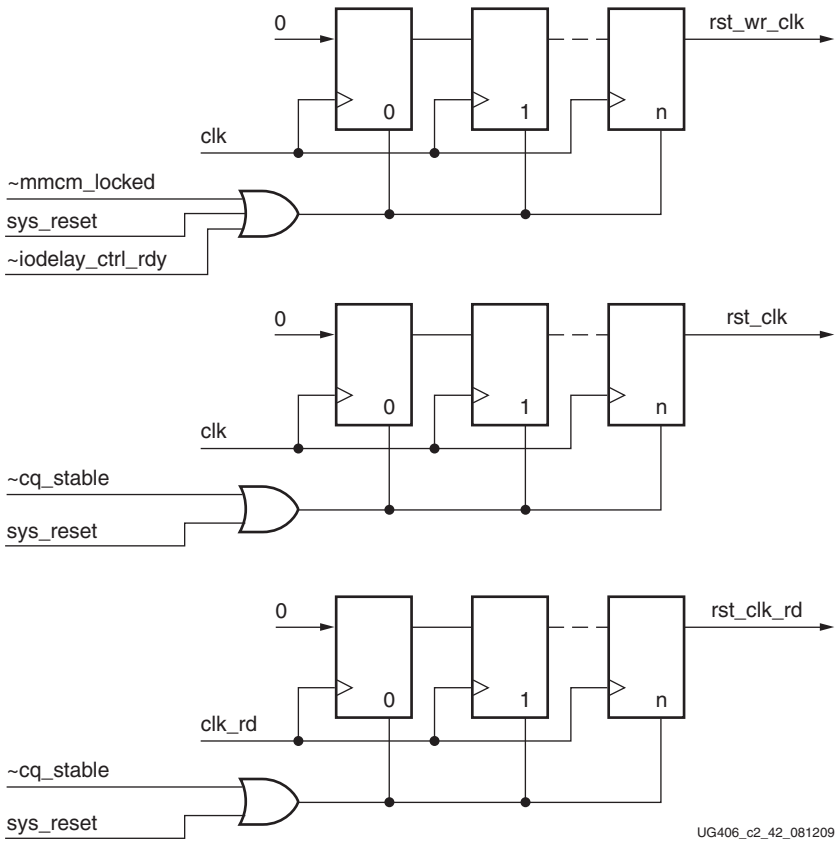


Figure 2-42: **Reset Synchronization**

Table 2-12 indicates the QDRII+ SRAM memory interface solution read latency.

Table 2-12: **Read Latency of the QDRII+ SRAM Memory Interface Solution**

Parameter	Number of Half-Frequency clk Cycles	Description
User command to memory	3.5	1 half clk cycle to align the OSERDESE1 input data. 2.5 half clk cycles of latency inside the OSERDESE1.
Memory read command to valid data available in the BUFR domain	4.5	1 half clk cycle (2 memory clk cycles) read latency from the memory. 2 half clk cycles inside the ISERDESE1. 1.5 half clk cycles to align read data to the rising edge of the ISERDESE1 capture clock (BUFR half frequency clock).
Read data from the BUFR domain to the half clk cycle domain	5	3 half clk cycles for data transfer to the half clk cycle domain through the circular buffer built using distributed RAM. 1 half clk cycle to register circular buffer output data. 1 half clk cycle to align the read output from all memory devices to the same clock edge.
Total read latency	13	

IDELAYCTRL

An IDELAYCTRL is required in any bank that uses IODELAYs. IODELAYs are associated with the data read group, data write group and address/control group. Any bank/clock region where these signals are used requires an IDELAYCTRL.

The MIG tool instantiates one IODELAYCTRL and then uses the IODELAY_GROUP attribute (see the `iodelay_ctrl.v/.vhd` module). Based on this attribute, the ISE software properly replicates IODELAYCTRLs as needed within the design.

The IDELAYCTRL reference frequency varies based on the selected design frequency. If the selected design frequency is 480 MHz and above, the IDELAYCTRL reference clock frequency is 300 MHz, otherwise it is 200 MHz. If multi-controller designs (for example, DDR3 SDRAM multi-controller designs or combinations of DDR3 SDRAM and QDRII controllers) are generated such that one of the controller frequencies is 480 MHz and above and the other controller frequency is below 480 MHz, then IDELAYCTRLs with reference frequencies of both 200 MHz and 300 MHz are generated. When the MIG tool generates a multi-controller design, the MIG tool only instantiates one IODELAYCTRL with this primitive if only one of the 200 MHz or 300 MHz IODELAYCTRLs is required, and allows the tools to replicate. If the design is generated such that both 200 MHz and 300 MHz IODELACYCTRLs are required, the MIG tool instantiates two IODELAYCTRLs with primitives and passes the IODELAY_GROUP parameters accordingly. The MIG tool generates the design such that all 200 MHz IODELAY elements and IODELAYCTRLs use the IODELAY_GROUP parameter value of IODELAY200_MIG and all 300 MHz IODELAYCTRLs and IODELAYs use the IODELAY_GROUP parameter value of IODELAY300_MIG. Based on whether the IODELAY_GROUP attribute is set, the ISE software replicates the IODELAYCTRLs for each region where the IODELAY blocks exist.

Customizing the Core

The Virtex-6 FPGA QDRII+ SRAM memory interface solution is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top level of the core. These parameters are summarized in Table 2-13.

Table 2-13: **Virtex-6 FPGA QDRII+ SRAM Memory Interface Solution Configurable Parameters**

Parameter	Value	Description
ADDR_WIDTH		This is the memory address bus width.
BURST_LEN	4	This is the memory data burst length.
CLK_PERIOD(1)	5,000–16,667 ps	This is the FPGA fabric clock period (ps). This value should be twice that of the memory device clock period and is determined by the speed grade of the FPGA.
CLK_STABLE	(See memory vendor)	This is the number of cycles to wait until the echo clocks are stable.
DATA_WIDTH		This is the memory data bus width and can be set through the MIG tool. A maximum DATA_WIDTH of 72 is supported.
BW_WIDTH		This must be set to DATA_WIDTH/9.
IODELAY_GRP		This is a unique name for the IODELAY_CTRL that is provided when multiple IP cores are used in the design.
DEVICE_ARCH	virtex6	This indicates the targeted device family.
FIXED_LATENCY_MODE	0, 1	This indicates whether or not to use a predefined latency for a read response from the memory to the client interface. If set to 0, the minimum possible latency is used.
MEM_TYPE	QDR2PLUS	This indicates the type of QDR memory device attached. QDR2PLUS is the only supported value for this parameter.
NUM_DEVICES		This is the number of memory devices.
PHY_LATENCY	19 to 30	This indicates the desired latency through the PHY for a read from the time the read command is issued until the read data is returned on the client interface.
REFCLK_FREQ	200.0, 300.0	This is the reference clock frequency for IODELAYCTRLs. This value can be set to 200.0 for any speed grade device or 300.0 for a -2 or -3 device. For more information, see the IODELAYE1 Attribute Summary table in the *Virtex-6 FPGA SelectIO Resources User Guide* [Ref 4].
RST_ACT_LOW	0, 1	This is the active-Low or active-High reset.
SIM_CAL_OPTION	“NONE” “SKIP_CAL” “FAST_CAL”	This parameter is only used during simulation. If this parameter is set to a value other than NONE, it does not work in hardware. This should only be used to speed up simulations. See SIM_CAL_OPTION, page 222.
SIM_INIT_OPTION	“NONE” “SIM_MODE”	This parameter is used only during simulation to speed it up. It should be set to “NONE” to work in hardware.
PHASE_DETECT	“ON” “OFF”	The phase detector logic compensates for any voltage and temperature variation.

Table 2-13: **Virtex-6 FPGA QDRII+ SRAM Memory Interface Solution Configurable Parameters *(Cont'd)***

Parameter	Value	Description
DEBUG_PORT	"ON" "OFF"	Turning on the debug port allows for use with the Virtual I/O (VIO) of the ChipScope analyzer. This allows the user to change the tap settings within the PHY based on those selected though the VIO. This parameter is always set to OFF in the `sim_tb_top` module of the `sim` folder, because debug mode is not required for functional simulations.
IBUF_LPWR_MODE	"ON" "OFF"	This enables or disables low power mode for the input buffers.
IODELAY_HP_MODE	"ON" "OFF"	This enables or disables high-performance mode within the IODELAY primitive. When set to OFF, the IODELAY operates in low power mode at the expense of performance.
INPUT_CLK_TYPE	"DIFFERENTIAL", "SINGLE_ENDED"	This parameter indicates whether the system uses single-ended or differential system clocks/reference clocks. Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, sys_clk_p/sys_clk_n must be used. For single-ended clocks, sys_clk must be used.
CLKFBOUT_MULT_F		This is the MMCM voltage-controlled oscillator (VCO) multiplier. It is set by the MIG tool based on the frequency of operation.
CLKOUT_DIVIDE		This is the VCO output divisor for fast memory clocks. This value is set by the MIG tool based on the frequency of operation.
DIVCLK_DIVIDE		This is the MMCM VCO divisor. This value is set by the MIG tool based on the frequency of operation.

Notes:

1. The lower limit (maximum frequency) is pending characterization.

FIXED_LATENCY_MODE

If desired, the PHY can operate in fixed-latency mode. This is done by setting FIXED_LATENCY_MODE to 1. The latency is measured from when a read command is issued on the physical interface to when the read response is present on the client interface. When set to 1, the desired latency should be set in the PHY_LATENCY parameter.

PHY_LATENCY

This parameter indicates the desired latency from when a read command is issued on the physical interface to when the read response is present on the client interface. This value is only used when the FIXED_LATENCY_MODE parameter is also set. If the value of this parameter is less then the minimum possible latency, the core issues an error through the error port in the top level `user_top` module.

The best way to calculate the PHY_LATENCY value for a specific system is to run the system with FIXED_LATENCY_MODE set to `0` and record the results of the dbg_valid_lat debug signal. To guarantee the latency across multiple controllers, the largest value of

dbg_valid_lat should be used as the value of PHY_LATENCY for all of the controllers in a system.

SIM_CAL_OPTION

Core initialization during simulation can be greatly reducing by using the SIM_CAL_OPTION parameter. Two simulation modes are supported. When set to "FAST_CAL," calibration is performed on only one bit per memory device, and this value is used across the remaining data bits. When set to "SKIP_CAL," no calibration is performed and the incoming clocks and data are assumed to be aligned. This parameter should be set to "NONE" when implementing the design to generate a bitstream, or the core does not function properly in hardware.

Design Guidelines

While the Virtex-6 family offers many advanced I/O and clocking-related features to greatly simplify memory interface design, attention must still be paid to basic board design criteria for a reliable and high-performance interface.

Specifically, the source-synchronous read and write path interfaces require matched board trace lengths for the interface clock, data, and control signals. For example, the trace lengths of the QDRII+ SRAM input signals (qdr_k_p, qdr_k_n, qdr_w_n, qdr_r_n, qdr_sa, qdr_bw_n, and qdr_d) must be well matched to present the control, address, and data lines to the memory device with adequate setup and hold margins. The implementation of the physical interface ensures that these signals are center aligned to the qdr_k_p and qdr_k_n clock edges when leaving the FPGA device outputs. The board traces must ensure that this relationship continues to the memory device inputs.

Similarly, the QDRII+ SRAM output signals (qdr_q, qdr_cq_p, qdr_cq_n) must have well-matched trace lengths for the signals to all arrive edge aligned at the inputs to the Virtex-6 FPGA. This trace length matching is critical to the implementation of the direct-clocking read data capture methodology. Any reasonable board design tool can match these traces within an acceptable tolerance with little effort.

Trace Length Requirements

Trace lengths described here are for high-speed operation and can be relaxed depending on the application's target bandwidth requirements. The package delay should be included when determining the effective trace length. The most accurate and recommended method for determining the delay is to use the L and C values for each pin from the IBIS models. The delay value is determined as the square root of (L × C). Alternatively, a less accurate method is to use the PARTGen utility. These internal delays can be found using the FPGA Editor tool. These rules indicate the maximum skew between QDRII+ SRAM signals:

- The maximum skew between any bit in the data bus, D, and its associated K/K# clocks should be ±15 ps.
- The maximum skew between any Q and its associated CQ/CQ# should be ±15 ps.
- The maximum skew between any address and control signals and the corresponding K/K# should be ±50 ps.
- There is no relation between CQ and the K clocks. K should be matched with D, and CQ should be matched with Q (read data).

Pinout Requirements

After generating a core through the MIG tool, the most optimal pin out has been selected for the design. Manual changes through the UCF are not recommended. However, if the UCF needs to be altered, these rules must be taken into consideration:

- The K/K# clocks should be placed in the same bank as the address and control. For maximum performance, this bank should be located on an inner column. For design frequencies less than 300 MHz, address/control signals can be placed in the outer column. (The address and control can be shared across multiple devices.)
- Write data (D) placement is done based on the address/control bank placement:
  - When the address/control group is allocated in inner column banks, only the inner column banks that reside one row above, one row below, and on the same row of the allocated MMCM can be used for write data (D) selection.
  - When the address/control group is allocated in outer column banks, only the banks that lie one above and/or below the address/control bank can be used for write data (D).
- The write data (D) and byte writes BW_N should be placed in the same bank for a given memory device.
- CQ and CQ# must each be placed on the p-side of a multi-region clock-capable I/O.
- The CQ/CQ# clocks should be together in the same bank as data Q.
- If CQ/CQ#, and data Q do not all fit into one bank, any remaining bits of data Q should be placed in an adjacent bank of the same column.

I/O Standards

The MIG tool generates the appropriate UCF for the core with select I/O standards based on the type of input or output to the Virtex-6 FPGA. These standards should not be changed. Table 2-14 contains a list of the ports together with the I/O standard used.

Table 2-14: **I/O Standards**

Signal[1]	Direction	I/O Standard
qdr_bw_n	Output	HSTL_I
qdr_cq_p, qdr_cq_n	Input	HSTL_I_DCI
qdr_d	Output	HSTL_I
qdr_k_p, qdr_k_n	Output	HSTL_I
qdr_q	Input	HSTL_I_DCI
qdr_r_n	Output	HSTL_I
qdr_sa	Output	HSTL_I
qdr_w_n	Output	HSTL_I

Notes:

1. All signals operate at 1.5V.

Debugging Virtex-6 FPGA QDRII+ SRAM Designs

This section defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the memory interface design process.

Introduction

The QDRII+ memory interfaces in Virtex-6 FPGAs simplify the challenges associated with memory interface design. However, every application environment is unique and proper due diligence is required to ensure a robust design. Careful attention must be given to functional testing through simulation, proper synthesis and implementation, adherence to PCB layout guidelines, and board verification through IBIS simulation and signal integrity analysis.

This section defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the design process. Details are provided on:

- Functional verification using the UNISIM simulation models
- Design implementation verification
- Board layout verification
- Using the QDRII+ SRAM physical layer to debug board-level issues
- General board-level debug techniques

The two primary issues encountered during verification of a memory interface are:

- Calibration not completing properly
- Data corruption during normal operation

Problems might be seen in simulation, hardware, or both due to various root causes. Figure 2-43 shows the overall flow for debugging problems associated with these two general types of issues.

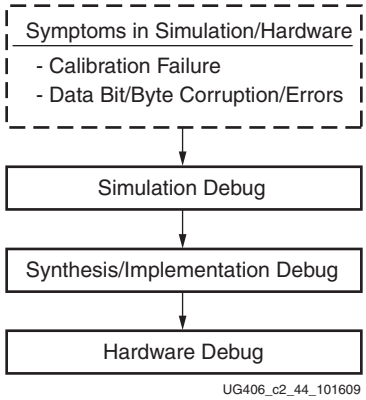


Figure 2-43: **Virtex-6 FPGA QDRII+ SRAM MIG Tool Debug Flowchart**

Debug Tools

Many tools are available to debug memory interface design issues. This section indicates which resources are useful for debugging a given situation.

Example Design

QDRII+ SRAM design generation using the MIG tool produces an example design and a user design. The example design includes a synthesizable test bench that has been fully verified in simulation and hardware. This design can be used to observe the behavior of the MIG tool design, and can also aid in identifying board-related problems. Quick Start Example Design, page 174 provides complete details about the example design. This section also describes using the example design to verify setup of a proper simulation environment and to perform hardware validation.

Debug Signals

The MIG tool includes a Debug Signals Control option on the FPGA Options screen. Enabling this feature allows calibration, tap delay, and read data signals to be monitored using the ChipScope analyzer. Selecting this option port maps the debug signals to VIO modules of the ChipScope analyzer in the design top module. Getting Started, page 174, provides details on enabling this debug feature.

ChipScope Pro Tool

The ChipScope Pro tool inserts logic analyzer, bus analyzer, and VIO software cores directly into the design. The ChipScope Pro tool allows the user to set trigger conditions to capture application and the MIG tool signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool [Ref 6].

Simulation Debug

Figure 2-44 shows the debug flow for simulation.

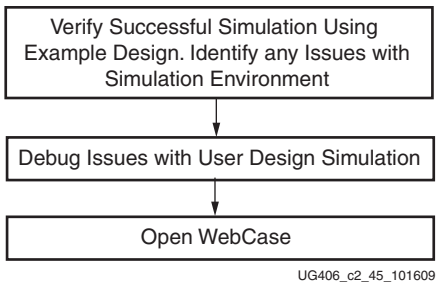


Figure 2-44: **Simulation Debug Flowchart**

Verifying the Simulation Using the Example Design

The example design generated by the MIG tool includes a simulation test bench and parameter file based on memory selection in the MIG tool, and a ModelSim `.do` script file. The MIG tool does not provide a QDRII+ memory model. A QDRII+ memory model must be provided and added to the simulation by the user. Refer to Quick Start Example Design, page 174 for detailed steps on running the example design simulation. Successful completion of this example design simulation verifies a proper simulation environment. This shows that the simulation tool and Xilinx libraries are set up correctly. For detailed information on setting up Xilinx libraries, refer to COMPXLIB in the *Command Line Tools User Guide* [Ref 7] and the *Synthesis and Simulation Design Guide* [Ref 8]. For simulator tool support, refer to the *Virtex-6 FPGA Memory Interface Solutions Data Sheet* [Ref 9].

A working example design simulation completes memory initialization and runs traffic in response to the test bench stimulus. Successful completion of memory initialization and calibration results in the assertion of the cal_done signal. When this signal is asserted, the test bench takes control and begins executing writes and reads according to its parameterization.

Table 2-15 and Table 2-16 show the signals and parameters of interest, respectively, during simulation.

Table 2-15: **Signals of Interest During Simulation**

Signal Name	Usage
cal_done	This signal indicates completion of calibration.
compare_error	This signal indicates a mismatch between the data written from the UI and data received during a read on the UI. This signal is a part of the example design. A single error asserts this signal and is held until the design is reset.
cmp_err	This signal indicates a mismatch between the data written from the UI and the data received during a read on the UI. This signal is a part of the example design. This signal is asserted each time a data mismatch occurs.
user_rd_addr	This is the address provided for the read command.
user_rd_cmd	This signal indicates that the read address is valid for a read command.
user_rd_data	This is the read data being returned from the memory device.
user_rd_valid	This signal is asserted when user_rd_data is valid.
user_wr_addr	This is the address provided for the write command.
user_wr_bw_n	This signal is the byte write control.
user_wr_cmd	This signal indicates that the write address and write data are valid for a write command.
user_wr_data	This is the write data for a write command.

Table 2-16: **Parameters of Interest During Simulation**

Signal Name	Usage
SIM_INIT_OPTION	This parameter sets the simulation initialization procedure.
SIM_CAL_OPTION	This parameter sets the simulation calibration procedure.

When the SIM_INIT_OPTION is set to SIM_MODE, and the SIM_CAL_OPTION is set to FAST_CAL, the MIG tool design executes an abbreviated calibration sequence. For the design to properly initialize and calibrate the full memory array in hardware, the top-level MIG tool design file (`example_top.v/vhd`) cannot use any abbreviated values for these parameters. The MIG tool output properly sets the abbreviated values in the test bench and the full range of values in the top-level design module.

Figure 2-45 shows a high-level view of a successful simulation using the provided example design with the abbreviated simulation parameters set, as described in Table 2-15, page 226. The simulation can be divided into these main sections: Memory Initialization, Calibration, and Test Bench.

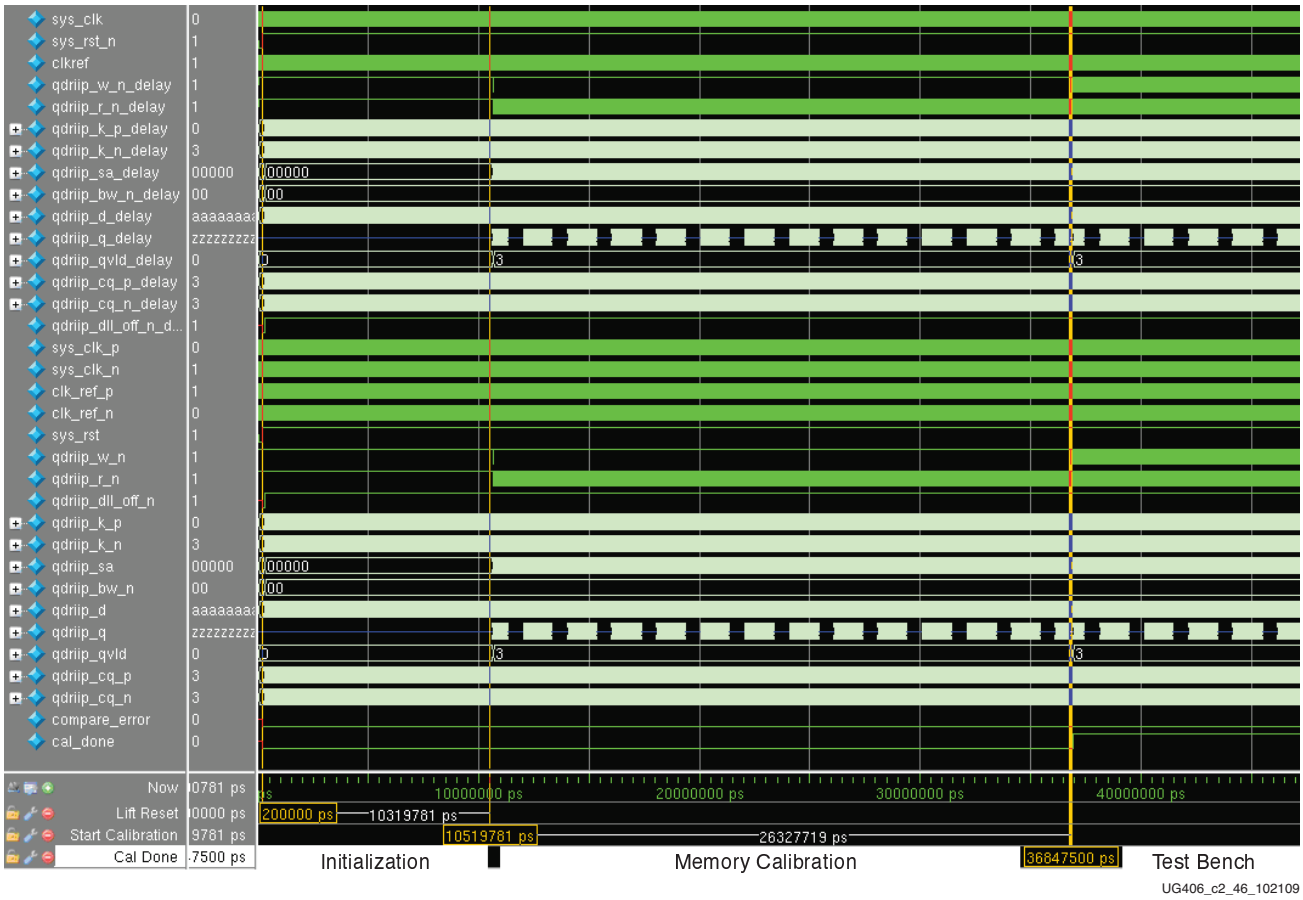


Figure 2-45: **Waveforms and Simulation Transcripts Showing Successful Example Design Completion**

Memory Initialization

The QDRII+ memories do not require an elaborate initialization procedure. However, the user must ensure that the Doff_n signal is provided to the memory as required by the vendor. The QDRII+ SRAM interface design provided by the MIG tool drives the Doff_n signal from the FPGA. After the internal MMCM has locked after a wait period of 200 µs, the Doff_n signal is asserted High. After the Doff_n signal assertion and following CLK_STABLE (set to 2048) number of CQ clock cycles, commands are issued to the memory.

For memory devices that require the Doff_n signal to be terminated at the memory and not be driven from the FPGA, the user must perform the required termination procedure.

Calibration

Calibration completes read leveling, write calibration, and read enable calibration. This is completed over two stages. This sequence successfully completes when the cal_done signals asserts. For more details, refer to PHY, page 102.

The first stage performs per-bit read leveling calibration. The data pattern used during this stage is `00ff00ff00ffff00`. The data pattern is first written to the memory, as shown in Figure 2-46.

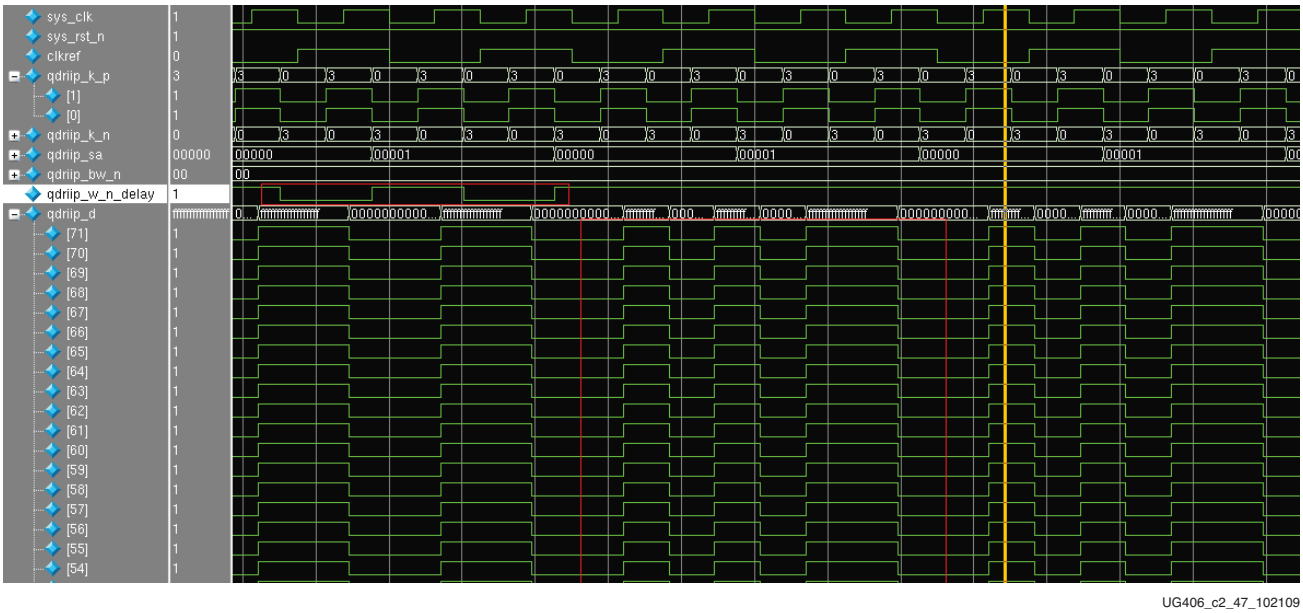


Figure 2-46: **Writes for First Stage Read Calibration**

This pattern is then continuously read back while the per-bit calibration is completed, as shown in Figure 2-47.

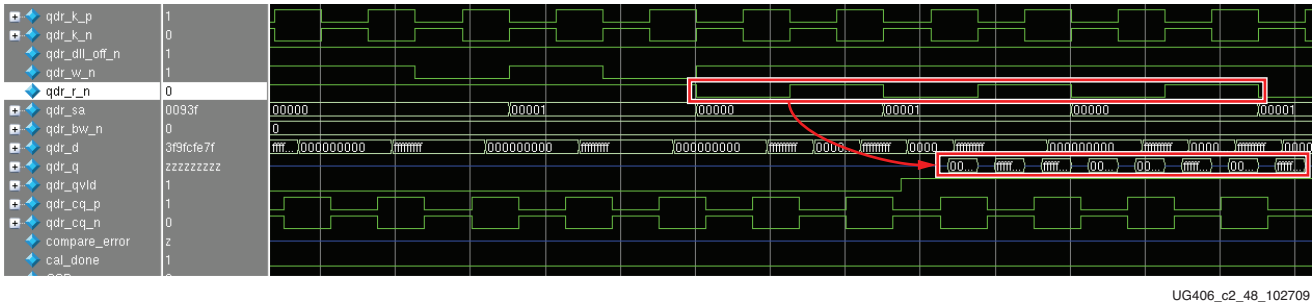


Figure 2-47: **Reads for First Stage Read Calibration**

The second stage performs a read enable calibration. The data pattern used during this stage is AAAA. The data pattern is first written to the memory, and then read back for the read enable calibration, as shown in Figure 2-48.

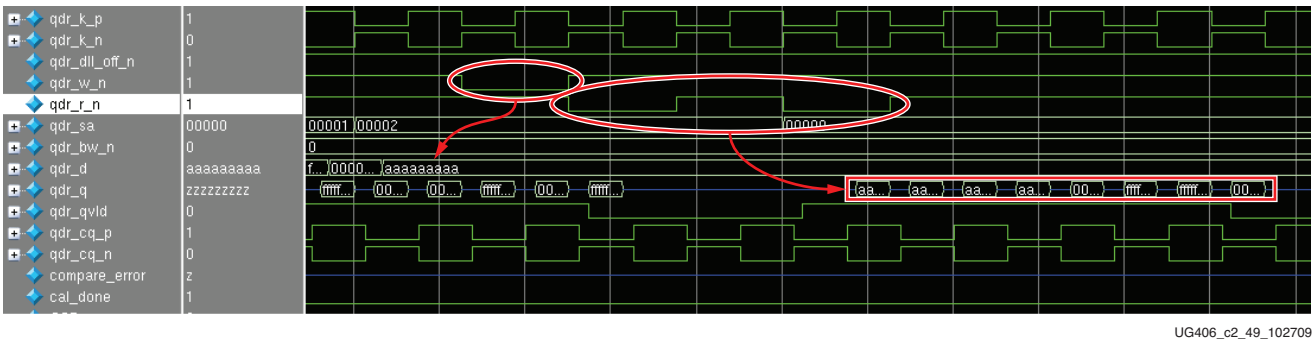


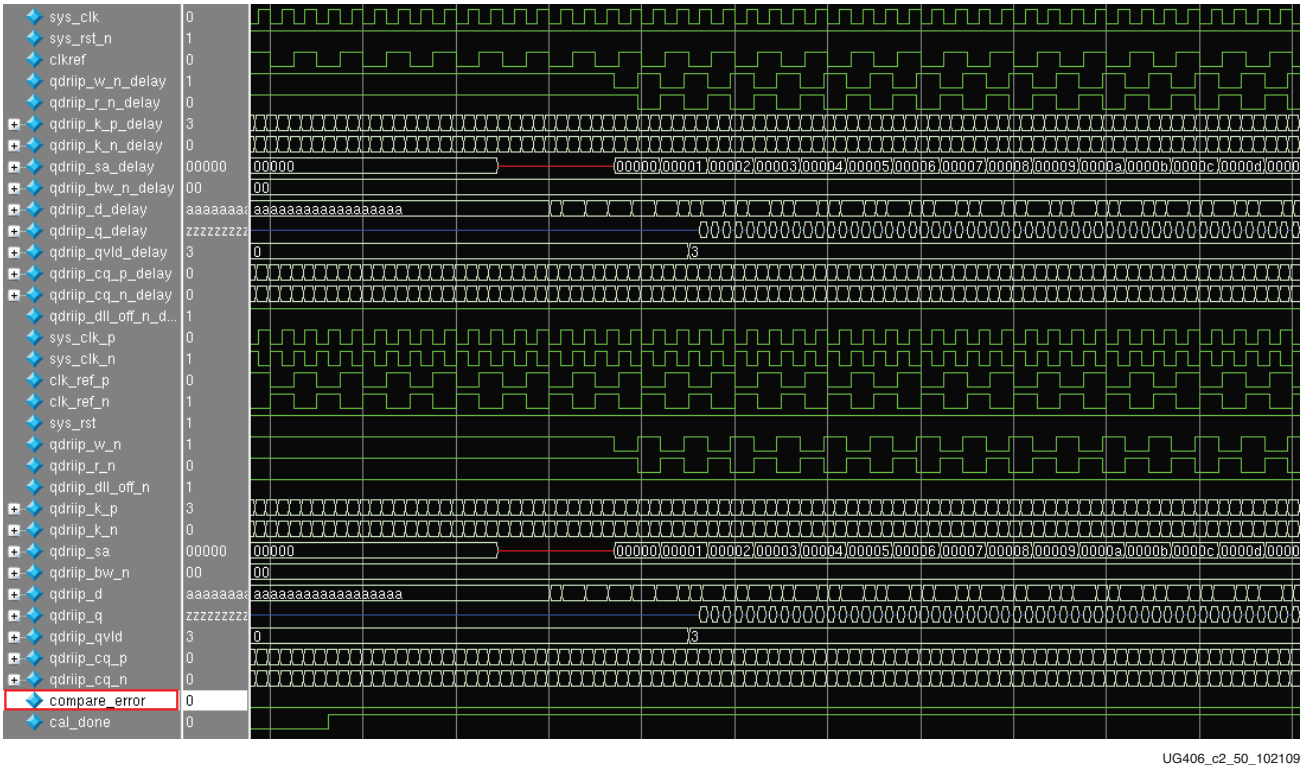
Figure 2-48: **Write and Read for Second Stage Read Calibration**

An additional read is performed so the read bus is driven to a different value. This is mostly required in hardware to make sure that the read calibration can distinguish the correct data pattern.

After second stage calibration completes, cal_done asserts, signifying successful completion of the calibration process.

Test Bench

After cal_done asserts, the test bench takes control, writing to and reading from the memory. The data written is compared to the data read back. Any mismatches trigger an assertion of the error signal. Figure 2-49 shows a successful implementation of the test bench with no assertions on error.



UG406_c2_50_102109

Figure 2-49: **Test Bench Operation After Completion of Calibration**

Debug Issues with User Design Simulation

After the simulation environment and parameter settings are verified by successful simulation of the example design, issues with the user design simulation can be investigated. Because the environment and parameters are verified to work properly, calibration of the user design completes without error as long as no RTL changes exist.

Data Errors

Issues that might be seen with user design simulation exist within the generation of user writes and reads. Thus, it is crucial to understand how to drive the UI to properly send writes and reads. For more information, refer to User Interface, page 68 and Interfacing to the Core, page 113.

Proper Write and Read Commands

When sending write and read commands, the user must properly assert and deassert the corresponding UI inputs. Refer to User Interface, page 68 and Interfacing to the Core, page 113 for full details. The test bench design provided within the example design can be used as a further source of proper behavior on the UI.

To debug data errors on the QDRII+ SRAM interface, it is necessary to pull the UI signals into the simulation waveform.

In the ModelSim Instance window, highlight **u_ip_top** to display the necessary UI signals in the Objects window, as shown in Figure 2-50. Highlight the user interface signals noted in Table 2-15, page 226, right-click, and select **Add** → **To Wave** → **Selected Signals**.

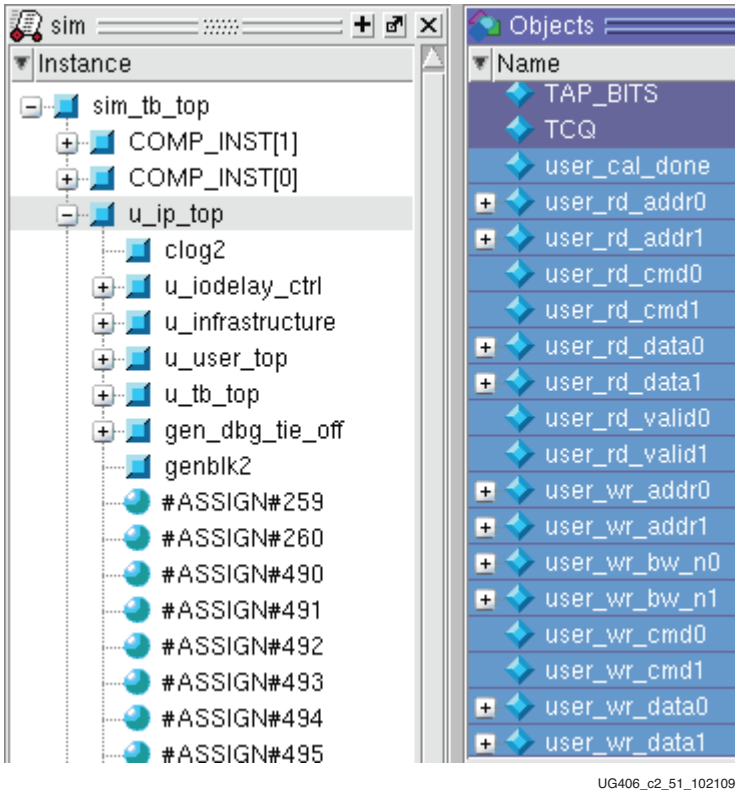


Figure 2-50: **ModelSim Instance Window**

Figure 2-51 and Figure 2-52 show example waveforms of a write and read on both the user interface and QDR interface.

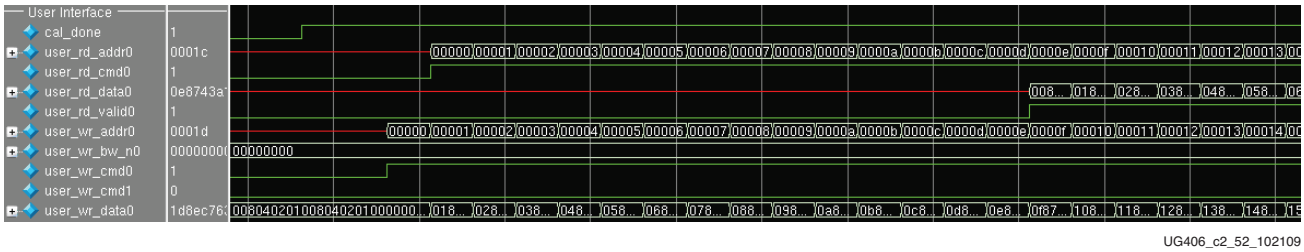


Figure 2-51: **User Interface Write and Read**

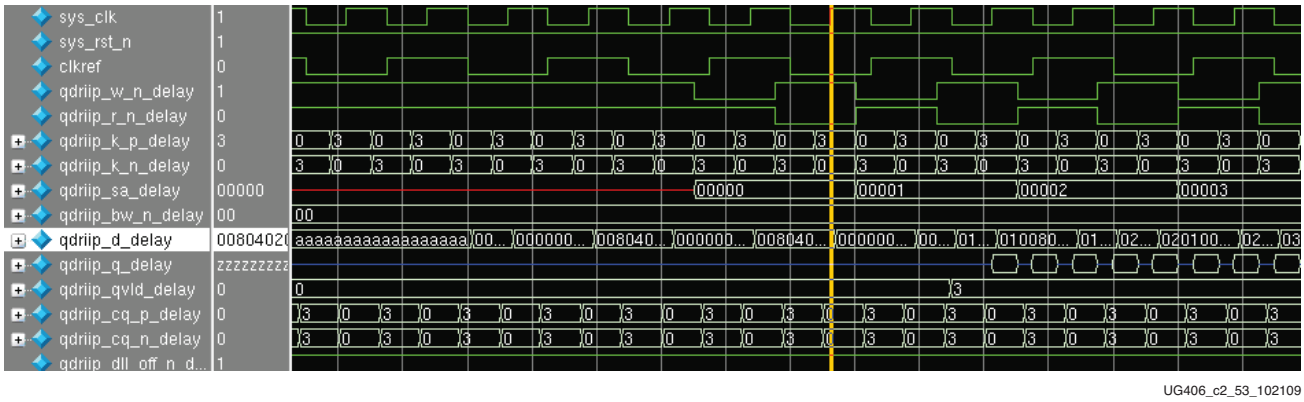


Figure 2-52: **QDRII+ Interface Write and Read**

Synthesis and Implementation Debug

Figure 2-53 shows the debug flow for synthesis and implementation.

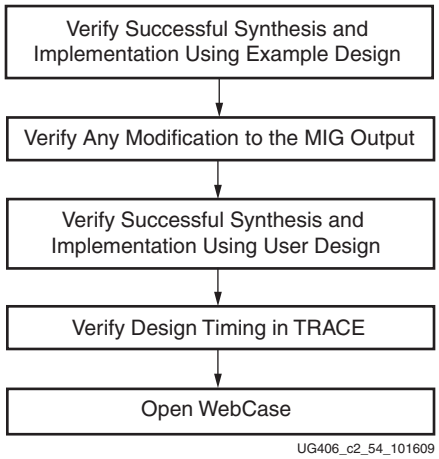


Figure 2-53: **Synthesis and Implementation Debug Flowchart**

Verify Successful Synthesis and Implementation

The example design and user design generated by the MIG tool include synthesis/implementation script files and user constraint files (.ucf). These files should be used to properly synthesize and implement the targeted design and generate a working bitstream.

The synthesis/implementation script file, called `ise_flow.bat`, is located in both `example_design/par` and `user_design/par` directories. Execution of this script runs either the example design or the user design through synthesis, translate, MAP, PAR, TRACE, and BITGEN. The options set for each of these processes are the only options that have been tested with the QDRII+ SRAM MIG tool designs. A successfully implemented design completes all processes with no errors (including zero timing errors).

Verify Modifications to the MIG Tool Output

The MIG tool allows the user to select the FPGA banks for the memory interface signals. Based on the banks selected, the MIG tool outputs a UCF with all required location constraints. This file is located in both the `example_design/par` and `user_design/par` directories and should not be modified.

The MIG tool outputs open source RTL code parameterized by top-level HDL parameters. These parameters are set by the MIG tool and should not be modified manually. If changes are required, such as decreasing or increasing the frequency, the MIG tool should be rerun to create an updated design. Manual modifications are not supported and should be verified independently in behavioral simulation, synthesis, and implementation.

Identifying and Analyzing Timing Failures

The MIG tool QDRII+ SRAM designs have been verified to meet timing using the example design across a wide range of configurations. However, timing violations might occur, such as when integrating the MIG tool design with the user's specific application logic.

Any timing violations that are encountered must be isolated. The timing report output by TRACE (`.twx/.twr`) should be analyzed to determine if the failing paths exist in the MIG tool QDRII+ SRAM design or the UI (backend application) to the MIG tool design. If failures are encountered, the user must ensure the build options (that is, XST, MAP, PAR) specified in the `ise_flow.bat` file are used.

If failures still exist, Xilinx has many resources available to aid in closing timing. The PlanAhead™ tool [Ref 10] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 11] provides valuable information on all available Xilinx constraints.

Hardware Debug

Figure 2-54 shows the debug flow for hardware.

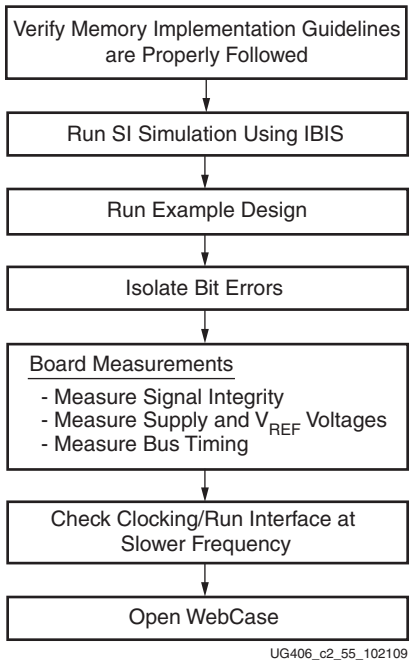


Figure 2-54: **Hardware Debug Flowchart**

Verify Design Guidelines

See Design Guidelines, page 222 for specifications on termination, I/O standards, and trace matching. The guidelines provided therein are specific to the QDRII+ SRAM. It is important to verify that these guidelines have been referred to during board layout. Failure to follow these guidelines or modifications to a MIG tool provided pinout, or both, can result in problematic behavior in hardware as discussed in this debugging section.

Clocking

The external clock source should be measured to ensure frequency, stability (jitter), and usage of the expected FPGA pin. The designer must ensure that the design follows all clocking guidelines. If clocking guidelines have been followed, the interface should be run at a slower speed. Not all designs or boards can accommodate slower speeds. Lowering the frequency increases the marginal setup or hold time, or both, due to PCB trace mismatch, poor signal integrity, or excessive loading. When lowering the frequency, the MIG tool should be rerun to regenerate the design with the lower clock frequency. Portions of the calibration logic are sensitive to the CLK_PERIOD parameter; thus, manual modification of the parameter is discouraged.

Verify Board Pinout

The user should ensure that the pinout provided by the MIG tool is used without modification. Then, the board schematic should be compared to the `<design_name>.pad` report generated by PAR. This step ensures that the board pinout matches the pins assigned in the implemented design.

Run Signal Integrity Simulation with IBIS Models

To verify that board layout guidelines have been followed, signal integrity simulations must be run using the I/O buffer information specification (IBIS). These simulations should always be run for both pre-board and post-board layouts. The purpose of running these simulations is to confirm the signal integrity on the board.

The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 12] can be used as a guideline. This chapter provides a detailed look at signal integrity correlation results for the ML561 board. It can be used as an example for signal integrity analysis. It also provides steps to create a design-specific IBIS model to aid in setting up the simulations. While this guide is specific to Virtex-5 devices and the ML561 development board, the principles therein can be applied to Virtex-6 FPGA MIG designs.

Run the Example Design

The MIG tool provided example design is a fully verified design that can be used to test the memory interface on the board. It rules out any issues with the backend logic interfacing with the MIG tool core. In addition, the test bench provided by the MIG tool can be modified to send out different data patterns that test different board-level concerns.

Debugging Common Hardware Issues

When calibration failures and data errors are encountered in hardware, the ChipScope analyzer should be used to analyze the behavior of MIG tool core signals. For detailed information about using the ChipScope analyzer, refer to the *ChipScope Pro 11.1 Software and Cores User Guide* [Ref 14].

A good starting point in hardware debug is to load the provided example_design onto the board in question. This is a known working solution with a test bench design that checks for data errors. This design should complete successfully with the assertion of cal_done and no assertions of compare_error. Assertion of cal_done signifies successful completion of calibration while no assertions of compare_error signifies that the data is written to and read from the memory compare with no data errors.

The cmp_err signal can be used to indicate if a single error was encountered or if multiple errors are encountered. With each error encountered, cmp_err is asserted so that the data can be manually inspected to help track down any issues.

Isolating Bit Errors

An important hardware debug step is to try to isolate when and where the bit errors occur. Looking at the bit errors, these should be identified:

- Are errors seen on data bits belonging to certain CQ clock groups?
- Are errors seen on accesses to certain addresses of memory?
- Do the errors only occur for certain data patterns or sequences?

  This can indicate a shorted or open connection on the PCB. This can also indicate an SSO or crosstalk issue.

It might be necessary to isolate whether the data corruption is due to writes or reads. This case can be difficult to determine because if writes are the cause, read back of the data is bad as well. In addition, issues with control or address timing affect both writes and reads. Some experiments that can be tried to isolate the issue are:

- If the errors are intermittent, have the design issue a small initial number of writes, followed by continuous reads from those locations. If the reads intermittently yield bad data, there is a potential read problem.
- Check/vary only write timing:
  - Check that the external termination resistors are populated on the PCB.
  - Use ODELAY to vary the phase of D relative to the K clocks.
- Vary only read timing:
  - Check the IDELAY values after calibration. Look for variations between IDELAY values. IDELAY values should be very similar for Qs in the same CQS group.
  - Vary the IDELAY taps after calibration for the bits that are returning bad data. This affects only the read capture timing.

Debugging the Core

The Debug port is a set of input and output signals that either provide status (outputs) or allow the user to make adjustments as the design is operating (inputs). When generating the QDRII+ SRAM design through the MIG tool, an option is provided to turn the Debug Port on or off. When the Debug port is turned off, the outputs of the debug port are still generated but the inputs are ignored. When the Debug port is turned on, the inputs are valid and must be driven to a logical value. Driving the signals incorrectly on the debug port might cause the design to fail or have less read data capture margin.

When running the core in hardware, a few key signals should be inspected to determine the status of the design. The dbg_phy_status bus described in Table 2-17 consists of status bits for various stages of calibration. Checking the dbg_phy_status bus gives initial information that can aid in debugging an issue that might arise, determining which portion of the design to look at, or looking for some common issues.

Table 2-17: **Physical Layer Simple Status Bus Description**

Debug Port Signal	Name	Description	If Problems Arise
dbg_phy_status[0]	iodelay_ctrl_rdy	IODELAY blocks are ready to be used.	Check that the IODELAY clock is supplied properly with the expected frequency.
dbg_phy_status[1]	mmcm_locked	MMCM has locked and is generating the system clocks.	Check that the system clock is supplied properly with the expected frequency. Check the polarity of the reset.
dbg_phy_status[2]	init_done	QDRII+ SRAM initialization sequence is complete.	N/A
dbg_phy_status[3]	cal_stage1_start	Stage 1 read calibration start signal.	N/A

Table 2-17: **Physical Layer Simple Status Bus Description *(Cont'd)***

Debug Port Signal	Name	Description	If Problems Arise
dbg_phy_status[4]	cal_stage2_start	Stage 2 read calibration start signal after stage 1 calibration is completed.	If this signal does not go High, then stage 1 has not completed. Make sure the expected data is being returned from the memory.
dbg_phy_status[5]	dbg_pd_calib_start	Phase detector calibration start signal.	If this signal does not go High, then stage2 calibration has not completed. Check the IODELAY values set for stage 1 read calibration and check the data for stage 2.
dbg_phy_status[6]	dbg_pd_calib_done	Phase detector calibration complete.	Check the system clock frequency as the phase detector has a lower frequency limit. (The phase detector should be off below 250 MHz.)
dbg_phy_status[7]	cal_done	Calibration complete.	N/A

Notes:

1. N/A indicates that as long as previous stages have completed this stage is also completed.

The results of read calibration are provided as part of the Debug port as various output signals. These signals can be used to capture and evaluate the results of read calibration. Read calibration uses the IODELAY to center the capture clock in the data valid window for captured data. The algorithm shifts the IODELAY values and looks for edges of the data valid window on a per-bit basis as part of the calibration procedure.

DEBUG_PORT Signals

The top-level wrapper, `user_top`, provides several output signals that can be used to debug the core if the debug option is checked when generating the design through the MIG tool. Each debug signal output is prefixed with "dbg_." The DEBUG_PORT parameter is always set to OFF in the `sim_tb_top` module of the `sim` folder that disables the debug option for functional simulations. These signals are listed in Table 2-18 along with descriptions of the data they provide.

Table 2-18: **Core Debug Signals**

Signal	Valid Clock Domain	Direction	Description
dbg_phy_wr_cmd_n[1:0]	clk	Output	This active-Low signal is the internal wr_cmd used for debug with the ChipScope analyzer.
dbg_phy_rd_cmd_n[1:0]	clk	Output	This active-Low signal is the internal rd_cmd used for debug with the ChipScope analyzer.

Table 2-18: **Core Debug Signals *(Cont'd)***

Signal	Valid Clock Domain	Direction	Description
dbg_phy_addr[ADDR_WIDTH × 4 – 1:0]	clk	Output	This is the control addresses bus used for debug with the ChipScope analyzer.
dbg_phy_wr_data[DATA_WIDTH × 4 – 1:0]	clk	Output	This is the data being written that is used for debug with the ChipScope analyzer.
dbg_inc_cq_all	clk	Input	This signal increments taps on all CQ clock bits.
dbg_inc_cqn_all	clk	Input	This signal increments taps on all CQ# clock bits.
dbg_inc_q_all	clk	Input	This signal increments taps on all data Q bits.
dbg_dec_cq_all	clk	Input	This signal decrements taps on all CQ clock bits.
dbg_dec_cqn_all	clk	Input	This signal decrements taps on all CQ# clock bits.
dbg_dec_q_all	clk	Input	This signal decrements taps on all data Q bits.
dbg_inc_cq	clk	Input	This signal increments the select clock CQ bit.
dbg_dec_cq	clk	Input	This signal decrements the select clock CQ bit.
dbg_sel_cq[CQ_BITS – 1:0]	clk	Input	This is the selected CQ bit to modify.
dbg_inc_cqn	clk	Input	This signal increments the select clock CQ# bit.
dbg_dec_cqn	clk	Input	This signal decrements the select clock CQ# bit.
dbg_sel_cqn[CQ_BITS – 1:0]	clk	Input	This is the selected CQ# bit to modify.
dbg_inc_q	clk	Input	This signal increments the select data Q bit.
dbg_dec_q	clk	Input	This signal decrements the select data Q bit.
dbg_sel_q[Q_BITS – 1:0]	clk	Input	This is the selected Q bit to modify.
dbg_pd_off	clk	Input	This input should be driven High to disable the read phase detector.
dbg_cq_tapcnt[TAP_BITS × NUM_DEVICES – 1:0]	clk	Output	This is the current CQ tap setting for each device.
dbg_cqn_tapcnt[TAP_BITS × NUM_DEVICES – 1:0]	clk	Output	This is the current CQ# tap setting for each device.

Table 2-18: **Core Debug Signals *(Cont'd)***

Signal	Valid Clock Domain	Direction	Description
dbg_q_tapcnt[TAP_BITS × NUM_DEVICES – 1:0]	clk	Output	This is the current Q tap setting for each device.
dbg_clk_rd[NUM_DEVICES – 1:0]	clk_rd	Output	This is the aligned read clock.
dbg_rd_stage1_cal[255:0]	clk	Output	These are the debug signals for stage 1 calibration. See Table 2-19 for a signal map.

Read Stage 1 Calibration Debug

Table 2-19 indicates which bits within the dbg_rd_stage1_cal bus map to which debug signals in the PHY. These signals can all be found within the `phy_read_state1_cal` module and are all valid in the clk domain.

Table 2-19: **Read Stage 1 Debug Signal Map**

Bits	PHY Signal Name	Description
dbg_rd_stage1_cal[4:0]	tap_ctr_cs	This is the current state of the tap-centering state machine.
dbg_rd_stage1_cal[18:5]	cal_cs	This is the current state of the calibration stage 1 state machine.
dbg_rd_stage1_cal[23:19]	cq_dly_tap	This is the current tap setting for the active CQ clock.
dbg_rd_stage1_cal[28:24]	cqn_dly_tap	These are the current tap settings for the active CQ# clock.
dbg_rd_stage1_cal[33:29]	q_dly_tap	These are the current tap settings for the active Q data.
dbg_rd_stage1_cal[34]	window_vld	This bit indicates if the data being captured is valid for the target rising or falling window.
dbg_rd_stage1_cal[35]	opp_window_vld	This bit indicates if the data being captured is valid for the opposite of the targeted rising or falling window.
dbg_rd_stage1_cal[36]	data_rdy	This signal indicates when all tap adjustments are in place.
dbg_rd_stage1_cal[37]	en_tap_adj	This bit enables a tap adjustment.
dbg_rd_stage1_cal[38]	found_left0	This bit indicates that the left0 edge is found. The left0 edge is the left edge of the data window and is found by delaying the clock.
dbg_rd_stage1_cal[39]	found_left1	This bit indicates that the left1 edge is found. The left1 edge is the left edge of the data window and is found by delaying the data.
dbg_rd_stage1_cal[40]	found_right	This bit indicates that the right edge of the data window is found by delaying the clock.
dbg_rd_stage1_cal[41]	try_clk_inv	This bit indicates that no definite valid tap setting was found, and clock inversion is either required or will be tried for better results.
dbg_rd_stage1_cal[42]	tap_offset	This is the distance from the true middle for the selected setting.

Table 2-19: **Read Stage 1 Debug Signal Map *(Cont'd)***

Bits	PHY Signal Name	Description
dbg_rd_stage1_cal[43]	optimal_tap	This bit indicates the optimal tap setting of the current clock or data bit being calibrated.
dbg_rd_stage1_cal[44]	cdt_selected	This bit indicates if delaying the clock achieves the best result. If not asserted, it indicates that delaying the data provides a better result.
dbg_rd_stage1_cal[45]	ctr_done	This bit indicates that the tap centering state machine is complete and has found the tap settings for the data edges.
dbg_rd_stage1_cal[46]	q_mem_0	This bit indicates if the first bit in the memory component is being calibrated.
dbg_rd_stage1_cal[47]	re_captured	This bit indicates when the rising edge has been captured.
dbg_rd_stage1_cal[48]	fe_captured	This bit indicates when the falling edge has been captured.
dbg_rd_stage1_cal[49]	det_opt_done	This bit indicates when the optimal Q0 settings have been determined.
dbg_rd_stage1_cal[50]	det_ovr_done	This bit indicates when the overall Q0 settings have been determined.
dbg_rd_stage1_cal[51]	qbit_det_done	This bit indicates when the current Q tap settings have been determined.
dbg_rd_stage1_cal[52]	qbit_set_done	This bit indicates when the Q bit tap setting no longer needs adjustments.
dbg_rd_stage1_cal[57:53]	rei_optimal_tap	These are the optimal tap settings from the tap centering state machine for the rising edge data.
dbg_rd_stage1_cal[62:58]	fe_optimal_tap	These are the optimal tap settings from the tap centering state machine for the falling edge data.
dbg_rd_stage1_cal[63]	rei_cdt_selected	This bit indicates if the optimal tap setting for the rising edge data required clock delay for centering. If it is not asserted, data delay was required to center align.
dbg_rd_stage1_cal[64]	fe_cdt_selected	This bit indicates if the optimal tap setting for the falling edge data required clock delay for centering. If it is not asserted, data delay was required to center align.
dbg_rd_stage1_cal[65]	q_bit_clkinv	This is the ISERDES clk/clkb polarity inversion control.
dbg_rd_stage1_cal[66]	polarity_done	This bit indicates when the polarity change is done and fully propagated to the clk_rd domain.
dbg_rd_stage1_cal[67]	cal_rise	This bit indicates if the rising edge is being calibrated against. If not asserted, falling edge data is being calibrated.
dbg_rd_stage1_cal[72:68]	left0_tap	This is the tap setting for the left edge of the data window found by delaying the clock.
dbg_rd_stage1_cal[77:73]	left1_tap	This is the tap setting for the left edge of the data window found by delaying the data.
dbg_rd_stage1_cal[82:78]	right_tap	This is the tap setting for the right edge of the data window found by delaying the clock.

Table 2-19: **Read Stage 1 Debug Signal Map *(Cont'd)***

Bits	PHY Signal Name	Description
dbg_rd_stage1_cal[87:83]	re_optimal_tap	These are the optimal tap settings from the tap centering state machine for the rising edge data.
dbg_rd_stage1_cal[88]	re_cdt_selected	This bit indicates if the optimal tap setting for the rising edge data required clock delay for centering. If it is not asserted, data delay was required to center align.
dbg_rd_stage1_cal[89]	invert_clk	This bit indicates if the ISERDES clk/clkb inputs were inverted.
dbg_rd_stage1_cal[90]	rise_cdt_delayed	This bit indicates that the rising edge data (with or without clock inversion) was center aligned by delaying the clock.
dbg_rd_stage1_cal[95:91]	cq_tap	These are the CQ tap settings to be loaded in the IODELAY for the target CQ.
dbg_rd_stage1_cal[100:96]	cqn_tap	These are the CQ# tap settings to be loaded in the IODELAY for the target CQ#.
dbg_rd_stage1_cal[105:101]	q_tap	These are the Q tap settings to be loaded in the IODELAY for the target Q.
dbg_rd_stage1_cal[106]	capture_adj	This bit indicates that adjustment values should be loaded for the tap settings.
dbg_rd_stage1_cal[107]	load_init	This bit indicates that the previous tap settings should be loaded.
dbg_rd_stage1_cal[108]	prev_adj_req	This bit indicates that adjustments to the tap settings for the previous Q bits are needed.
dbg_rd_stage1_cal[113:109]	prev_q_adj	These bits indicate by how much the previous Q bits must be adjusted overall.
dbg_rd_stage1_cal[118:114]	q_adj_val	These bits indicate how many adjustments remain for the Q bit currently being adjusted.
dbg_rd_stage1_cal[119]	q_bit_adj_done	This bit indicates that adjustments to the previous Q bits have been completed.
dbg_rd_stage1_cal[120]	phase0_data_vld	This bit indicates that a phase setting of 0 in the `phy_read_data_align` module provides valid data for the target device.
dbg_rd_stage1_cal[121]	phase1_data_vld	This bit indicates that a phase setting of 1 in the `phy_read_data_align` module provides valid data for the target device.
dbg_rd_stage1_cal[122]	phase_error	This bit indicates that the target CQ# indicator is stable and control can be issued for the IODELAY.
dbg_rd_stage1_cal[123]	c_num_rdy	This bit indicates that the target CQ indicator is stable and control can be issued for the IODELAY.
dbg_rd_stage1_cal[124]	q_bit_rdy	This bit indicates that the target Q indicator is stable and control can be issued for the IODELAY.
dbg_rd_stage1_cal[125]	cq_num_ce	This is the IODELAY clock enable for the target CQ.
dbg_rd_stage1_cal[126]	cqn_num_ce	This is the IODELAY clock enable for the target CQ#.

Table 2-19: **Read Stage 1 Debug Signal Map *(Cont'd)***

Bits	PHY Signal Name	Description
dbg_rd_stage1_cal[127]	q_bit_ce	This is the IODELAY clock enable for the target Q.
dbg_rd_stage1_cal[128]	tap_done	This bit indicates that any IODELAYs that had taps incremented or decremented from the calibration logic have had adequate time to take effect.
dbg_rd_stage1_cal[129]	load_done	This bit indicates that any IODELAYs that were loaded with new tap settings issued by the calibration logic have had adequate time to take effect.
dbg_rd_stage1_cal[134:130]	cq_num_load	These bits indicate the CQ IODELAY load value for the target cq_num.
dbg_rd_stage1_cal[135]	cq_num_rst	This bit indicates that the target cq_num CQ IODELAY should be reset.
dbg_rd_stage1_cal[140:136]	cqn_num_load	These bits indicate the CQ# IODELAY load value for the target cq_num.
dbg_rd_stage1_cal[141]	cqn_num_rst	This bit indicates that the target cq_num CQ# IODELAY should be reset.
dbg_rd_stage1_cal[145:141]	q_bit_load	These bits indicate the Q IODELAY load value for the target q bit.
dbg_rd_stage1_cal[146]	q_bit_rst	This bit indicates that the target q-bit Q IODELAY should be reset.
dbg_rd_stage1_cal[147]	rst_done	This bit indicates that any IODELAY resets that were issued by the calibration logic have had adequate time to take effect.
dbg_rd_stage1_cal[152:148]	window_size	These bits indicate the size of the data window in number of taps.
dbg_rd_stage1_cal[255:148]	–	These bit are unused.

Within the read path are some additional signals that can be used for more extensive debug. These signals can be accessed from the `phy_read_top` module and are listed in Table 2-20.

Table 2-20: **Additional Read Path Debug Signals**

Signal	Module	Valid Clock Domain	Description
dbg_valid_lat[4:0]	phy_read_vld_gen	clk	This is the latency in cycles of the delayed read command.
dbg_cq_num[CQ_BITS – 1:0]	phy_read_stage1_cal	clk	This signal indicates the current CQ/CQ# being calibrated.
dbg_q_bit[Q_BITS – 1:0]	phy_read_stage1_cal	clk	This signal indicates the current Q being calibrated.

Table 2-20: **Additional Read Path Debug Signals *(Cont'd)***

Signal	Module	Valid Clock Domain	Description
dbg_error_max_latency[NUM_DEVICES – 1:0]	phy_read_stage2_cal	clk	This signal indicates that the latency could not be measured before the counter overflowed. There is one error bit for each device.
dbg_error_adj_latency	phy_read_stage2_cal	clk	This signal indicates that the target PHY_LATENCY could not be achieved.
dbg_stage2_cal[127:0]	phy_read_stage2_cal	clk	This signal is unused, but phy_read_stage2_cal signals can be added if debug is required.
dbg_phase[NUM_DEVICES – 1:0]	phy_read_data_align	clk_rd	This signal indicates whether or not to realign the data to correct the CLK/CLKB relationship relative to the CLKDIV in the ISERDES. There is one dbg_phase bit per device.
dbg_inc_latency[NUM_DEVICES – 1:0]	phy_read_dcb	clk	This signal indicates that latency through the DCB should be increased. There is one increment latency signal for each device, and they are all concatenated together.
dbg_dcb_wr_ptr[NUM_DEVICES × 5 – 1:0]	phy_read_dcb	clk_rd	This is the write pointer into the data block RAM. The pointer for each device is four bits long and concatenated together.
dbg_dcb_rd_ptr[NUM_DEVICES × 5 – 1:0]	phy_read_dcb	clk	This is the read pointer into the data block RAM. The pointer for each device is four bits long and concatenated together.
dbg_dcb_din[NUM_DEVICES × MEMORY_WIDTH × 4 – 1:0]	phy_read_dcb	clk_rd	This is the data input into the data circular buffer (DCB). The data for the DCB in each device is MEMORY_WIDTH × 4 and each is concatenated together.
dbg_dcb_dout[NUM_DEVICES × MEMORY_WIDTH × 4 – 1:0]	phy_read_dcb	clk	This is the data output from the DCB. The data for the DCB in each device is MEMORY_WIDTH × 4 and each is concatenated together.

When checking the results of read calibration check the tap values for the capture clock (cq_dly_tap and cqn_dly_tap) and the tap settings for each data bit (q_dly_tap). For a single clock group, the IODELAY tap settings should not vary widely. Figure 2-55 shows stage 1 read calibration running for Q bit 0, as well as some signals of interest.

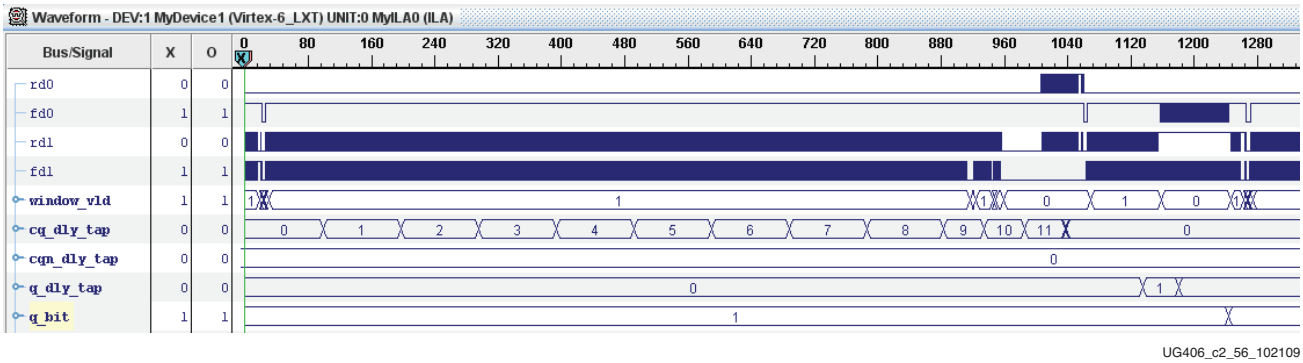


Figure 2-55: **ChipScope Analyzer Capture of Stage 1 Read Calibration**

Manual control signals are provided in the debug port for adjusting IODELAY tap values during normal operation. These can be adjusted to check for issues or to measure the data valid window timing using the VIO port of the ChipScope analyzer that is provided along with the Debug port.

Board Measurements

The signal integrity of the board and bus timing must be analyzed. The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 12] describes expected bus signal integrity. While this guide is specific to Virtex-5 devices and the ML561 development board, the principles therein can be applied to Virtex-6 FPGA MIG tool designs.

Other important board measurements are the reference voltage levels. It is important that these voltage levels are measured when the bus is active. These levels can be correct when the bus is idle, but might drop when the bus is active.

XILINX®

Chapter 3

RLDRAM II Memory Interface Solution

Introduction

The RLDRAM II memory interface solution is a physical layer for interfacing Virtex®-6 FPGA user designs to RLDRAM II devices. RLDRAM II memory can transfer up to two, four, or eight words of data per request and is commonly used in applications such as look-up tables (LUTs), L3 cache, and graphics.

The RLDRAM II memory solutions core is composed of a user interface (UI), memory controller (MC) and physical layer (PHY) that takes simple user commands and converts them to the RLDRAM II protocol before sending them to the memory. Unique capabilities of the Virtex-6 family allow the PHY to maximize performance and simplify read data capture within the FPGA. The full solution is complete with a synthesizable reference design.

This chapter describes the core architecture and information about using, customizing, and simulating a LogiCORE™ IP RLDRAM II memory interface core for the Virtex-6 FPGA. Although this soft memory controller core is a fully verified solution with guaranteed performance, termination and trace routing rules for PCB design need to be followed to have the best design. For detailed board design guidelines, see Design Guidelines, page 299.

For detailed information and updates about the Virtex-6 FPGA RLDRAM II memory interface core, refer to the Virtex-6 FPGA data sheets [Ref 9] [Ref 13] on the Virtex-6 FPGA memory interface product page.

Getting Started

This section provides a step-by-step guide to using the CORE Generator™ tool to generate an RLDRAM II memory interface core in a Virtex-6 device, run the design through implementation with the Xilinx tools, and simulate the example design using the provided synthesizable test bench.

System Requirements

These are needed to implement the RLDRAM II memory interface core:

- Windows XP Professional (32-/64-bit)
- ISE® Design Suite, version 13.4

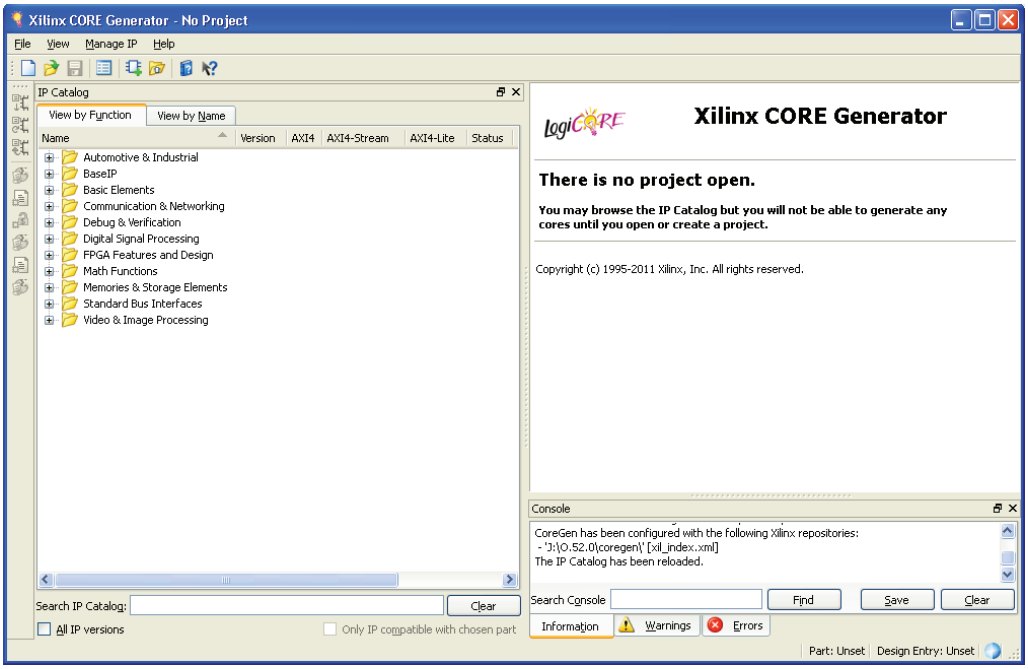
Customizing and Generating the Core

Generation through the Graphical User Interface

The Memory Interface Generator is a self-explanatory wizard tool that can be invoked under the CORE Generator software. This section is intended to help in understanding the various steps involved in using the MIG tool.

These steps should be followed to generate a Virtex-6 FPGA RLDRAM II design:

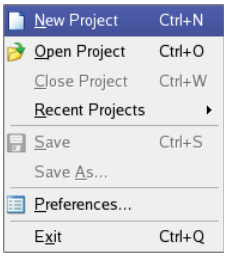
1. Launch the CORE Generator software by selecting **Start** → **Xilinx ISE Design Suite 13.4** → **ISE** → **Accessories** → **CORE Generator** (Figure 3-1).



UG406_c3_01_041411

Figure 3-1: **Xilinx CORE Generator Software**

2. Choose **File** → **New project** to open the New Project dialog box. Create a new project named *Virtex6_MIG_Example_Design* (Figure 3-2).



UG406_c3_02_081109

Figure 3-2: **New CORE Generator Software Project**

3. Enter a project name and location. Click **Save** (Figure 3-3).

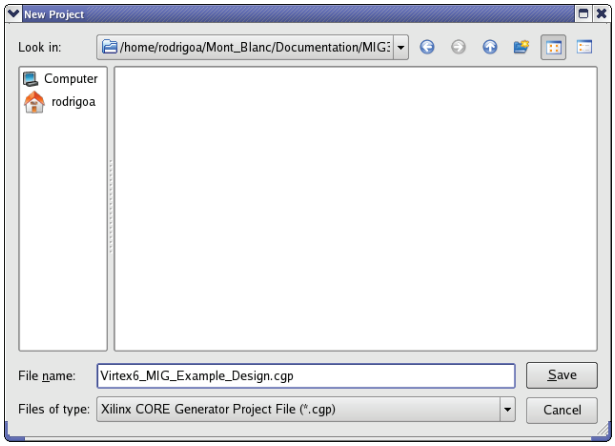


Figure 3-3: **New Project Menu**

4. Select these project options for the part (Figure 3-4):
   a. Family: **Virtex-6**
   b. Device: **xc6vlx240t**
   c. Package: **ff1156**
   d. Speed Grade: **-2**

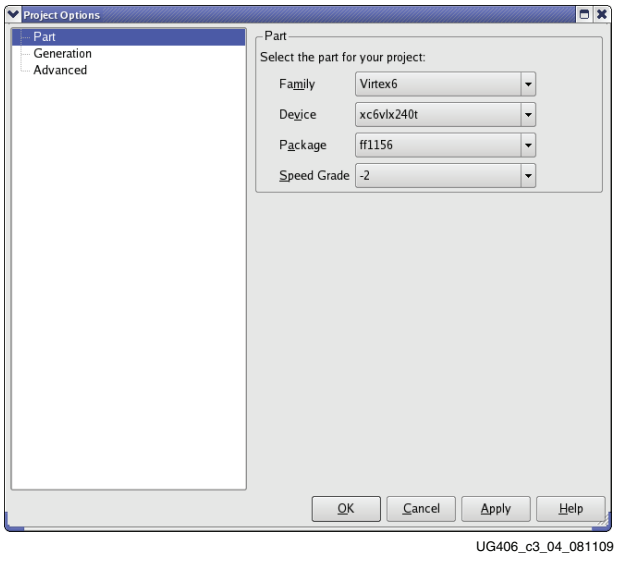
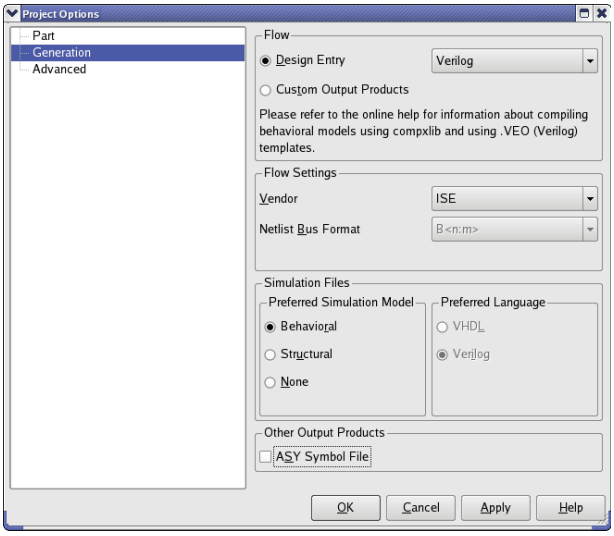


Figure 3-4: **CORE Generator Software Project Options**

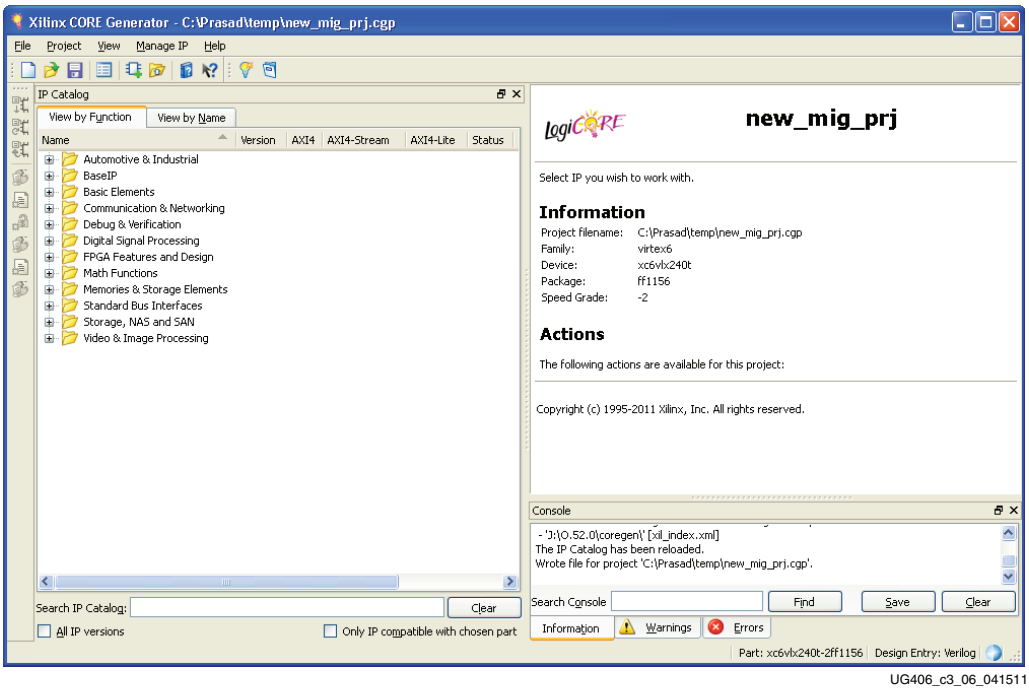
5. Select **Generation** from the menu on the left. From this menu, select **Verilog** or **VHDL** as the Design Entry and **ISE** for the Vendor Flow Setting. Click **OK** to finish the Project Options setup (Figure 3-5).



UG406_c3_05_081109

Figure 3-5: **CORE Generator Software Design Flow Settings**

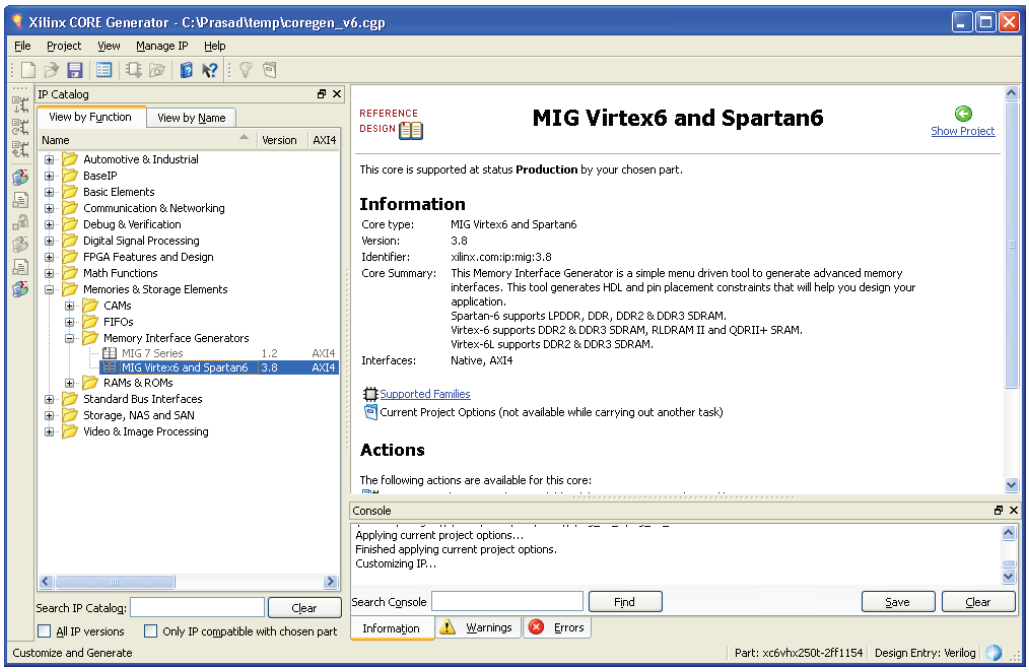
6. Select **Memory Interface Generator (MIG)** by expanding **Memories & Storage Elements** (Figure 3-6).



UG406_c3_06_041511

Figure 3-6: **Virtex-6_MIG_Example_Design Project Page**

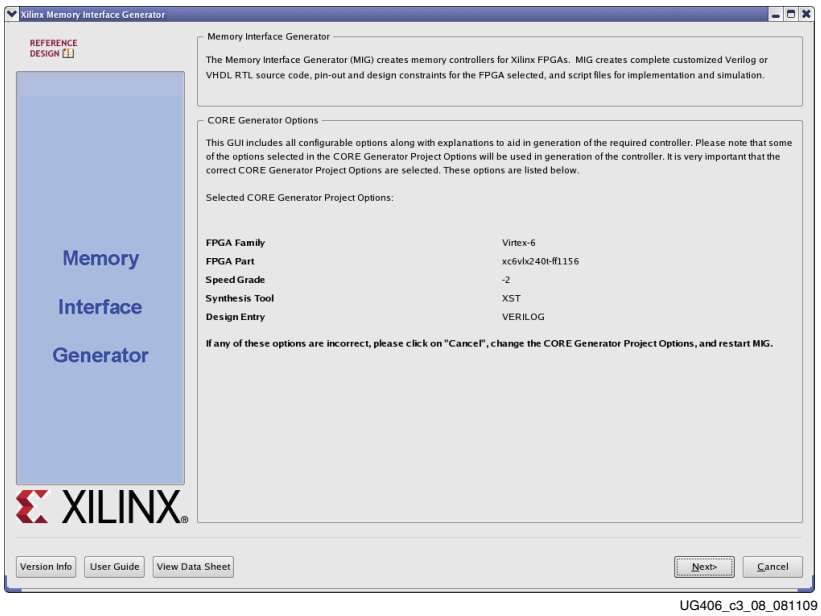
7. Launch the MIG tool wizard by selecting
Memories & Storage Elements → **Memory Interface Generators** → **MIG**
(Figure 3-7).



UG406_c3_07_041511

Figure 3-7: **Starting the MIG Tool Wizard**

8. The options screen in the CORE Generator software displays the details of the selected CORE Generator software options that are selected before invoking the MIG tool (Figure 3-8).



UG406_c3_08_081109

Figure 3-8: **Virtex-6 FPGA Memory Interface Generator Front Page**

9. Click **Next** to display the Output Options window.

MIG Tool Output Options

1. Select the **Create Design** radio button to create a new memory core. Enter a component name in the Component Name field (Figure 3-9).

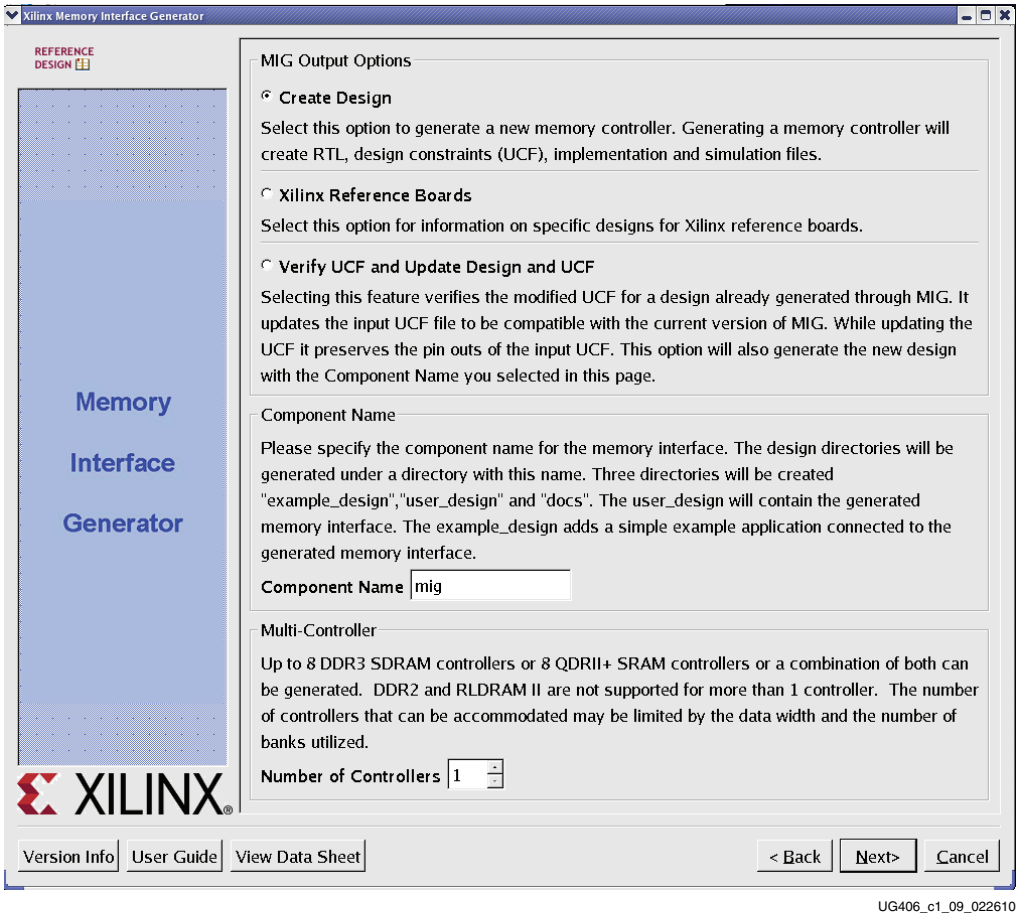


Figure 3-9: **MIG Tool Output Options**

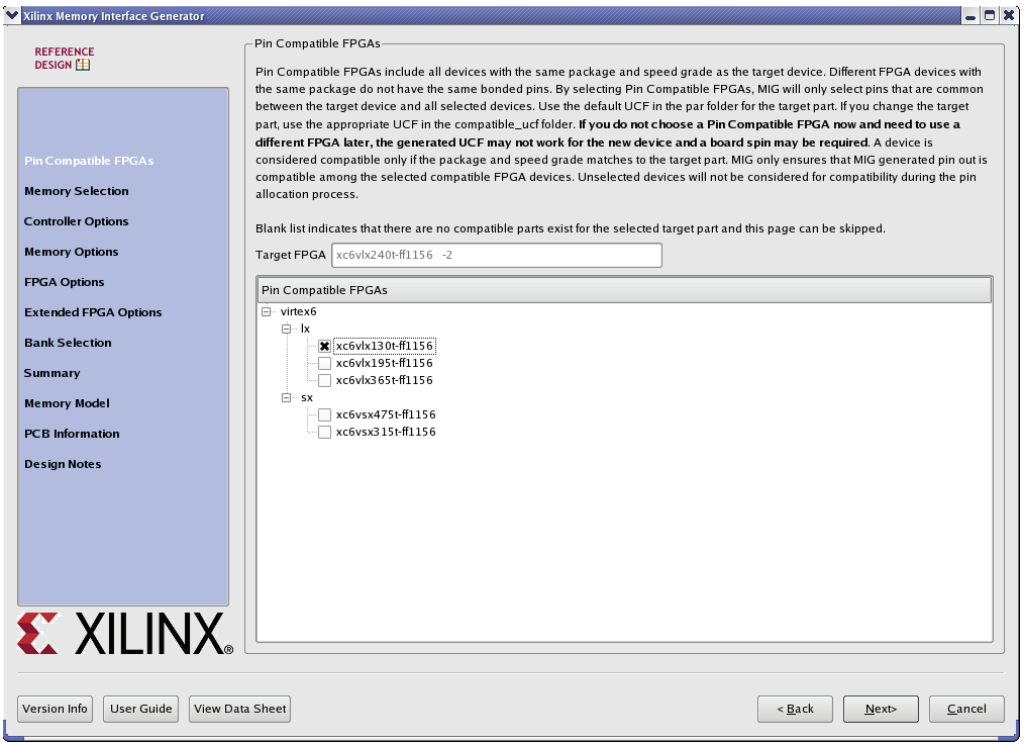
The MIG tool outputs are generated with the folder name `<component name>`.

Note: Only alphanumeric characters can be used for `<component name>`. Special characters cannot be used. This name should always start with an alphabetical character and can end with an alphanumeric character.

2. Click **Next** to display the Pin Compatible FPGAs window.

Pin Compatible FPGAs

The Pin Compatible FPGAs window lists FPGAs in the selected family having the same package. If the generated pinout from the MIG tool needs to be compatible with any of these other FPGAs, this option should be used to select the FPGAs with which the pinout has to be compatible (Figure 3-10).



UG406_C3_10_081109

Figure 3-10: **Pin-Compatible Virtex-6 FPGAs**

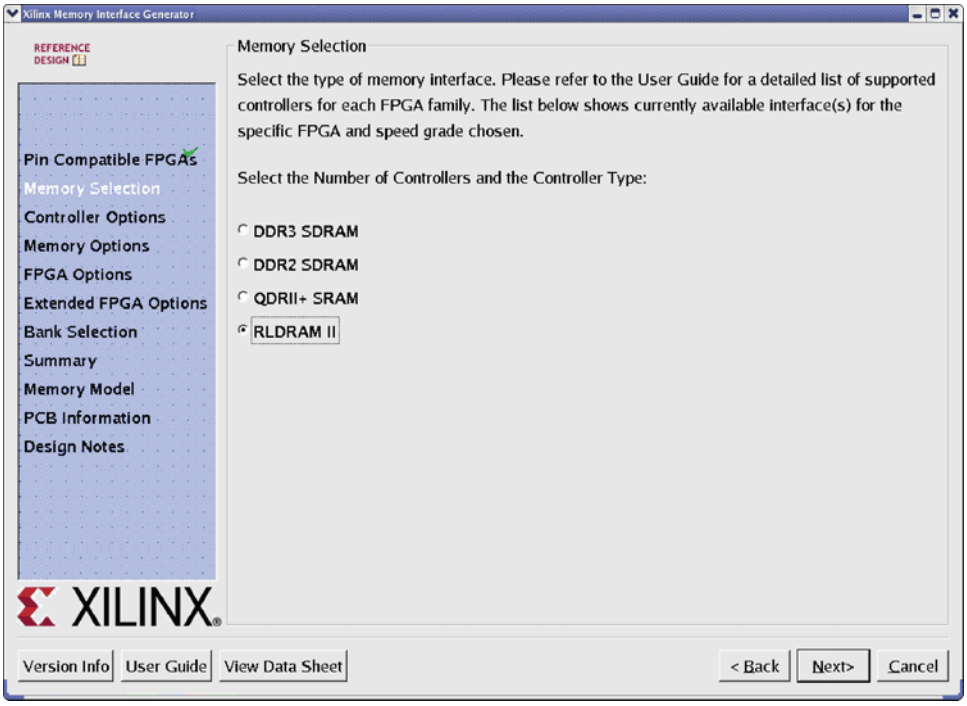
1. Select any of the compatible FPGAs in the list. Only the common pins between the target and selected FPGAs are used by the MIG tool. The name in the text box signifies the target FPGA selected.
2. Click **Next** to display the Memory Selection window.

Creating the Virtex-6 FPGA RLDRAM II Memory Design

Memory Selection

This page displays all memory types that are supported by the selected FPGA family.

1. Select the RLDRAM II controller type.
2. Click **Next** to display the Controller Options window (Figure 3-11).

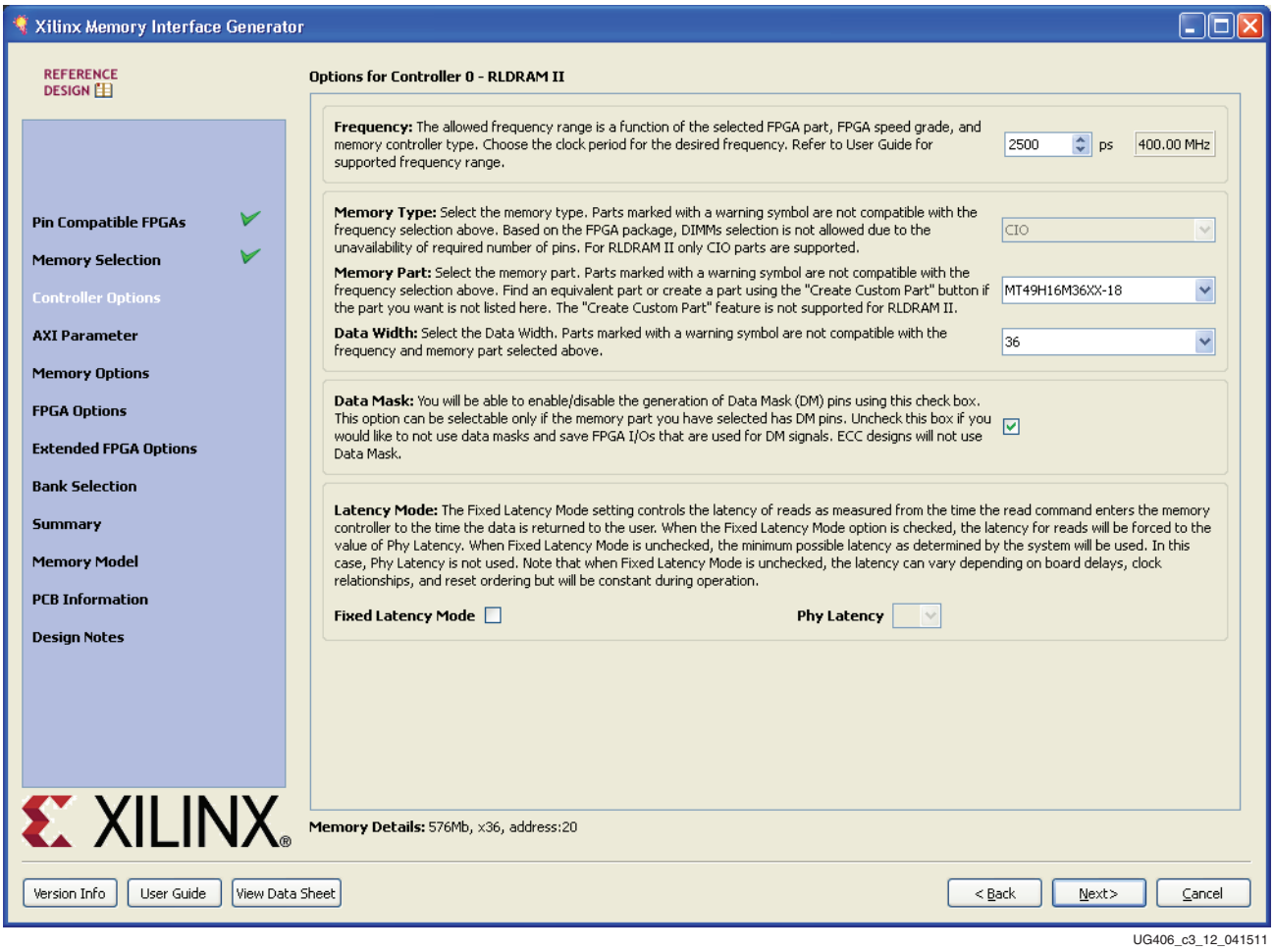


UG406_c3_11_072009

Figure 3-11: **Memory Type and Controller Selection**

Controller Options

This page shows the various controller options that can be selected (Figure 3-12).

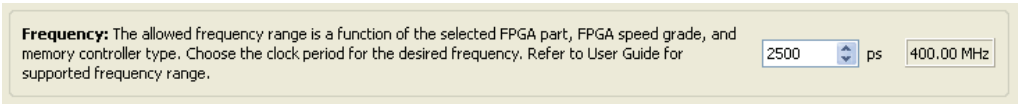


UG406_c3_12_041511

Figure 3-12: **Controller Options**

The controller options page also contains these pull-down menus to modify different features of the design:

- **Frequency**: This feature indicates the operating frequency for all the controllers (Figure 3-13). The frequency block is limited by factors such as the selected FPGA and device speed grade.



UG406_c3_13_041511

Figure 3-13: **Frequency Selection**

- **Memory Type**: This option selects the memory type for the design. Only common I/O (CIO) devices are supported. (Figure 3-14).

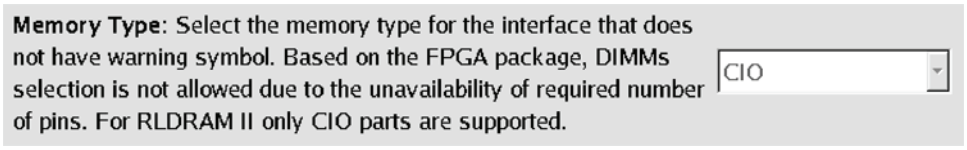


Figure 3-14: **Memory Type Selection**

- **Memory Part**: This option selects the memory part for the design. Selections can be made from the list, or if the part is not listed, a new part can be created (Figure 3-15).

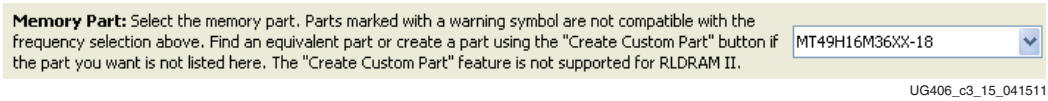


Figure 3-15: **Memory Part Selection**

- **Data Width**: The data width value can be selected here based on the memory part selected. The MIG tool supports values in multiples of the individual device data widths (Figure 3-16).

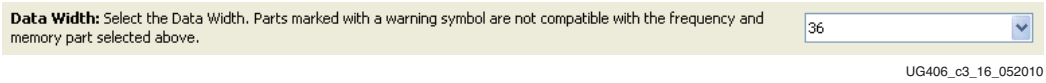


Figure 3-16: **Data Width Selection**

- **Data Mask**: This option allocates data mask pins when selected (Figure 3-17). This option should be deselected to deallocate data mask pins and increase pin efficiency. This option is disabled for memory parts that do not support data mask.

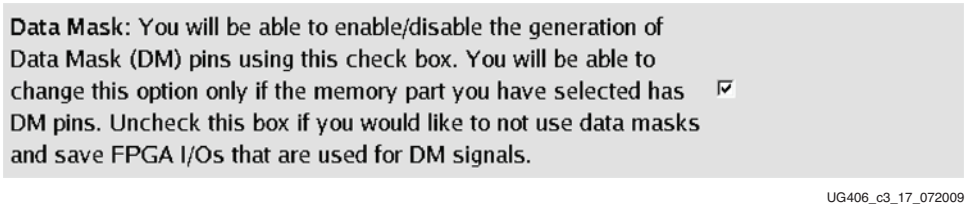


Figure 3-17: **Data Mask Selection**

- **Latency Mode**: If fixed latency through the core is needed, the Fixed Latency Mode option allows the user to select the desired latency. This option can be used if the user design needs a read response returned in a predictable number of clock cycles. To use this mode, select the **Fixed Latency Mode** box. After enabling fixed latency, the pull-down box allows the user to select the number of cycles until the read response is returned to the user. This value ranges from 19 to 30 cycles (Figure 3-18). If Fixed Latency Mode is not used, the core uses the minimum number of cycles through the system.

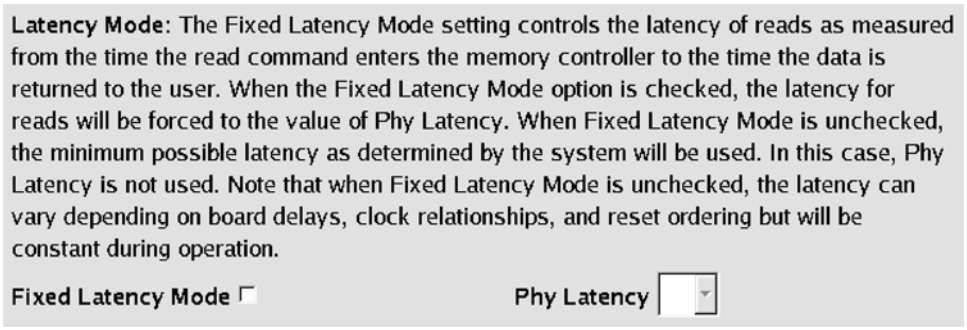
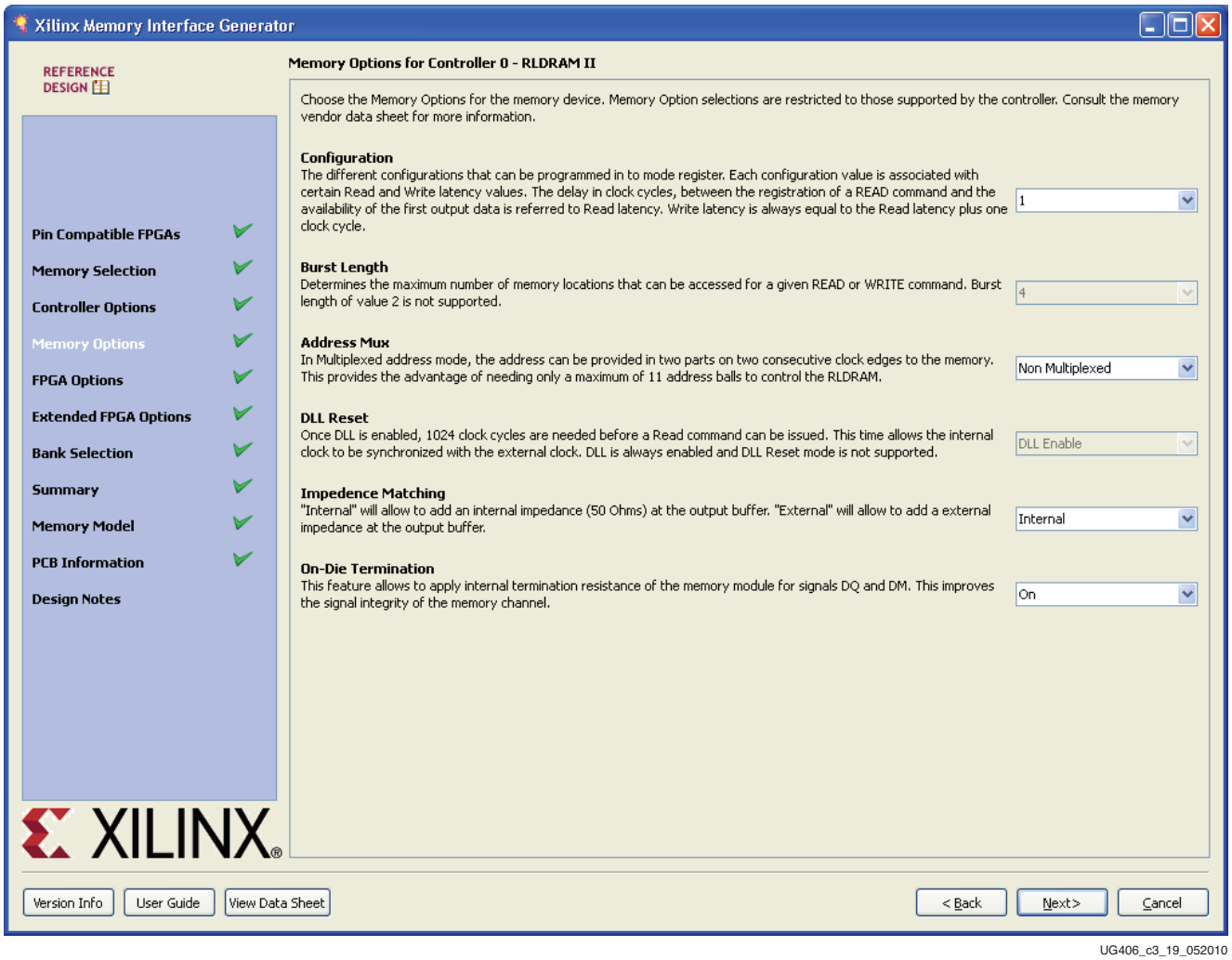


Figure 3-18: **Latency Selection**

Setting RLDRAM II Memory Parameter Option

This feature allows the selection of various memory mode register values, as supported by the controller's specification (Figure 3-19).

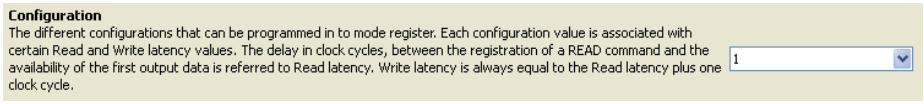


UG406_c3_19_052010

Figure 3-19: **Setting Memory Mode Options**

The mode register value is loaded into the load mode register during initialization.

- **Configuration**: This option sets the configuration value that is associated with write and read latency values. Values of 1, 2, and 3 are available that are controlled based on the selected design frequency.



UG406_c3_72_052010

Figure 3-20: **Configuration Selection**

- **Burst Length**: This option sets the length of a burst for a single memory transaction. This option is a trade-off between granularity and bandwidth and should be determined based on the application. Values of 4 and 8 are available (Figure 3-21).

Figure 3-21: **Burst Length Selection**

- **Address Multiplexing**: This option minimizes the number of address pins required for a design, because the address is provided using less pins but over two consecutive clock cycles. This option is not supported with a burst length of 2 (Figure 3-22).

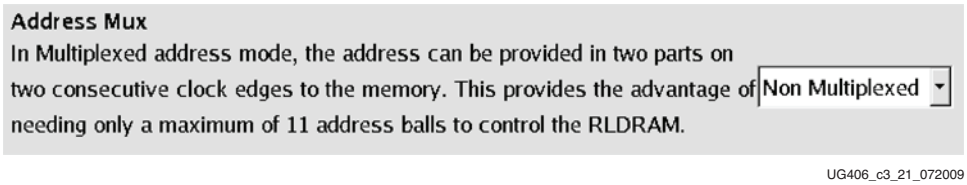


Figure 3-22: **Address Multiplexing Selection**

- **DLL Reset**: This option turns off the DLL inside the memory device allowing the memory to run at much lower frequencies. This option is not supported; thus, it is always set to DLL_ENABLE. (Figure 3-23).

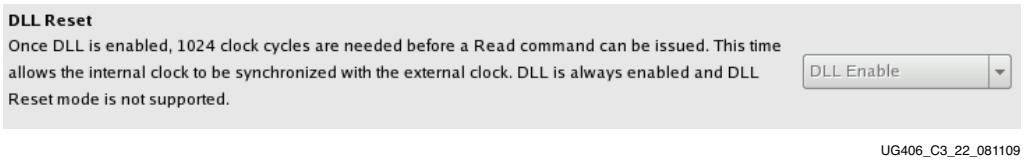


Figure 3-23: **DLL Reset Selection**

- **Impedance Matching**: This option is used to determine how the memory device tunes its outputs, either via an internal setting or using an external reference resistor connected to the ZQ input of the memory device (Figure 3-24).

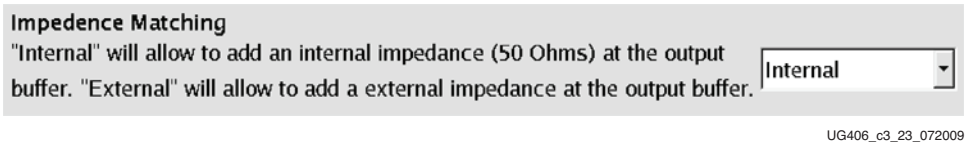


Figure 3-24: **Impedance Matching Selection**

On-Die Termination: This option is used to apply termination to the DQ and DM signals at the memory device during write operations. When set, the memory device dynamically switches off ODT when driving the bus during a read command (Figure 3-25).

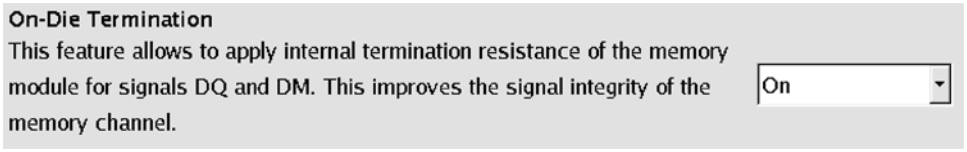


Figure 3-25: **On-Die Termination Selection**

Click **Next** to display the FPGA Options window (Figure 3-26).

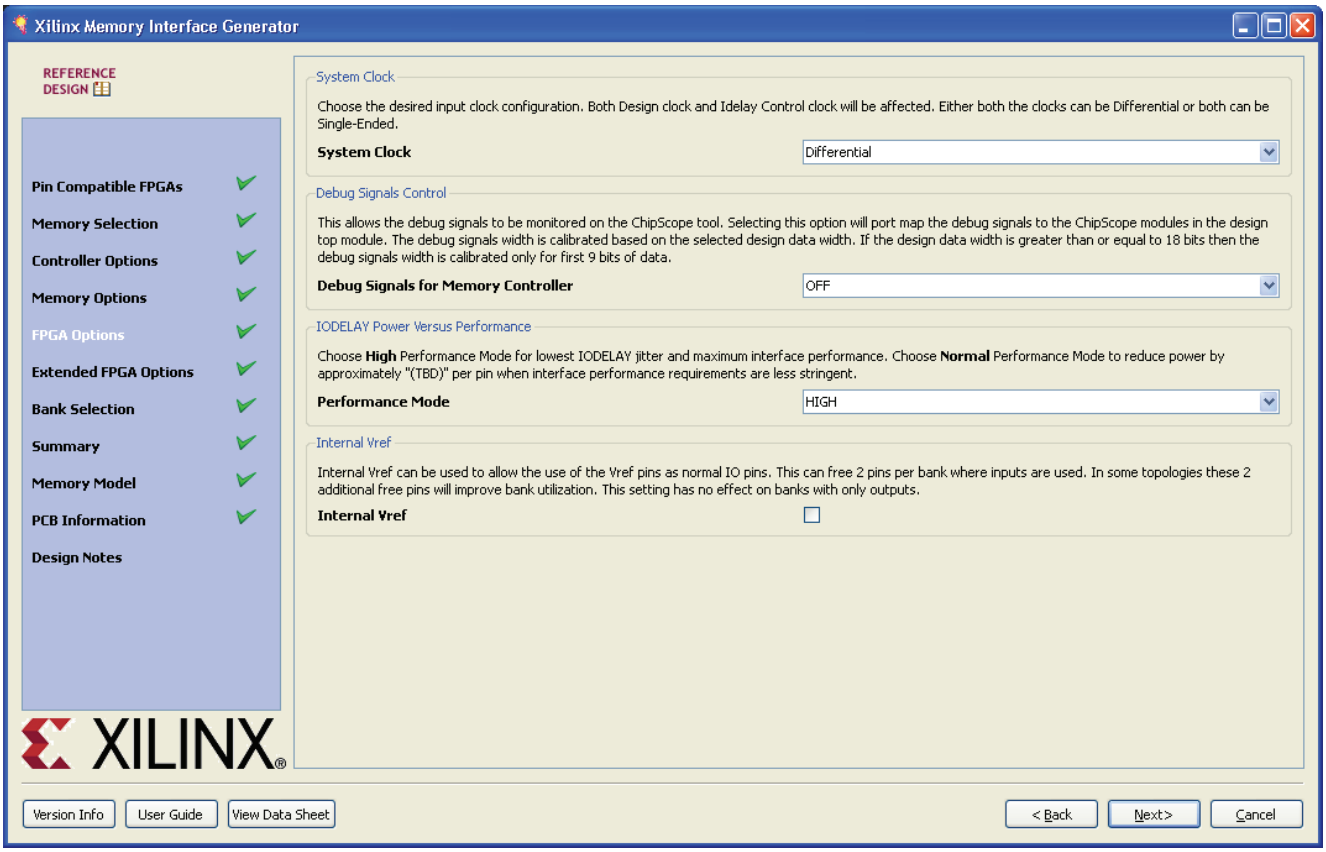


Figure 3-26: **FPGA Options**

- **System Clock**: The MIG tool supports the use of differential system and reference clocks or single-ended clocks. This should be set to the type of input clock that will be used (Figure 3-27).

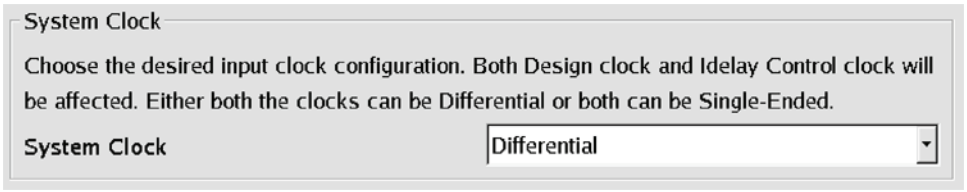


Figure 3-27: **System Clock Selection**

- **Debug Signals Control**: This option indicates whether the ChipScope™ analyzer should be incorporated into the generated design (Figure 3-28). See Debugging Virtex-6 FPGA RLDRAM II Memory Designs, page 301 for more details on the signals that are provided when this option is turned on.

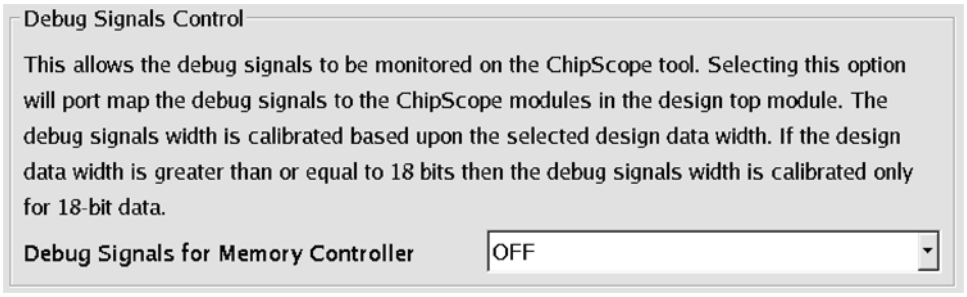


Figure 3-28: **Debug Enable Selection**

- Internal Vref Selection. Internal Vref can be used for data read banks to allow the use of the VREF pins for normal I/O usage.

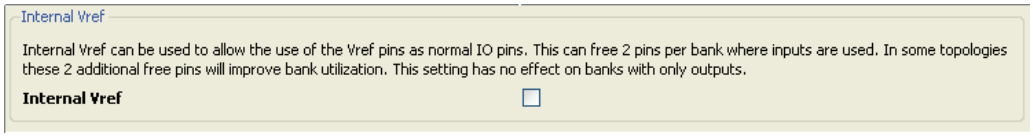


Figure 3-29: **Internal VREF Selection**

- **Digitally Controlled Impedance (DCI)**: When selected, this option internally terminates the signals from the RLDRAM II read path (Figure 3-30).
- **I/O Voltage Option**: The I/O voltage level for the memory interface can be set to either 1.8V or 1.5V (Figure 3-30). The memory device must have the appropriate voltage level provided on the PCB for correct functionality.

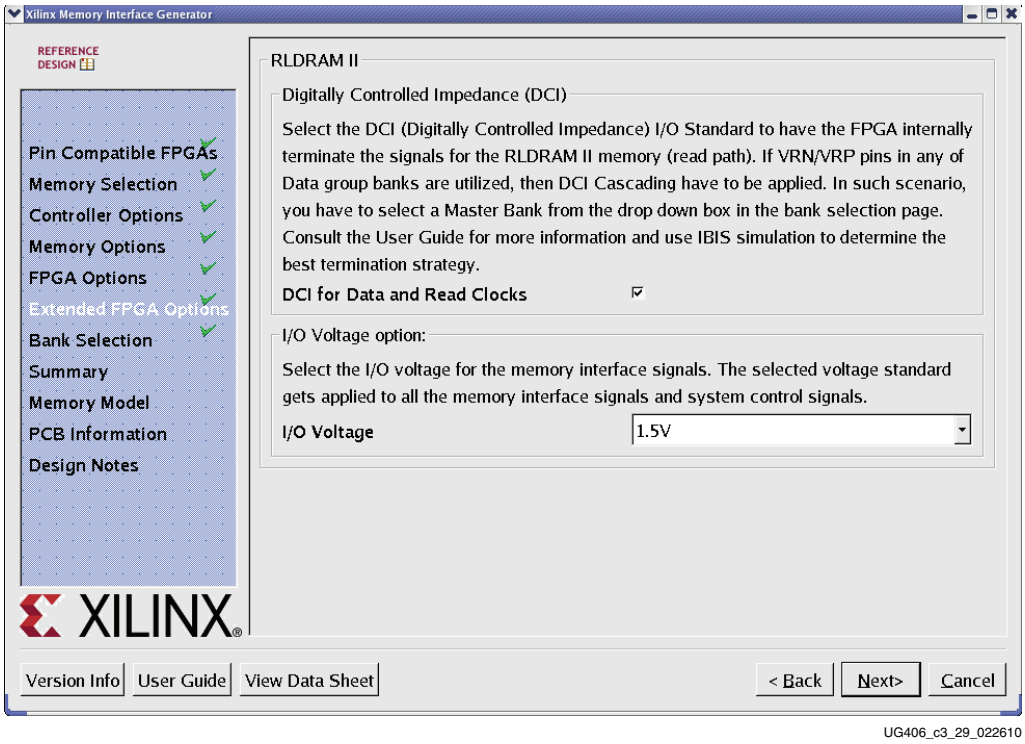
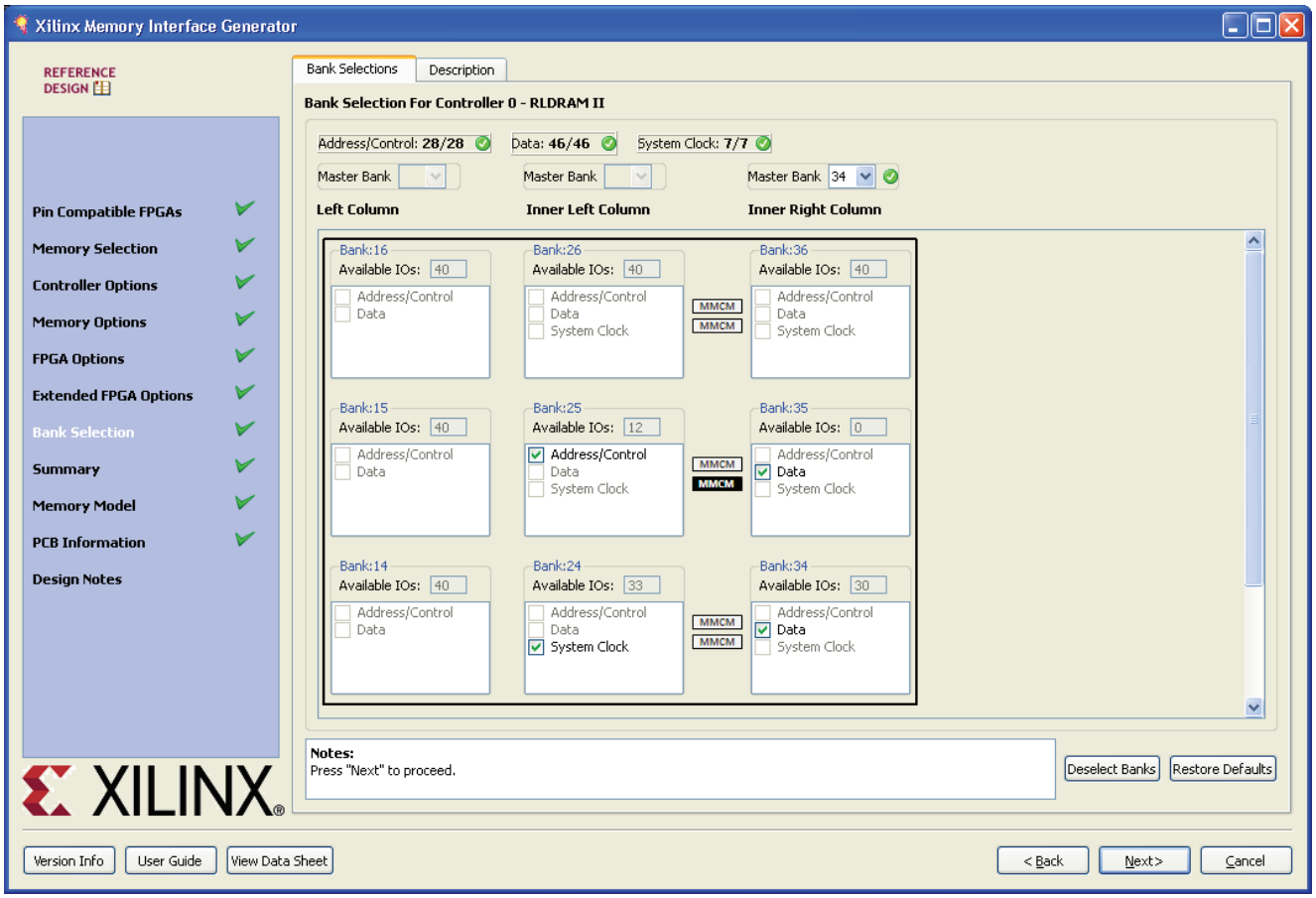


Figure 3-30: **DCI Selection and I/O Voltage Option**

Bank Selections

This page allows the selection of banks for the memory interface (Figure 3-31). Banks can be selected for different classes of memory signals, such as:

- Address and control signals
- Data signals
- System control signals
- System clocks



UG406_c3_30_052010

Figure 3-31: **Bank Selections Page**

For customized settings, click **Deselect Banks** and select the appropriate bank and memory signals. When selecting banks, read the Description tab for a list of bank selection and pin allocation rules that must be followed. Click **Next** to select the default settings and move to the next page.

- **Address Group Selection**: Select the address/control group from one of the white banks. Only the inner columns are allowed (Figure 3-32).

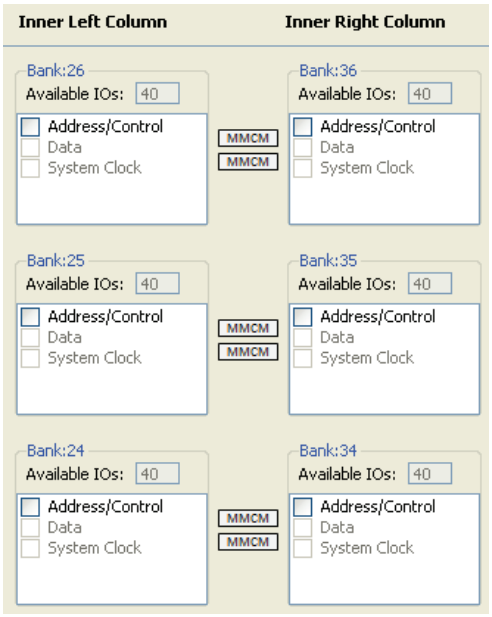


Figure 3-32: **Address Group Selection**

- **Data Selection**: After the address group is assigned, the data group must be selected. The data group is grayed out in banks that are invalid based on the selected address group while the available banks are outlined with a black box. To complete the data selection, enough banks must be selected to hold the data and data mask signals (Figure 3-33).

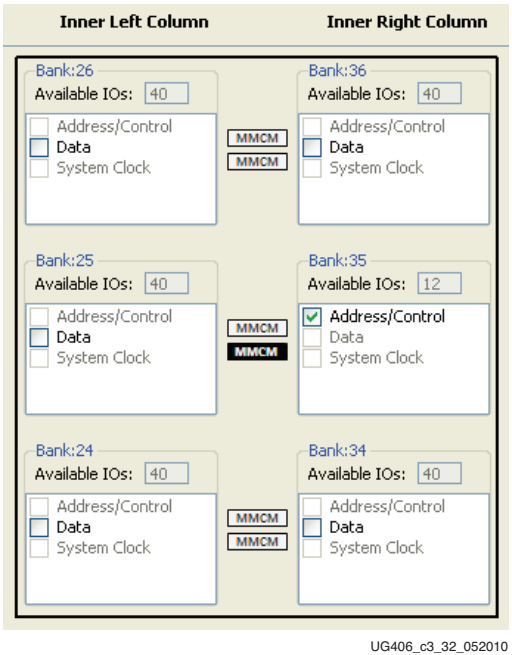


Figure 3-33: **Data Group Selection**

- **System Clocks Selection**: After the data group is assigned, the system clocks group must be selected. Select this group from any of the enabled banks (Figure 3-34).

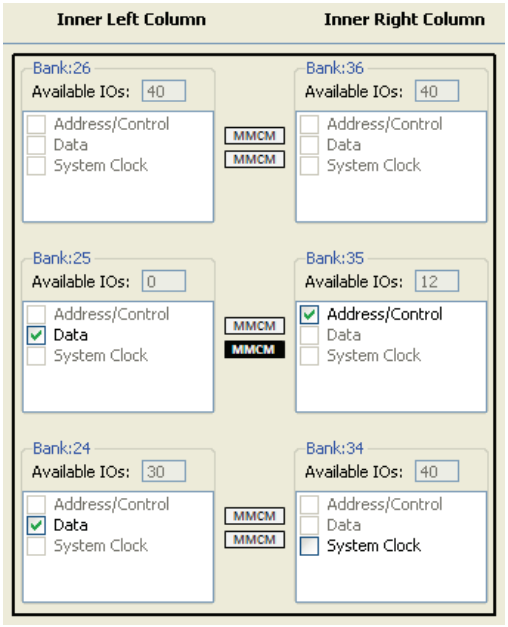


Figure 3-34: **System Clocks Group Selection**

- **Master Bank Selection**: Two extra pins are required to set up a DCI reference that provides better signal integrity. Select the master bank from the pull-down menu (Figure 3-35).

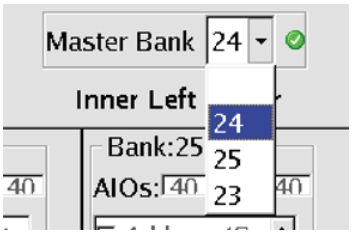
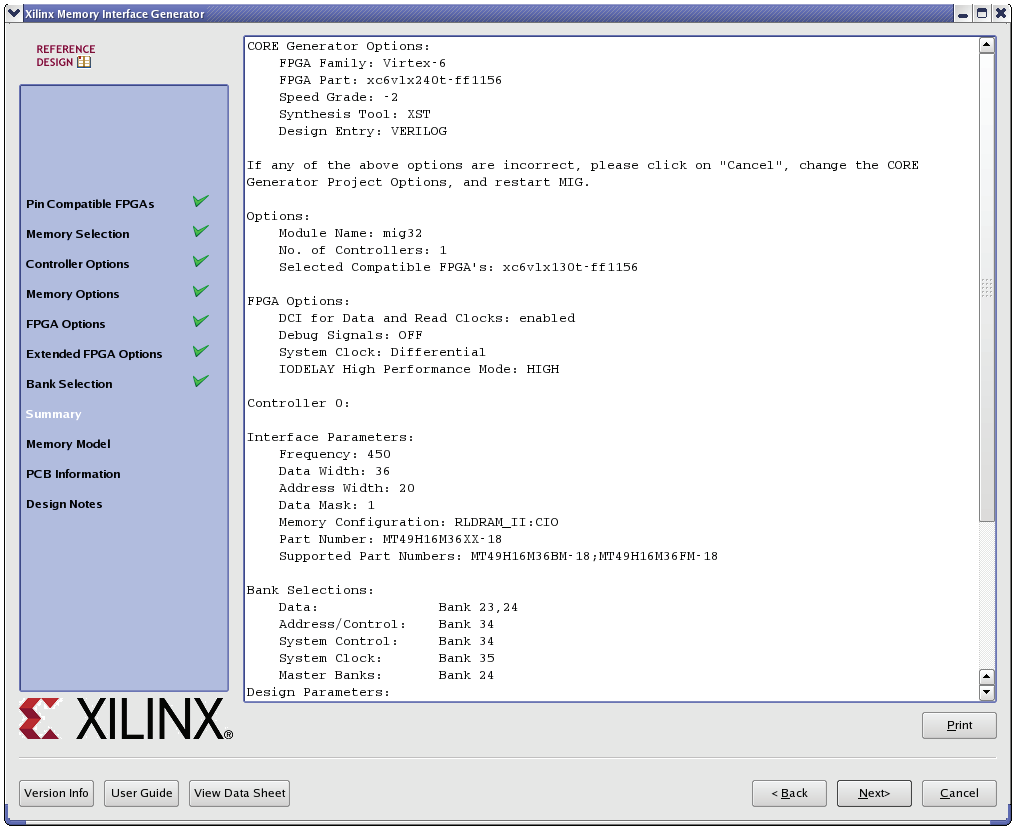


Figure 3-35: **Master Bank Selection**

After the banks have been assigned, click **Next** to view the summary.

Summary

The summary window provides the complete details about the Virtex-6 FPGA memory core selection, interface parameters, CORE Generator software options, and FPGA options of the active project (Figure 3-36).

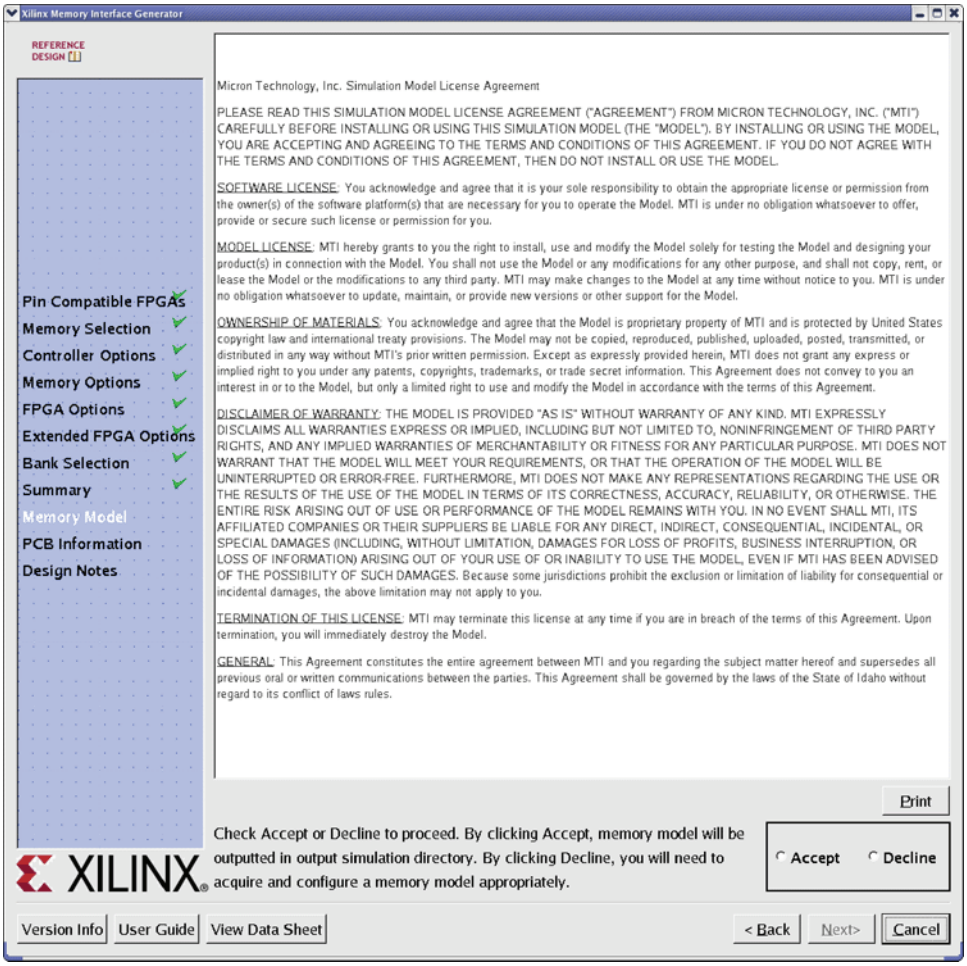


UG406_C3_36_081109

Figure 3-36: **Summary**

Memory Models

The MIG tool can output a chosen vendor's memory model for simulation purposes. To access the models in the output `sim` folder, click the license agreement (Figure 3-37). Read the license agreement and check the **Accept License Agreement** box to accept it. If the license agreement is not agreed to, the memory model is not made available. A memory model is necessary to simulate the design.



UG406_c3_37_072009

Figure 3-37: **Memory Models Page**

Click **Next** to move to the PCB Information page.

PCB Information

This page displays the PCB-related information to be considered while designing the board that uses the MIG tool generated designs (Figure 3-38).

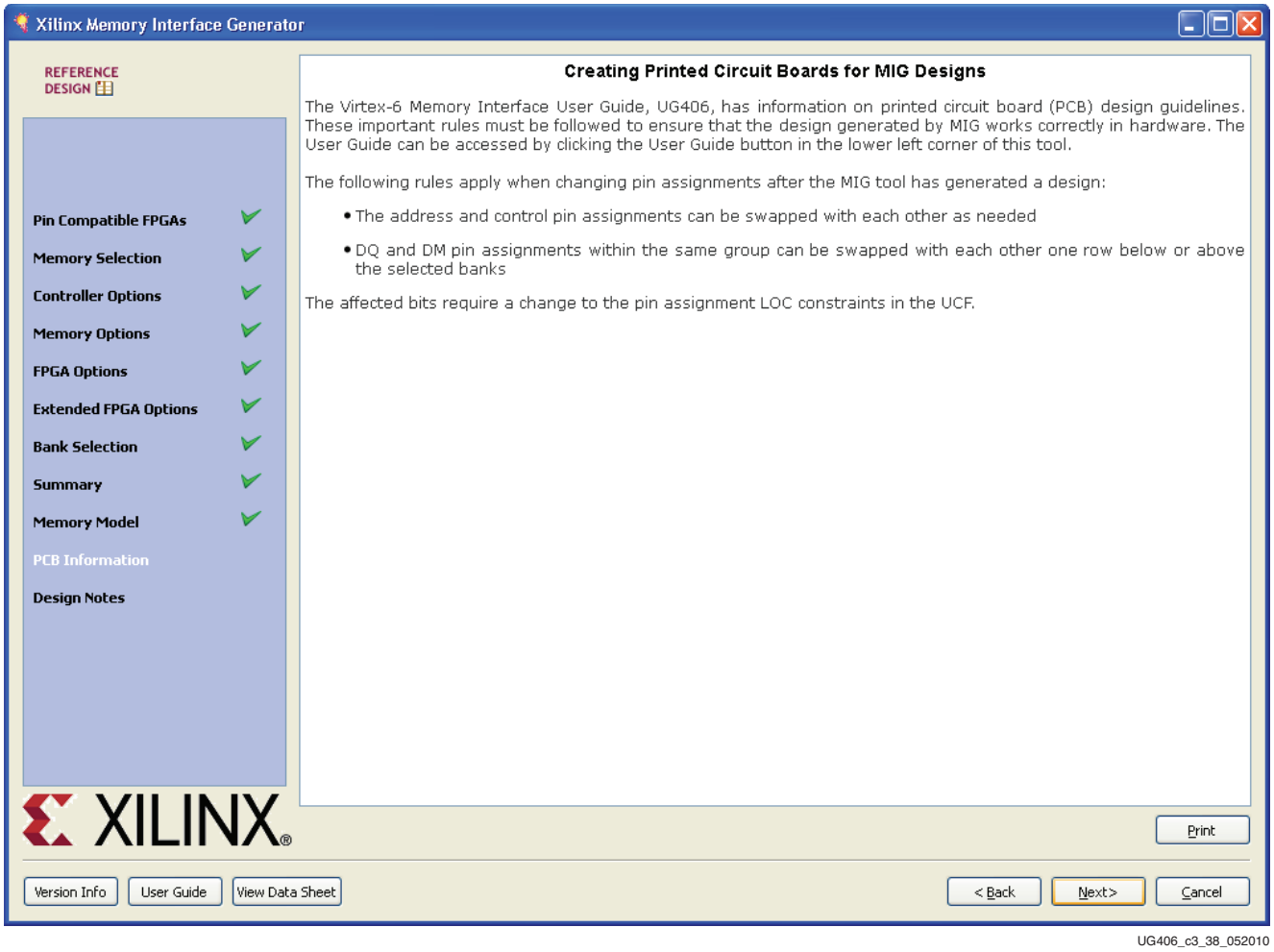


Figure 3-38: **PCB Information Page**

Click **Next** to move to the Design Notes Information page.

Design Notes

The MIG tool outputs some useful design notes that should be considered before proceeding (Figure 3-39).

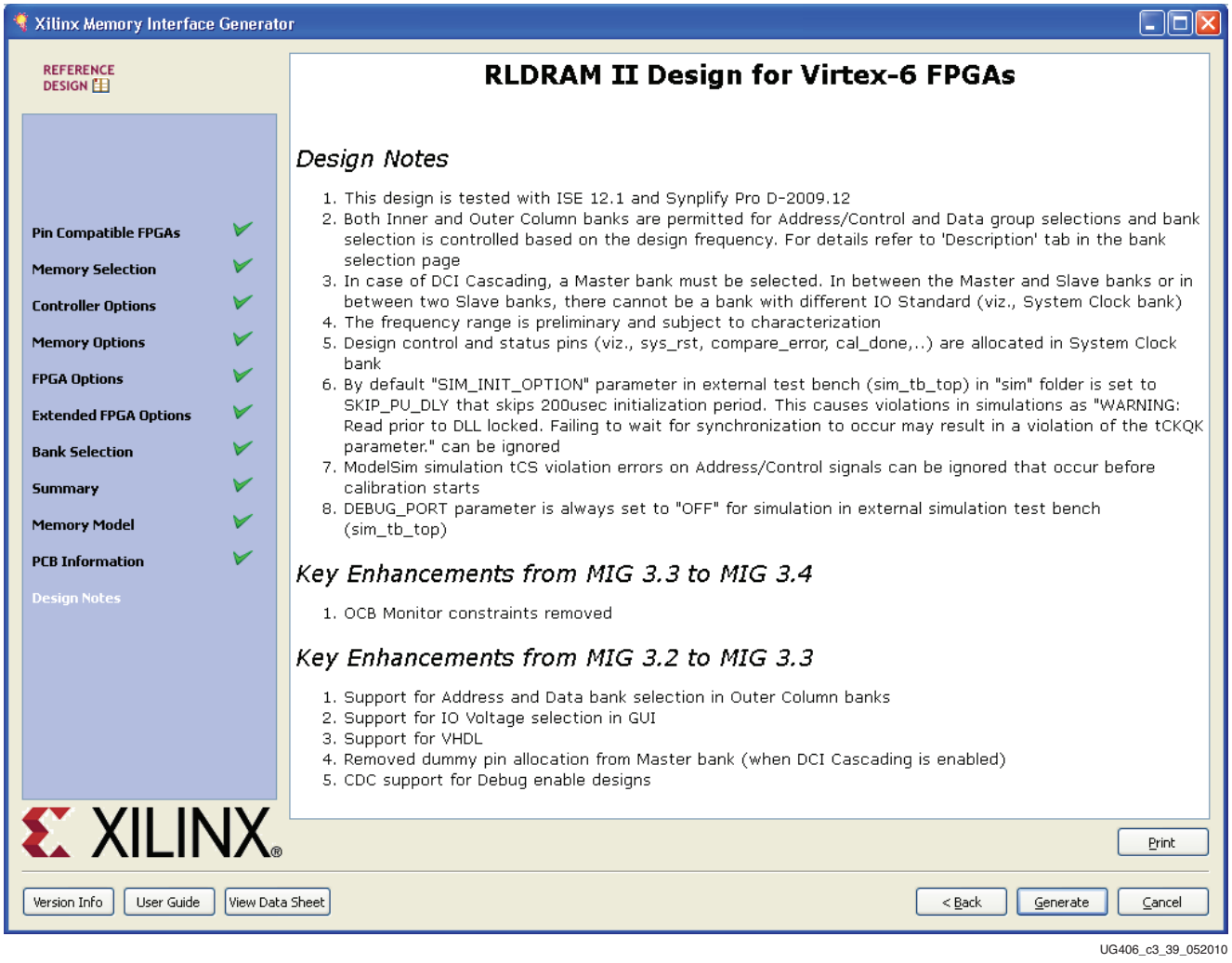


Figure 3-39: **Readme Page**

Click **Generate** to generate the design files. The MIG tool generates three output directories: `docs`, `example_design`, and `user_design`. See Directory Structure and File Descriptions, page 268 for more details on the contents of these directories.

After generating the design, a Readme page is displayed with the CORE Generator software output descriptions (Figure 3-40).

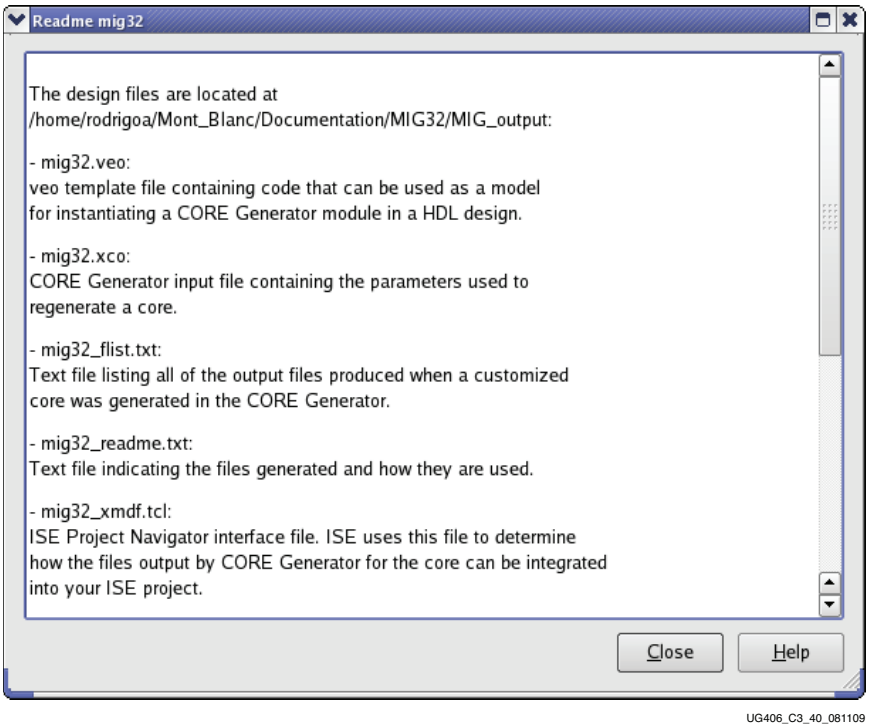


Figure 3-40: **Readme Page**

Click **Close** to return to the CORE Generator software.

Directory Structure and File Descriptions

Overview

Output Directory Structure

The MIG tool outputs are generated with the folder name `<component name>`.

Figure 3-41 shows the output directory structure of the selected design from the MIG tool. In the `<component name>` directory, three folders are created:

- docs
- example_design
- user_design

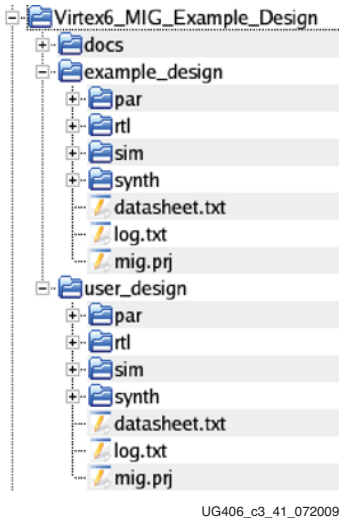


Figure 3-41: **MIG Tool Output Directory Structure**

The `example_design` and `user_design` directories contain most of the same files. However, they are provided separately for easy simulation and integration into the user's design. The example_design directory provides an example user application that sends traffic through the core. This example design is used for simulation and contains the complete synthesizable test bench. The `user_design` directory provides only those files needed to integrate the core into the user's logic and does not include the simulation or test bench files.

Directory and File Contents

The Virtex-6 device core directories and their associated files are listed in this section.

`<component name>/docs`

The `docs` folder contains the PDF documentation.

`<component name>/example_design`

The `example_design` folder contains four folders, namely, `par`, `rtl`, `sim`, and `synth`.

Table 3-1 to Table 3-8 list the contents of these directories along with file descriptions.

`<component name>/example_design/par`

Table 3-1 lists the files in the `example_design/par` directory.

Table 3-1: **Files in `example_design/par` Directory**

Name	Description
`bitgen_options.ut`	This file contains the bitgen options when running the design through implementation.
`create_ise.bat`	This is a batch file to generate an ISE project for implementing the design using the GUI.
`example_top.ucf`	This is the UCF generated from the bank selections.

Table 3-1: **Files in `example_design/par` Directory *(Cont'd)***

Name	Description
`ise_flow.bat`	This is a batch file for running the design through the implementation tools through the command-line interface.
`rem_files.bat`	This is a batch file used to remove implementation files when rerunning a design through the tools. This file is called by `ise_flow.bat` to ensure that previous implementation results are discarded.
`xst_options.txt`	This file contains the XST synthesis options when running the design through implementation.
`set_ise_prop.tcl`	This file is used by `create_ise.bat` to set the ISE project options.

`<component name>/example_design/rtl`

Table 3-2 lists the files in the `example_design/rtl` directory.

Table 3-2: **Files in `example_design/rtl` Directory**

Name	Description
`clk_ibuf.v/vhd`	This module instantiates the system clock input buffers.
`example_top.v/vhd`	This top-level module serves as an example for connecting the user design to the Virtex-6 FPGA memory interface core.
`iodelay_ctrl.v/vhd`	This module instantiates the IDELAYCTRL primitive needed for IODELAY use.
`phy_d_q_io.v/vhd`	This is the I/O module for the entire DQ bus for a single byte lane.
`phy_oserdes_io.v/vhd`	This is the I/O module for a single bit of data going to the memory.
`phy_read_clk_io.v/vhd`	This is the I/O module for the incoming CQ/CQ# echo clocks from the memory.
`phy_read_data_align.v/vhd`	This module realigns the incoming data.
`phy_read_dcb.v/vhd`	This module transfers the data from the clk_rd domain into the clk domain.
`phy_read_dly_ctrl.v/vhd`	This module drives the IODELAY control for each clock and data I/O based on the control from the calibration logic.
`phy_read_stage1_cal.v/vhd`	This module contains the logic for stage 1 calibration.
`phy_read_stage2_cal.v/vhd`	This module contains the logic for stage 2 calibration.
`phy_read_sync.v/vhd`	This module synchronizes control signals from the clk domain to the clk_rd domain.
`phy_read_top.v/vhd`	This is the top-level of the read path.

Table 3-2: **Files in `example_design/rtl` Directory *(Cont'd)***

Name	Description
phy_read_vld_gen.v/vhd	This module contains the logic to generate the valid signal for the read data returned on the user interface.
phy_reset_sync.v/vhd	This module synchronizes control signals from the clk domain to the clk_rd domain.
phy_v6_d_q_io.v/vhd	This is the I/O module for a single DQ bit coming from the memory.
qdr_rld_infrastructure.v/vhd	This modules helps in clock generation and distribution.
qdr_rld_phy_ocb_mon.v/vhd	This module contains the logic for aligning two clock signals for phase detection.
qdr_rld_phy_pd.v/vhd	This is the top-level module for the phase detector.
rld_mc.v/vhd	This module implements the memory controller.
rld_phy_iob.v/vhd	This module instantiates the modules that use IOBs.
rld_phy_top.v/vhd	This is the top-level module for the physical layer.
rld_phy_write_control_io.v/vhd	This module contains the logic for the control signals going to the memory.
rld_phy_write_data_io.v/vhd	This module contains the logic for the data and byte writes going to the memory.
rld_phy_write_init_sm.v/vhd	This module contains the logic for the initialization state machine.
rld_phy_write_top.v/vhd	This is the top-level wrapper for the write path.
rld_tb_addr_gen.v/vhd	This module generates the addresses used in the example test bench for simulation.
rld_tb_top.v/vhd	This is the top-level of the synthesizable test bench.
rld_tb_wr_rd_sm.v/vhd	This is the test bench write/read state machine that issues commands during simulation.
rld_top.v/vhd	This is the top-level wrapper for the memory controller, user interface, and the PHY.
rld_ui_addr.v/vhd	This module generate the FIFOs used to buffer address and commands for the user interface.
rld_ui_top.v/vhd	This is the top-level wrapper for the user interface.
rld_ui_wr.v/vhd	This module generate the FIFOs used to buffer write data for the user interface.
tb_cmp_data.v/vhd	This is the comparison module for the data returning from the PHY in a simulation.

Table 3-2: **Files in `example_design/rtl` Directory *(Cont'd)***

Name	Description
`tb_cmp_data_bits.v/vhd`	This is the comparison module for a single bit of data.
`tb_data_gen.v/vhd`	This module generates the data to use for write requests in the example test bench.

`<component name>/example_design/sim`

Table 3-3 lists the files in the example_design/sim directory.

Table 3-3: **Files in `example_design/sim` Directory**

Name	Description
`glbl.v`	This file is used for initializing the simulation environment.
`sim.do`	This is the script used for running a ModelSim simulation.
`sim_tb_top.v/vhd`	This is the top-level simulation file.

`<component name>/example_design/synth`

Table 3-4 lists the files in the example_design/synth directory.

Table 3-4: **Files in `example_design/synth` Directory**

Name	Description
`example_top.lso`	This is a library search order file provided for XST.
`example_top.prj`	This is the ISE software project file used for synthesis.

`<component name>/user_design/par`

Table 3-5 lists the files in the user_design/par directory.

Table 3-5: **Files in `user_design/par` Directory**

Name	Description
`bitgen_options.ut`	This file contains the bitgen options when running the design through implementation.
`create_ise.bat`	This is a batch file to generate an ISE project for implementing the design using the GUI.
`<component name>.ucf`	This is the UCF generated from the bank selections.
`ise_flow.bat`	This is a batch file for running the design through the implementation tools through the command-line interface.
`rem_files.bat`	This is a batch file used to remove implementation files when rerunning a design through the tools. This file is called by `ise_flow.bat` to ensure that previous implementation results are discarded.
`xst_options.txt`	This file contains the XST synthesis options when running the design through implementation.
`set_ise_prop.tcl`	This file is used by `create_ise.bat` to set the ISE project options.

<component name>/user_design/rtl

Table 3-6 lists the files in the user_design/rtl directory.

Table 3-6: **Files in user_design/rtl Directory**

Name	Description
clk_ibuf.v/vhd	This module instantiates the system clock input buffers.
<component name>.v/vhd	This top-level module serves as an example for connecting the user design to the Virtex-6 FPGA memory interface core.
iodelay_ctrl.v/vhd	This module instantiates the IDELAYCTRL primitive needed for IODELAY use.
phy_d_q_io.v/vhd	This is the I/O module for the entire DQ bus for a single byte lane.
phy_oserdes_io.v/vhd	This is the I/O module for a single bit of data going to the memory.
phy_read_clk_io.v/vhd	This is the I/O module for the incoming CQ/CQ# echo clocks from the memory.
phy_read_data_align.v/vhd	This module realigns the incoming data.
phy_read_dcb.v/vhd	This module transfers the data from the clk_rd domain into the clk domain.
phy_read_dly_ctrl.v/vhd	This module drives the IODELAY control for each clock and data I/O based on the control from the calibration logic.
phy_read_stage1_cal.v/vhd	This module contains the logic for stage 1 calibration.
phy_read_stage2_cal.v/vhd	This module contains the logic for stage 2 calibration.
phy_read_sync.v/vhd	This module synchronizes control signals from the clk domain to the clk_rd domain.
phy_read_top.v/vhd	This is the top-level of the read path.
phy_read_vld_gen.v/vhd	This module contains the logic to generate the valid signal for the read data returned on the user interface.
phy_reset_sync.v/vhd	This module synchronizes control signals from the clk domain to the clk_rd domain.
phy_v6_d_q_io.v/vhd	This is the I/O module for a single DQ bit coming from the memory.
qdr_rld_infrastructure.v/vhd	This modules helps in clock generation and distribution.
qdr_rld_phy_ocb_mon.v/vhd	This module contains the logic for aligning two clock signals for phase detection.
qdr_rld_phy_pd.v/vhd	This is the top-level module for the phase detector.
rld_mc.v/vhd	This module implements the memory controller.

Table 3-6: **Files in `user_design/rtl` Directory *(Cont'd)***

Name	Description
rld_phy_iob.v/vhd	This module instantiates the modules that use IOBs.
rld_phy_top.v/vhd	This is the top-level module for the PHY.
rld_phy_write_control_io.v/vhd	This module contains the logic for the control signals going to the memory.
rld_phy_write_data_io.v/vhd	This module contains the logic for the data and byte writes going to the memory.
rld_phy_write_init_sm.v/vhd	This module contains the logic for the initialization state machine.
rld_phy_write_top.v/vhd	This is the top-level wrapper for the write path.
rld_top.v/vhd	This is the top-level wrapper for the memory controller, user interface, and the PHY.
rld_ui_addr.v/vhd	This module generate the FIFOs used to buffer address and commands for the user interface.
rld_ui_top.v/vhd	This is the top-level wrapper for the user interface.
rld_ui_wr.v/vhd	This module generate the FIFOs used to buffer write data for the user interface.

`<component name>/user_design/sim`

Table 3-7 lists the files in the `user_design/sim` directory.

Table 3-7: **Files in `user_design/sim` Directory**

Name	Description
glbl.v	This file is used for initializing the simulation environment.
sim.do	This is the script used for running a ModelSim simulation.
sim_tb_top.v/vhd	This is the top-level simulation file.
tb_cmp_data.v/vhd	This is the comparison module for the data returning from the PHY in a simulation.
tb_cmp_data_bits.v/vhd	This is the comparison module for a single bit of data.
tb_data_gen.v/vhd	This module generates the data to use for write requests in the example test bench.
rld_tb_addr_gen.v/vhd	This module generates the addresses used in the example test bench for simulation.
rld_tb_top.v/vhd	This is the top-level of the synthesizable test bench.
rld_tb_wr_rd_sm.v/vhd	This is the test bench write/read state machine that issues commands during simulation.

<component name>/user_design/synth

Table 3-8 lists the files in the user_design/synth directory.

Table 3-8: **Files in user_design/synth Directory**

Name	Description
<component name>.lso	This is a library search order file provided for XST.
<component name>.prj	This is the ISE software project file used for synthesis.

Verify UCF and Update Design and UCF Rules

Verify UCF and Update Design and UCF verifies the input UCF for bank selection, pin allocation, and constraint allocation rules, and generates warnings or error reports for any issue. It does not verify the input .prj file. This feature is useful to verify any UCF pinout changes after the design is generated from the MIG tool. The user must load the MIG generated .prj file (the original .prj file) without any modifications. The verification report is not correct if any of the parameters in the original .prj file are altered. In the CORE Generator tool, the recustomization option should be selected to reload the project. The design can be generated only when Verify UCF does not report an error in the verification report. Warnings can be ignored while generating a design.

These rules are verified from the input UCF:

- If a pin is allocated to more than one signal, the tool reports an error.
  - The tool stops further verification if the UCF does not adhere to the uniqueness property.
- The associative property is verified:
  - If the output data clock pair is allocated to a single-region clock-capable (SRCC) pair, all its associated signals should be allocated in the same bank.
  - If the output data clock pair is allocated to an MRCC pin, all its associated signals should be allocated within the banks above and below.
- Banks should be allocated for the group's address and data within the vicinity arena.
  - An error occurs if a bank is allocated outside the vicinity arena.
- The system clock should be selected either to the GC bank (24, 25, 34, and 35) or to the bank adjacent to the capture clock bank.
  - The system clock bank can be selected adjacent to the capture clock bank only when the frequency of this controller is not repeated to any other controllers. If the frequency of this controller is repeated to another controller, the system clock group must be allocated to any one of the GC banks (24, 25, 34, and 35).
  - The signal pairs sys_clk and clk_ref should be allocated to the CC pair or GC pair pins (for banks adjacent to the capture clock bank) or to the GC pair pins (for GC banks).
- The memory clock pairs should be allocated to the differential signals.
- In the DCI CASCADE syntax, the selected configuration should require the master bank.
  - The slave banks provided should be valid.
- A valid MMCM constraint value should be provided, otherwise a warning is generated.

If the UCF satisfies the above rules, the updated design can be generated. The design:

- Provides the latest HDL.
- Updates the UCF with the latest clock constraints or any TIGs provided by keeping the same pinout.
- Generates even the compatible UCFs if the project loaded contains the compatible FPGA selection.

Error Messages

This section describes the error messages that are generated when verifying the UCF. The reference UCF must follow the MIG naming conventions (see the UCF generated by the MIG tool, or names used for the ML605 board).

- **Uniqueness**: If two or more signals are allocated to the same pins in the reference UCF, an error message is listed in the directed file with a user-assigned name.

  The error message format is "<*signalname1*> and <*signalname2*> are allocated to the same pin." For example, if rld2_dq[0] and rld2_qk[0] are allocated to the same pin, such as:

  ```
  NET "rld2_dq[0]" LOC = "D12";
  NET "rld2_qk[0]" LOC = "D12";
  ```

  Then this error message is displayed:

  ```
  ERROR: rld2_dq[0] and rld2_qk[0] are allocated to the same pin. Pins
  are not unique.
  ```

- **Association**: If the output Data Clock pins are allocated to the SRCC pins, these error messages are displayed:

  ```
  ERROR: Pin Names (rld2_qk[0]) and (rld2_dq[32]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[0]) and (rld2_dq[34]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[0]) and (rld2_dq[33]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[0]) and (rld2_dq[35]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[68]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[69]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[65]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[64]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[67]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[66]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[70]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ERROR: Pin Names (rld2_qk[3]) and (rld2_dq[71]) should be allocated
  in the same bank as the strobe pins are allocated to 'SRCC P' pin.
  ```

- **Vicinity Verification**: Error messages are displayed when the pins are allocated out of the vicinity arena.
  - If the data write bank selected is out of the vicinity arena, these error messages are displayed:

```
ERROR: c0_rld2_dq[0](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[1](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[2](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[3](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[4](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[5](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27,37,38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[6](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.

ERROR: c0_rld2_dq[7](Data) should not be allocated to bank 42. The
rule is, it can only be moved within the bank(s) "27, 37, 38"
specified in the input mig.prj file for "Data" group.
```

- **Differential Pair Verification**: If the system clock pins are not allocated to the differential pairs, these error messages are displayed:

```
ERROR: "sys_clk_p" Should be allocated to either CC P pin or GC P
pin.

ERROR: "sys_clk_n" Should be allocated to either CC N pin or GC N
pin.

ERROR: "sys_clk_p" and "sys_clk_n" Should be allocated to either CC
or GC P/N pair.

ERROR: "clk_ref_p" Should be allocated to either CC P pin or GC P
pin.

ERROR: "clk_ref_n" Should be allocated to either CC N pin or GC N
pin.

ERROR: "clk_ref_p" and "clk_ref_n" Should be allocated to either CC
or GC P/N pair.
```

- **Absence of Signals**: If one or more signal pin pairs is missing and/or commented in the given UCF against the selected inputs, the verification result indicates the absence of these signal pin pairs as a warning.

  The warning message format is "Signal <*signal_name*> is expected, but not present in the UCF." For example:

```
WARNING: Signal "rld2_dq[15]" expected, but not present in the UCF.

WARNING: Signal "rld2_dq[16]" expected, but not present in the UCF.

WARNING: Signal "rld2_dq[17]" expected, but not present in the UCF.
```

- **Master Bank Verification**: This verifies whether the provided master bank is valid for the selected DCI banks in the column. This error message is displayed when the valid master bank is not provided for the column:

```
ERROR: the master bank "23" provided is not valid master bank.
Following are the valid master bank "24, 25" for the column "1".
```

Quick Start Example Design

After the core is successfully generated, the example design HDL can be processed through the Xilinx Implementation toolset. The MIG tool provides a simple synthesizable test bench to generate traffic to test the core. An architectural overview of the test bench is shown in Figure 3-42, page 278. The top level of the test bench (`sim_tb_top`) is located in `<project_dir>/sim` and contains the memory model to simulate against the top level of the example design (`ip_top`). The `ip_top` folder contains the infrastructure module, the IODELAY controller, and the simple test bench. The infrastructure module generates all the clocking signals needed by the core. In `tb_top` are the modules used to generate commands, data, and addresses, as well as a comparator module that checks the responses to verify whether or not the correct data was returned.

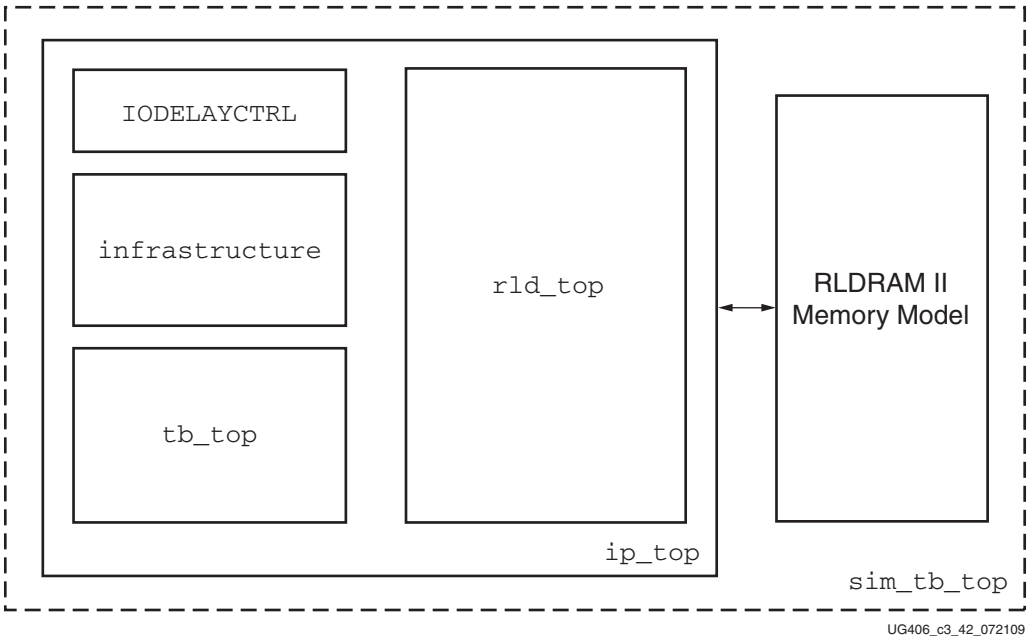


Figure 3-42: **Top Level of Test Bench**

Simulating the Example Design

The Xilinx® UNISIM library must be mapped into the simulator. The test bench provided with the example design supports these pre-implementation simulations:

- The test bench along with vendor's memory model used in the example design
- The RTL files of the MC and the PHY core, created by the MIG tool

The simulation can be run from this directory:

```
<project_dir>/<component_name>/sim
```

ModelSim is the only supported simulation tool. The simple test bench can be run using ModelSim by executing the `sim.do` script.

Implementing the Example Design

The `xst.bat` script file runs the design through synthesis, translate, map, and par. This script file sets all the required options and should be referred to for the recommended build options for the design.

Designing with the Core

The core is bundled with an example design that can be simulated. The example design can be used as a starting point for the user design or as a reference for debugging purposes. Only supported modifications should be made to the configuration of the core. See Customizing the Core, page 296 for supported configuration parameters.

Core Architecture

This section describes the design implementation of the PHY.

Overview

Figure 3-43 is a high-level block diagram of the Virtex-6 FPGA RLDRAM II memory interface solution. This figure shows both the internal FPGA connections to the client interface for initiating read and write commands, and the external interface to the memory device.

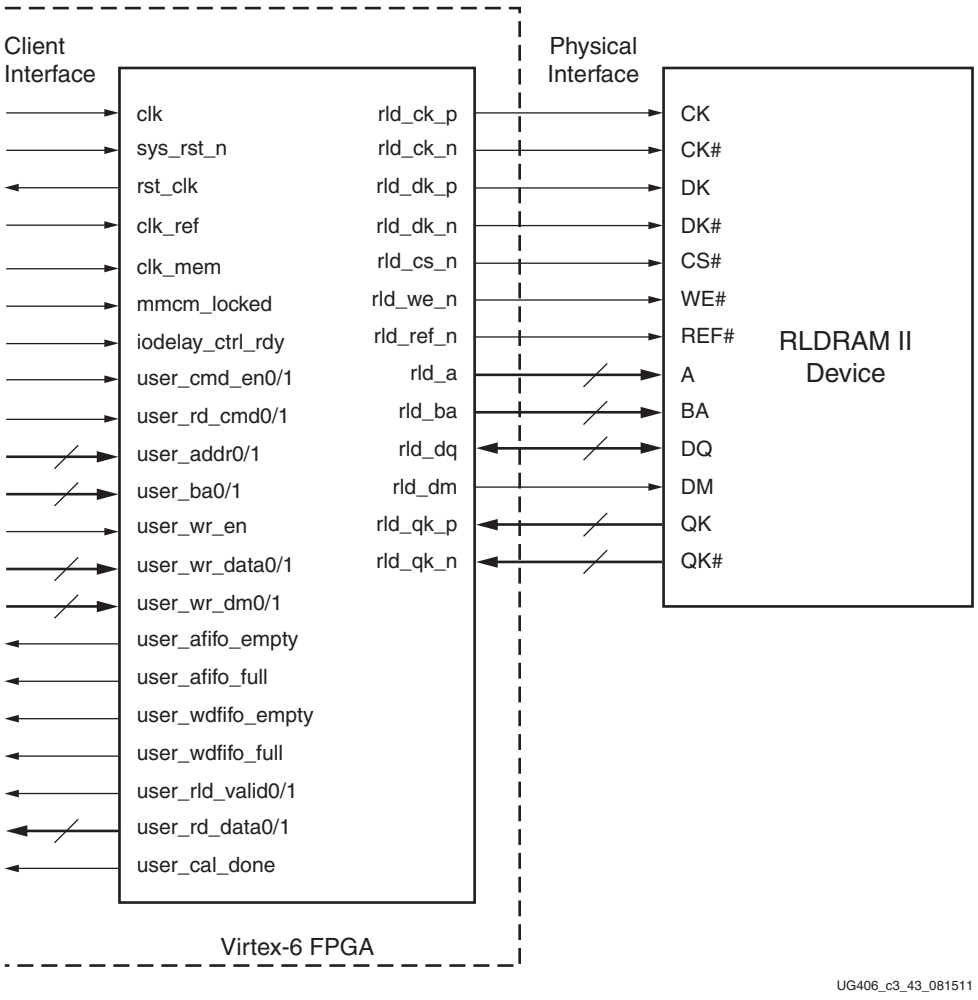


Figure 3-43: **High-Level Block Diagram of the Memory Interface Solution**

The core is composed of these elements, as shown in Figure 3-44:

- Client interface
- Memory controller
- Physical interface
- Read path
- Write path

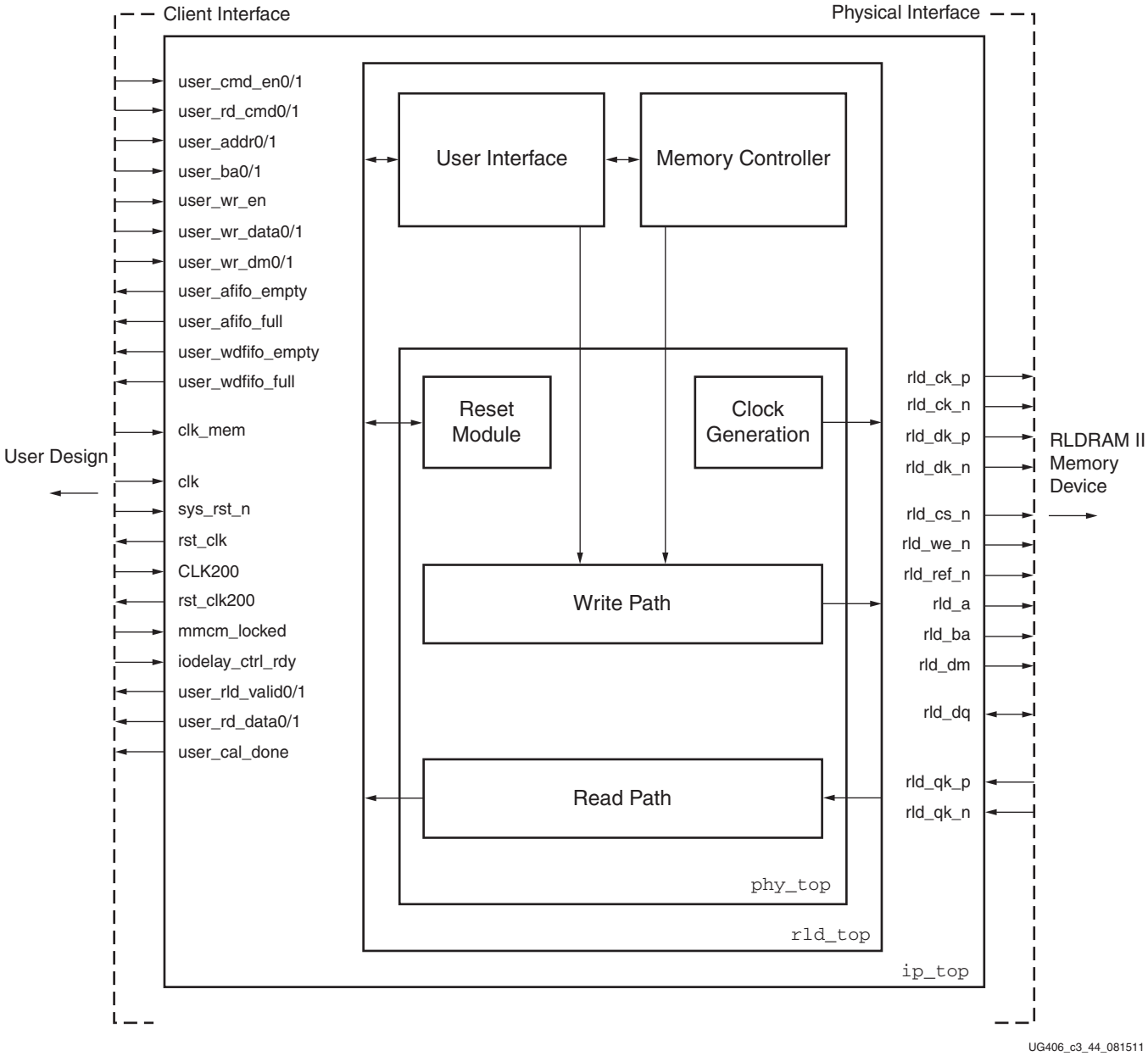


Figure 3-44: **Components of the RLDRAM II Memory Interface Solution**

The client interface (also known as the user interface) uses a simple protocol based entirely on SDR signals to make read and write requests. Refer to Client Interface, page 282 for more details describing this protocol.

The memory controller takes commands from the user interface and adheres to the protocol requirements of the RLDRAM II memory device. Refer to Memory Controller, page 290 for more details.

The physical interface generates the proper timing relationships and DDR signaling to communicate with the external memory device, while conforming to the RLDRAM II protocol and timing requirements. Refer to Physical Interface, page 286 for more details.

Within the PHY, logic is broken up into read and write paths. The write path generates the RLDRAM II signaling for generating read and write requests. This includes clocking, control signals, address, data, and data mask signals. The read path is responsible for calibration and providing read responses back to the user with a corresponding valid signal. Refer to Calibration, page 294 for more details describing this process.

Client Interface

The client interface connects the Virtex-6 FPGA user design to the RLDRAM II memory solutions core to simplify interactions between the user and the external memory device.

Command Request Signals

The client interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in Table 3-9 and are listed assuming four-word or eight-word burst architectures. When using a burst length of two, some additional signals are required, as listed in Table 3-10, page 284. Although the top level contains debug signals, these are left out of Table 3-9 and are described further in Debugging Virtex-6 FPGA RLDRAM II Memory Designs, page 301.

Table 3-9: **Client Interface Request Signals**

Signal	Direction	Description
user_cmd_en0	Input	**Command Enable**. This signal issues a read or write request and indicates that the corresponding command signals are valid.
user_rd_cmd0	Input	**Read Command**. This signal issues a read request. When user_cmd_en0 is asserted, this signal is active High for a read command and active Low for a write command.
user_addr0[ADDR_WIDTH – 1:0]	Input	**Command Address**. This is the address to use for a command request. It is valid when user_cmd_en is asserted.
user_ba0[BANK_WIDTH – 1:0]	Input	**Command Bank Address**. This is the address to use for a write request. It is valid when user_cmd_en is asserted.
user_wr_en	Input	**Write Data Enable**. This signal issues the write data and data mask. It indicates that the corresponding user_wr_* signals are valid.

Table 3-9: **Client Interface Request Signals *(Cont'd)***

Signal	Direction	Description
user_wr_data0[DATA_WIDTH – 1:0]	Input	**Write Data 0**. This is the data to use for a write request and is composed of the rise and fall data concatenated together. It is valid when user_wr_en is asserted.
user_wr_data1[DATA_WIDTH – 1:0]	Input	**Write Data 1**. This is the data to use for a write request and is composed of the rise and fall data concatenated together. It is valid when user_wr_en is asserted.
user_wr_dm0[NUM_DEVICES – 1:0]	Input	**Write Data Mask 0**. When active High, the write data for a given selected device is masked and not written to the memory. It is valid when user_wr_en is asserted.
user_wr_dm1[NUM_DEVICES – 1:0]	Input	**Write Data Mask 0**. When active High, the write data for a given selected device is masked and not written to the memory. It is valid when user_wr_en is asserted.
user_afifo_empty	Output	**Address FIFO empty**. If asserted, the command buffer is empty.
user_wdfifo_empty	Output	**Write Data FIFO empty**. If asserted, the write data buffer is empty.
user_afifo_full	Output	**Address FIFO empty**. If asserted, the command buffer is full, and any writes to the FIFO are ignored until deasserted.
user_wdfifo_full	Output	**Write Data FIFO empty**. If asserted, the write data buffer is full, and any writes to the FIFO are ignored until deasserted.
user_rd_valid0	Output	**Read Valid 0**. This signal indicates that data read back from memory is available on user_rd_data0 and should be sampled.
user_rd_valid1	Output	**Read Valid 1**. This signal indicates that data read back from memory is available on user_rd_data1 and should be sampled.
user_rd_data0[DATA_WIDTH – 1:0]	Output	**Read Data 0**. This is the data read back from the read command.
user_rd_data1[DATA_WIDTH – 1:0]	Output	**Read Data 1**. This is the data read back from the read command.
user_cal_done	Output	**Calibration Done**. This signal indicates back to the user design that read calibration is complete and requests can now take place.

Table 3-10: **Additional Client Interface Request Signals used for BL2**

Signal	Direction	Description
user_cmd_en1	Input	Reserved for future use. Tie Low.
user_rd_cmd1	Input	Reserved for future use. Tie Low.
user_addr1[ADDR_WIDTH – 1:0]	Input	Reserved for future use. Tie Low.
user_ba1[BANK_WIDTH – 1:0]	Input	Reserved for future use. Tie Low.

Interfacing with the Core through the Client Interface

The client interface protocol is shown in Figure 3-45 for the four-word burst architecture.

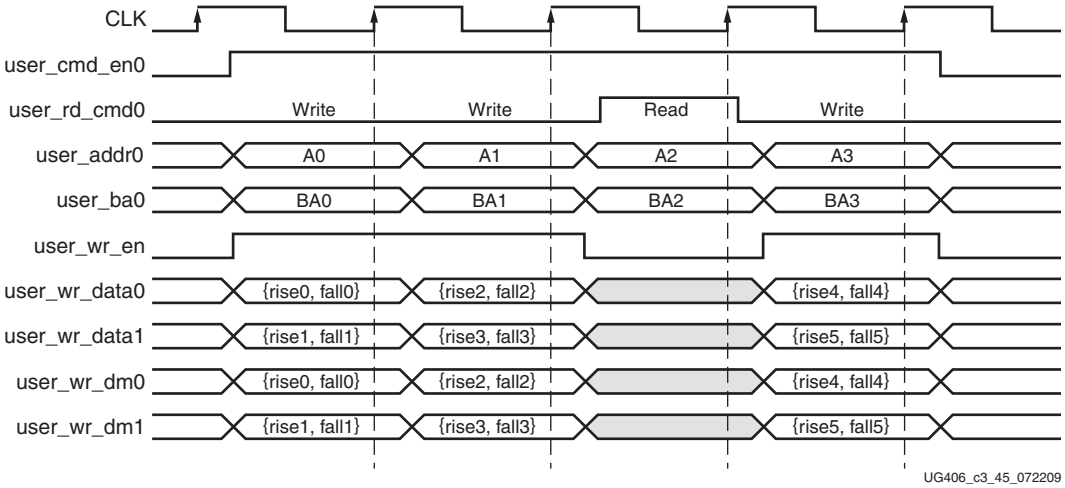


Figure 3-45: **Client Interface Protocol (Four-Word Burst Architecture)**

Before any requests can be accepted, the rst_clk signal must be deasserted Low. After the rst_clk signal is deasserted, the user interface FIFOs can accept commands and data for storage. The user_cal_done signal is asserted after the memory initialization procedure and PHY calibration are complete, and the core can begin to service client requests.

A command request is issued by asserting user_cmd_en0 as a single cycle pulse. At this time, the user_rd_cmd0, user_addr0, and user_ba0 signals must be valid. To issue a read request, user_rd_cmd0 is asserted active High, while for a write request, user_rd_cmd0 is kept Low. For a write request, the data is to be issued in the same cycle as the command by asserting the user_wr_en signal High and presenting valid data on user_wr_data0, user_wr_data1, user_wr_dm0, and user_wr_dm1. For an eight-word burst architecture, an extra cycle of data is required for a given write command, as shown in Figure 3-46. Any gaps in the command flow required can be filled with read commands, if desired.

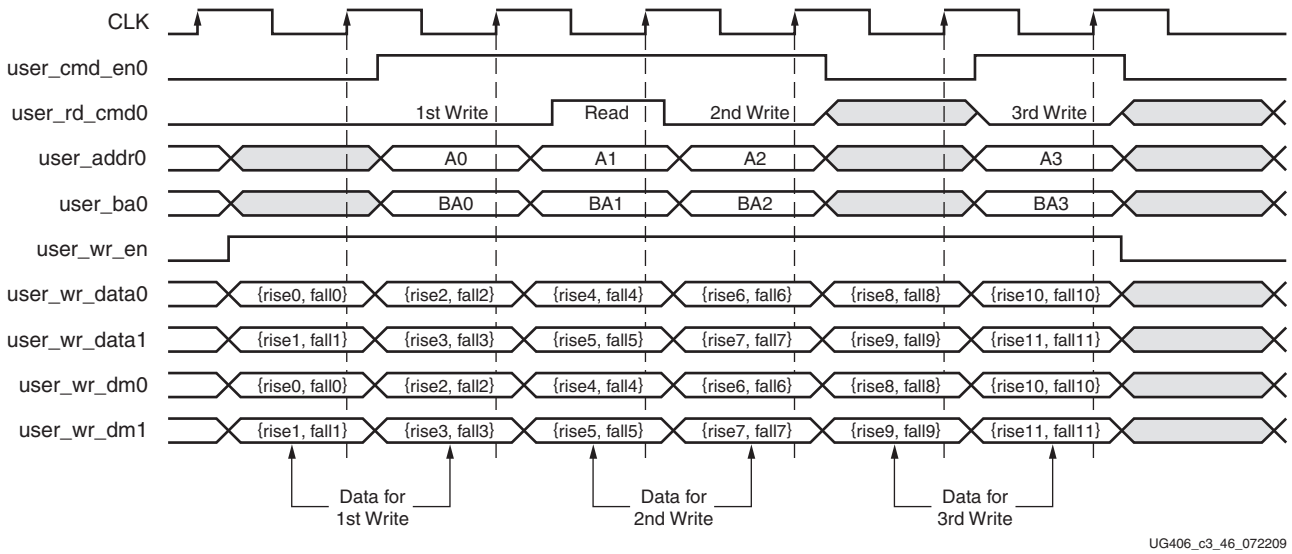


Figure 3-46: **Client Interface Protocol (Eight-Word Burst Architecture)**

When a read command is issued some time later (based on the configuration and latency of the system), the user_rd_vld0 signal is asserted, indicating that user_rd_data0 is now valid, while user_rd_vld1 is asserted indicating that user_rd_data1 is valid, as shown in Figure 3-47. The read data should be sampled on the same cycle that user_rd_vld0 and user_rd_vld1 are asserted because the core does not buffer returning data. This functionality can be added in by the user, if desired.

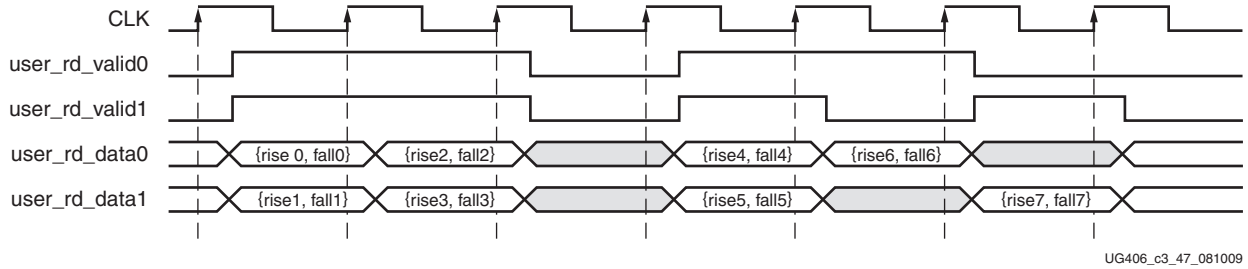


Figure 3-47: **Client Interface Protocol Read Data**

Core Clocking and Reset Requirements

The PHY requires several clocks to function properly. These clocks are described in Table 3-11. As part of calibration, the reset signals used by the core must be tightly controlled, and it is highly recommended that these signals are not altered. To control these signals, a reset module exists within the PHY that synchronizes all reset signals and provides them back through the client interface for use. These signals are also listed in Table 3-11.

Table 3-11: **Client Interface Clocking and Reset Signals**

Signal	Direction	Description
clk	Input	**Divided Clock**. This clock is half the frequency of the memory clock and used as the main system clock.
sys_rst	Input	**System Reset**. This is the asynchronous reset to be synchronized in the reset module within the PHY. This signal must be held for at least 3 clk cycles.
rst_clk	Output	**Divided Clock Reset**. This is the synchronized reset provided from the PHY back to the user's client interface.
clk_ref	Input	**Reference Clock**. This is the reference clock used for the IODELAY controller. Allowable speeds are 200 MHz (-1, -2, and -3 devices) or 300 MHz (-2 and -3 devices only).
clk_mem	Input	**Full Frequency Memory Clock**. This is a full-frequency clock provided from the MMCM and should only be used as an input to the OSERDES.
mmcm_locked	Input	**MMCM Locked**. This signal indicates that the MMCM is locked.
iodelay_ctrl_rdy	Input	**IODELAY Controller Ready**. This signal from the IODELAY controller indicates that the IODELAYs are ready to be used. The PHY is held in reset until the controller is ready.

Physical Interface

The physical interface is the connection from the FPGA memory interface solution to an external RLDRAM II device. The I/O signals for this interface are shown in Table 3-12. These signals can be directly connected to the corresponding signals on the RLDRAM II device.

Table 3-12: **Physical Interface Signals**

Signal	Direction	Description
rld_ck_p	Output	**System Clock CK**. This is the address/command clock to the memory device.
rld_ck_n	Output	**System Clock CK#**. This is the inverted system clock to the memory device.
rld_dk_p	Output	**Write Clock DK**. This is the write clock to the memory device.
rld_dk_n	Output	**Write Clock DK#**. This is the inverted write clock to the memory device.
rld_a	Output	**Address**. This is the address supplied for memory operations.
rld_ba	Output	**Bank Address**. This is the bank address supplied for memory operations.
rld_cs_n	Output	**Chip Select CS#**. This is the active-Low chip select control signal for the memory.

Table 3-12: **Physical Interface Signals *(Cont'd)***

Signal	Direction	Description
rld_we_n	Output	**Write Enable WE#**. This is the active-Low write enable control signal for the memory.
rld_cs_n	Output	**Refresh REF#**. This is the active-Low refresh control signal for the memory.
rld_dq	Input/Output	**Data DQ**. This is a bidirectional data port, driven by the FPGA for writes and by the memory for reads.
rld_qk_p	Input	**Read Clock QK**. This is the read clock returned from the memory edge aligned with read data on rld_dq. This clock (in conjunction with QK#) is used by the PHY to sample the read data on rld_dq.
rdd_qk_n	Input	**Read Clock QK#**. This is the inverted read clock returned from the memory. This clock (in conjunction with QK) is used by the PHY to sample the read data on rld_dq.

PHY-Only Interface

The PHY-only interface is described here to facilitate designing a controller to interface with the PHY. The client interface signals described in Table 3-11, page 286 and the physical interface signals described in Table 3-12, page 286 are still required for the PHY-only interface.

When using a custom controller, care must be taken to abide by the memory specifications. Because the PHY is clocked at half the memory clock frequency, two commands must be issued per clock cycle. In a half-frequency design, internal FPGA logic timing is easier to meet, but more signals are required for a given internal clock cycle. This is because for each internal clock cycle, there are two fast clock cycles for the memory interface. The PHY takes the two commands and sends them to the memory in the order received, where the signal names ending in 0 go out first before the signal names ending in 1. The 0 signals comprise the command and data for the first fast clock cycle, while the 1 signals comprise the command and data for the second fast clock cycle. Before cal_done is asserted, the PHY controls the command, address, and data bus of the memory. The input signals listed in Table 3-13 must be valid after cal_done is asserted from the PHY, because control over the memory interface is transferred to the controller, and the input signals listed in Table 3-13 are used to send out commands and data over the memory interface.

Table 3-13: **PHY-Only Interface Signals**

Signal	Direction	Description
cs0[NUM_DEVICES – 1:0]	Input	**Memory Chip Select 0**. This signal is active High, and one exists per device. This is the first chip select sent out.
cs1[NUM_DEVICES – 1:0]	Input	**Memory Chip Select 1**. This signal is active High, and one exists per device. This is the second chip select sent out after cs0.
we0	Input	**Memory Write Enable 0**. This active-High signal is the first write enable sent out.

Table 3-13: **PHY-Only Interface Signals *(Cont'd)***

Signal	Direction	Description
we1	Input	**Memory Write Enable 1**. This active-High signal is the second write enable sent out after we0.
ref0	Input	**Memory Refresh 0**. This active-High signal is the first refresh sent out.
ref1	Input	**Memory Refresh 1**. This active-High signal is the second refresh sent out after ref0.
addr0[ADDR_WIDTH – 1:0]	Input	**Memory Address 0**. This is the first address and is associated with cs0, we0, and ref0.
addr1[ADDR_WIDTH – 1:0]	Input	**Memory Address 1**. This is the second address and is associated with cs1, we1, and ref1.
ba0[BANK_WIDTH – 1:0]	Input	**Memory Bank Address 0**. This is the first bank address and is associated with cs0, we0, and ref0.
ba1[BANK_WIDTH – 1:0]	Input	**Memory Bank Address 1**. This is the second bank address and is associated with cs1, we1, and ref1.
wr_en0	Input	**Write Enable 0**. This signal is necessary to control the 3-state OSERDES inputs for bidirectional interfaces.
wr_en1	Input	**Write Enable 1**. This signal is necessary to control the 3-state OSERDES inputs for bidirectional interfaces.
wr_data0[DATA_WIDTH × 2 – 1:0]	Input	**Write Data 0**. This is the data to use for a write request. It is composed of the rise and fall data concatenated together.
wr_data1[DATA_WIDTH × 2 – 1:0]	Input	**Write Data 1**. This is the data to use for a write request. It is composed of the rise and fall data concatenated together.
wr_dm0[NUM_DEVICES × 2 – 1:0]	Input	**Write Data Mask 0**. When active High, the write data for a given selected device is masked and not written to the memory.
wr_dm1[NUM_DEVICES × 2 – 1:0]	Input	**Write Data Mask 1**. When active High, the write data for a given selected device is masked and not written to the memory.
cal_done	Output	**Calibration Done**. This signal indicates back to the controller that read calibration is complete and hands over control to the controller.

Figure 3-48 shows the timing diagram for a typical configuration 3, burst length of four with commands being sent to the PHY from a controller. After cal_done is asserted, the controller begins issuing commands. A single write command is issued by asserting the cs0 and we0 signals (with ref0 being held Low) and ensuring that addr0 and ba0 are valid. Because this is a burst length of four configuration, the second command that must be issued is a No Operation (NOP), that is, all the control signals (cs1, we1, ref1) are held Low. Two clock cycles later, the wr_en0/1 signals are asserted, and the wr_data0/1 and wr_dm0/1 signals are valid for the given write command. In this same clock cycle, a single read command is issued by asserting cs0 (with we0 and ref0 being held Low) and placing the associated addresses on addr0 and ba0. Two refresh commands are issued by asserting cs0/1, ref0/1, and ba0/1. The refresh commands can be issued in the same clock cycle as long as the memory banking rules are met.

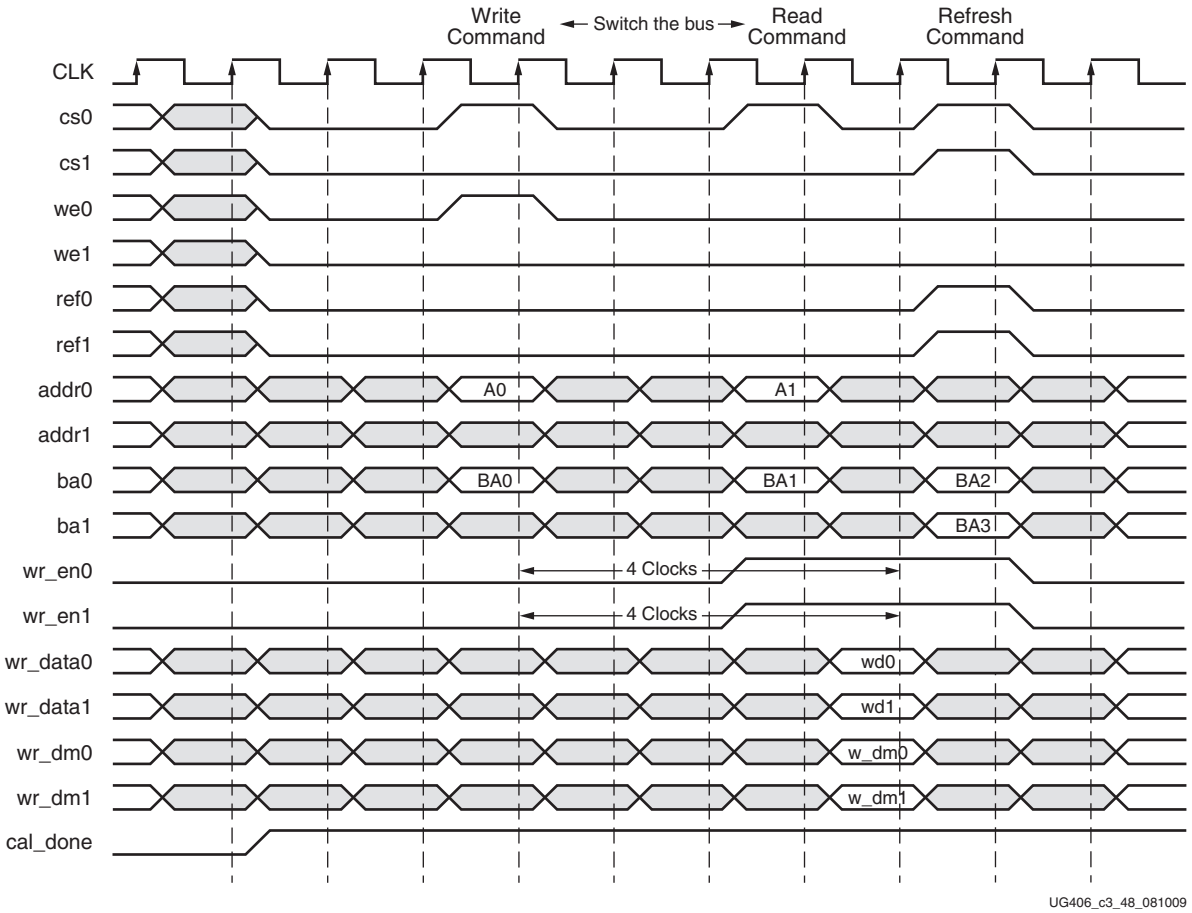


Figure 3-48: **PHY-Only Interface for Burst Length 4, Configuration 3, and Address Multiplexing OFF**

The controller sends the wr_en0/1 signals and data at the necessary time based on the configuration setting. This time changes depending on the configuration. Table 3-14 details when the wr_en0/1 signals should be asserted with the data valid for a given configuration. If Address Multiplexing is used, the PHY handles rearranging the address signals and outputting the address over two clock cycles rather than one.

Table 3-14: **Command to Write Enable Timing**

Address Multiplexing	Configuration	Command to Write Enable (Clock Cycles)
ON	1	3
	2	4
	3	5
OFF	1	2
	2	3
	3	4(1)

Notes:

1. Shown in Figure 3-48.

For CIO memory interfaces, the wr_en0/1 signals are required to be asserted an extra clock cycle before the first wr_en0/1 signal is asserted, and held for an extra clock cycle after deassertion. This is required to ensure that the shared bus has time to change from read to write and from write to read. The physical layer has a requirement of two clock cycles of no operation (NOP) when transitioned from a write to a read, and from a read to a write. This two clock cycle requirement is dependent on the PCB and might need to be increased for different board layouts.

Memory Controller

The memory controller is designed to enforce the RLDRAM II memory access requirements and interface with the PHY. The controller processes commands in order, so the order of commands presented to the controller is the order in which they are presented to the memory device.

The memory controller first receives commands from the user interface and determines if the command can be processed right away or needs to wait. When all requirements are met, the command is placed on the PHY interface. For a write command, the controller generates a signal for the user interface to provide the write data to the PHY. This signal is generated based on the memory configuration to ensure the proper command-to-data relationship. Auto-refresh commands are inserted into the command flow by the controller to meet the memory device refresh requirements.

For CIO devices, the data bus is shared for read and write data. Switching from read commands to write commands and vice versa introduces gaps in the command stream due to switching the bus. For better throughput, changes in the command bus should be minimized when possible.

Write Path

The write path to the RLDRAM II memory includes the address, data, and control signals necessary to execute any memory operation. The control strobes rld_cs_n, rld_we_n, and rld_ref_n, and addresses rld_a and rld_ba to the memory all use SDR formatting. The write data values rld_dq and rld_dm also utilize DDR formatting to achieve the required two-word, four-word, or eight-word burst within the given clock periods. Figure 3-49, page 291 shows a high-level block diagram of the write path and its submodules.

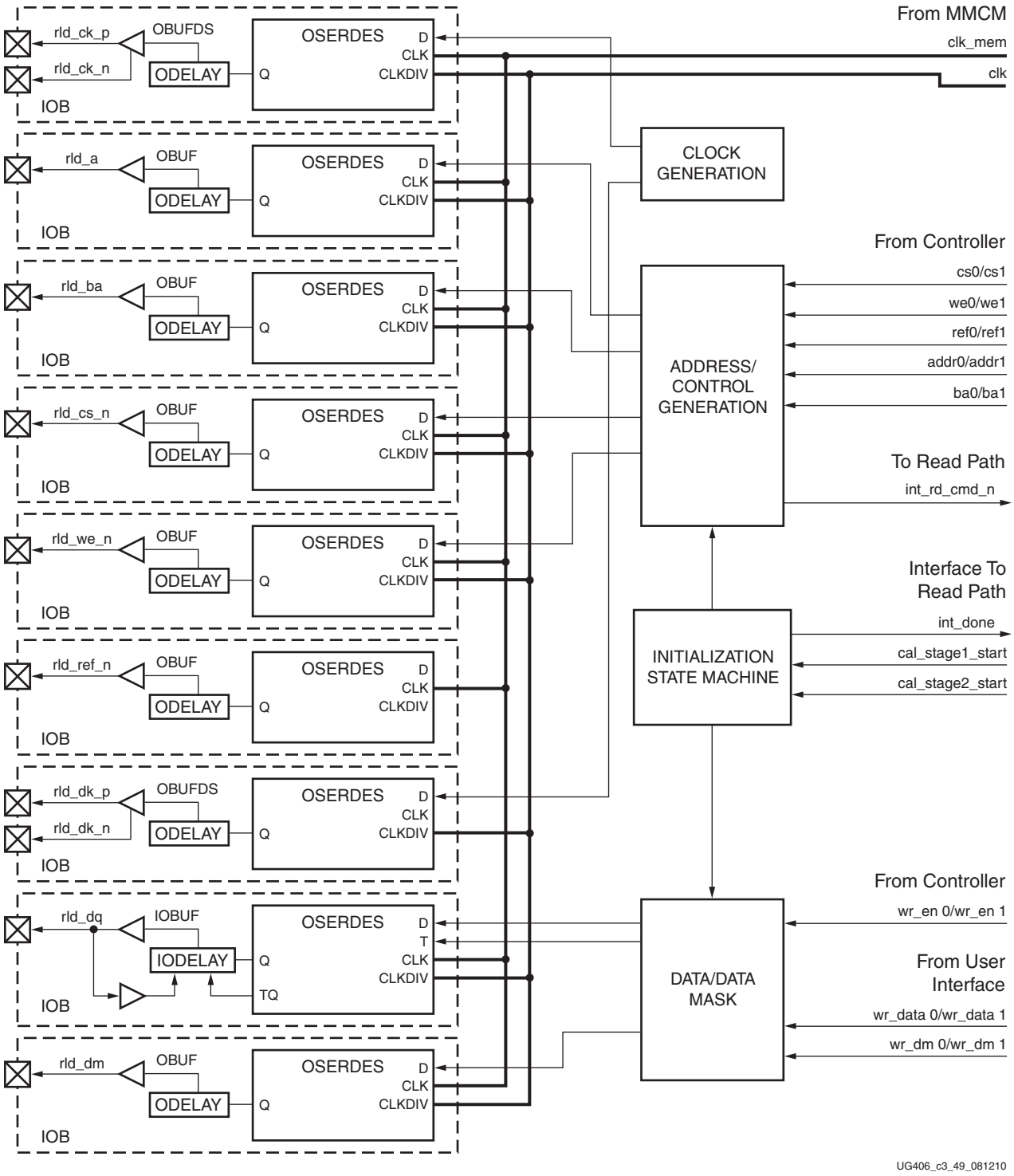


Figure 3-49: **Write Path**

I/O Architecture

The RLDRAM II memory interface solution uses the OSERDES primitive found in the Virtex-6 FPGA I/O blocks for clocking all outputs of the PHY to the memory device. Built-in Virtex-6 FPGA OSERDES functions simplify the task of generating the proper

clock, address, data, and control signaling for communication with the memory device. The flow through the OSERDES uses two different input clocks to achieve maximum performance. Data input ports D1/D2 or D3/D4 are clocked in using the clock provided on the CLKDIV input port (clk in this case), and passed through a parallel-to-serial conversion block.

Figure 3-50 shows a high-level block diagram of the OSERDES flow. The OSERDES is used to clock all outputs from the PHY to the memory device.

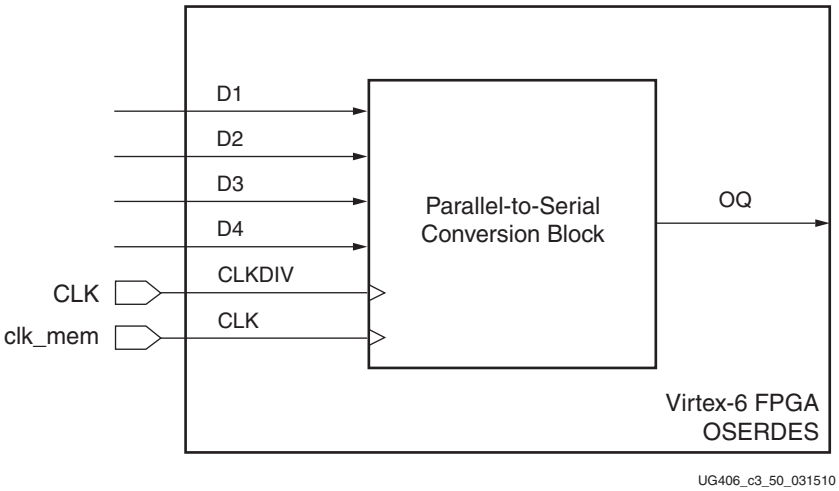


Figure 3-50: **OSERDES Flow**

After exiting the OSERDES, the data needs to be optionally realigned. All the output signals must be presented center aligned with respect to the generated clocks CK/CK#. For this reason, the IODELAY blocks within the I/O blocks are also used in conjunction with the OSERDES to shift the controls and clocks into alignment. The shift is placed on the control and clocks rather than the data to minimize the jitter on the datapath.

In addition to generating the output signals to the memory, the write path also assists the read path with calibration. This logic performs reads and writes based on signals provided from the read path indicating which stage of calibration it is in. The state machine begins by asserting the init_done signal indicating that calibration can begin. This signal is asserted only after the OSERDES block is ready to accept data and the RLDRAM II initialization sequence is complete. Stage one calibration is ready to begin when cal_stage1_start is asserted. During this stage, the pattern `0x00FF_0F0F` is written to the memory device and read back continuously until signaled to begin stage two. During the first stage, QK and QK# are calibrated along with data DQ. Stage two calibration requires one write of the pattern `0xAAAA` followed by one read used to calibrate the valid signal for read responses.

Read Path

The QK-based data capture scheme enables capture of read data from the memory at very high clock rates. The read path captures the returning data and provides a valid strobe back to the user indicating that the return data is on the client interface. Before any read can take place, calibration must occur. Calibration is the main function of the read path and occurs once on reset followed by continuous dynamic calibration. For every bit in the memory, calibration is done against both QK and QK#. After the initial settings are in place, dynamic calibration takes over to account for any voltage and temperature changes that might affect the system's once-ideal settings.

Data Capture

Figure 3-51, page 293 shows a high-level block diagram of the path that data takes from entering the FPGA until given to the user. To capture the data, the read path utilizes an IODELAY and ISERDES that exist in every I/O block on the Virtex-6 device. The IODELAY is used to shift the clocks or data entering the FPGA to adjust its alignment relative to other signals. Following this shift, data then passes through the ISERDES. After the data is retrieved, it enters a data alignment module and optionally realigns as seen in Figure 3-51, page 293. More details about this alignment are provided in Calibration, page 294. Data is transferred from the clk_rd domain to the clk domain through a circular buffer built using distributed RAM. In the clk domain, the valid signal is generated and provided with data back to the user.

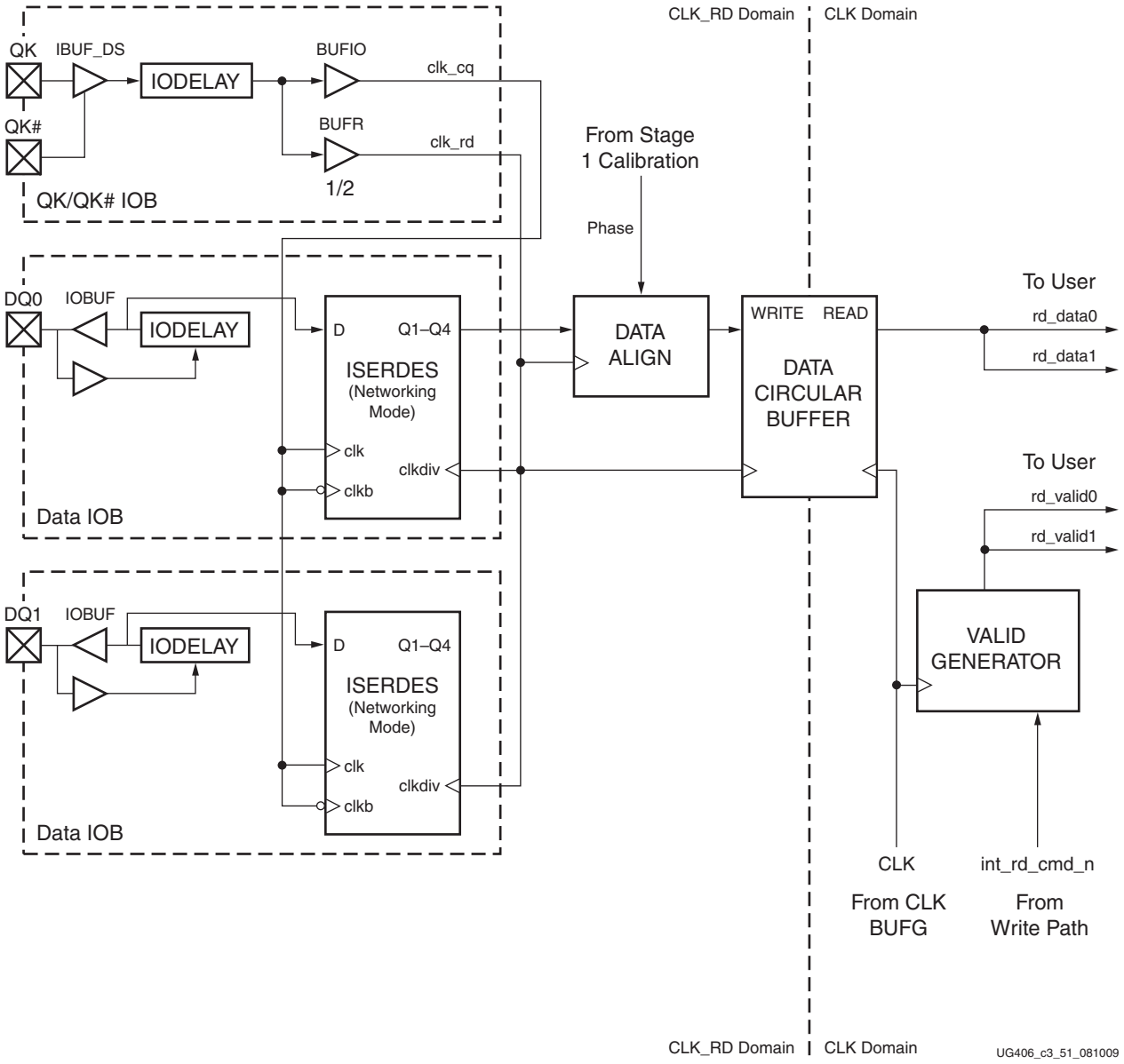


Figure 3-51: **Read Capture**

Calibration

The calibration logic consists of a one-time initial calibration followed by continuous real-time calibration. This logic provides the requested amount of delay on read data (DQ) and the read clocks (QK/QK#) to align the clock in the data valid window. This is done using the IODELAY elements within the I/O block of the Virtex-6 FPGA. The IODELAY elements delay the input in increments of 75 ps, up to a maximum delay of 2.4 ns when using a reference clock frequency of 200 MHz. For a reference clock frequency of 300 MHz (in -3 devices only), the delay increment is 52 ps for a maximum delay of 1.6 ns. An IODELAYCTRL module is needed in conjunction with the IODELAYs to maintain the resolution of the IODELAY elements.

Calibration begins after the RLDRAM II initialization sequence is complete. Prior to this point, all read path logic is held in reset. Calibration is performed in two stages:

1. Calibration of QK with respect to DQ followed by data realignment
2. Resolving latency and valid generation

Calibration of QK, QK# and DQ, and Data Realignment

When the data is returned from memory, it is initially edge aligned to the clocks QK and QK#. To safely capture the data, a sample must be taken from the center of the data. Center aligning QK and QK# to DQ provides the most possible margin for a successful capture. To assist this stage of calibration by ensuring that the calibration logic has predefined data to calibrate against, the write path performs an initial write of `0x0FF0_0F0F` to an address location in each memory bank followed by continuous reads from these locations.

During calibration, delay adjustments are made by delaying the clock or data through the use of IODELAYs. The basic flow through this stage of calibration is:

1. Find the best tap setting to center align QK/QK# and DQ[0]'s rising and falling edge data for each memory.
2. Perform a fine phase alignment of the ISERDES outputs and find the best tap setting for QK/QK# and DQ[0].
3. Determine which phase alignment of the ISERDES output provides the best results (the best result is determined by which polarity delays the data the least and which is most accurately found in the center of DQ).
4. Set the selected fine phase alignment.
5. Each subsequent bit is now calibrated to remove any skew differences relative to DQ[0].
6. This process repeats for each memory device on the interface.

Resolving Latency and Valid Generation

This phase of calibration serves several purposes:

- Sets the latency for fixed-latency mode
- Matches the latency for each memory when wider memories are derived from small memories
- Sends the determined latency to the valid generation logic.

This stage is required to generate the valid signal associated with the data on the client interface. During this stage of calibration, a write pattern of `0xAAAA` is written to memory and read back. Doing this allows the read logic to count the number cycles before the expected data returns. The basic flow through this phase is:

1. Count cycles until the read data arrives for each memory device.
2. Determine what value to use as the fixed latency. This value can be either the set value indicated by the user from the PHY_LATENCY parameter, or the maximum latency across all memory devices.
3. Calibrate the generation of the read valid signal. Using the value determined in step 2, delay the read valid signal to align with the read data for the user.
4. Assert cal_done.

Dynamic Calibration

After calibration is complete, an ideal tap setting is found to center align clocks QK/QK# to the data. However, due to changes in voltage or temperature, this relationship might shift over time and no longer be ideal. To compensate for this, real-time calibration is needed to add taps to QK/QK# when necessary.

Reset Module

The reset module synchronizes all the reset signals across all clock domains. This includes the resets from the client interface for the IODELAYCTRL reference clock, the system reset, write path reset, and the read path reset. This logic is contained within the PHY because strict timing must be met on the read path reset with respect to the system reset due to the synchronization logic between these two domains. In addition, the read path must remain in reset until the echo clocks QK/QK# are stable from the memory.

The system reset is provided asynchronously to the reset module and gated with the appropriate signals to use for synchronization across four different clock domains. All reset signals used by the PHY are asynchronously asserted and synchronously deasserted. Figure 3-52, page 296 shows the reset scheme used within this module.

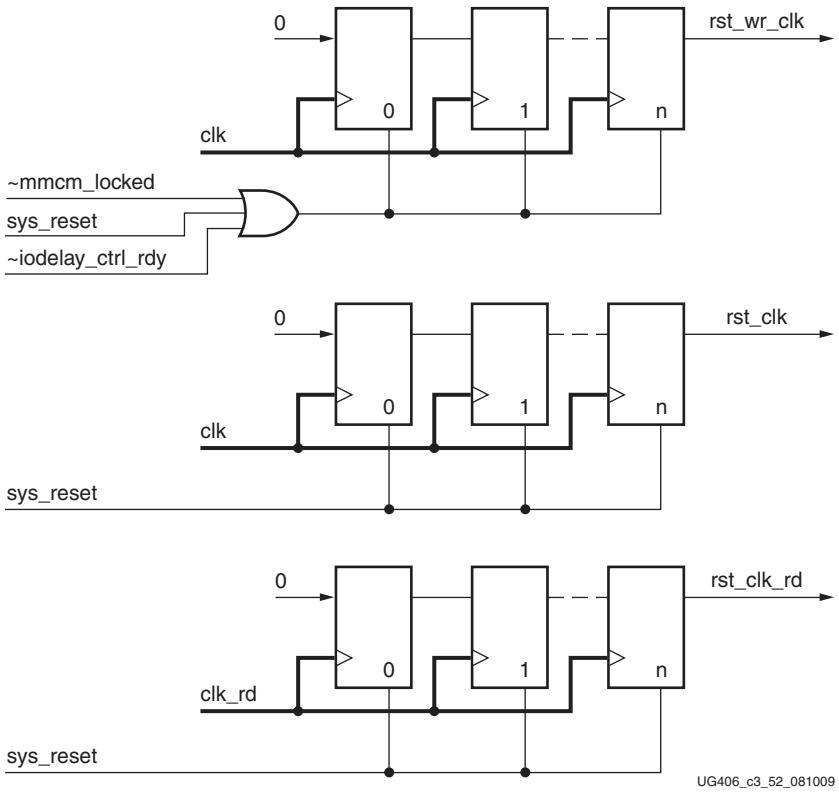


Figure 3-52: **Reset Synchronization**

IDELAYCTRL

An IDELAYCTRL is required in any bank that uses IODELAYs. IODELAYs are associated with the data group and address/control group. Any bank/clock region where these signals are used requires an IDELAYCTRL.

The MIG tool instantiates one IODELAYCTRL and then uses the IODELAY_GROUP attribute (see the `iodelay_ctrl.v/.vhd` module). Based on this attribute, the ISE software properly replicates IODELAYCTRLs as needed within the design.

Customizing the Core

The Virtex-6 FPGA RLDRAM II memory interface solution is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top level of the core, as shown in Table 3-15.

Table 3-15: **RLDRAM II Memory Solution Configurable Parameters**

Parameter	Value	Description
ADDR_WIDTH	19–22	This is the memory address bus width.
BANK_WIDTH	3	This is the memory bank address bus width.
DATA_WIDTH	9, 18, 36, 72	This is the memory data bus width.

Table 3-15: **RLDRAM II Memory Solution Configurable Parameters *(Cont'd)***

Parameter	Value	Description
QK_WIDTH	2 per x18/x36 device 1 per x9 device	This is the memory read clock bus width.
DK_WIDTH	2 per x36 device 1 per x9/x18 device	This is the memory write clock bus width.
NUM_DEVICES	1+	This is the number of memory devices.
BURST_LEN	4, 8	This is the memory data burst length.
CLK_PERIOD(1)	3752–11428	This is the FPGA fabric clock period (ps).
REFCLK_FREQ	200.0, 300.0	This is the reference clock frequency for IODELAYCTRLs. This value can be set to 200.0 for any speed grade device or 300.0 for a -2 or -3 device. For more information, refer to the IODELAYE1 Attribute Summary table in the *Virtex-6 FPGA SelectIO Resources User Guide* [Ref 4].
FIXED_LATENCY_MODE(2)	0, 1	This parameter indicates whether to use a predefined latency for a read response from the memory to the client interface. If set to 0, the minimum possible latency is used.
PHY_LATENCY(3)	19 to 30	This parameter indicates the desired latency through the PHY for a read from the time the read command is issued until the read data is returned on the client interface.
DM_PORT	"ON" "OFF"	This parameter is used to enable and disable the generation of the data mask ports.
MRS_CONFIG	1, 2, 3	This parameter is used to set the configuration setting in the RLDRAM II memory register.
MRS_ADDR_MUX	"ON" "OFF"	This parameter is used to set the address multiplexing setting in the RLDRAM II memory register.
MRS_DLL_RESET	"DLL_ON"	This parameter is used to set the DLL setting in the RLDRAM II memory register.
MRS_IMP_MATCH	"INTERNAL" "EXTERNAL"	This parameter is used to set the impedance setting in the memory register.
MRS_ODT	"ON" "OFF"	This parameter is used to set the ODT setting in the memory register.

Table 3-15: **RLDRAM II Memory Solution Configurable Parameters *(Cont'd)***

Parameter	Value	Description
MEM_TYPE	RLD2_CIO	This indicates the type of memory device attached.
SIM_INIT_OPTION	"SKIP_PU_DLY" "NONE"	This parameter is for simulation only. It is used to skip some power-up initialization time.
SIM_CAL_OPTION(4)	"FAST_CAL" "SKIP_CAL" "NONE"	This parameter is for simulation only. It is used to speed up calibration.
PHASE_DETECT	"ON" "OFF"	This is the phase detector and adjusts capture for voltage and temperature compensation. This parameter is currently set to "OFF" below 250 MHz and "ON" above 250 MHz.
DEVICE_ARCH	"virtex6"	This parameter is reserved for future use.
RST_ACT_LOW	0, 1	This parameter is used to set the polarity of sys_rst.
DEBUG_PORT	"ON" "OFF"	This parameter is always set to OFF in the `sim_tb_top` module of the `sim` folder, because debug mode is not required for functional simulations. Turning on the debug port allows for use with the VIO of the ChipScope analyzer. This allows the user to change the tap settings within the PHY based on those selected though the VIO.
IODELAY_GRP	Default: "IODELAY_MIG"	This is an ASCII character string to define an IODELAY group. It is usually used when multiple designs are implemented on the same FPGA.
IODELAY_HP_MODE	"ON" "OFF"	This parameter enables IODELAY High Performance Mode.
IBUF_LPWR_MODE	"ON" "OFF"	This parameter enables Input Buffer Low Performance Mode.
INPUT_CLK_TYPE	"DIFFERENTIAL" "SINGLE_ENDED"	This parameter indicates whether the system uses single-ended or differential system/reference clocks. Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, sys_clk_p/n must be used. For single-ended clocks, sys_clk must be used.

Table 3-15: **RLDRAM II Memory Solution Configurable Parameters *(Cont'd)***

Parameter	Value	Description
CLKFBOUT_MULT_F		This is the MMCM VCO multiplier. It is set by the MIG tool based on the frequency of operation.
CLKOUT_DIVIDE		This is the VCO output divisor for fast memory clocks. This value is set by the MIG tool based on the frequency of operation.
DIVCLK_DIVIDE		This is the MMCM VCO divisor. This value is set by the MIG tool based on the frequency of operation.

Notes:

1. The lower limit (maximum frequency) is pending characterization.
2. If desired, the physical layer can operate in fixed-latency mode. This is done by setting FIXED_LATENCY_MODE to 1. The latency is measured from when a read command is issued on the PHY to when the read response is present on the client interface. When set to 1, the desired latency should be set in the PHY_LATENCY parameter.
3. PHY_LATENCY indicates the desired latency from when a read command is issued on the PHY to when the read response is present on the client interface. This value is only used when the FIXED_LATENCY_MODE parameter is also set. If the value of this parameter is less than the minimum possible latency, the core issues an error through the error port in the top-level module, user_top. The best way to calculate the PHY_LATENCY value for a specific system is to run the system with FIXED_LATENCY_MODE set to 0 and record the results of the dbg_valid_lat debug signal. To guarantee the latency across multiple controllers, the largest value of dbg_valid_lat should be used as the value of PHY_LATENCY for all of the controllers in a system.
4. Core initialization during simulation can be greatly reduced by using SIM_CAL_OPTION. Two simulation modes are supported. Setting SIM_CAL_OPTION to "FAST_CAL" causes calibration to occur on only one bit per memory device. This value is then used across the remaining data bits. When SIM_CAL_OPTION is set to "SKIP_CAL", no calibration occurs, and the incoming clocks and data are assumed to be aligned. SIM_CAL_OPTION should be set to "NONE" for implementation or the core does not function properly.

Design Guidelines

While the Virtex-6 family offers many advanced I/O and clocking-related features to greatly simplify memory interface design, attention must still be paid to basic board design criteria for a reliable and high-performance interface.

Specifically, the source synchronous nature of the read and write path interfaces requires matched board trace lengths for the interface clock, data, and control signals. For example, the trace lengths of the RLDRAM II memory device input signals (rld_ck_p, rld_ck_n, rld_cs_n, rld_we_n, rld_ref_n, rld_a, rld_ba, rld_dk_p, rld_dk_n, rld_dq, and rld_dm) must be well matched to present the control, address, and data lines to the memory device with adequate setup and hold margins. The implementation of the physical interface ensures that these signals are center aligned to the rld_ck_p/n and rld_dk_p/n clock edges when leaving the FPGA device outputs. The board traces must ensure that this relationship continues to the memory device inputs.

Similarly, the RLDRAM II memory device output signals (rld_dq, rld_qk_p, and rld_qk_n) must have well-matched trace lengths for the signals to all arrive edge aligned at the inputs to the Virtex-6 device. This trace-length matching is critical to the implementation of the direct-clocking read data capture methodology. Any reasonable board design tool can match these traces within an acceptable tolerance with little effort.

Trace Length Requirements

The trace lengths described here are for high-speed operation and can be relaxed depending on the application's target bandwidth requirements. The package delay should be included when determining the effective trace length. The most accurate and recommended method for determining the delay is to use the L and C values for each pin from the IBIS models. The delay value is determined as the square root of (L × C). Alternatively, a less accurate method is to use the PARTGen utility. These internal delays can be found through the FPGA Editor tool. These rules indicate the maximum skew between RLDRAM II memory signals:

- The maximum skew between any DQ/DM and DK/DK# should be ±15 ps.
- The maximum skew between any DQ and its associated QK/QK# should be ±15 ps.
- The maximum skew between any address and control signals and the corresponding CK/CK# should be ±50 ps.
- The maximum skew between any DK/DK# and CK/CK# should be ±25 ps.

Pinout Requirements

Any changes to the pinout require changes within the UCF produced by the MIG tool. When altering the pinout, these rules must be taken into consideration:

- The clocks CK/CK# should be placed in the same bank as the address and control. For maximum performance, this bank should be located on an inner column. For design frequencies less than 400 MHz, address/control signals can be placed in the outer column. (The address and control can be shared across multiple devices.)
- All memory output signals should be placed in a clock region three banks high starting from the MMCM location horizontal row, one bank up and one bank down to minimize clock skew.
- QK/QK# must be placed on multi-region clock-capable I/Os.
- For design frequencies higher than 400 MHz, only inner column banks are allowed for data group selection. For design frequencies 400 MHz and below, both inner and outer column banks are allowed for data group selection.
- Inner and outer column banks that reside one row above, one row below, and on the same row of banks consisting of the CK[0] and CK#[0] pins are enabled for data group pin selection. This restriction is represented by a boundary box called the vicinity box.
- The read clocks QK/QK# and write clocks DK/DK# should be together in the same bank as data DQ and data mask DM.
- If QK/QK#, DK/DK#, DQ, and DM do not all fit into one bank, any remaining bits of DQ should be placed in an adjacent bank (one bank up or down) of the same column.

I/O Standards

The MIG tool generates the appropriate UCF for the core with SelectIO™ interface standards based on the type of input or output to the Virtex-6 FPGA. These standards should not be changed. Table 3-16 contains a list of the ports with the I/O standard used.

Table 3-16: **I/O Standards**

Signal	Direction	I/O Standard
rld_ck_p, rld_ck_n	Output	DIFF_HSTL_I
rld_dk_p, rld_dk_n	Output	DIFF_HSTL_I
rld_cs_n	Output	HSTL_I
rld_we_n	Output	HSTL_I
rld_ref_n	Output	HSTL_I
rld_a	Output	HSTL_I
rld_ba	Output	HSTL_I
rld_dm	Output	HSTL_I
rld_dq	Input/Output	HSTL_II_T_DCI
rld_qk_p, rld_qk_n	Input	DIFF_HSTL_II_T_DCI

Notes:

1. All signals operate at 1.5V.

Debugging Virtex-6 FPGA RLDRAM II Memory Designs

This section defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the memory interface design process.

Introduction

The RLDRAM II memory interfaces in the Virtex-6 FPGA simplify the challenges associated with memory interface design. However, every application environment is unique and proper due diligence is required to ensure a robust design. Careful attention must be given to functional testing through simulation, proper synthesis and implementation, adherence to PCB layout guidelines, and board verification through IBIS simulation and signal integrity analysis.

This section defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the design process. Details are provided on:

- Functional verification using the UNISIM simulation models
- Design implementation verification
- Board layout verification
- Using the RLDRAM II PHY to debug board-level issues
- General board-level debug techniques

The two primary issues encountered during verification of a memory interface are:

- Calibration not completing properly
- Data corruption during normal operation

Problems might be seen in simulation, hardware, or both due to various root cause causes. Figure 3-53 shows the overall flow for debugging problems associated with these two general types of issues.

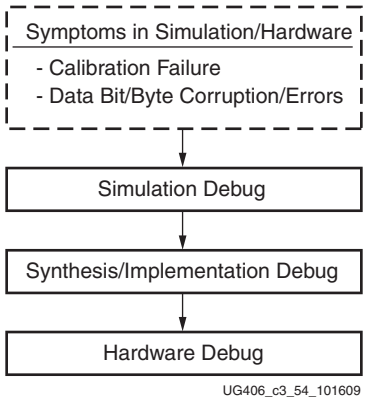


Figure 3-53: **Virtex-6 FPGA RLDRAM II Memory Debug Flowchart**

Debug Tools

Many tools are available to debug memory interface design issues. This section indicates which resources are useful for debugging a given situation.

Example Design

Generation of an RLDRAM II design using the MIG tool produces an example design and a user design. The example design includes a synthesizable test bench that has been fully verified in simulation and hardware. This example design can be used to observe the behavior of the MIG tool design and can also aid in identifying board-related problems. Refer to Quick Start Example Design, page 278 for complete details on the example design. This section also describes using the example design to verify setup of a proper simulation environment and to perform hardware validation.

Debug Signals

The MIG tool includes a Debug Signals Control option on the FPGA Options screen. Enabling this feature allows calibration, tap delay, and read data signals to be monitored using the ChipScope analyzer. Selecting this option port maps the debug signals to VIO modules of the ChipScope analyzer in the design top module. Refer to Getting Started with the CORE Generator Software, page 11 for details on enabling this debug feature.

ChipScope Pro Tool

The ChipScope Pro tool inserts logic analyzer, bus analyzer, and virtual I/O software cores directly into the design. The ChipScope Pro tool allows the user to set trigger conditions to capture application and the MIG tool signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool [Ref 6].

Simulation Debug

Figure 3-54 shows the debug flow for simulation.

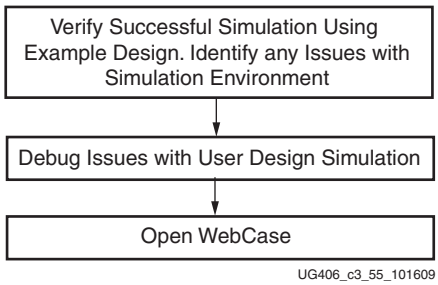


Figure 3-54: **Simulation Debug Flowchart**

Verifying the Simulation Using the Example Design

The example design generated by the MIG tool includes a simulation test bench and parameter file based on memory selection in the MIG tool, and a ModelSim `.do` script file. The MIG tool does not provide an RLDRAM II memory model so that must be provided and added to the simulation by the user. Refer to Quick Start Example Design, page 278 for detailed steps on running the example design simulation. Successful completion of this example design simulation verifies a proper simulation environment. This shows that the simulation tool and Xilinx libraries are set up correctly. For detailed information on setting up Xilinx libraries, refer to COMPXLIB in the *Command Line Tools User Guide* [Ref 7] and the *Synthesis and Simulation Design Guide* [Ref 8]. For simulator tool support, refer to the *Virtex-6 FPGA Memory Interface Solutions Data Sheet* [Ref 9].

A working example design simulation completes memory initialization and runs traffic in response to the test bench stimulus. Successful completion of memory initialization and calibration results in the assertion of the cal_done signal. When this signal is asserted, the test bench takes control and begins executing writes and reads according to its parameterization.

Table 3-17 and Table 3-18 show the signals and parameters of interest, respectively, during simulation.

Table 3-17: **Signals of Interest During Simulation**

Signal Name	Usage
cal_done	This signal indicates completion of calibration.
compare_error	This signal indicates a mismatch between the data written from the UI and data received during a read on the UI. This signal is a part of the example design. A single error asserts this signal and is held until the design is reset.
cmp_err	This signal indicates a mismatch between the data written from the UI and the data received during a read on the UI. This signal is a part of the example design. This signal is asserted each time a data mismatch occurs.
user_cmd_en	This signal indicates if a command is valid.
user_rd_cmd	This signal is asserted if the command is a read operation request.
user_addr	This is the address location for the current command.
user_ba	This is the bank address location for the current command.

Table 3-17: **Signals of Interest During Simulation**

Signal Name	Usage
user_wr_en	This signal is asserted when the user_wr_data is valid. This signal is necessary for write commands.
user_wr_data	This signal is the write data provided for write commands.
user_wr_dm	This signal is the data mask for masking off, and not writing, all of the data in a given write transaction.
user_afifo_empty	This signal indicates that the command and address FIFO is empty.
user_afifo_full	This signal indicates that the command and address FIFO is full. When asserted additional commands and data is not accepted.
user_wdfifo_empty	This signal indicates that the write data FIFO is empty.
user_wdfifo_full	This signal indicates that the write data FIFO is full. When asserted additional Write data is not accepted.
user_rd_valid	Asserted when user_rd_data is valid.
user_rd_data	Read data returned from the memory as a result of a read command.

Table 3-18: **Parameters of Interest During Simulation**

Signal Name	Usage
SIM_INIT_OPTION	This parameter sets the simulation initialization procedure.
SIM_CAL_OPTION	This parameter sets the simulation calibration procedure.

When SIM_INIT_OPTION is set to SKIP_PU_DLY, and SIM_CAL_OPTION is set to FAST_CAL, the MIG tool design executes an abbreviated calibration sequence. For the design to properly initialize and calibrate the full memory array in hardware, the top-level MIG tool design file (`example_top.v/vhd`) cannot use any abbreviated value for these parameters.The MIG tool output properly sets the abbreviated values in the test bench and the full range values in the top-level design module.

Figure 3-55 shows a high-level view of a successful simulation using the provided example design with the abbreviated simulation parameters set, as described in Table 3-17, page 303. The simulation can be divided into these main sections: Memory Initialization, Calibration, and Test Bench.

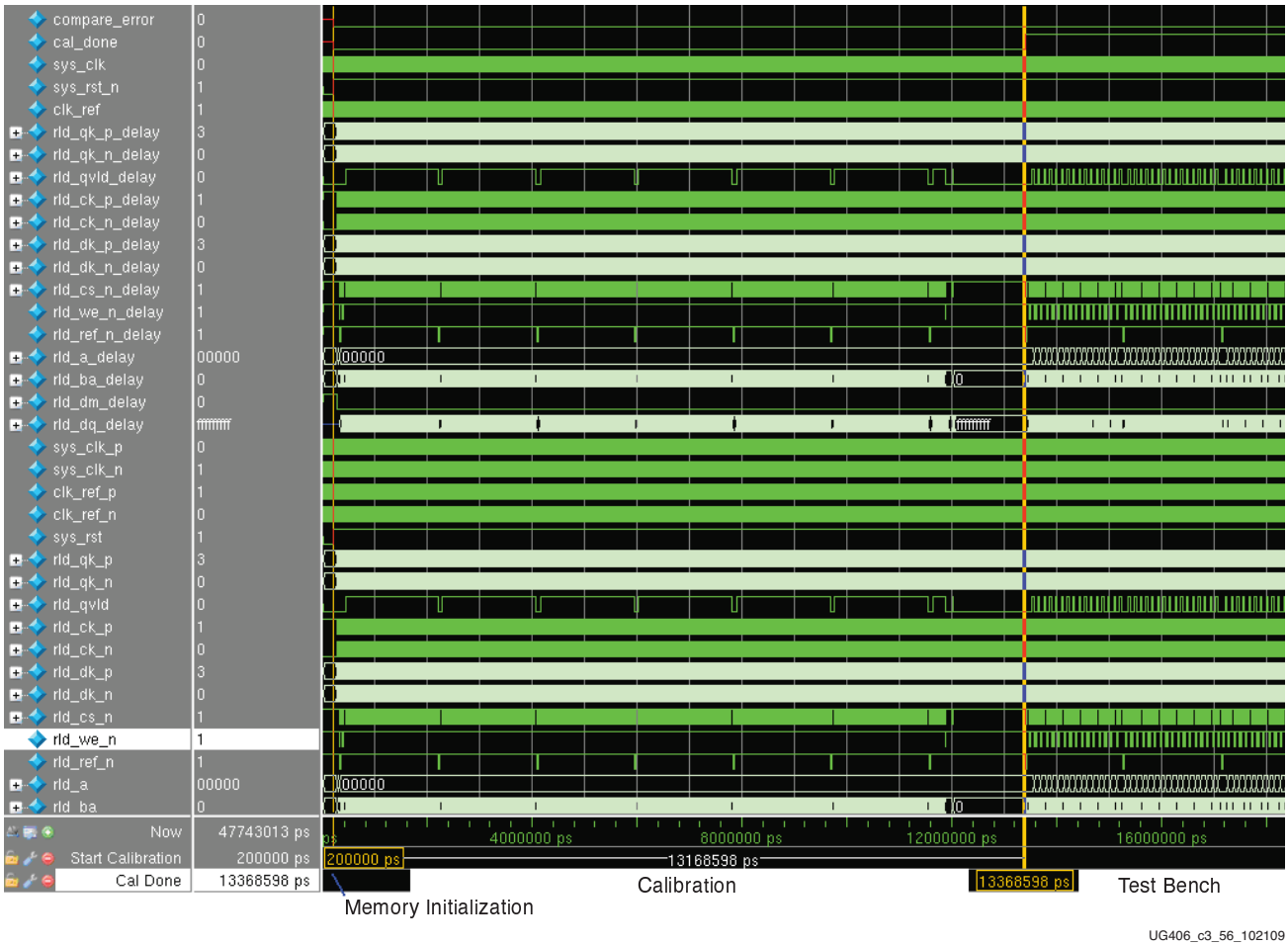


Figure 3-55: **Waveforms and Simulation Transcripts Showing Successful Example Design Completion**

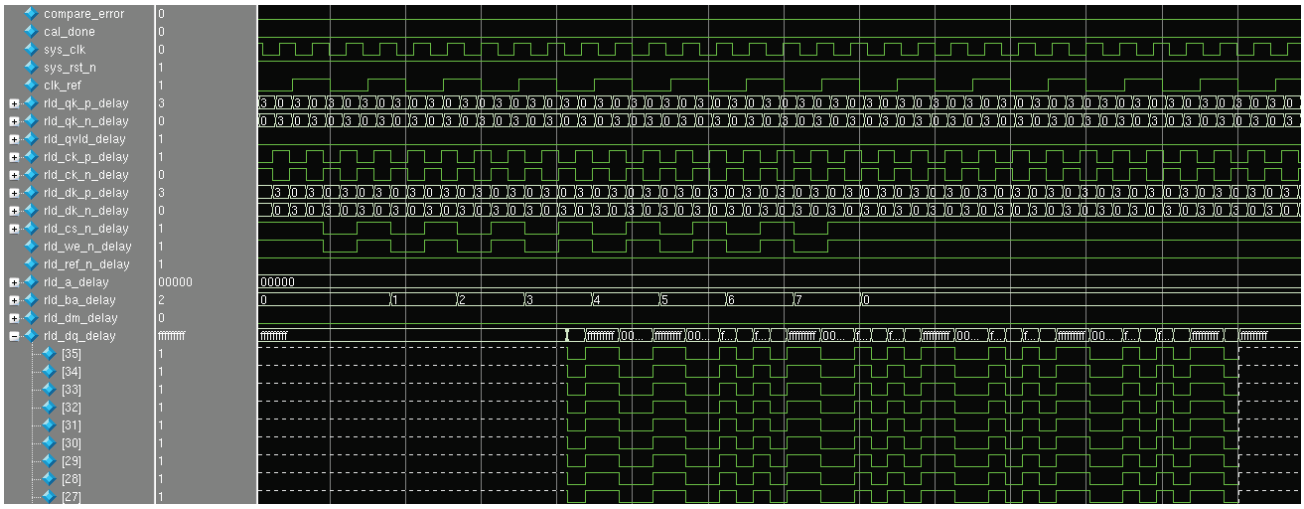
Memory Initialization

The memory initialization skips the initial 200 µs delay but completes all remaining steps as defined by the Micron RLDRAM II specification.

Calibration

Calibration completes read leveling, write calibration, and read enable calibration. This is completed over two stages. This sequence successfully completes when the cal_done signal asserts. For more details, refer to PHY, page 102.

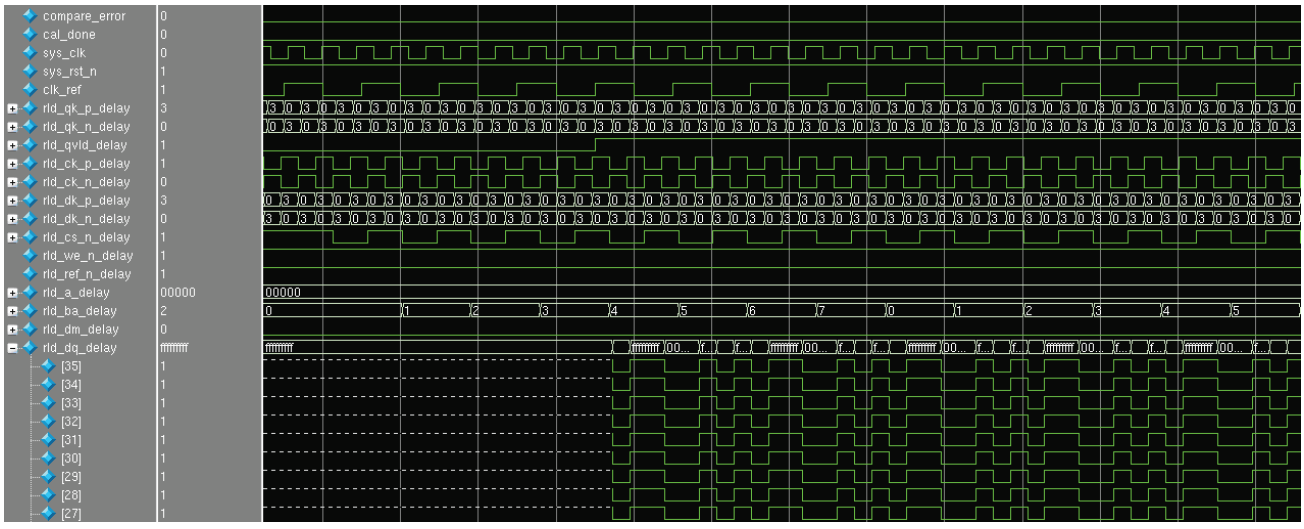
The first stage performs per-byte read leveling calibration. The data pattern used during this stage is `00ff00ff00ffff00` and is first written to the memory as shown in Figure 3-56.



UG406_c3_57_102109

Figure 3-56: **Writes for First Stage Read Calibration**

This pattern is then continuously read back while the per-bit calibration is completed, as shown in Figure 3-57.



UG406_c3_58_102109

Figure 3-57: **Reads for First Stage Read calibration**

The second stage performs a read enable calibration. The data pattern used during this stage is AAAA. The data pattern is first written to the memory, as shown in Figure 3-58.

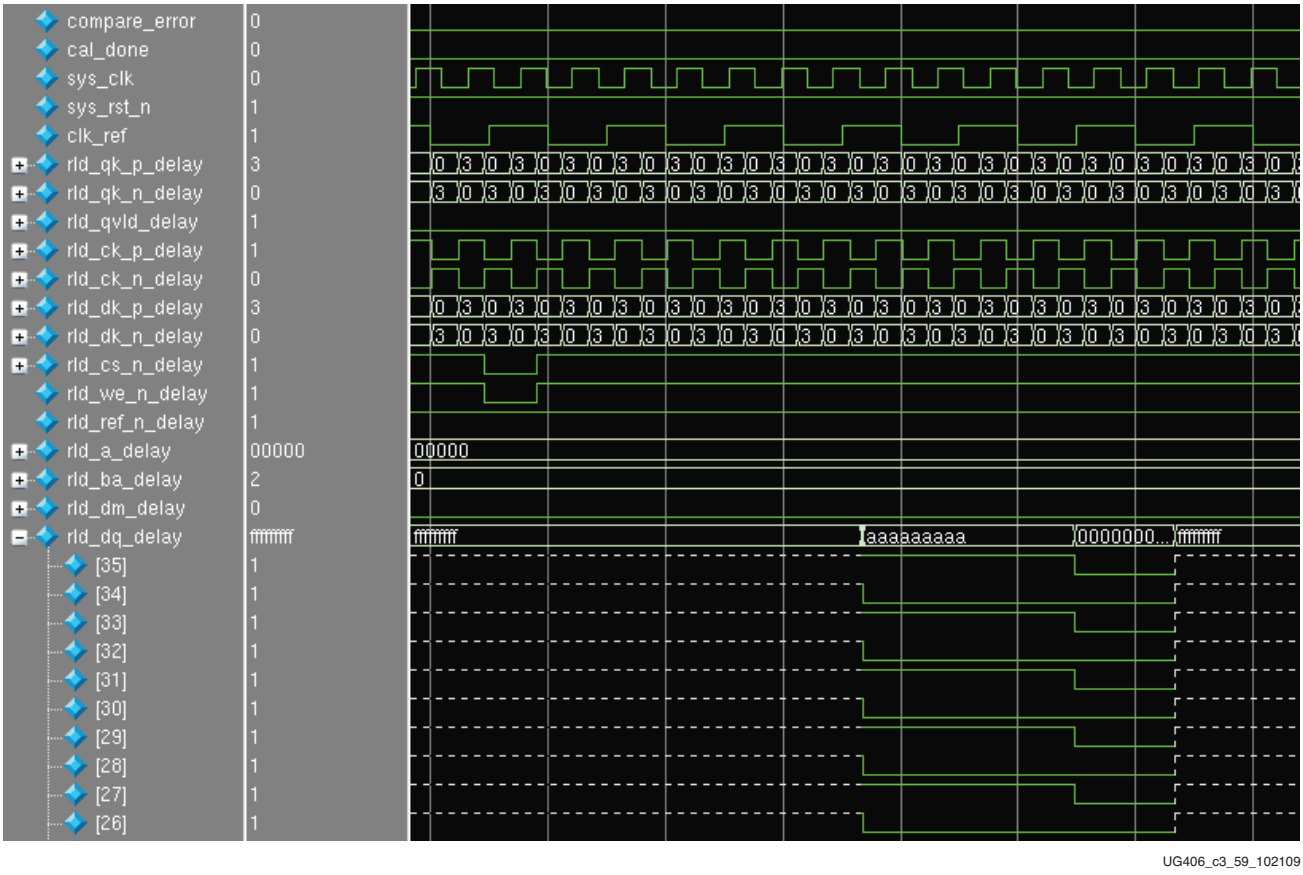


Figure 3-58: **Write for Second Stage Read Calibration**

The same location is read some time later for read enable calibration. An additional read is performed so the read bus is driven to a different value. This is mostly required in

hardware to ensure that the read calibration can distinguish the correct data pattern, as shown in Figure 3-59.

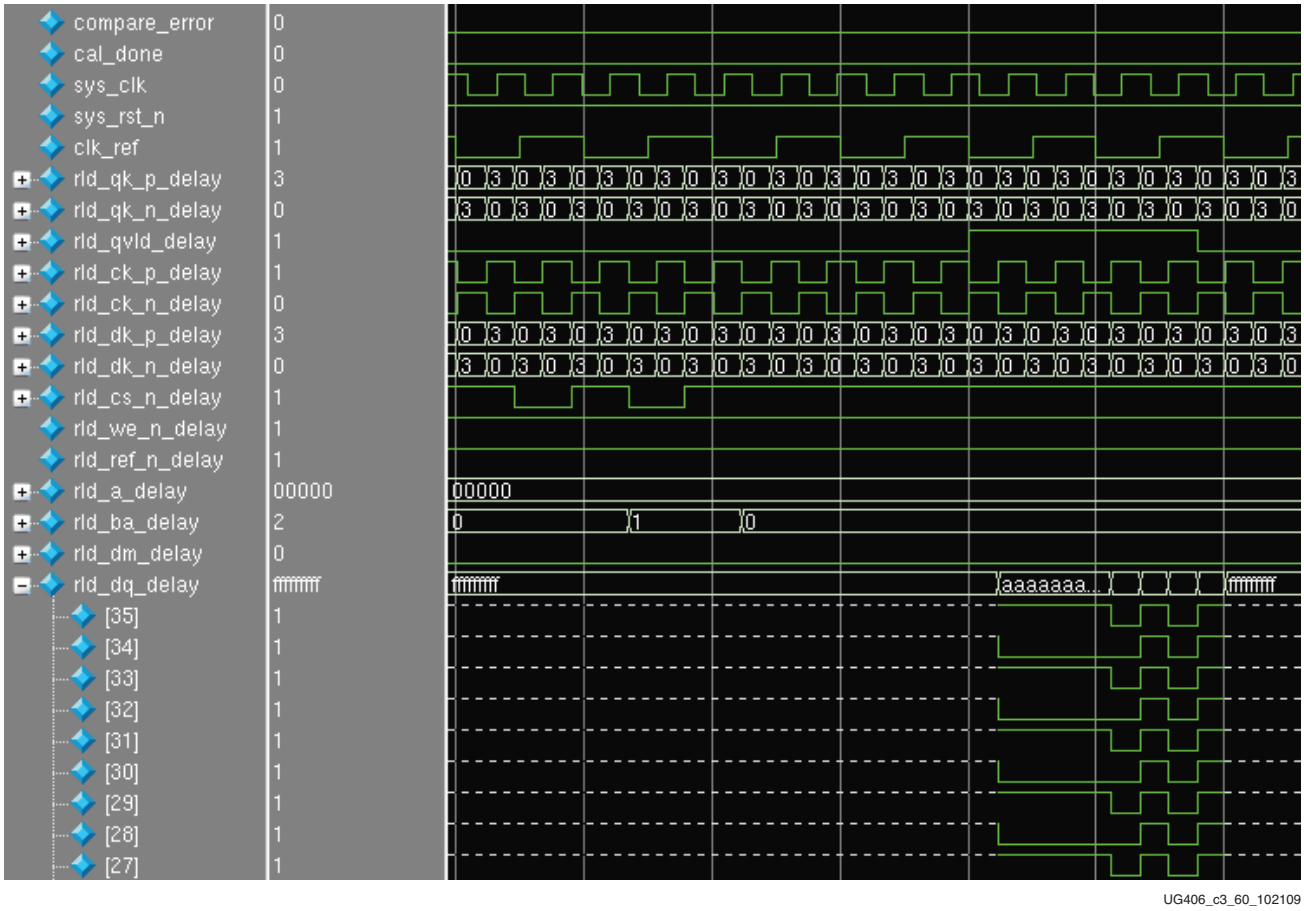


Figure 3-59: **Reads for Second Stage Read Calibration**

After second stage calibration completes, cal_done asserts signifying successful completion of the entire calibration process.

Test Bench

After cal_done asserts, the test bench takes control writing to and reading from the memory. The data written is compared to the data read back. Any mismatches trigger an

assertion of the error signal. Figure 3-60 below shows successful implementation of the test bench with no assertions on error.

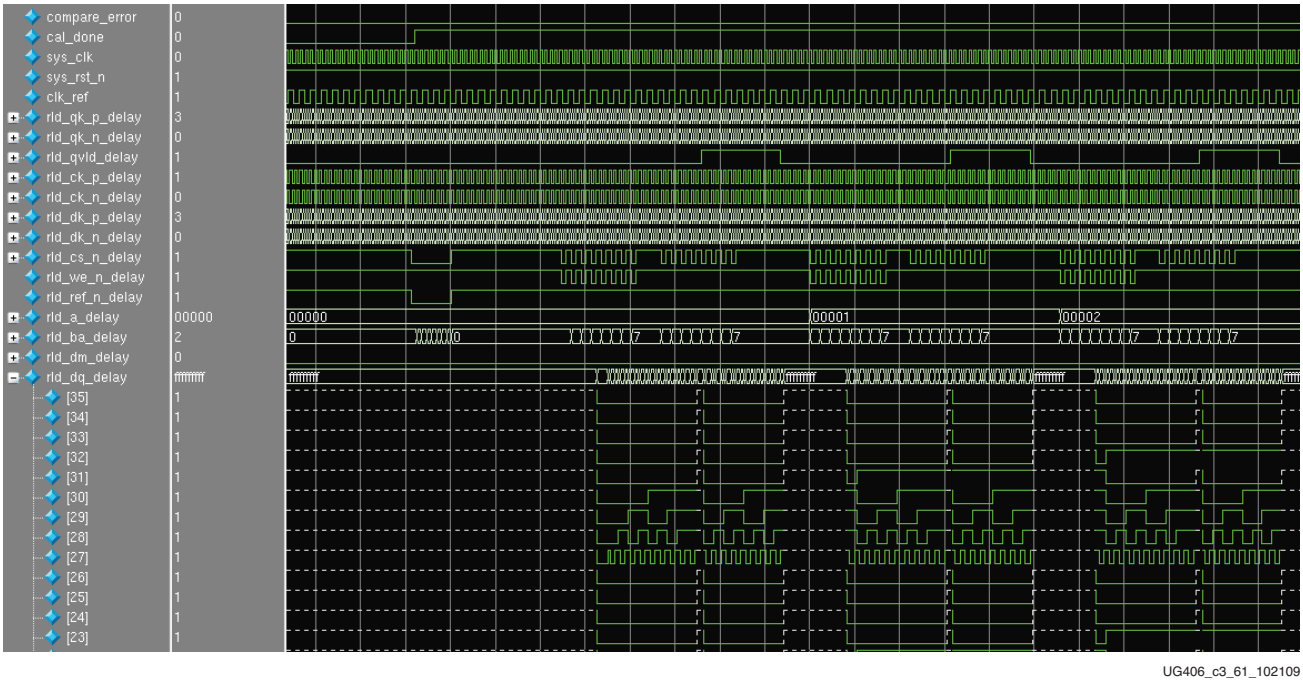


Figure 3-60: **Test Bench Operation After Completion of Calibration**

Debug Issues with User Design Simulation

After the simulation environment and parameter settings are verified by successful simulation of the example design, issues with the user design simulation can be investigated. Because the environment and parameters are verified to work properly, calibration in the user design completes without error as long as no RTL changes exist.

Data Errors

Issues that might be seen with user design simulation exist within the generation of user writes and reads. Thus, it is crucial to understand how to drive the UI to properly send writes and reads. For more information, refer to User Interface, page 68 and Interfacing to the Core, page 113.

Proper Write and Read Commands

When sending write and read commands, the user must properly assert and deassert the corresponding UI inputs. Refer to User Interface, page 68 and Interfacing to the Core, page 113 for full details. The test bench design provided within the example design can be used as a further source of proper behavior on the UI.

To debug data errors on the RLDRAM II interface, UI signals must be pulled into the simulation waveform.

In the ModelSim Instance window, highlight **u_ip_top** to display the necessary UI signals in the Objects window, as shown in Figure 3-61. Highlight the user interface signals noted in Table 3-17, page 303, right-click, and select **Add** → **To Wave** → **Selected Signals**.

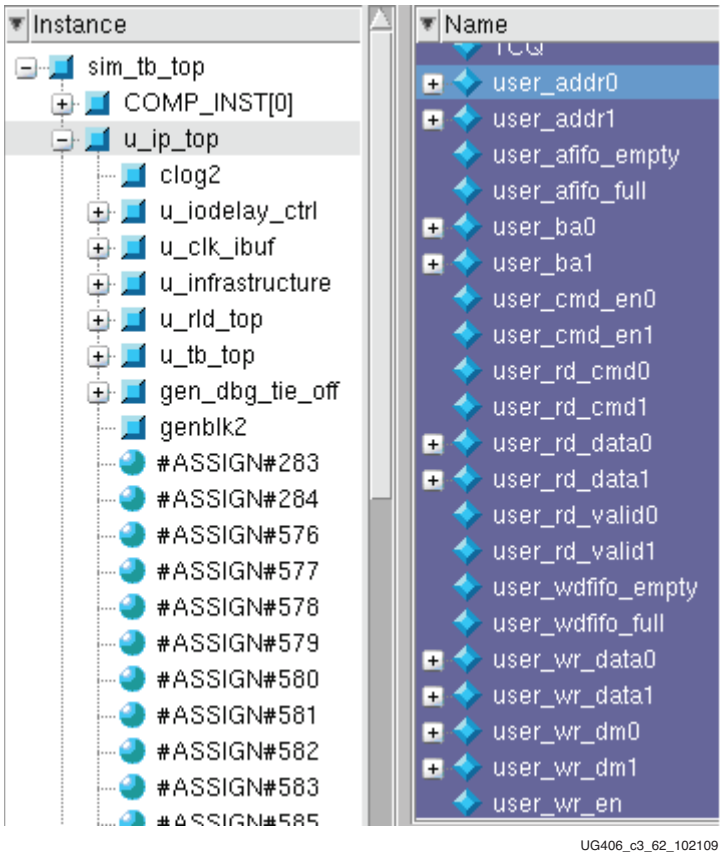


Figure 3-61: **ModelSim Instance Window**

The waveforms shown in Figure 3-63 through Figure 3-65 provide examples of a write and read on both the UI and RLDRAM II interface.

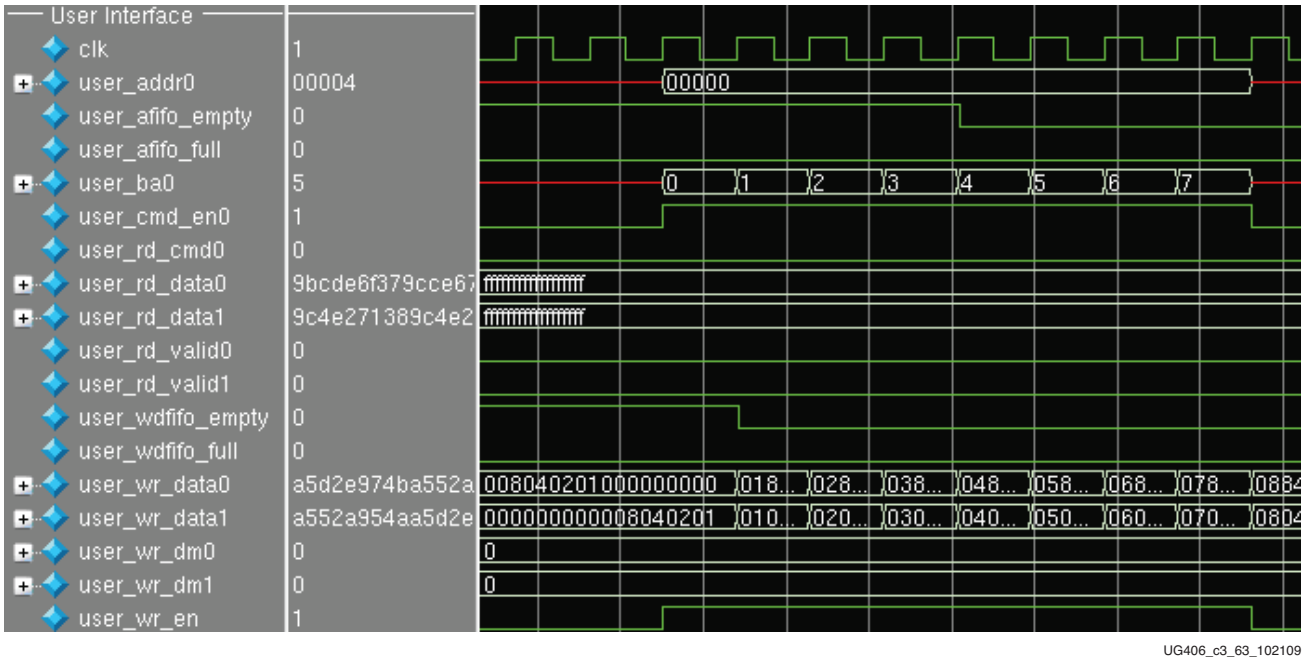


Figure 3-62: **User Interface Write**

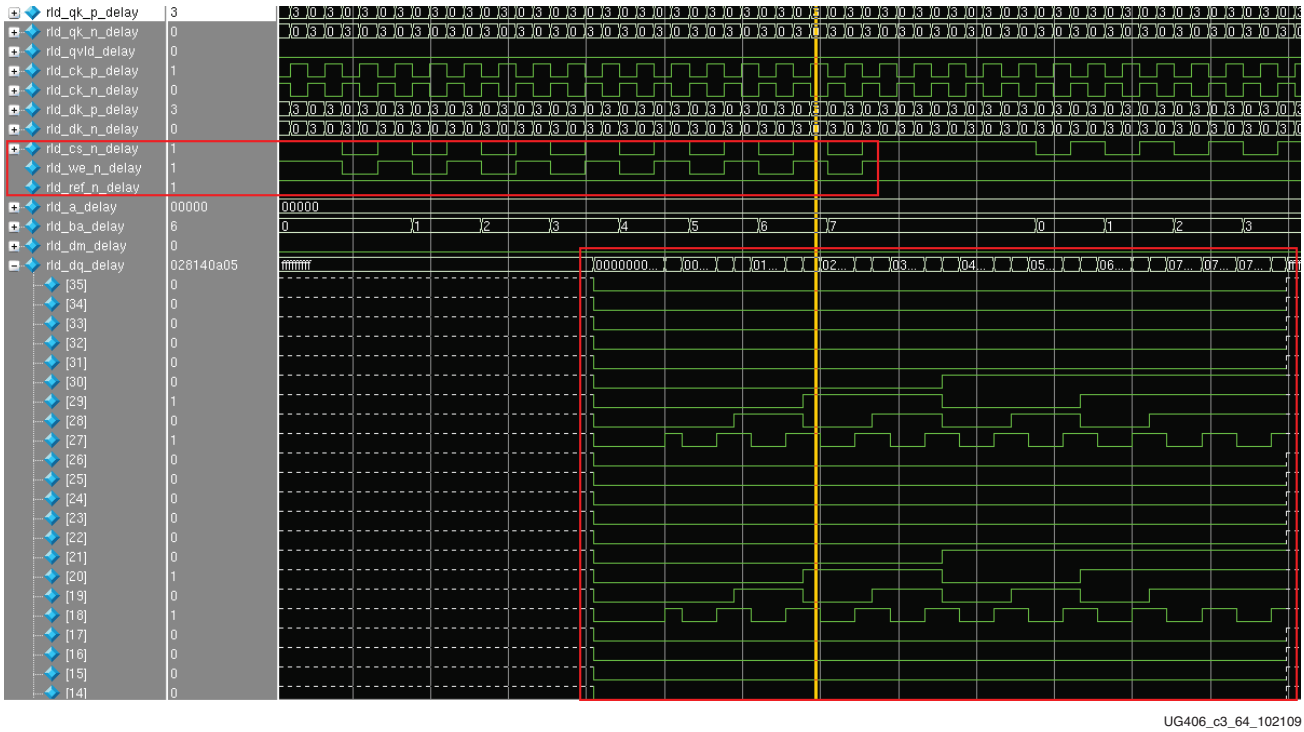


Figure 3-63: **RLDRAM II Interface Write**

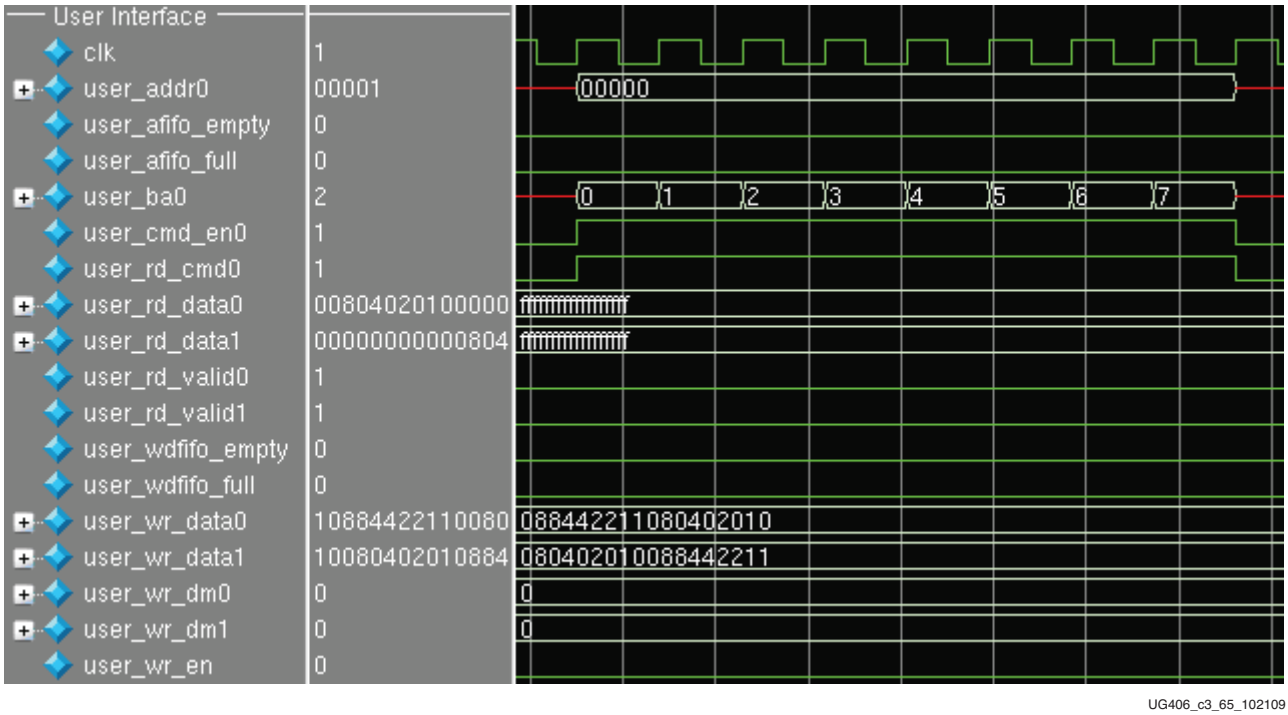


Figure 3-64: **User Interface Read**

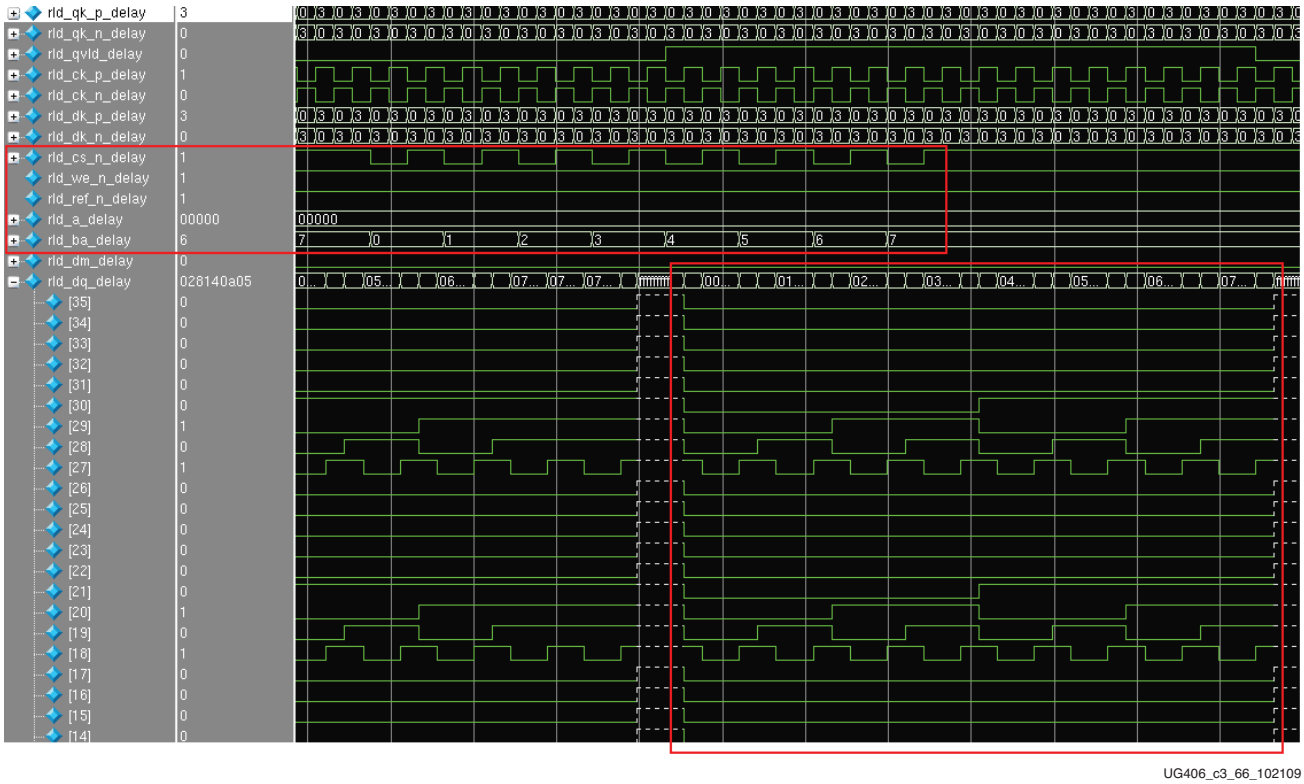


Figure 3-65: **RLDRAM II Interface Read**

Synthesis and Implementation Debug

Figure 3-66 shows the debug flow for synthesis and implementation.

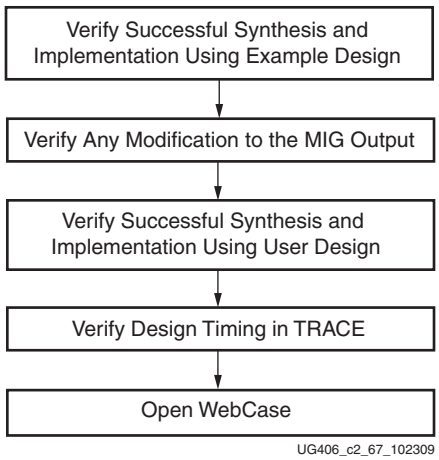


Figure 3-66: **Synthesis and Implementation Debug Flowchart**

Verify Successful Synthesis and Implementation

The example design and user design generated by the MIG tool include synthesis/implementation script files and user constraint files (`.ucf`). These files should be used to properly synthesize and implement the targeted design and generate a working bitstream. The synthesis/implementation script file, called `ise_flow.bat`, is located in both `example_design/par` and `user_design/par` directories. Execution of this script runs either the example design or the user design through synthesis, translate, MAP, PAR, TRACE, and BITGEN. The options set for each of these processes are the only options that have been tested with the RLDRAM II MIG tool designs. A successfully implemented design completes all processes with no errors (including zero timing errors).

Verify Modifications to the MIG Tool Output

The MIG tool allows the user to select the FPGA banks for the memory interface signals. Based on the banks selected, the MIG tool outputs a UCF with all required location constraints. This file is located in both `example_design/par` and `user_design/par` directories and should not be modified.

The MIG tool outputs open source RTL code parameterized by top-level HDL parameters. These parameters are set by the MIG tool and should not be modified manually. If changes are required, such as decreasing or increasing the frequency, the MIG tool should be rerun to create an updated design. Manual modifications are not supported and should be verified independently in behavioral simulation, synthesis, and implementation.

Identifying and Analyzing Timing Failures

The MIG tool RLDRAM II designs have been verified to meet timing using the example design across a wide range of configurations. However, timing violations might occur, such as when integrating the MIG tool design with the user's specific application logic. Any timing violations that are encountered must be isolated. The timing report output by TRACE (`.twx/.twr`) should be analyzed to determine if the failing paths exist in the MIG tool RLDRAM II design or the UI (backend application) to the MIG tool design. If failures

are encountered, the user must ensure the build options (that is, XST, MAP, PAR) specified in the `ise_flow.bat` file are used.

If failures still exist, Xilinx has many resources available to aid in closing timing. The PlanAhead™ tool [Ref 10] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 11] provides valuable information on all available Xilinx constraints.

Hardware Debug

Figure 3-67 shows the debug flow for hardware.

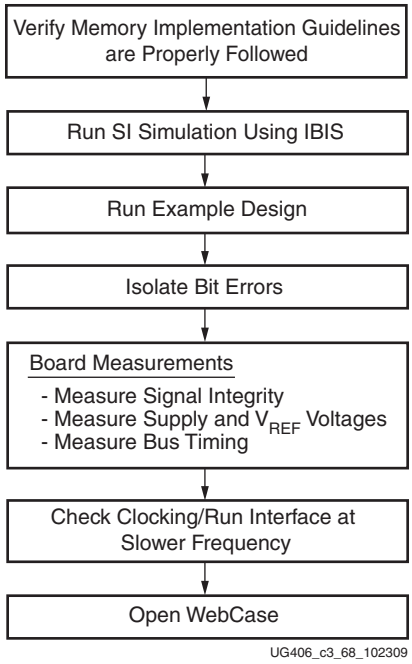


Figure 3-67: **Hardware Debug Flowchart**

Verify Design Guidelines

See Design Guidelines, page 299 for specifications on termination, I/O standards, and trace matching. The guidelines provided therein are specific to the RLDRAM II memory. It is important to verify that these guidelines have been referred to during board layout. Failure to follow these guidelines and/or modifications to a MIG tool provided pinout, or both, can result in problematic behavior in hardware as discussed in this debugging section.

Clocking

The external clock source should be measured to ensure frequency, stability (jitter), and usage of the expected FPGA pin. The designer must ensure that the design follows all clocking guidelines. If clocking guidelines have been followed, the interface should be run at a slower speed. Not all designs or boards can accommodate slower speeds. Lowering the frequency increases the marginal setup or hold time, or both, due to PCB trace mismatch, poor signal integrity, or excessive loading. When lowering the frequency, the MIG tool should be rerun to regenerate the design with the lower clock frequency. Portions

of the calibration logic are sensitive to the CLK_PERIOD parameter; thus manual modification of the parameter is discouraged.

Verify Board Pinout

The user should ensure that the pinout provided by the MIG tool is used without modification. Then, the board schematic should be compared to the `<design_name>.pad` report generated by PAR. This step ensures that the board pinout matches the pins assigned in the implemented design.

Run Signal Integrity Simulation with IBIS Models

To verify that board layout guidelines have been followed, signal integrity simulations must be run using the I/O buffer information specification (IBIS). These simulations should always be run for both pre-board and post-board layouts. The purpose of running these simulations is to confirm the signal integrity on the board.

The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 12] can be used as a guideline. This chapter provides a detailed look at signal integrity correlation results for the ML561 board. It can be used as an example for signal integrity analysis. It also provides steps to create a design-specific IBIS model to aid in setting up the simulations. While this guide is specific to Virtex-5 devices and the ML561 development board, the principles therein can be applied to Virtex-6 FPGA MIG tool designs.

Run the Example Design

The MIG tool provided example design is a fully verified design that can be used to test the memory interface on the board. It rules out any issues with the backend logic interfacing with the MIG tool core. In addition, the test bench provided by the MIG tool can be modified to send out different data patterns that test different board-level concerns.

Debugging Common Hardware Issues

When calibration failures and data errors are encountered in hardware, the ChipScope analyzer should be used to analyze the behavior of MIG tool core signals. For detailed information about using the ChipScope analyzer, refer to the *ChipScope Pro 11.1 Software and Cores User Guide* [Ref 14].

A good starting point in hardware debug is to load the provided example_design onto the board in question. This is a known working solution with a test bench design that checks for data errors. This design should complete successfully with the assertion of cal_done and no assertions of compare_error. Assertion of cal_done signifies successful completion of calibration while no assertions of compare_error signifies that the data is written to and read from the memory compare with no data errors.

The cmp_err signal can be used to indicate if a single error was encountered or if multiple errors are encountered. With each error encountered, cmp_err is asserted so that the data can be manually inspected to help track down any issues.

Isolating Bit Errors

An important hardware debug step is to try to isolate when and where the bit errors occur. Looking at the bit errors, these should be identified:

- Are errors seen on data bits belonging to certain CQ clock groups?
- Are errors seen on accesses to certain addresses of memory?

- Do the errors only occur for certain data patterns or sequences?

  This can indicate a shorted or open connection on the PCB. This can also indicate an SSO or crosstalk issue.

It might be necessary to isolate whether the data corruption is due to writes or reads. This case can be difficult to determine because if writes are the cause, read back of the data is bad as well. In addition, issues with control or address timing affect both writes and reads. Some experiments that can be tried to isolate the issue are:

- If the errors are intermittent, have the design issue a small initial number of writes, followed by continuous reads from those locations. If the reads intermittently yield bad data, there is a potential read problem.
- Increase the time between write and read transactions to check if collisions might be occurring on the data bus.
- Check/vary only write timing:
  - Check that the external termination resistors are populated on the PCB or if the ODT mode is set properly.
  - Use ODELAY to vary the phase of D relative to the K clocks.
- Vary only read timing:
  - Check the IDELAY values after calibration. Look for variations between IDELAY values. IDELAY values should be very similar for DQs in the same QK group.
  - Vary the IDELAY taps after calibration for the bits that are returning bad data. This affects only the read capture timing.

Debugging the Core

The Debug port is a set of input and output signals that either provide status (outputs) or allow the user to make adjustments as the design is operating (inputs). When generating the RLDRAM II design through the MIG tool, an option is provided to turn the Debug Port on or off. When the Debug port is turned off, the outputs of the debug port are still generated but the inputs are ignored. When the Debug port is turned on, the inputs are valid and must be driven to a logical value. Driving the signals incorrectly on the debug port might cause the design to fail or have less read data capture margin.

When running the core in hardware, a few key signals should be inspected to determine the status of the design. The dbg_phy_status bus described in Table 3-19 consists of status bits for various stages of calibration. Checking the dbg_phy_status bus gives initial information that can aid in debugging an issue that might arise, determining which portion of the design to look at, or looking for some common issues.

Table 3-19: **Physical Layer Simple Status Bus Description**

Debug Port Signal	Name	Description	If Problems Arise
dbg_phy_status[0]	iodelay_ctrl_rdy	IODELAY blocks are ready to be used.	Check that the IODELAY clock is supplied properly with the expected frequency.
dbg_phy_status[1]	mmcm_locked	MMCM has locked and is generating the system clocks.	Check that the system clock is supplied properly with the expected frequency. Check the polarity of the reset.

Table 3-19: **Physical Layer Simple Status Bus Description**

Debug Port Signal	Name	Description	If Problems Arise
dbg_phy_status[2]	init_done	RLDRAM 2 initialization sequence is complete.	N/A
dbg_phy_status[3]	cal_stage1_start	Stage 1 read calibration start signal.	Verify the expected data is being returned from the memory
dbg_phy_status[4]	cal_stage2_start	Stage 2 read calibration start signal.	Check the IODELAY values set for stage 1 read calibration and check the data for stage 2.
dbg_phy_status[5]	dbg_pd_calib_start	Phase detector calibration start signal.	N/A
dbg_phy_status[6]	dbg_pd_calib_done	Phase detector calibration complete.	Check the system clock frequency as the phase detector has a lower frequency limit. (The phase detector should be off below 250 MHz.)
dbg_phy_status[7]	cal_done	Calibration complete.	N/A

Notes:

1. N/A indicates that as long as previous stages have completed this stage is also completed.

The results of read calibration are provided as part of the Debug port as various output signals. These signals can be used to capture and evaluate the results of read calibration. Read calibration uses the IODELAY to center the capture clock in the data valid window for captured data. The algorithm shifts the IODELAY values and looks for edges of the data valid window on a per-bit basis as part of the calibration procedure.

DEBUG_PORT Signals

The top-level file, ip_top, provides several output signals that can be used to debug the core. Each debug signal output is prefixed with dbg_. These signals are listed in Table 3-20, page 318 along with descriptions of the data they provide.

Table 3-20: **Core Debug Signals**

Signal	Valid Clock Domain	Direction	Description
dbg_phy_cmd_n[5:0]	clk	Output	These are internal command signals sent to the IOBs that are useful for controller or initialization verification: [0] = ref[1] [1] = we[1] [2] = cs[1] [3] = ref[0] [4] = we[0] [5] = cs[0]
dbg_phy_addr[ADDR_WIDTH × 2 – 1:0]	clk	Output	This is the internal address sent to the IOBs (rise0, rise1).
dbg_phy_ba[BANK_WIDTH × 2 – 1:0]	clk	Output	This is the internal bank address sent to the IOBs (rise0, rise1).
dbg_phy_wr_data[DATA_WIDTH × 4 – 1:0]	clk	Output	This is the write data sent to the IOBs (rise, fall).
dbg_phy_init_sm[31:0]	clk	Output	This is the general-purpose initialization debug port.
dbg_rd_stage1_cal[255:0]	clk	Output	These are the debug signals for stage 1 calibration. See Table 2-19, page 238 for a signal map.
dbg_pd_calib_start[QK_WIDTH – 1:0]	clk	Output	This signal indicates that the phase detector is starting.
dbg_pd_calib_done[QK_WIDTH – 1:0]	clk	Output	This signal indicates that the phase detector is complete.
dbg_phy_status[7:0]	clk	Output	This is the general debug status. It is made up of these signals: [0] = iodelay_ctr_rdy [1] = mmcm_locked [2] = init_done [3] = cal_stage1_start [4] = cal_stage2_start [5] = dbg_pd_calib_start [6] = dbg_pd_calib_done [7] = cal_done
dbg_cq_tapcnt[TAP_BITS × QK_WIDTH – 1:0]	clk	Output	This is a bus made up of IODELAY tap counts for various CQ bits.
dbg_cqn_tapcnt[TAP_BITS × QK_WIDTH – 1:0]	clk	Output	This is a bus made up of IODELAY tap counts for various CQ# bits.
dbg_q_tapcnt[TAP_BITS × DATA_WIDTH – 1:0]	clk	Output	This is a bus made up of IODELAY tap counts for various DQ bits.

Table 3-20: **Core Debug Signals *(Cont'd)***

Signal	Valid Clock Domain	Direction	Description
dbg_inc_cq_all	clk	Input	This signal increments taps on all CQ clock bits.
dbg_inc_cqn_all	clk	Input	This signal increments taps on all CQ# clock bits.
dbg_inc_q_all	clk	Input	This signal increments taps on all data Q bits.
dbg_dec_cq_all	clk	Input	This signal decrements taps on all CQ clock bits.
dbg_dec_cqn_all	clk	Input	This signal decrements taps on all CQ# clock bits.
dbg_dec_q_all	clk	Input	This signal decrements taps on all data Q bits.
dbg_inc_cq	clk	Input	This signal increments the select clock CQ bit.
dbg_inc_cqn	clk	Input	This signal decrements the select clock CQ# bit.
dbg_inc_q	clk	Input	This signal increments the select data Q bit.
dbg_dec_cq	clk	Input	This signal decrements the select clock CQ bit.
dbg_dec_cqn	clk	Input	This signal decrements the select clock CQ# bit.
dbg_dec_q	clk	Input	This signal decrements the selected DQ IODELAY tap value.
dbg_sel_cq[CQ_BITS – 1:0]	clk	Input	This is the selected QK bit to modify.
dbg_sel_cqn[CQ_BITS – 1:0]	clk	Input	This signal is not used.
dbg_sel_q[Q_BITS – 1:0]	clk	Input	This is the selected DQ bit to modify.
dbg_pd_off	clk	Input	This input should be driven High to disable the read phase detector.
dbg_clear_error	clk	Input	This is the signal to clear test bench errors. It is useful in checking for single bit errors or measuring a read window.

Additional read path debug signals are listed in Table 3-21.

Table 3-21: **Additional Read Path Debug Signals**

Signal	Module	Valid Clock Domain	Description
dbg_valid_lat[4:0]	phy_read_vld_gen	clk	This is the latency in cycles of the delayed read command.
dbg_cq_num[CQ_BITS – 1:0]	phy_read_stage1_cal	clk	This signal indicates the current CQ/CQ# being calibrated.
dbg_q_bit[Q_BITS – 1:0]	phy_read_stage1_cal	clk	This signal indicates the current Q being calibrated.
dbg_error_max_latency[QK_WIDTH – 1:0]	phy_read_stage2_cal	clk	This signal indicates that the latency could not be measured before the counter overflowed. There is one error bit for each device.
dbg_error_adj_latency	phy_read_stage2_cal	clk	This signal indicates that the target PHY_LATENCY could not be achieved.
dbg_rd_stage2_cal[127:0]	phy_read_stage2_cal	clk	This is a general-purpose stage 2 calibration bus.
dbg_phase[QK_WIDTH – 1:0]	phy_read_data_align	clk	This signal indicates whether or not to realign the data to correct the CLK/CLKB relationship relative to the CLKDIV in the ISERDES. There is one dbg_phase bit per QK clock pair.
dbg_inc_latency[QK_WIDTH – 1:0]	phy_read_dcb	clk	This signal indicates that latency through the DCB should be increased. There is one increment latency signal for each device, and they are all concatenated together.
dbg_dcb_wr_ptr[5 × QK_WIDTH – 1:0]	phy_read_dcb	clk_rd	This is the write pointer into the data block RAM. The pointer for each device is four bits long and concatenated together.
dbg_dcb_rd_ptr[5 × QK_WIDTH – 1:0]	phy_read_dcb	clk	This is the read pointer into the data block RAM. The pointer for each device is four bits long and concatenated together.
dbg_dcb_din[4 × DATA_WIDTH – 1:0]	phy_read_dcb	clk_rd	This is the data input into the DCB.
dbg_dcb_dout[4 × DATA_WIDTH – 1:0]	phy_read_dcb	clk	This is the data output from the DCB.
dbg_clk_rd[QK_WIDTH – 1:0]	rld_phy_iob	clk_rd	This is the aligned read clock.

Table 3-21: **Additional Read Path Debug Signals *(Cont'd)***

Signal	Module	Valid Clock Domain	Description
dbg_iserdes_0[DATA_WIDTH × 2 – 1:0]	rld_phy_top	clk_rd	These are the ISERDES outputs {rise0, fall0}.
dbg_iserdes_1[DATA_WIDTH × 2 – 1:0]	rld_phy_top	clk_rd	These are the ISERDES outputs {rise1, fall1}.

When checking the results of read calibration, check the tap values for the capture clock (cq_dly_tap and cqn_dly_tap) and the tap settings for each data bit (q_dly_tap). For a single clock group, the IODELAY tap settings should not vary widely. Figure 3-68 shows stage 1 read calibration running for Q bit 0, as well as some signals of interest.

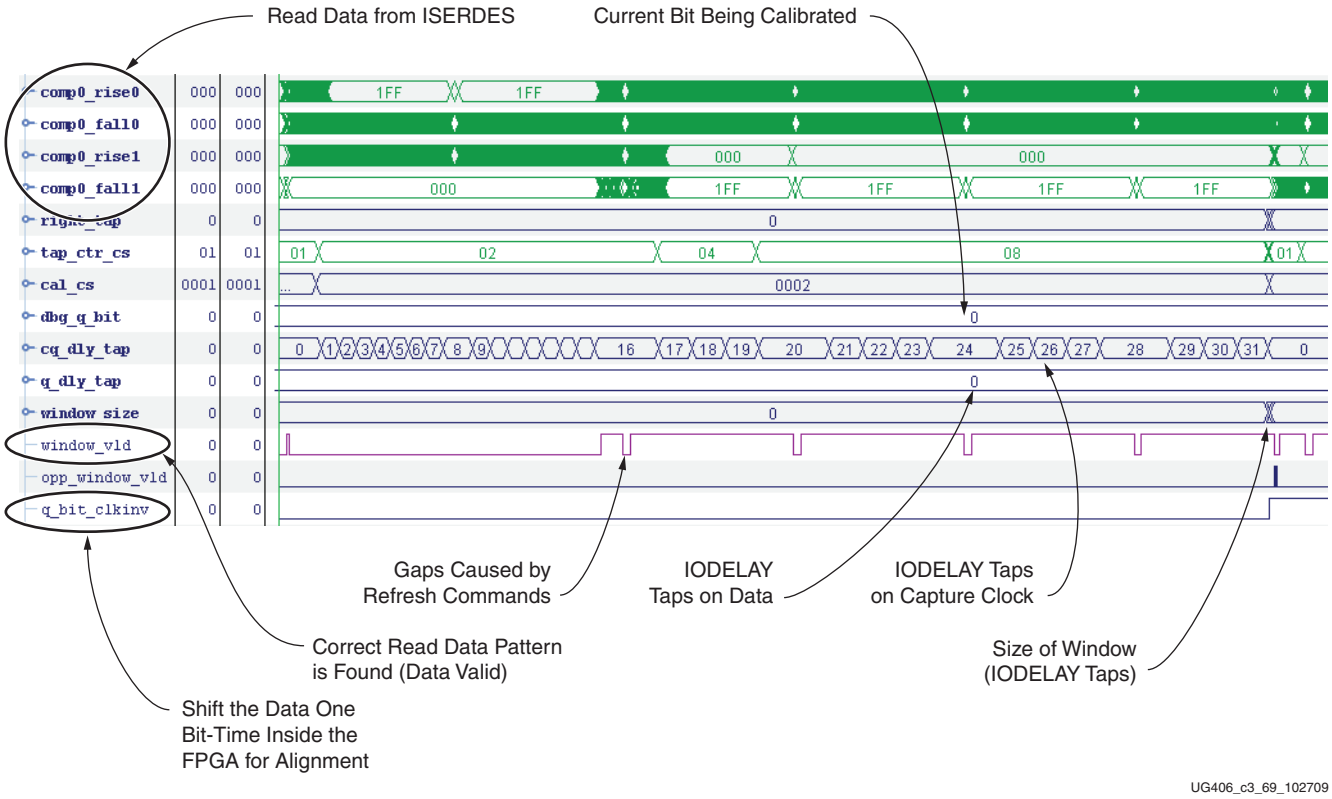


Figure 3-68: **Stage 1 Read Calibration**

The RLDRAM II memory signals being provided to the IOB logic are fed back in the debug port. The user can inspect the commands and command sequence presented to the memory device (dbg_phy_cmd_n, dbg_phy_addr, dbg_phy_ba, dbg_phy_wr_data) as shown in Table 3-69. These signals can be used to verify controller functionality or to verify that the initialization sequence is being generated as expected.

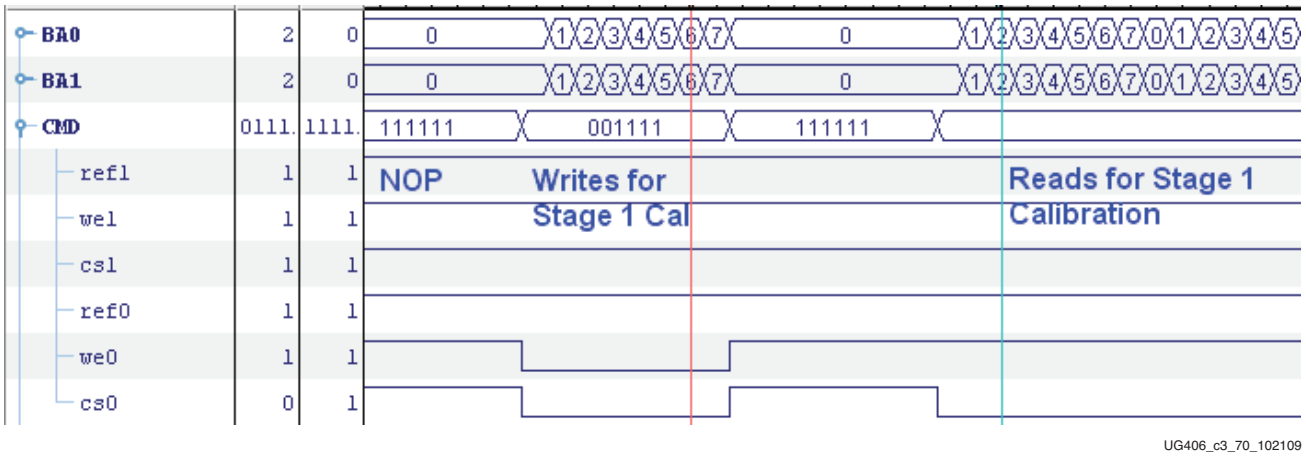


Figure 3-69: **Bank Address and Commands for Stage 1 Calibration**

Manual control signals are provided in the debug port for adjusting IODELAY tap values during normal operation. These can be adjusted to check for issues or to measure the data valid window timing. Figure 3-70 shows one such example setup in the VIO window of the ChipScope analyzer.

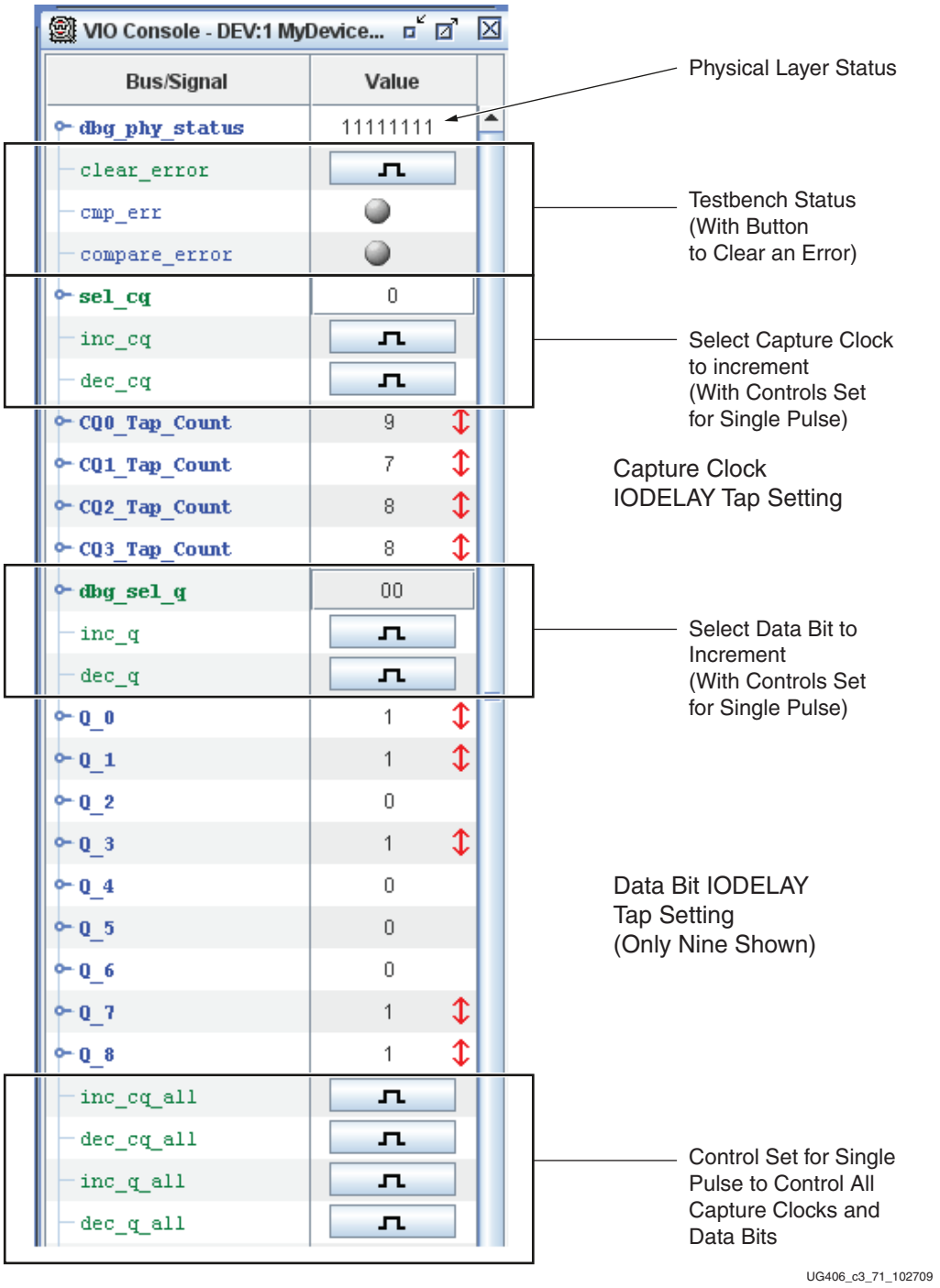


Figure 3-70: **Debug Port Control Signals**

To measure the read capture window, the phase detector must be verified as not running. This can be accomplished by generating a design without the phase detector enabled (PHASE_DETECT set to off) or by asserting the dbg_pd_off signal. The phase detector has control over the capture clock IODELAY taps. The IODELAY value is adjusted by the phase detector in the VIO window of the ChipScope analyzer, while the data bit IODELAY values are switching between the values used during reads and writes (always 0 taps for writes). After the phase detector is turned off, the IODLEAY taps can be moved manually.

The read capture clock taps can be adjusted by incrementing the taps. A single pulse on the inc_cq button increments the taps on the capture clock by one. The taps should be incremented until the compare_error signal is asserted. This indicates that the test bench has detected an error the read data window is at the edge. It should be noted where this edge occurred. The capture clock IODELAY taps should be decremented back to the original location found during calibration. After the IODELAY taps are back to the starting location, the clear_error control signal should be pulsed to clear the test bench error latch. The capture clock IODELAY taps should be decremented until the compare_error signal, the other edge of the window, is again asserted. It should be noted where this edge occurs. The capture clock back should either be returned back to the original IODELAY setting from calibration and the test bench error cleared, or the design should be reset. The read window is the first edge minus the second edge in IODELAY taps for the given clock group being adjusted. This tap value should be multiplied by the IODELAY tap resolution to obtain the window in ps.

Board Measurements

The signal integrity of the board and bus timing must be analyzed. The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 12] describes expected bus signal integrity. While this guide is specific to Virtex-5 devices and the ML561 development board, the principles therein can be applied to Virtex-6 FPGA MIG tool designs.

Other important board measurements are the reference voltage levels. It is important that these voltage levels be measured when the bus is active. These levels can be correct when the bus is idle, but might drop when the bus is active.

XILINX®

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

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

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

The Solution Center specific to Virtex®-6 FPGA Memory Interface Solutions is located at Xilinx MIG Solution Center.

References

These references provide supplemental information useful for this document:

1. ARM® AMBA® Specifications
   http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
2. UG683, *EDK Concepts, Tools, and Techniques*
3. UG111, *Embedded System Tools Reference Manual*
4. UG361, *Virtex-6 FPGA SelectIO Resources User Guide*
5. TN-47-01, *DDR2-533 Memory Design Guide For Two-DIMM Unbuffered Systems*. Micron Technology, Inc.
6. ChipScope Pro Logic Analyzer tool
   http://www.xilinx.com/tools/cspro.htm
7. UG628, *Command Line Tools User Guide*, COMPXLIB
8. UG626, *Synthesis and Simulation Design Guide*
9. DS186, *Virtex-6 FPGA Memory Interface Solutions Data Sheet*
10. PlanAhead™ Design Analysis tool
    http://www.xilinx.com/tools/planahead.htm
11. UG612, *Xilinx Timing Constraints User Guide*
12. UG199, *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide*
13. DS152, *Virtex-6 FPGA Data Sheet*

14. UG029, *ChipScope Pro Software and Cores User Guide*
15. Virtex-6 FPGA ML605 Reference Designs
http://www.xilinx.com/products/boards/ml605/reference_designs.htm

XILINX®

Appendix B

Simulation Help

The generic simulation test bench supports MIG generated designs. With this test bench, the user can simulate the design generated from the MIG tool.

Introduction

The `sim` folder provides the simulation environment for the generated design in both the ModelSim and ISim environments. The folder instantiates the various entities or modules required to simulate the design properly, and generates the required system input signals such as clocks and resets into the design. Figure B-1 depicts a block diagram of the simulation environment.

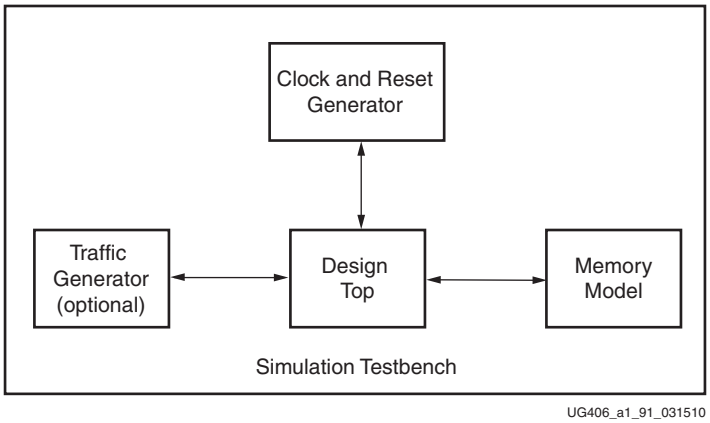


Figure B-1: **Block Diagram of Simulation Environment**

The simulation test bench module integrates the complete system through port maps, a design clock, and reset generation logic. The complete system can be divided into these blocks:

- Clock and Reset Generator: This block generates the clocks and system reset signals.
- Traffic Generator (optional instance): This block is a part of the test bench module for the user design. The traffic generator is a synthesizable module that provides test inputs such as design data, addresses, and commands. In designs with a test bench, the test bench is part of the design top module.
- Design Top: In the example design, this block connects with the clocks, reset, memory interface signals, and status signals. In the user design, The Design Top block includes the controller, infrastructure, and optional traffic generator.
- Memory Model: This block is provided with the memory core of the component selected. The MIG tool provides memory models in Verilog only. VHDL models are

not provided. The MIG tool provides Micron vendor memory models only. If the user is using other vendor memories (Samsung or Cypress), the memory models should be downloaded from the web.

The simulation test bench module can be in Verilog or VHDL, depending on the HDL used in the design. Simulation test bench Top refers to the `sim_tb_top` module in the `sim` folder.

Supported Features

Supported features include:

- All component widths
- Designs with or without a test bench
- All supported components and DIMMs (UDIMMs, SODIMMs, and RDIMMs)
- Twin-die components and dual-rank DIMMs for DDR2 and dual-rank DIMMs for DDR3 SDRAM designs
- A multi-controller simulation test bench for the Virtex-6 FPGA QDRII+ SRAM and DDR3 SDRAM.

Unsupported Features

VHDL memory models are not supported.

Note: The simulation test bench top file generated is unique for the design generated from the MIG tool. Design parameters should not be changed after generating the design, except for the RST_ACT_LOW parameter.

Simulating the Design

Support is provided for these simulators:

- ModelSim
- ISim

Simulations Using ModelSim

The design can be simulated using the ModelSim GUI either manually, or by calling the `sim.do` file in ModelSim.

Method 1: Manual Simulation

1. Invoke ModelSim.
2. Create a new project.
3. Add design files from the `rtl` folder and simulation files from the `sim` folder. Only `.v` or `.vhd` files are required to be added into the project. It is not necessary to add the `.vh` files in the `sim` folder.
4. Map the `unisim` or `unsims_ver` libraries. This is required to compile Xilinx primitives used in the design.
5. Compile the design.
6. Load the design and map the library to it. For example, the following command is used to load a Virtex-6 FPGA DDR2 SDRAM design in Verilog:

```
vsim -t ps +notimingchecks -L unisims_ver work.sim_tb_top glbl
```

The following command loads the same design in VHDL:

```
vsim -t ps +notimingchecks -L unisim work.sim_tb_top glbl
```

7. After loading the design, run the simulation for an amount of time based on the traffic to be generated (typically 200 µs, if memory initialization is skipped).

Method 2: Using the `sim.do` File

1. Invoke ModelSim.
2. Create a new project. (Optional.)
3. Change the directory to the working `sim` folder. (Optional.)
4. If the directory is set to the working directory, execute this command:

```
do sim.do
```

For example:

```
do E:/simulations/test1/example_design/sim/sim.do
```

Changing Simulation Run Time

Memory write and read can be performed correctly after calibration/initialization of the design at a given frequency. Run time also depends upon the calibration period. In the `sim.do` file, the run time is determined by this portion of the ModelSim command:

```
when {/sim_tb_top/phy_init_done = 1} {
if {[when -label a_100] == ""} {
when -label a_100 { $now = 50 us } {
nowhen a_100..........}}}}
```

The `when` command checks for completion of calibration (run time is set for 50 µs). To increase the run time after completion of calibration, 50 µs should be changed to some other value, such as 100 µs. There must be a space between the value and the unit.

This command is useful in case the previous condition fails to complete within, for example, 800 µs:

```
when {$now = @800 us} {stop}
```

ModelSim pauses in this case.

Note: The second `when` command run time value (800 µs) should always be greater than the first one (50 µs), otherwise the simulation result that is displayed at the end shows a "calibration/initialization failed" message.

Changing the Breakpoint Condition

In the run time example, `/sim_tb_top/phy_init_done = 1` indicates the signal on which the breakpoint is set. The user can change this condition by changing the path and value of the breakpoint. Refer to the ModelSim command reference for more information.

Simulations Using ISim

The design can be simulated using the ISim simulator by running the `isim_run.bat` file located in the `sim` folder. This file compiles and elaborates the design. The `isim_run.bat` file also generates the simulation executable using the fuse command and invokes the ISim GUI.

By default, all signals in the `sim_tb_top` module are added to the waveform viewer. The user can add required signals from the objects window to the waveform viewer. The simulation can be run for a specified time using the run <time> command in the ISim GUI.

Files in sim Folder

The MIG tool generates all of the design files in the `rtl` folder and the simulation files in the `sim` folder. Table B-1 lists the files generated in the `sim` folder for Virtex-6 FPGA DDR3 SDRAM designs.

Table B-1: **Virtex-6 FPGA DDR3 SDRAM Directory Files**

File Name	Description
`sim_tb_top.v/vhd`	This is the Verilog or VHDL external test bench file. It integrates the system and provides system inputs.
`wiredly.v/vhd`	This module inserts a delay into Verilog or VHDL designs.
`ddr3_model.v`	This is the Verilog memory model for the DDR3 SDRAM memory from Micron. It is provided only in Verilog.
`ddr3_model_parameters.vh`	This is the parameter file used by Micron memory model `ddr3_model.v`. It lists all of the different parameters that define the memory type and timing parameters.
`glbl.v`	This is used to initialize the simulator environment.
`sim.do`	This lists the ModelSim commands required to run the test case using the ModelSim simulator.
`isim_run.bat`	This lists the ISim commands required to run the test case using the ISim simulator.
`isim_files.prj`	This contains the list of HDL files present in the design. It also contains the library and the source file names.
`isim_options.tcl`	This contains the TCL commands for simulation and resume-on-error.
`readme.txt`	This describes the steps to run simulations using the ModelSim or ISim simulator.

Table B-1: **Virtex-6 FPGA DDR3 SDRAM Directory Files *(Cont'd)***

File Name	Description
afifo, cmd_gen, cmd_prbs_gen, data_prbs_gen, init_mem_pattern_ctr, mcb_flow_control, mcb_traffic_gen, pipeline_inserter, rd_data_gen, read_data_path, read_posted_fifo, sp6_data_gen, tg_status, v6_data_gen, wr_data_gen, write_data_path (.v/vhd)	These are optional modules for designs without test benches. They are located in the rtl folder.

Table B-2 lists the files generated in the sim folder for Virtex-6 FPGA DDR2 SDRAM designs.

Table B-2: **Virtex-6 FPGA DDR2 SDRAM Directory Files**

File Name	Description
sim_tb_top.v/vhd	This is the Verilog or VHDL external test bench file. It integrates the system and provides system inputs.
wiredly.v/vhd	This module inserts delays into Verilog or VHDL designs.
ddr2_model.v	This is the Verilog memory model for DDR2 SDRAM from Micron. It is provided only in Verilog.
ddr2_model_parameters.vh	This is the parameter file used by Micron memory model ddr2_model.v. It lists all of the different parameters that define the memory type and timing parameters.
glbl.v	This is used to initialize the simulator environment.
sim.do	This lists the ModelSim commands required to run the test case using the ModelSim simulator.
isim_run.bat	This lists the ISim commands required to run the test case using the ISim simulator.
isim_files.prj	This contains the list of HDL files present in the design. It also contains the library and the source file names.
isim_options.tcl	This contains the TCL commands for simulation and resume-on-error.

Table B-2: **Virtex-6 FPGA DDR2 SDRAM Directory Files *(Cont'd)***

File Name	Description
readme.txt	This describes the steps to run simulations using the ModelSim or ISim simulator.
afifo, cmd_gen, cmd_prbs_gen, data_prbs_gen, init_mem_pattern_ctr, mcb_flow_control, mcb_traffic_gen, pipeline_inserter, rd_data_gen, read_data_path, read_posted_fifo, sp6_data_gen, tg_status, v6_data_gen, wr_data_gen, write_data_path (.v/vhd)	These are optional modules for designs without test benches. They are located in the rtl folder.

Table B-3 lists the files generated in the sim folder for Virtex-6 FPGA QDRII+ SDRAM designs.

Table B-3: **Virtex-6 FPGA QDRII+ SDRAM Directory Files**

File Name	Description
sim_tb_top.v/vhd	This is the Verilog or VHDL external test bench file. It integrates the system and provides system inputs.
glbl.v	This is used to initialize the simulator environment.
sim.do	This lists the ModelSim commands required to run the test case using the ModelSim simulator.
isim_run.bat	This lists the ISim commands required to run the test case using the ISim simulator.
isim_files.prj	This contains the list of HDL files present in the design. It also contains the library and the source file names.
isim_options.tcl	This contains the TCL commands for simulation and resume-on-error.
readme.txt	This describes the steps to run simulations using the ModelSim or ISim simulator.
tb_addr_gen, tb_cmp_data, tb_cmp_data_bits, tb_data_gen, tb_top, tb_wr_rd_sm (.v/vhd)	These are optional modules for designs without test benches. They are located in the rtl folder.

Table B-4 lists the files generated in the sim folder for Virtex-6 FPGA RLDRAM II SDRAM designs.

Table B-4: **Virtex-6 FPGA RLDRAM II Directory Files**

File Name	Description
sim_tb_top.v/vhd	This is the Verilog or VHDL external test bench file. It integrates the system and provides system inputs.
ldram2_cio_model.v	This is the Verilog memory model for RLDRAM II memory from Micron. It is provided only in Verilog.
rrldram2_cio_model_ parameters.vh	This is the parameter file used by the memory model.
rldram2_cio_model.v	This lists all of the different parameters that define the memory type and timing parameters.
glbl.v	This is used to initialize the simulator environment.
sim.do	This lists the ModelSim commands required to run the test case using the ModelSim simulator.
isim_run.bat	This lists the ISim commands required to run the test case using the ISim simulator.
isim_files.prj	This contains the list of HDL files present in the design. It also contains the library and the source file names.
isim_options.tcl	This contains the TCL commands for simulation and resume-on-error.
readme.txt	This describes the steps to run simulations using the ModelSim or ISim simulator.
rld_tb_addr_gen, tb_cmp_data, tb_cmp_data_bits, tb_data_gen, rld_tb_top, rld_tb_wr_rd_sm (.v/vhd)	These are optional modules for designs without test benches. They are located in the rtl folder.

Table B-5 lists the files generated in the sim folder for Virtex-6 FPGA multicontroller designs.

Table B-5: **Virtex-6 FPGA Multicontroller Directory Files**

File Name	Description
sim_tb_top.v/vhd	This is the Verilog or VHDL external test bench file. It integrates the system and provides system inputs.
wiredly.v/vhd	This module inserts delays into Verilog or VHDL designs.
cx_ddr3_model.v	This is the verilog memory model for the DDR3 SDRAM memory from Micron. it is provided only in Verilog. The controller number is represented by cx (for example, c0, c1).
cx_ddr3_model_parameters.vh	This is the parameter file used by Micron memory model ddr3_model.v. It lists all the different parameters that define the memory type and timing parameters. The controller number is represented by cx (for example, c0, c1).
glbl.v	This is used to initiate the simulator environment.
sim.do	This lists the ModelSim commands required to run the test case using the ModelSim simulator.
isim_run.bat	This lists the ISim commands required to run the test case using the ISim simulator.
isim_files.prj	This contains the list of HDL files present in the design. It also contains the library and the source file names.
sim_options.tcl	This contains the TCL commands for simulation and resume-on-error.
ireadme.txt	This describes the steps to run simulations using the ModelSim or ISim simulator.
fifo, cmd_gen, cmd_prbs_gen, data_prbs_gen, init_mem_pattern_ctr, mcb_flow_control, mcb_traffic_gen, pipeline_inserter, rd_data_gen, read_data_path, read_posted_fifo, sp6_data_gen, tg_status, v6_data_gen, wr_data_gen, write_data_path, tb_addr_gen, tb_cmp_data, tb_cmp_data_bits, tb_data_gen, tb_top, tb_wr_rd_sm (.v/vhd)	These are optional modules for designs without test benches. They are located in the rtl folder.

Design Notes

This section provides notes on the various designs discussed in this chapter:

1. When using ModelSim to simulate the design, the `sim.do` file has commands to suppress Numeric Std package and Arithmetic operation warnings.
2. After simulation in ModelSim, a test result is displayed based on whether or not the design generates an error signal. The displayed result does not consider the error or violations generated by the memory models or the simulator. The transcript file should be reviewed for any errors or warnings generated.
3. If the license agreement is not accepted while generating the design, a memory model is not generated in the `sim` folder. In this case, the memory model must be downloaded from the memory vendor site and placed in the `sim` folder. The files should be renamed accordingly, as described in Files in sim Folder, page 330 According to the design generated, memory model parameters are passed from the `sim.do` file. For example, this command is used for a Micron DDR2 SDRAM design:

   ```
   vlog  +incdir+. +define+x256Mb +define+sg3 +define+x8 ddr2_model.v
   ```

   In this example, `+define+x256Mb` indicates the device density. This parameter is not present in the downloaded memory model, so it should be ignored. `+define+sg3` indicates the memory speed grade, and `+define+x8` indicates the device data width.
4. For DIMM designs, the MIG tool uses instantiations of component models.
5. While simulating VHDL designs generated using the Debug Signals option enabled in the MIG tool, the simulator outputs warning messages such as:

   ```
   # ** Warning: (vsim-3473) Component instance "ila_inst : ila" is not
   bound.
   #    Time: 0 ps  Iteration: 0  Region: /sim_tb_top/u_mem_controller
   File: ../rtl/test13_vhd_X0.vhd
   # ** Warning: (vsim-3473) Component instance "vio_inst : vio" is not
   bound.
   #    Time: 0 ps  Iteration: 0  Region: /sim_tb_top/u_mem_controller
   File: ../rtl/test13_vhd_X0.vhd
   # ** Warning: (vsim-3473) Component instance "icon_inst : icon" is not
   bound.
   #    Time: 0 ps  Iteration: 0  Region: /sim_tb_top/u_mem_controller
   File: ../rtl/test13_vhd_X0.vhd
   ```

   These warning messages can be ignored because ila and icon instances are only useful for debugging the design on hardware.

Known Issues

This section discusses some known issues that can occur during simulation.

DDR3 SDRAM/DDR2 SDRAM

- ModelSim simulation error:

  ```
  "#sim_tb_top.comp_inst.mem_8_4.gen_mem[2].u_comp_ddr3.dqs_pos_timing_
  check: at time 18790687.0 ps WARNING: tWLS violation on DQS bit 0
  positive edge.
  ```

  Indeterminate CK capture is possible.

  ```
  #sim_tb_top.comp_inst.mem_8_4.gen_mem[2].u_comp_ddr3.dqs_pos_timing_ch
  eck: at time 18790687.0 ps Write Leveling @ DQS ck = 1"
  ```

  This error can be ignored for DDR3 SDRAM designs.

- For VHDL designs:
  - Warning in ModelSim simulator:

    ```
    #Warning:NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
    #Time: 0 ps
    Iteration:0
    Instance: /sim_tb_top/u_ip_top/u_memc_ui_top/u_mem_intfc/mc0/
    rank_mach0/\rank_cntrl_inst(0)\/rank_cntrl0.
    ```

  - Warning in ISim simulator:

    ```
    Instance /sim_tb_top/u_ip_top/u_memc_ui_top/u_mem_intfc/mc0/
    rank_mach0/\rank_cntrl_inst(0)\/rank_cntrl0/ : Warning: There is an
    'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be
    'X'(es).
    ```

    These warnings are due to metastable values during power-on. The user might get some warning messages in both ModelSim and ISim simulators. These messages are suppressed in the `sim.do` file for the ModelSim simulator. They can only appear if the design is simulated without using the `sim.do` file generated by the MIG tool. These warning messages should be ignored.

- The SIM simulator issues warnings such as “Parameter declaration becomes local in mc with formal parameter declaration list” and “Function clogb2 has no return value assignment.” These warnings can be ignored.
- In the case of x16/x8 components and data width of 8, this warning appears while compiling the design:

  ```
  #  ** Warning: [3] ../sim/sim_tb_top.vhd(316): Range 0 to -1 is null.
  ```

  This warning appears for VHDL designs only, and should be ignored.

6. If the design is rerun without deleting old files that were generated during simulation, this warning can appear:

   ```
   # ** Warning: (vlib-34) Library already exists at "work".
   ```

   This warning should be ignored.

QDRII+ SRAM

- Due to metastable values during power-on, warning messages might be displayed in both the ModelSim and ISim simulators. These messages are suppressed in the `sim.do` file for the ModelSim simulator. They appear only if the design is simulated without using the `sim.do` file generated by the MIG tool. These warning messages should be ignored.
- For the Cypress controller designs, the `sim_tb_top` module instantiates a sample Cypress memory model. The model name in the test bench (`sim_tb_top`) contains the two memory model instances (x36 and x18 memory models) with generate statements. For example, the model is named cyqdr2_b4 and cyqdr2_b4_18 for x36 and x18 parts, respectively. This model name should be appropriately modified based on the downloaded Cypress model.
- The ISim simulator warnings "Function clogb2 has no return value assignment" and "Comparison between arrays of unequal length always returns TRUE" can be ignored.

RLDRAM II

- Due to metastable values during power-on, warning messages might be displayed. These messages are suppressed in the `sim.do` file and appear only if the design is simulated without using the `sim.do` file generated by the MIG tool. These warning messages should be ignored.
- The SIM simulator warnings "Function clogb2 has no return value assignment" and "Comparison between arrays of unequal length always returns TRUE" can be ignored.
- ModelSim simulation tCS violation errors on address/control signals prior to calibration can be ignored.