

# 7 Series FPGAs Memory Interface Solutions

## *User Guide*

UG586 January 18, 2012



#### Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2011–2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, ARM, ARM1176JZ-S, CoreSight, Cortex, and PrimeCell are trademarks of ARM in the EU and other countries. All other trademarks are the property of their respective owners.

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/01/11	1.0	Initial Xilinx release.
06/22/11	1.1	MIG 1.2 release. Updated ISE Design Suite version to 13.2. Updated GUI screen captures throughout document.  <a href="#">Chapter 1</a> : Added <a href="#">Verify Pin Changes and Update Design</a> , <a href="#">Simulating the Example Design (for Designs with the AXI4 Interface)</a> , <a href="#">Simulation Considerations</a> , <a href="#">Error Correcting Code</a> , and <a href="#">Pinout Examples</a> sections. Added paragraph about SLRs to <a href="#">Pin Compatible FPGAs</a> , page 15. Added Input Clock Period and PHY to Controller bullets in <a href="#">Controller Options</a> , page 17. To <a href="#">Setting DDR3 Memory Parameter Option</a> , page 21, indicated that DDR3 SDRAM supports burst lengths of 8. Added Internal Termination for High Range Banks option under <a href="#">Figure 1-17</a> . Added bulleted item about Pin/Bank selection mode on <a href="#">page 24</a> . Added notes about chip select and data mask options on <a href="#">page 47</a> . Added app_correct_en_i to <a href="#">Table 1-17</a> . Added three command types to <a href="#">Command Path</a> , page 87. Added phy_mc_ctl_full, phy_mc_cmd_full, and phy_mc_data_full signals to <a href="#">Table 1-31</a> . Added paragraph about FIFOs at the end of <a href="#">Physical Layer Interface (Non-Memory Controller Design)</a> , page 114. Updated the description and options for DATA_BUF_ADDR_WIDTH in <a href="#">Table 1-36</a> . Added bullet about SLRs to <a href="#">Bank and Pin Selection Guides for DDR3 Designs</a> , page 126. Added LVCMOS15 and DIFF_SSTL15 I/O standards to <a href="#">Configuration</a> , page 128. Changed resistor values in <a href="#">Figure 1-70</a> , <a href="#">Figure 1-71</a> , and <a href="#">Figure 1-72</a> . Changed resistor values in FPGA DCI or IN_TERM column in <a href="#">Table 1-39</a> .  <a href="#">Chapter 2</a> : Added the <a href="#">Verify Pin Changes and Update Design</a> and <a href="#">Output Path</a> sections. Revised latency mode description on <a href="#">page 165</a> . Added bulleted item about Pin/Bank selection mode on <a href="#">page 170</a> . Added Internal Termination for High Range Banks option under <a href="#">Figure 2-16</a> . Updated <a href="#">Implementation Details</a> , page 196.  <a href="#">Chapter 3</a> : Added new chapter on RLDRAM II.

Date	Version	Revision
10/19/11	1.2	<p>MIG 1.3 release. Updated ISE Design Suite version to 13.3.</p> <ul style="list-style-type: none"> <li>• <a href="#">Chapter 1</a>: Added <a href="#">step 2 to MIG Output Options, page 14</a>. Added note about optional use of the memory controller to <a href="#">Controller Options, page 17</a>. Added arbitration scheme to <a href="#">AXI Parameter Options, page 20</a>. Added description of DCI Cascade under <a href="#">Figure 1-17</a>. Updated text about devices with SSI technology and SLRs on <a href="#">page 25</a> and <a href="#">page 127</a>. Changed error to <code>tg_compare_error</code> on <a href="#">page 27</a>. Replaced <a href="#">Table 1-8</a>. Added <code>qdr_wr_cmd_o</code>, <code>vio_fixed_instr_value</code>, <code>vio_fixed.bl_value</code>, <code>vio_pause_traffic</code>, and <code>vio_data_mask_gen</code> signals to <a href="#">Table 1-13</a>. Added signals to the User Interface in <a href="#">Figure 1-31</a> and <a href="#">Figure 1-33</a>. Added <code>app_sr_req</code>, <code>app_sr_active</code>, <code>app_ref_req</code>, <code>app_ref_ack</code>, <code>app_zq_req</code>, and <code>app_zq_ack</code> signals to <a href="#">Table 1-17</a>. Added <code>app_wdf_rdy</code>, <code>app_ref_req</code>, <code>app_ref_ack</code>, <code>app_zq_req</code>, <code>app_zq_ack</code>, <a href="#">Read Priority with Starve Limit (RD_PRI_REG_STARVE_LIMIT)</a>, <a href="#">Native Interface Maintenance Command Signals</a>, <a href="#">User Refresh</a>, and <a href="#">User ZQ</a> sections. Added <code>C_RD_WR_ARB_ALGORITHM</code> to <a href="#">Table 1-19</a>. Updated fields in <a href="#">Table 1-29</a>, changed Hi Index (Rank) to Rank Count, and added CAS slot field. Updated <a href="#">AXI Addressing</a> and <a href="#">Physical Layer Interface (Non-Memory Controller Design)</a>. Added <a href="#">Figure 1-57</a> through <a href="#">Figure 1-59</a> in <a href="#">Write Path</a>. In <a href="#">Table 1-35</a>, removed DISABLED option from <code>RTT_NOM</code> for DDR3_SDRAM, changed <code>RTT_NOM</code> to <code>RTT_WR</code> in <code>RTT_WR</code>, updated <code>SIM_BYPASS_INIT_CAL</code>, and updated table note 2. In <a href="#">Table 1-36</a>, updated <code>tZQI</code> and added <code>USER_REFRESH</code>. Added <a href="#">Table 1-37</a>. In <a href="#">Configuration</a>, updated constraints example and removed paragraph about SCL and SDA.</li> <li>• <a href="#">Chapter 2</a>: Added <a href="#">step 2 to MIG Output Options, page 162</a>. Added Input Clock Period description in <a href="#">Controller Options, page 165</a>. Added Debug Signals Control and Internal Vref Selection options to <a href="#">FPGA Options, page 168</a>. Added <a href="#">I/O Planning Options, page 170</a>. In <a href="#">System Pins Selection, page 173</a>, changed <code>cal_done</code> signal to <code>init_calib_complete</code> and error signal to <code>tg_compare_error</code>. Replaced <a href="#">Table 2-2</a>. Changed file names in <a href="#">Table 2-8</a>. Updated signal names in <a href="#">Figure 2-23</a>, <a href="#">Figure 2-24</a>, and <a href="#">Figure 2-25</a>. Updated signal names in <a href="#">Table 2-10</a>. Added <code>CPT_CLK_CQ_ONLY</code> and updated value for <code>SIM_BYPASS_INIT_CAL</code> in <a href="#">Table 2-13</a>. Added <a href="#">Table 2-14</a>. Updated pinout rules in <a href="#">Pinout Requirements, page 202</a>. Added paragraph about DCI and IN_TERM after <a href="#">Table 2-15</a>. Added <a href="#">Debugging QDRII+ SRAM Designs, page 204</a>.</li> <li>• <a href="#">Chapter 3</a>: Added <a href="#">step 2 to MIG Output Options, page 228</a>. Added Input Clock Period description in <a href="#">Controller Options</a>. Added Debug Signals Control and Internal Vref Selection options to <a href="#">FPGA Options, page 233</a>. In <a href="#">System Pins Selection</a>, changed <code>cal_done</code> signal to <code>init_calib_complete</code> and error signal to <code>tg_compare_error</code>. Changed file names in <a href="#">Table 3-8</a>. Removed Table 3-12, which contained Reserved signals not used. Added <code>rst_phaser_ref</code> to <a href="#">Table 3-12</a>. Removed PHY-Only Interface section. In <a href="#">Table 3-15</a>, added <code>RLD_ADDR_WIDTH</code>, <code>MEM_TYPE</code>, <code>CLKIN_PERIOD</code>, and <code>SIMULATION</code>, and renamed <code>CLKFBOUT_MULT</code>, <code>CLKOUT0_DIVIDE</code>, <code>CLKOUT1_DIVIDE</code>, <code>CLKOUT2_DIVIDE</code>, and <code>CLKOUT3_DIVIDE</code>. Updated <a href="#">Table 3-16</a>. Added paragraph about DCI and IN_TERM after <a href="#">Table 3-27</a>.</li> <li>• Added <a href="#">Chapter 4, Multicontroller Design</a>.</li> </ul>

Date	Version	Revision
01/18/12	1.3	<p>MIG 1.4 release. Updated ISE Design Suite version to 13.4. Updated GUI screen captures throughout document.</p> <ul style="list-style-type: none"> <li>• <a href="#">Chapter 1</a>: Added support for DDR2 SDRAM. Added option 3 to <a href="#">MIG Output Options</a>. Added <a href="#">EDK Clocking</a>. Updated <a href="#">Simulation Considerations</a>. Added Replaced <a href="#">Figure 1-26</a> and <a href="#">Figure 1-46</a>.</li> <li>• <a href="#">Chapter 2</a>: Removed Input Clock Period option from <a href="#">Controller Options</a>. Added <a href="#">Memory Options</a>. Added Reference Clock option to <a href="#">FPGA Options</a>. Updated <a href="#">Debug Signals</a>.</li> <li>• <a href="#">Chapter 3</a>: Removed Input Clock Period option from <a href="#">Controller Options</a>. Added Input Clock Period option to <a href="#">Memory Options</a>. Added Reference Clock option to <a href="#">FPGA Options</a>. Added Debugging RLDARAM II Designs.</li> </ul>

# *Table of Contents*

---

Revision History .....	2
<b>Chapter 1: DDR3 and DDR2 SDRAM Memory Interface Solution</b>	
Introduction .....	7
Features.....	7
Getting Started with the CORE Generator Tool .....	7
Getting Started with EDK .....	52
Simulation Considerations .....	55
Core Architecture .....	55
Designing with the Core .....	102
Interfacing to the Core .....	102
Customizing the Core .....	117
Design Guidelines .....	126
Supported Devices for 7 Series FPGAs.....	154
<b>Chapter 2: QDRII+ Memory Interface Solution</b>	
Introduction .....	155
Getting Started with the CORE Generator Tool .....	155
Core Architecture .....	181
Customizing the Core .....	197
Design Guidelines .....	202
Debugging QDRII+ SRAM Designs.....	204
<b>Chapter 3: RLDRAM II Memory Interface Solution</b>	
Introduction .....	221
Getting Started with the CORE Generator Tool .....	221
Core Architecture .....	250
Customizing the Core .....	266
Design Guidelines .....	272
Debugging RLDRAM II Designs.....	279
<b>Chapter 4: Multicontroller Design</b>	
Introduction .....	301
Getting Started with the CORE Generator Tool .....	301
MIG Directory Structure .....	306
<b>Appendix A: Additional Resources</b>	
Xilinx Resources .....	307

---

<b>Solution Centers .....</b>	307
<b>References .....</b>	307

# *DDR3 and DDR2 SDRAM Memory Interface Solution*

---

## Introduction

The Xilinx® 7 series FPGAs memory interface solutions core is a combined pre-engineered controller and physical layer (PHY) for interfacing 7 series FPGA user designs and AMBA® advanced extensible interface (AXI4) slave interfaces to DDR3 and DDR2 SDRAM devices. This user guide provides information about using, customizing, and simulating a LogiCORE™ IP DDR3 or DDR2 SDRAM memory interface core for 7 series FPGAs. In the Embedded Development Kit (EDK) this core is provided through the Xilinx Platform Studio (XPS) as the `axi_7series_ddrx` IP with a static AXI4 to DDR3 or DDR2 SDRAM architecture. The user guide describes the core architecture and provides details on customizing and interfacing to the core.

## Features

Enhancements to the Xilinx 7 series FPGA memory interface solutions from earlier memory interface solution device families include:

- Higher performance.
- New hardware blocks used in the physical layer: PHASER\_IN and PHASER\_OUT, PHY control block, and I/O FIFOs (see [Core Architecture, page 55](#)).
- Pinout rules changed due to the hardware blocks (see [Design Guidelines, page 126](#)).
- Controller and user interface operate at 1/4th the memory clock frequency.

## Getting Started with the CORE Generator Tool

This section is a step-by-step guide for using the CORE Generator™ tool to generate a DDR3 or DDR2 SDRAM memory interface in a 7 series FPGA, run the design through implementation with the Xilinx tools, and simulate the example design using the synthesizable test bench provided.

## System Requirements

- ISE® Design Suite, v13.4

## 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 tool 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 7 series FPGA DDR3 SDRAM design (most MIG options are same between DDR3 and DDR2 SDRAM interfaces):

**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 tool 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 7 Series Memory Controller** from the XPS IP catalog (when adding new IP to the system) or right-click the **axi\_7series\_ddrx** component in the XPS System Assembly View and select **Configure IP...**. Then skip to [MIG Output Options, page 14](#).

Otherwise, to launch the MIG tool from the CORE Generator tool, type **mig** in the search IP catalog box ([Figure 1-1](#)).

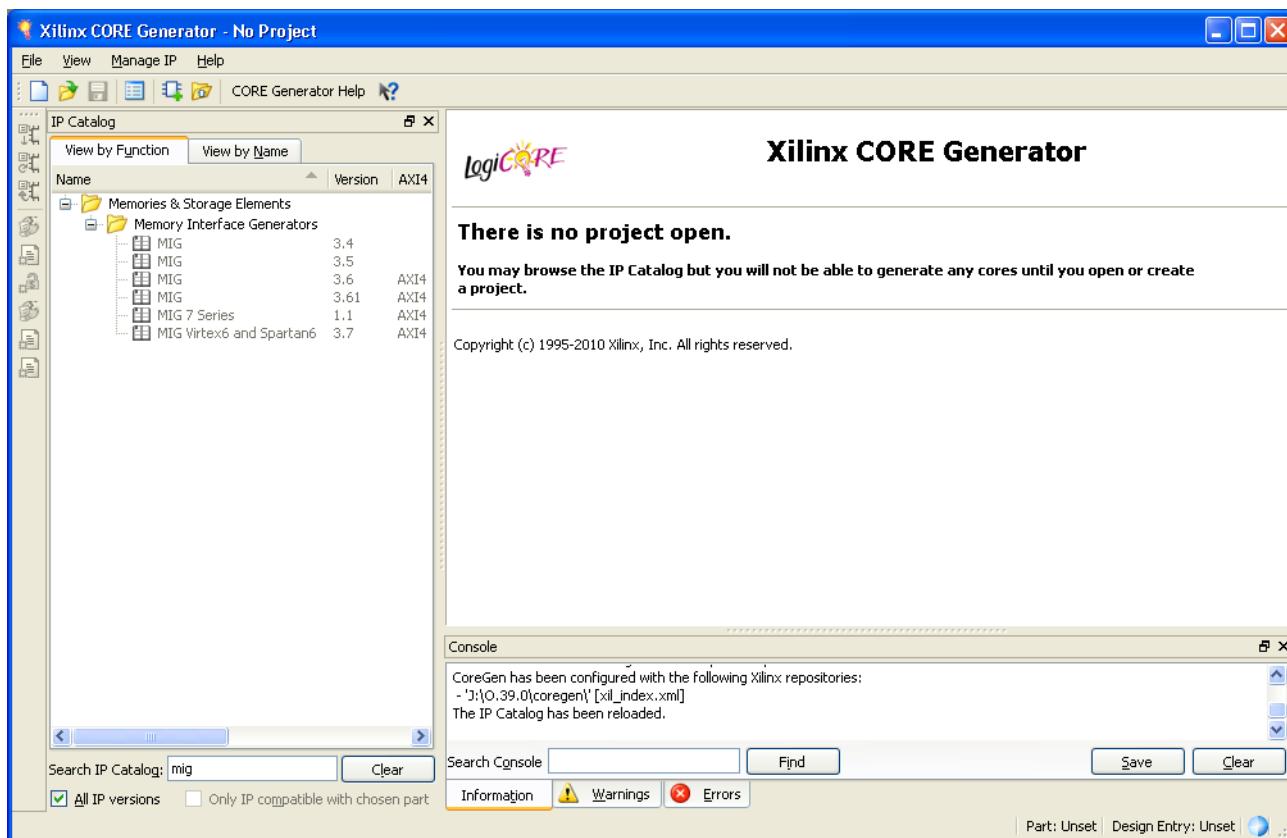


Figure 1-1: Xilinx CORE Generator Tool

2. Choose **File** → **New Project** to open the New Project dialog box. Create a new project named 7Series\_MIG\_Example\_Design (Figure 1-2).

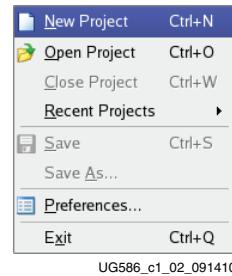


Figure 1-2: New CORE Generator Tool Project

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

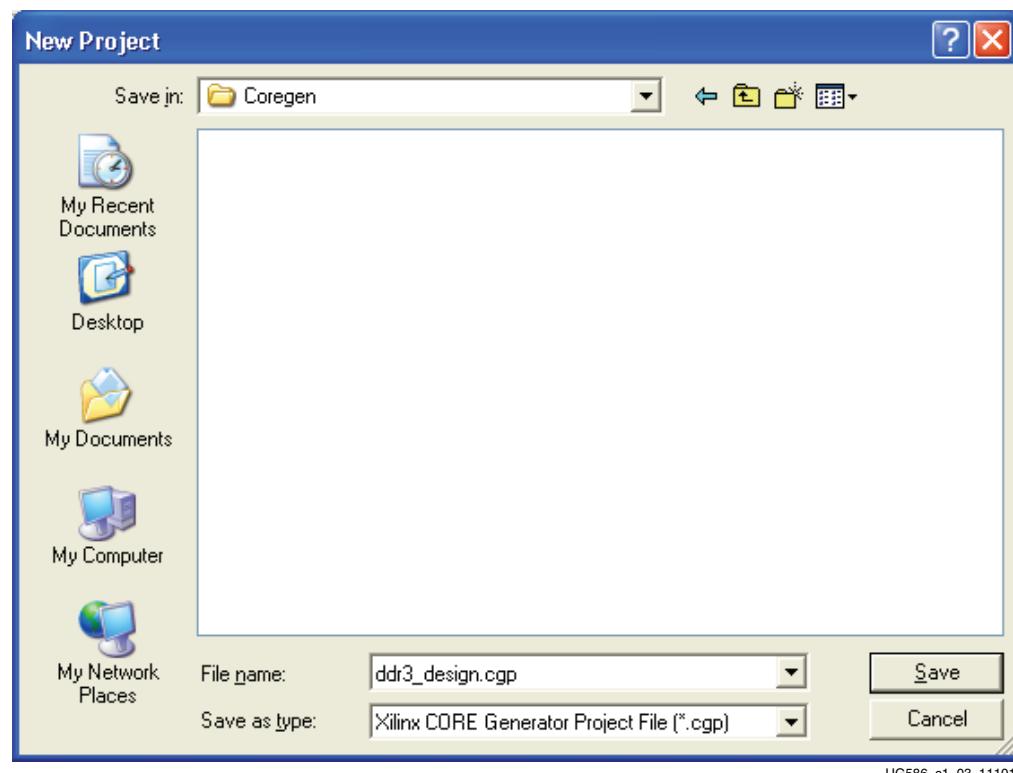


Figure 1-3: New Project Menu

4. Select these project options for the part (Figure 1-4):
- Select the target Kintex™-7 or Virtex®-7 device.

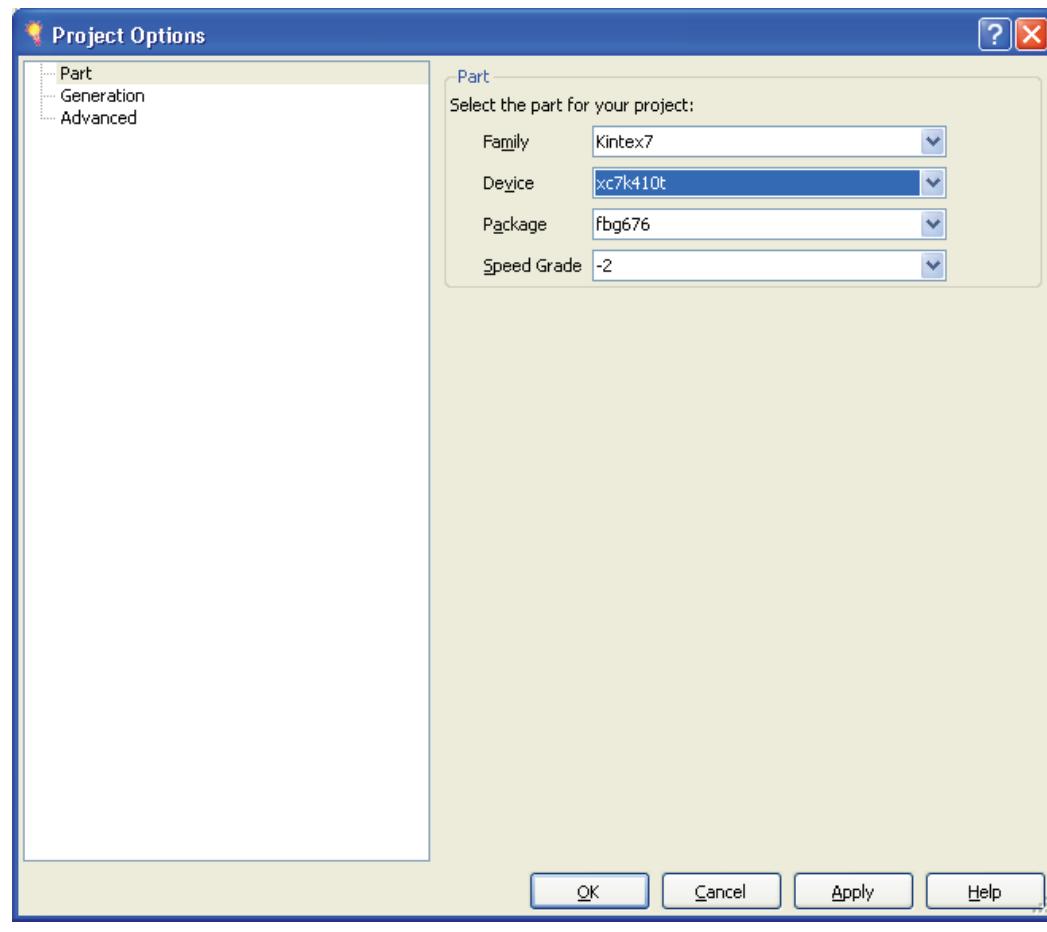


Figure 1-4: CORE Generator Tool Device Selection Page

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

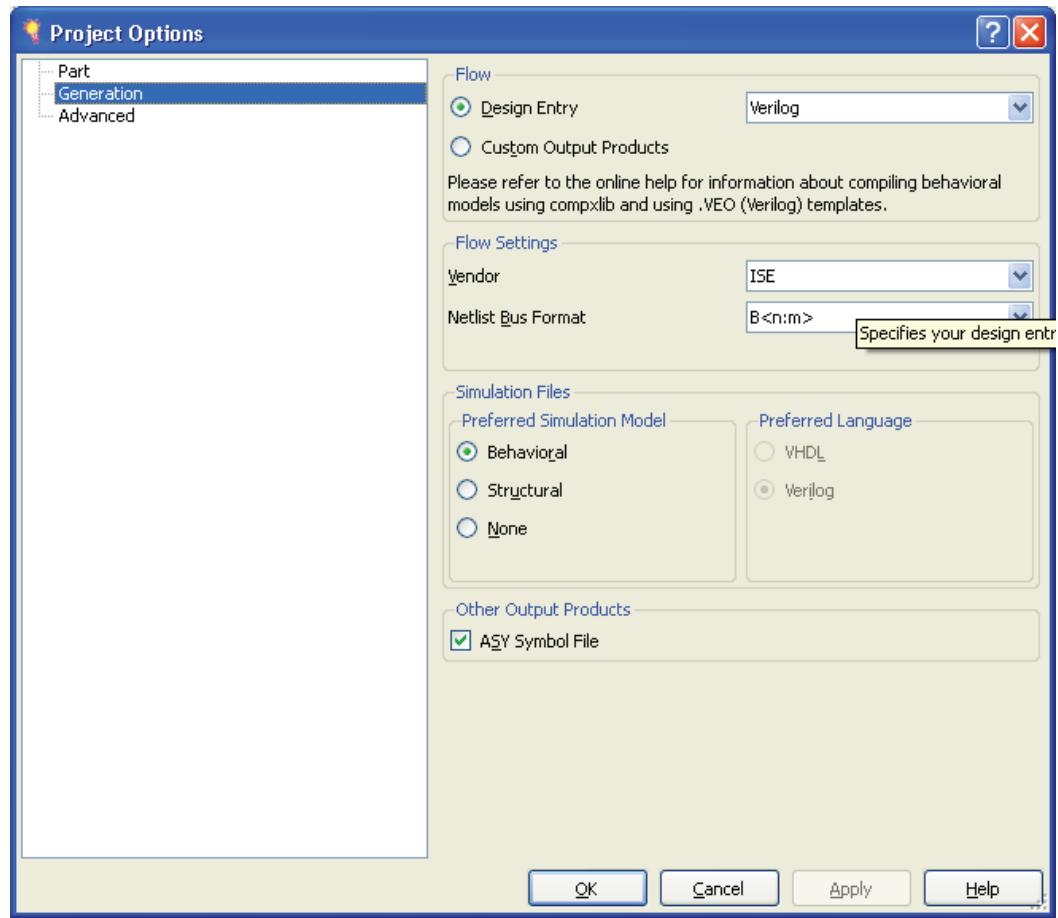


Figure 1-5: CORE Generator Tool Design Flow Setting Page

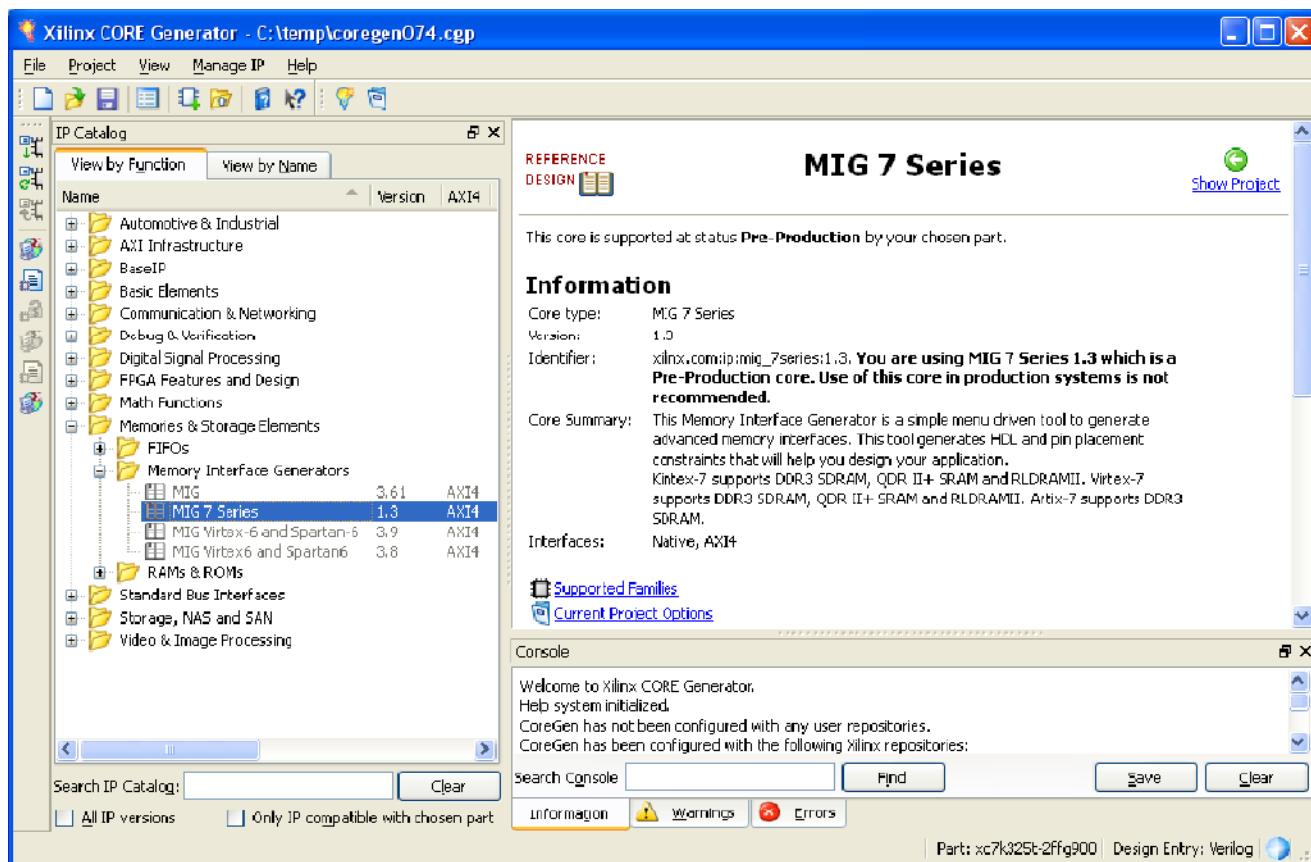
6. Select **MIG 7 Series 1.4** (Figure 1-6).

Figure 1-6: 7 Series FPGAs MIG Design Project Page

7. The options screen in the CORE Generator tool displays the details of the selected CORE Generator tool options that are selected before invoking the MIG tool (Figure 1-7).

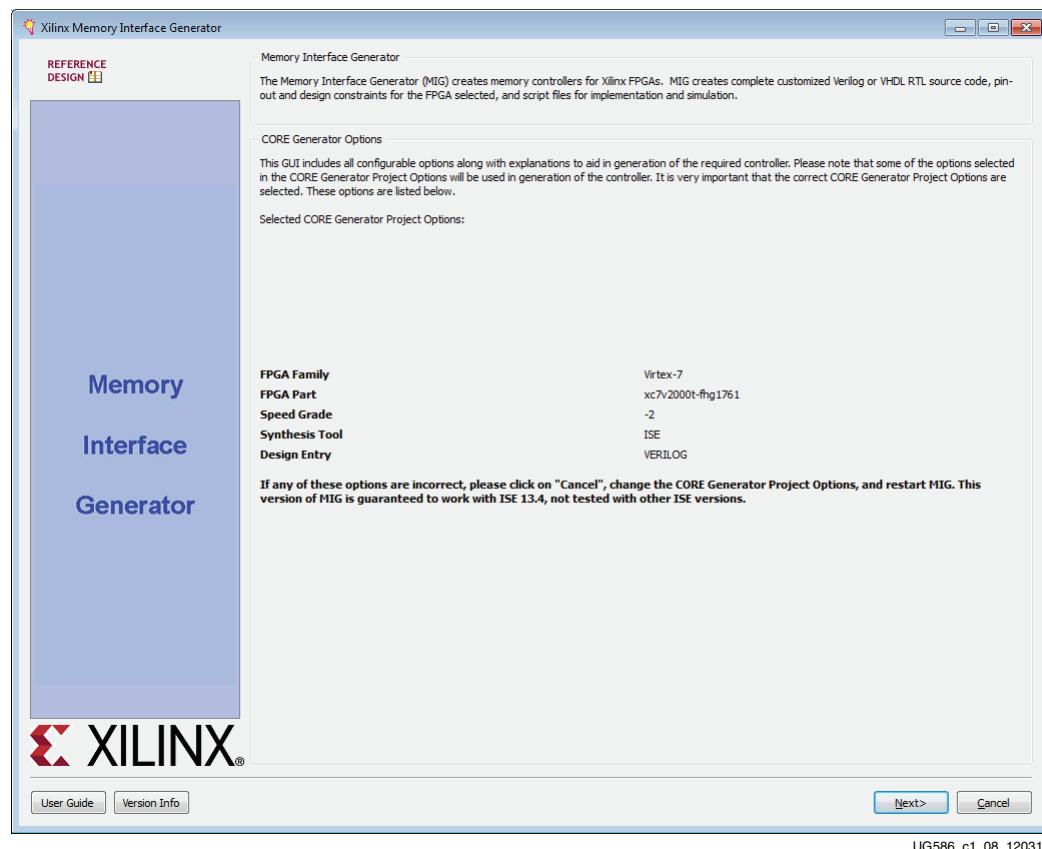
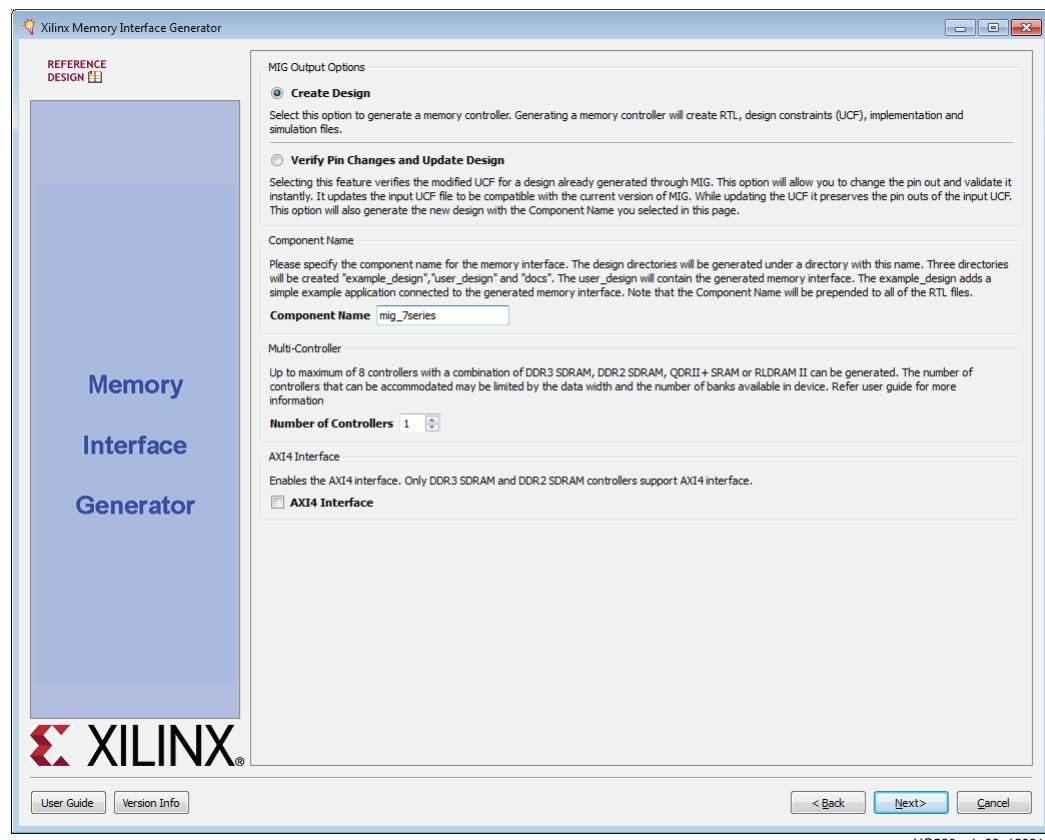


Figure 1-7: 7 Series FPGA Memory Interface Generator Front Page

8. Click **Next** to display the **Output Options** page.

## 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 1-8).
2. Choose the number of controllers to be generated. This option determines the replication of further pages.
3. DDR2 and DDR3 SDRAM designs support the memory-mapped AXI4 interface. The AXI4 interface is implemented in Verilog only. If an AXI4 interface is required, select the language as “Verilog” in the CORE Generator tool before invoking the MIG tool. If the AXI4 interface is not selected, the user interface (UI) is the primary interface. The `axi_7series_ddrx` IP from the EDK flow only supports DDR3 SDRAMs and has the AXI support always turned on.



**Figure 1-8: 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, the component name is corrected to be the IP instance name from XPS.

4. Click **Next** to display the **Pin Compatible FPGAs** page.

## Pin Compatible FPGAs

The Pin Compatible FPGAs page 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 1-9).

Xilinx 7 series devices using stacked silicon interconnect (SSI) technology have Super Logic Regions (SLRs). Memory interfaces cannot span across SLRs. If the device selected or a compatible device that is selected has SLRs, the MIG tool ensures that the interface does not cross SLR boundaries.

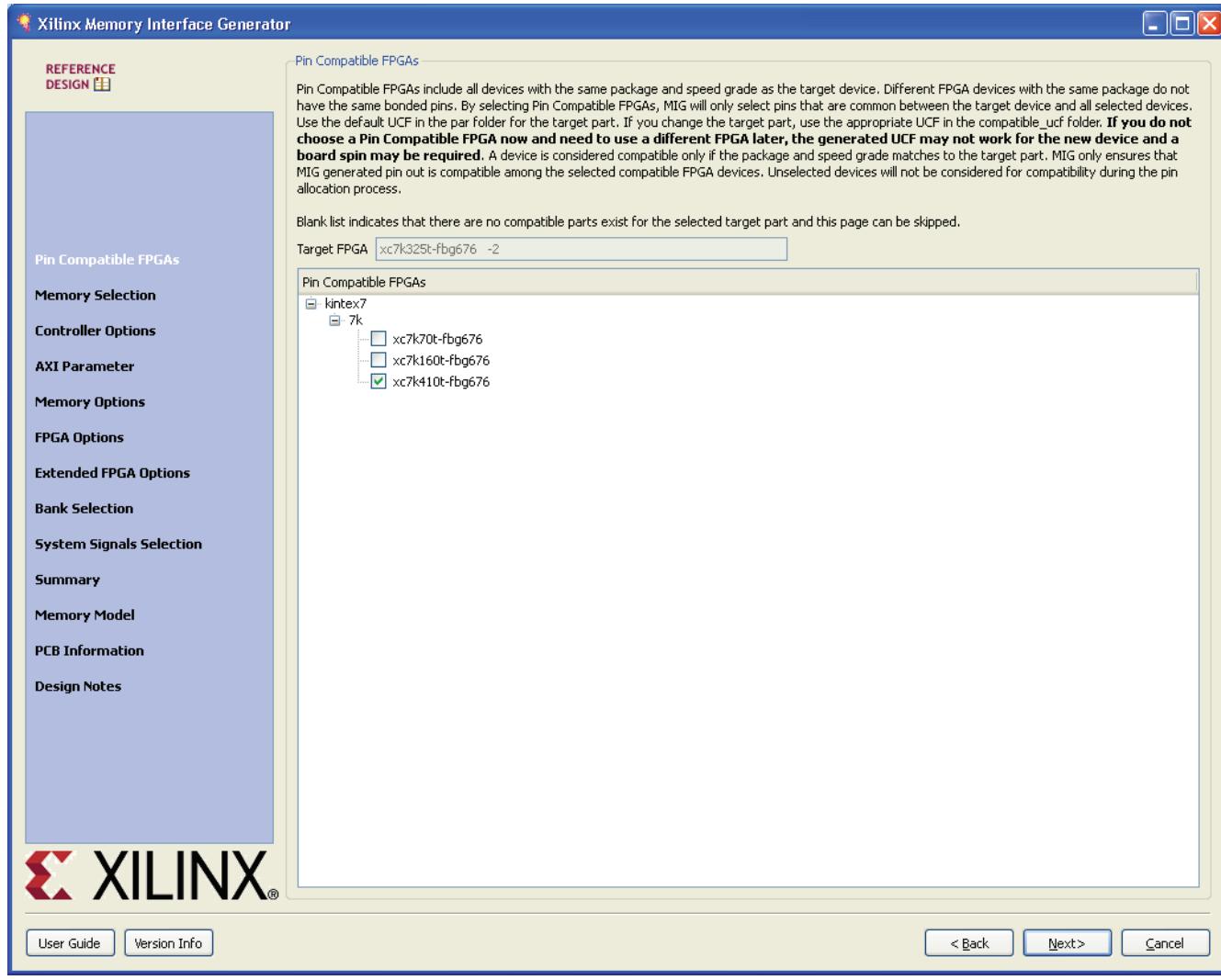


Figure 1-9: Pin-Compatible 7 Series FPGAs

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** page.

## Creating 7 Series 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** page (Figure 1-10).

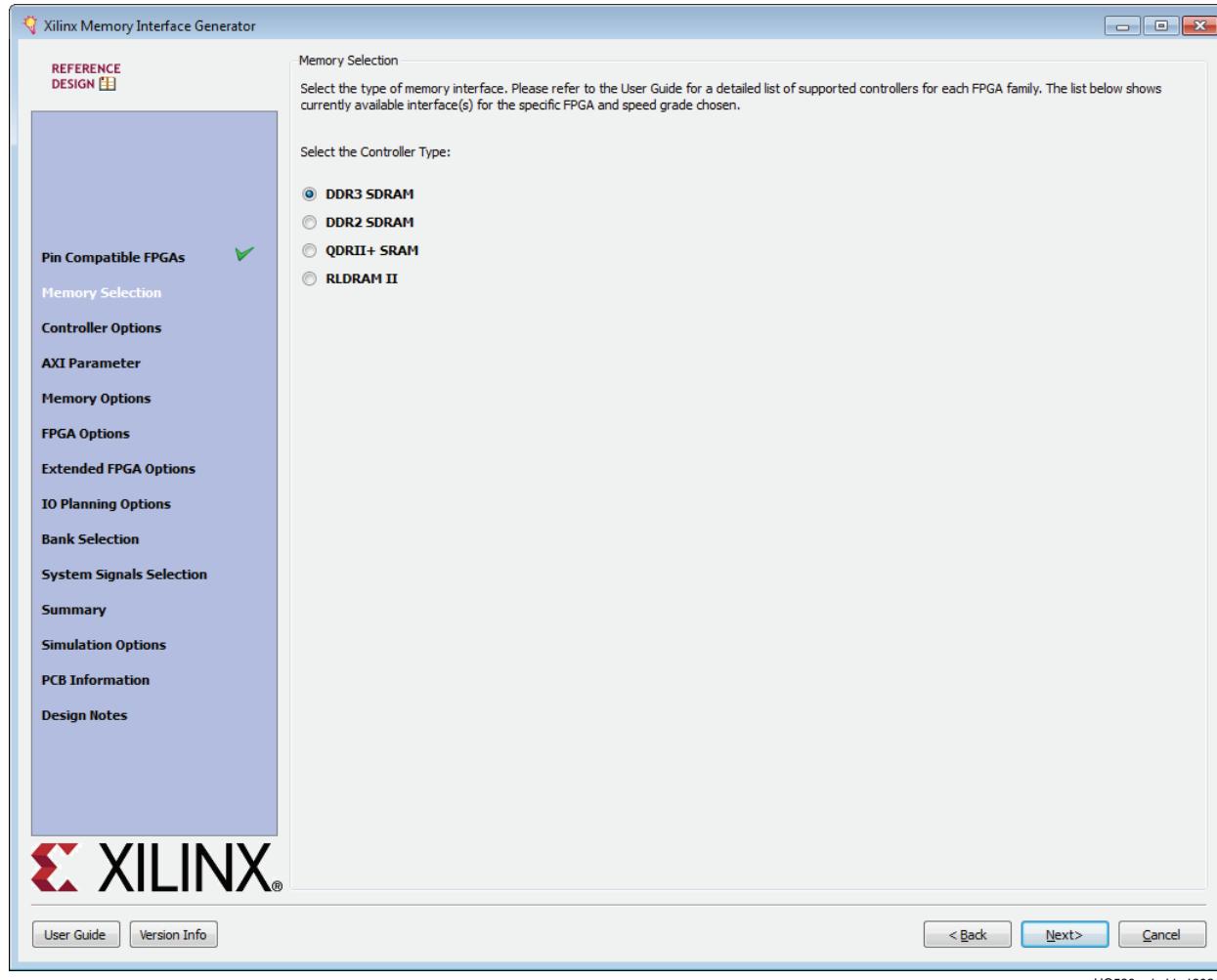


Figure 1-10: Memory Type and Controller Selection

## Controller Options

This page shows the various controller options that can be selected (Figure 1-11).

**Note:** The use of the memory controller is optional. The Physical Layer, or PHY, can be used without the memory controller. The memory controller RTL is always generated by the MIG tool, but this output need not be used. See [Physical Layer Interface \(Non-Memory Controller Design\), page 114](#) for more information. Controller only settings such as ORDERING are not needed in this case, and the defaults can be used. Settings pertaining to the PHY, such as the Clock Period, are used to set the PHY parameters appropriately.

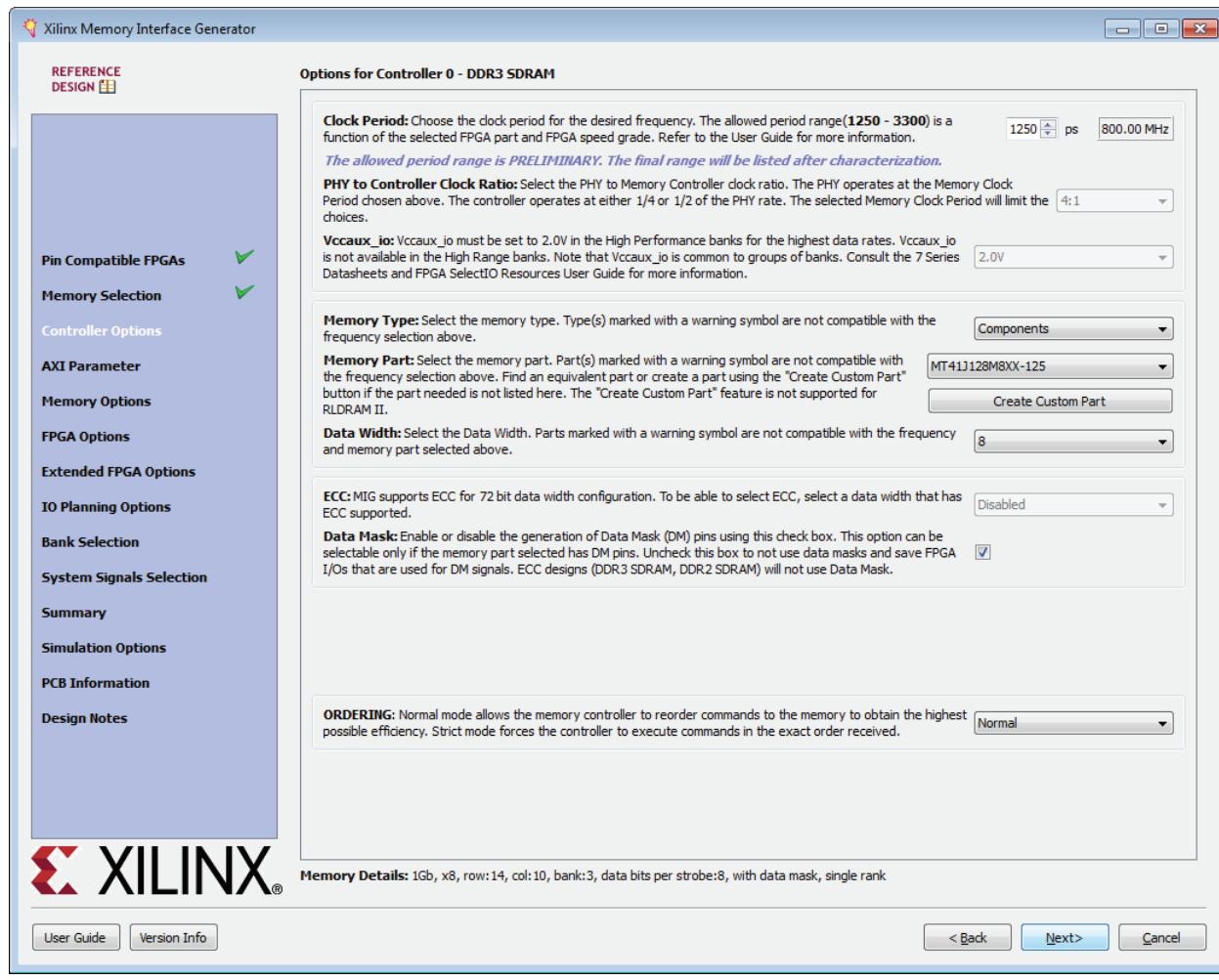


Figure 1-11: Controller Options Page

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 the controllers. 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.
- **Input Clock Period:** The desired input clock period is selected from the list. These values are determined by the memory clock period chosen and the allowable limits of the parameters. See [Design Guidelines, page 126](#) for more information on the PLL parameter limits.
- **PHY to Controller Clock Ratio:** This feature determines the ratio of the physical layer (memory) clock frequency to the controller and user interface clock frequency. The 2:1 ratio lowers the maximum memory interface frequency due to fabric timing limitations. The user interface data bus width of the 2:1 ratio is 4 times the width of the physical memory interface width, while the bus width of the 4:1 ratio is 8 times the physical memory interface width. The 2:1 ratio has lower latency. The 4:1 ratio is necessary for the highest data rates.
- **Vccaux\_io:** Vccaux\_io is set based on the period /frequency setting. 2.0V is required at the highest frequency settings in the High Performance column. The MIG tool automatically selects 2.0V when required. Either 1.8 or 2.0V can be used at lower frequencies. Groups of banks share the Vccaux\_io supply. See the *7 Series FPGAs SelectIO Resources User Guide* [Ref 1] and the *7 Series FPGAs Packaging and Pinout Specification* [Ref 2] for more information.
- **Memory Type:** This feature selects the type of memory parts used in the design.
- **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.
- **Data Width:** The data width value can be selected here based on the memory type selected earlier. 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.
- **Data Mask:** This option allocates data mask pins when selected. 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.
- **Ordering:** This feature allows the memory controller to reorder commands to improve the memory bus efficiency.
- **Memory Details:** The bottom of the Controller Options page ([Figure 1-11, page 17](#)) displays the details for the selected memory configuration ([Figure 1-12](#)).

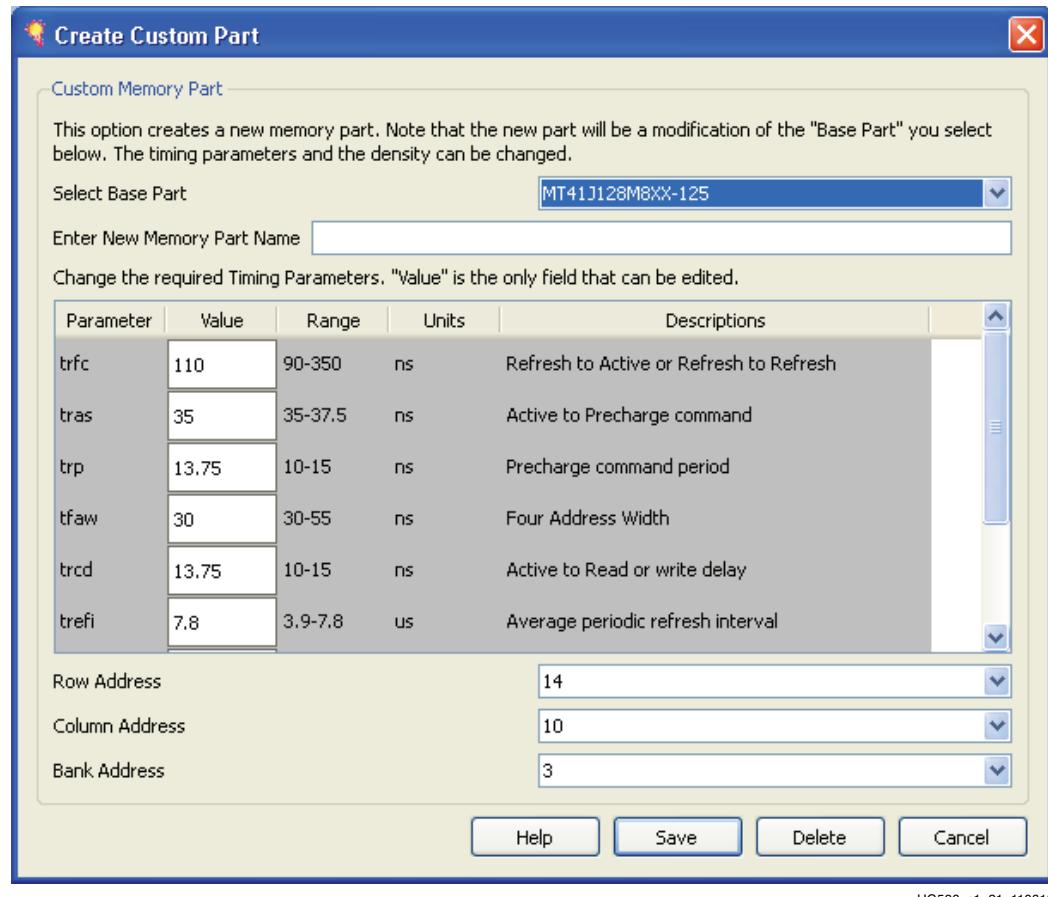
**Memory Details:** 1Gb, x8, row:14, col:10, bank:3, data bits per strobe:8, with data mask

UG586\_c1\_20\_091410

**Figure 1-12: Memory Details**

1. Select the appropriate frequency. 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. 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 page appears, as shown in [Figure 1-13](#).



*Figure 1-13: Create Custom Part*

The **Create Custom Part** page 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. It is also saved into the database for reuse and to produce the design.
8. Click **Next** to display the **Memory Options** page (or the AXI Parameter Options page 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 1-14). These are standard AXI parameters or parameters specific to the AXI4 interface. Details are available in the ARM® AMBA® specifications [Ref 3].

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.

- **Arbitration Scheme:** Selects the arbitration scheme between read and write address channels.

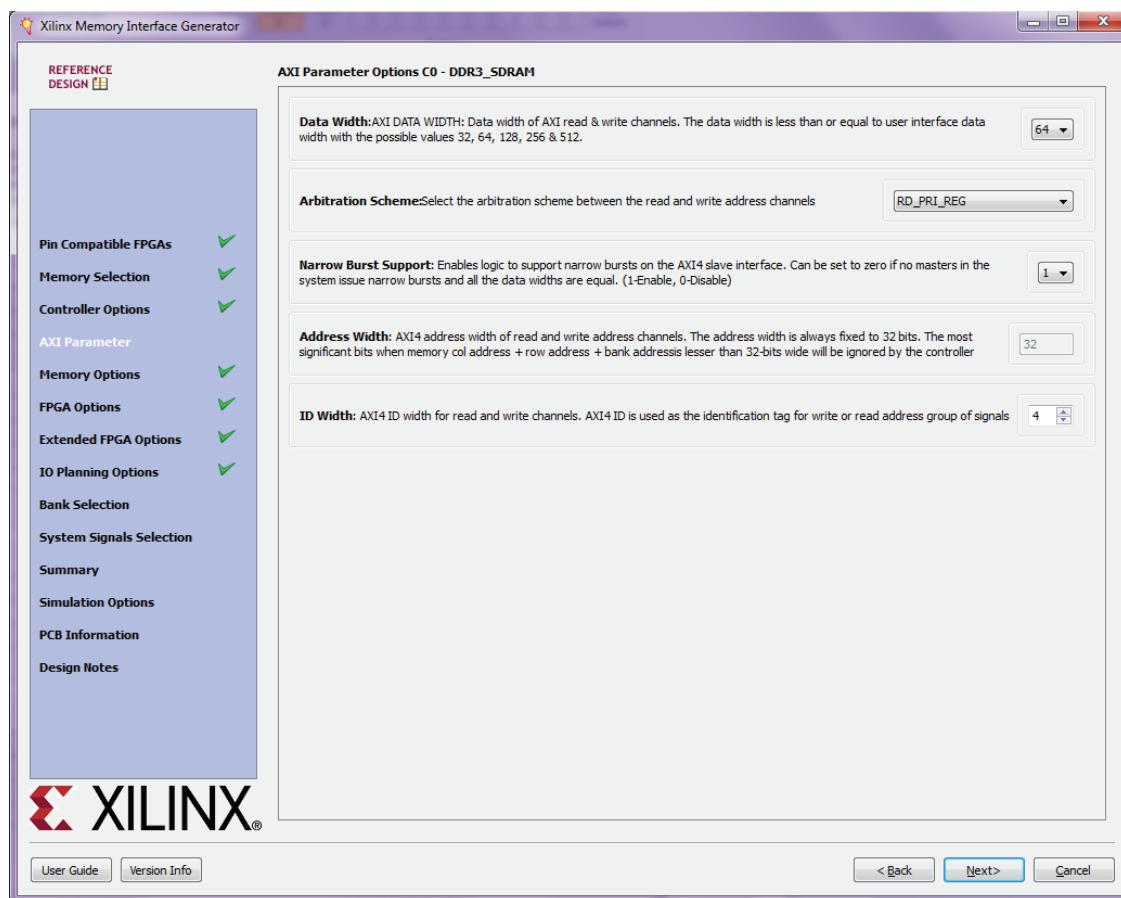


Figure 1-14: Setting AXI Parameter Options

UG586\_c1\_22\_090511

## Setting DDR3 Memory Parameter Option

This feature allows the selection of various memory mode register values, as supported by the controller's specification (Figure 1-15).

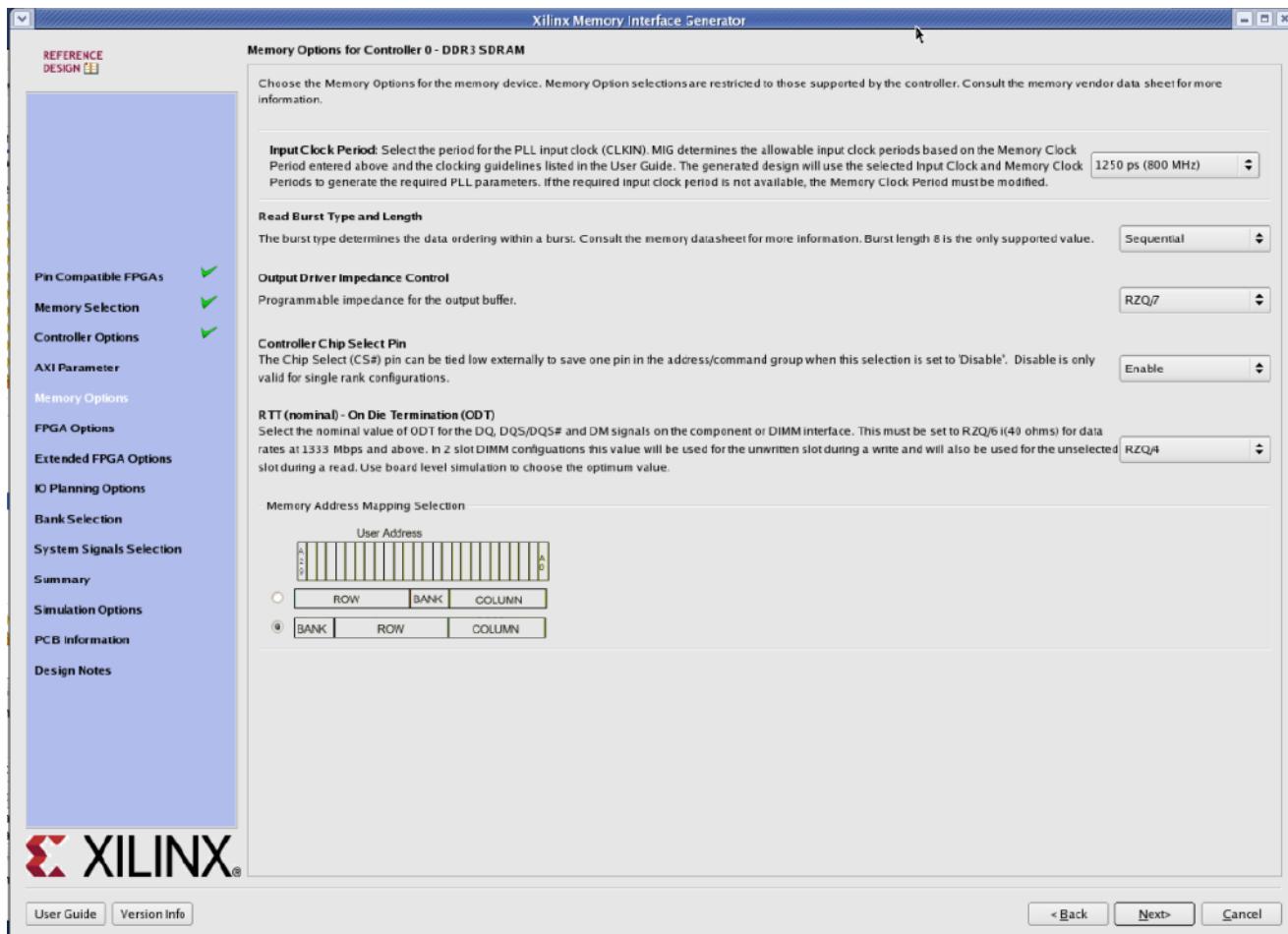


Figure 1-15: Setting Memory Mode Options

The mode register value is loaded into the load mode register during initialization. Only burst length 8 (BL8) is supported for DDR2 and DDR3 SDRAM.

Click **Next** to display the FPGA Options page.

## FPGA Options

Figure 1-16 shows the FPGA Options page.

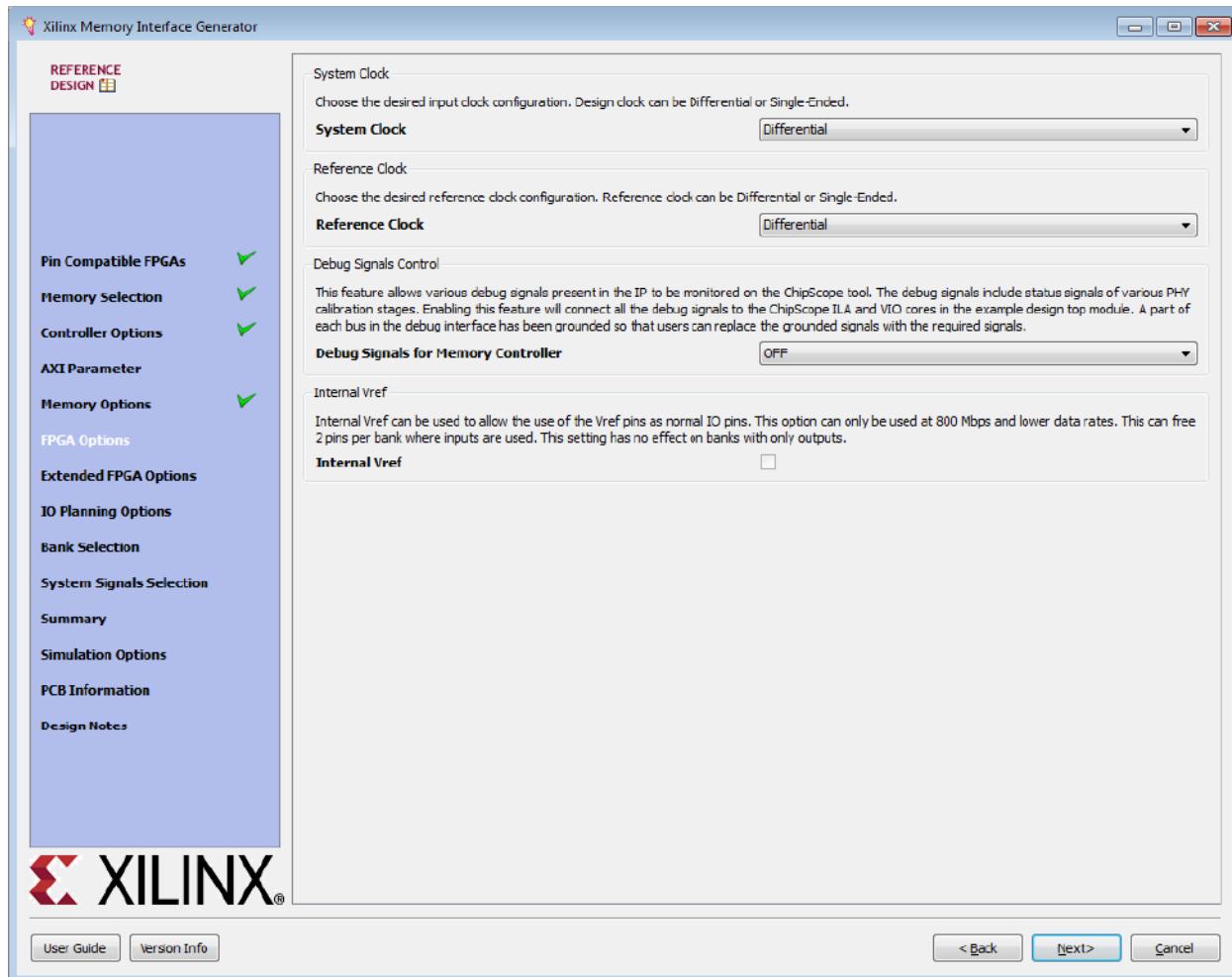


Figure 1-16: **FPGA Options**

- **System Clock.** This option (not available in the EDK flow) selects the clock type (Single-Ended or Differential) for the sys\_clk signal pair.
- **Reference Clock.** This option (not available in the EDK flow) selects the clock type (Single-Ended or Differential) for the ref\_clk signal pair.
- **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. 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 tool 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 and no ChipScope analyzer modules are instantiated in the design\_top module and no ChipScope analyzer modules are generated by the CORE Generator tool. The debug port is always disabled for functional simulations.

- **Internal Vref Selection.** Internal Vref can be used for data group bytes to allow the use of the VREF pins for normal I/O usage. Internal Vref should only be used for data rates of 800 Mb/s or below.

Click **Next** to display the DCI description page (Figure 1-17).

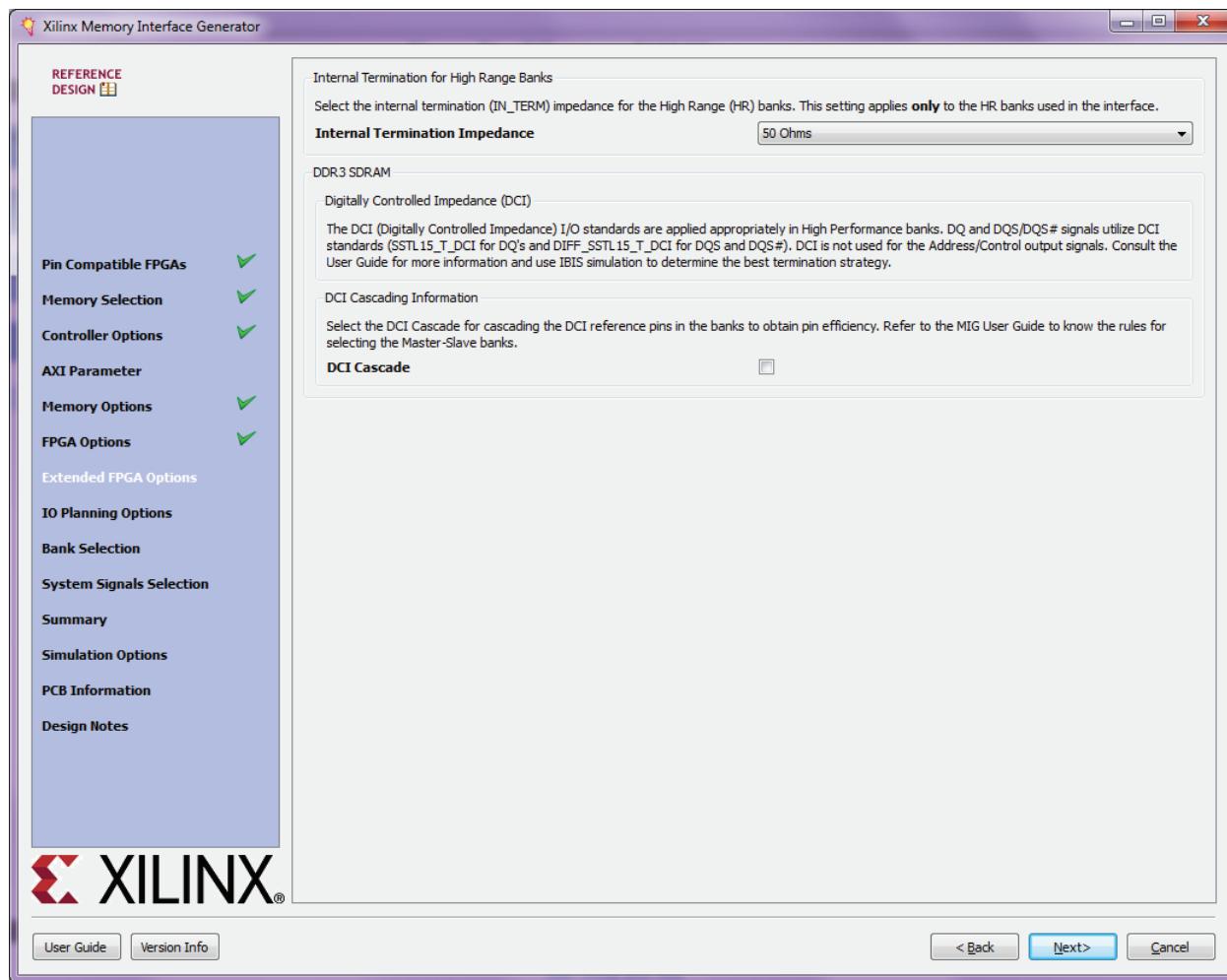


Figure 1-17: DCI Description

- **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. DCI is available in the High Performance Banks.
- **Internal Termination for High Range Banks.** The internal termination option can be set to 40, 50, or 60Ω or disabled. This selection is only for High Range banks.
- **DCI Cascade.** This selection enables the VRN/VRP pins that are available in High Performance banks to allocate for the address/control and reset\_n ports.

- **Pin/Bank Selection Mode.** This allows the user to specify an existing pinout and generate the RTL for this pinout, or pick banks for a new design. [Figure 1-18](#) shows the options for using an existing pinout. The user must assign the appropriate pins for each signal. A choice of each bank 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 against the MIG pinout rules. One cannot proceed until the MIG DRC has been validated by clicking the **Validate** button.

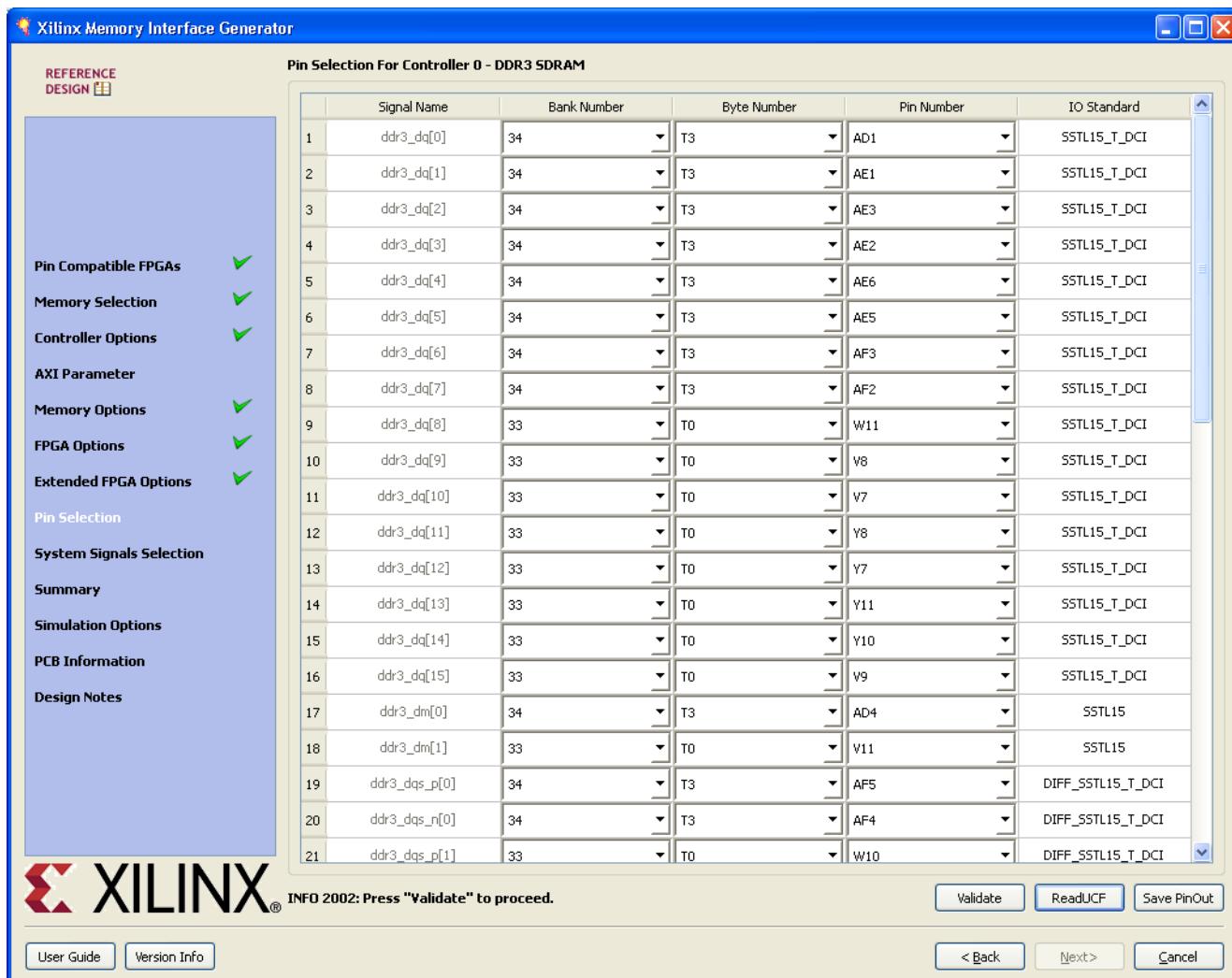


Figure 1-18: Pin/Bank Selection Mode

## Bank Selection

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

- Address and control signals
- Data signals

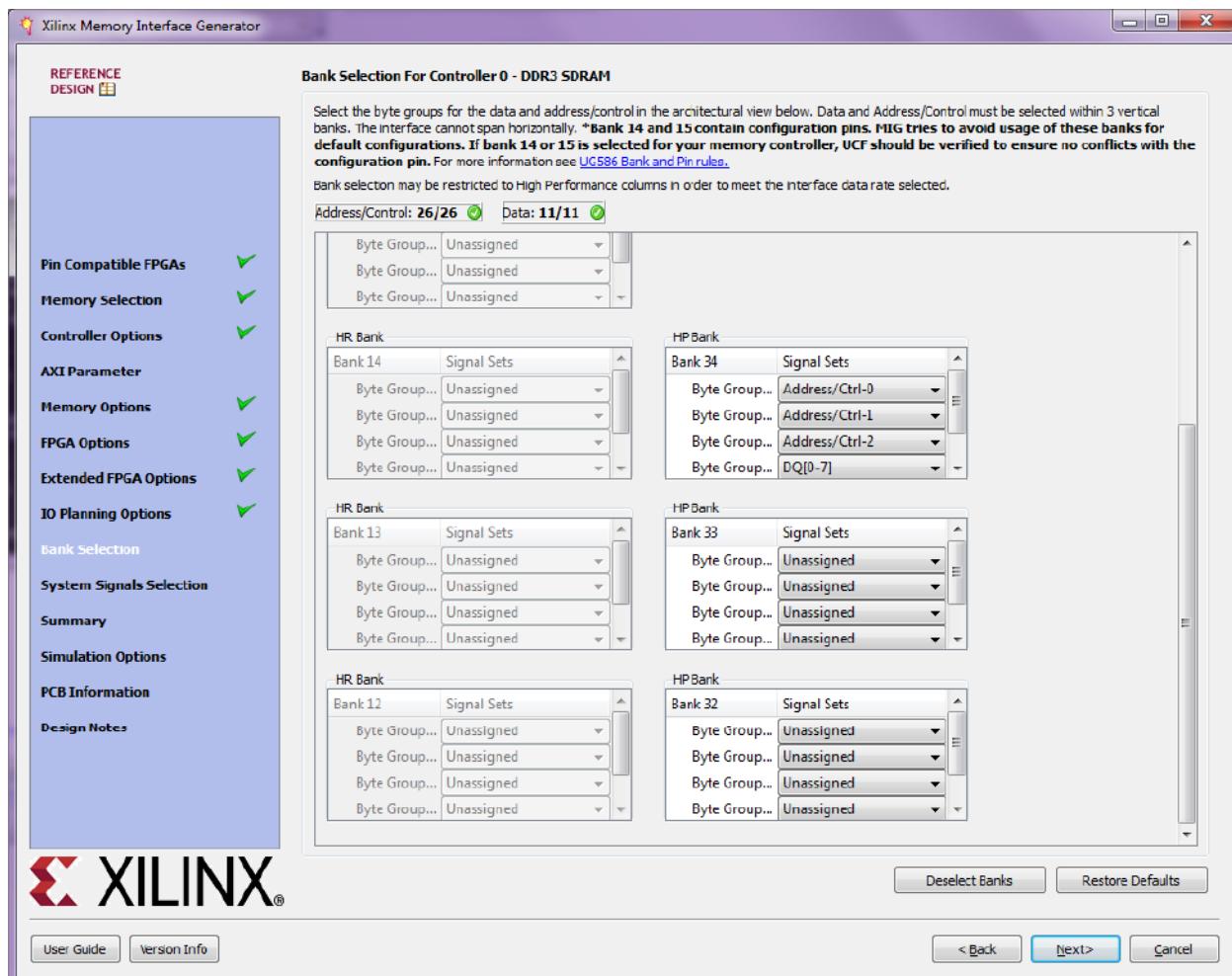


Figure 1-19: Bank Selection

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.

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

Vccaux\_io groups are shown for HP banks in devices with these groups using dashed lines. Vccaux\_io is common to all banks in these groups. The memory interface must have the same Vccaux\_io for all banks used in the interface. MIG automatically sets the VCCAUX\_IO constraint appropriately for the data rate requested.

For devices implemented with SSI technology, the SLRs are indicated by a number in the header in each bank, for example, SLR 1. Interfaces cannot span across Super Logic Regions.

Select the pins for the system signals on this page (Figure 1-20). The MIG tool allows the selection of either external pins or internal connections, as desired.

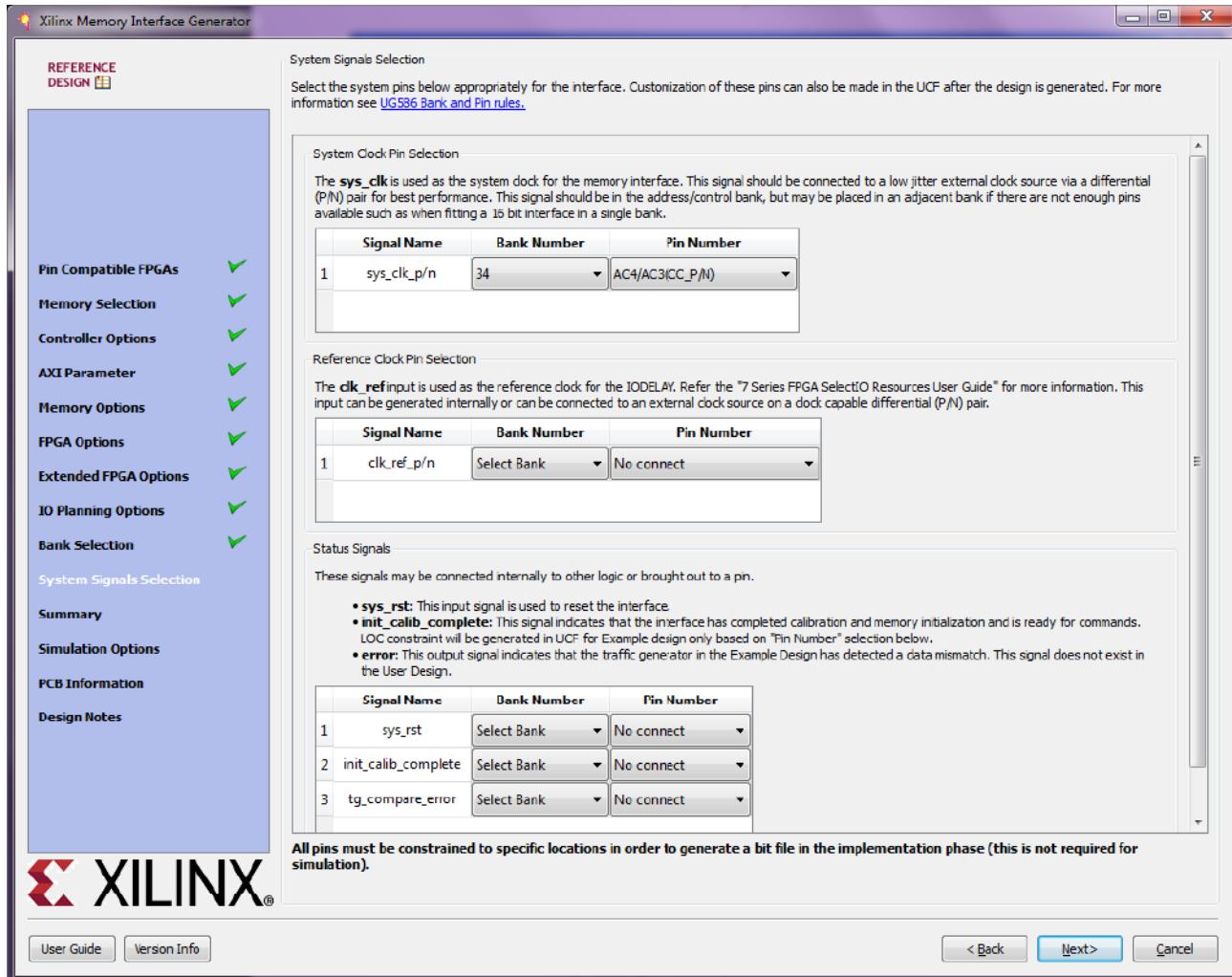


Figure 1-20: System Pins

- **sys\_clk:** This is the system clock input for the memory interface and is typically connected to a low-jitter external clock source. Either a single input or a differential pair can be selected based on the System Clock selection in the FPGA Options page (Figure 1-16). The sys\_clk input must be in the same column as the memory interface. If this pin is connected in the same banks as the memory interface, the MIG tool selects an I/O standard compatible with the interface, such as DIFF\_SSTL15 or SSTL15. If sys\_clk is not connected in a memory interface bank, the MIG tool selects an appropriate standard such as LVCMOS18 or LVDS. The UCF can be modified as desired after generation.
- **clk\_ref:** This is the reference frequency input for the IDELAY control. This is a 200 MHz input. The clk\_ref input can be generated internally or connected to an external source. A single input or a differential pair can be selected based on the System Clock selection in the FPGA Options page (Figure 1-16). The I/O standard is selected in a similar way as sys\_clk.

- **sys\_rst**: This is the system reset input that can be generated internally or driven from a pin. The MIG tool selects an appropriate I/O standard for the input such as SSTL15 if the input is within the interface banks, and LVCMOS18 if it is not.
- **init\_calib\_complete**: This output indicates that the memory initialization and calibration is complete and that the interface is ready to use. The init\_calib\_complete signal is normally only used internally, but can be brought out to a pin if desired.
- **tg\_compare\_error**: This output indicates that the traffic generator in the example design has detected a data compare error. This signal is only generated in the example design and is not part of the user design. This signal is not typically brought out to a pin but can be, if desired.

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

## Summary

This page provides the complete details about the 7 series FPGA memory core selection, interface parameters, CORE Generator tool options, and FPGA options of the active project ([Figure 1-21](#)). 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.

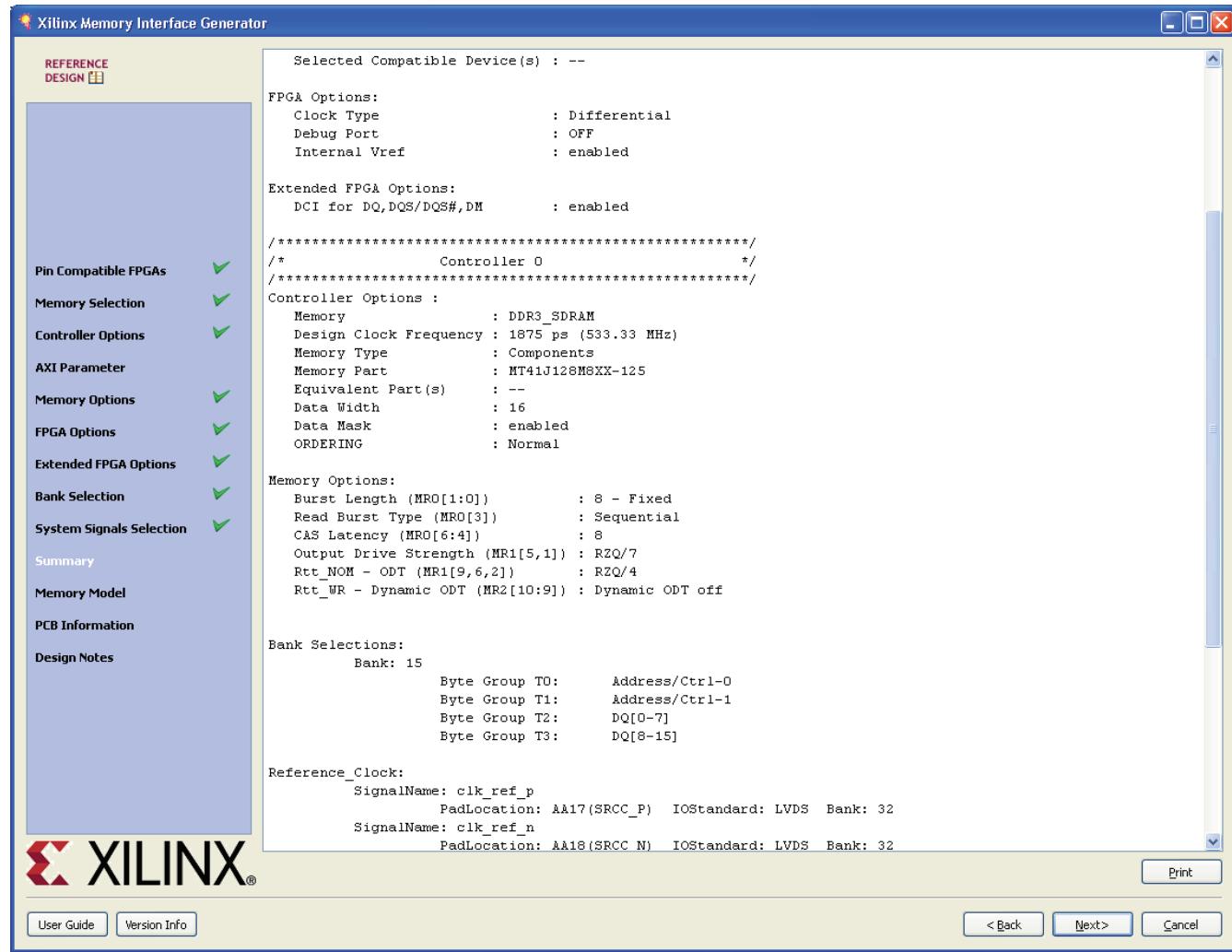


Figure 1-21: Summary

## 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 DDR2 or DDR3 SDRAMs. To access the models in the output sim folder, click the license agreement (Figure 1-22). 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 1-22: 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 1-23** shows the output directory structure of the selected memory controller (MC) design from the MIG tool. In the <component name> directory, three folders are created:

- docs
- example\_design
- user\_design

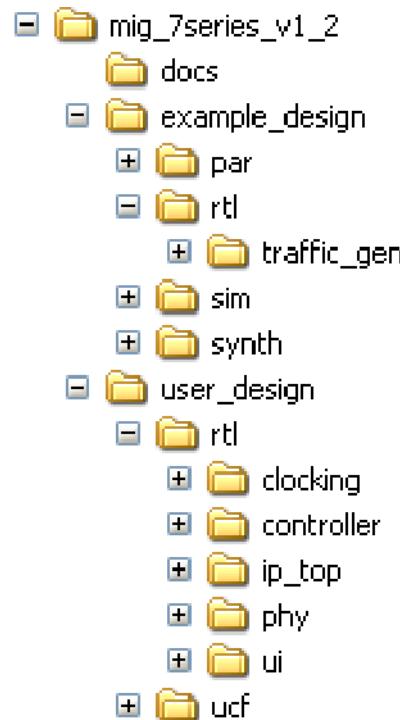


Figure 1-23: Directory Structure

## Directory and File Contents

The 7 series FPGAs 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.

#### **example\_design/rtl**

This directory contains the example design ([Table 1-1](#)).

**Table 1-1: Modules 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 7 series FPGAs memory interface core.

#### **example\_design/rtl/traffic\_gen**

This directory contains the traffic generator that provides the stimulus to the 7 series FPGAs memory controller ([Table 1-2](#)).

**Table 1-2: Modules in example\_design/rtl/traffic\_gen Directory**

Name	Description
memc_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.
memc_flow_vcontrol.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/vhd	This module generates timing control for writes and ready signals to mcb_flow_control.v/vhd.
s7ven_data_gen.v/vhd	This module generates different data patterns.
a_fifo.v/vhd	This is a synchronous FIFO using LUT RAMs.

**Table 1-2: Modules in example\_design/rtl/traffic\_gen Directory (Cont'd)**

Name	Description
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.
traffic_gen_top.v/vhd	This module is the top level of the traffic generator and comprises the memc_traffic_gen and init_mem_pattern_ctr modules.

**<component name>/example\_design/par**

**Table 1-3** lists the modules in the example\_design/par directory.

**Table 1-3: Modules in example\_design/par Directory**

Name	Description
example_top.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 tool 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 1-4** lists the modules in the example\_design/sim directory.

**Table 1-4: 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.
sim.do	This is the ModelSim simulator script file.
sim_tb_top.v/vhd	This is the simulation top file.

**<component name>/user\_design**

The user\_design folder contains two folders, namely, rtl, ucf and the user top level module <component\_name>.v/vhd. This top-level module serves as an example for connecting the user design to the 7 series FPGA memory interface core.

**`user_design/rtl/clocking`**

This directory contains the user design ([Table 1-5](#)).

*Table 1-5: Modules in `user_design/rtl/clocking` Directory*

Name	Description
<code>clk_ibuf.v/vhd</code>	This module instantiates the input clock buffer.
<code>iodelay_ctrl.v/vhd</code>	This module instantiates IDELAYCNTRL primitives needed for IDELAY use.
<code>infrastructure.v/vhd</code>	This module helps in clock generation and distribution, and reset synchronization.

**`user_design/rtl/controller`**

This directory contains the memory controller that is instantiated in the example design ([Table 1-6](#)).

*Table 1-6: Modules in `user_design/rtl/controller` Directory*

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

**`user_design/rtl/ip_top`**

This directory contains the user design ([Table 1-7](#)).

***Table 1-7: Modules in user\_design/rtl/ip\_top Directory***

Name	Description
<code>mem_intf.v/vhd</code>	This is the top-level memory interface block that instantiates the controller and the PHY.
<code>memc_ui_top.v/vhd</code>	This is the top-level memory controller module.

**`user_design/rtl/phy`**

This directory contains the 7 series FPGA memory interface PHY implementation ([Table 1-8](#)).

***Table 1-8: Modules in user\_design/rtl/phy Directory***

Name	Description
<code>ddr_byte_group_io</code>	This module contains the parameterizable I/O logic instantiations and the I/O terminations for a single byte lane.
<code>ddr_byte_lane</code>	This module contains the primitive instantiations required within an output or input byte lane.
<code>ddr_calib_top</code>	This is the top-level module for the memory physical layer interface.
<code>ddr_if_post_fifo</code>	This module extends the depth of a PHASER IN_FIFO up to four entries.
<code>ddr_mc_phy</code>	This module is a parameterizable wrapper instantiating up to three I/O banks, each with 4-lane PHY primitives.
<code>ddr_mc_phy_wrapper</code>	This wrapper file encompasses the MC_PHY module instantiation and handles the vector remapping between the MC_PHY ports and the user's DDR2 or DDR3 ports.
<code>ddr_of_pre_fifo</code>	This module extends the depth of a PHASER OUT_FIFO up to four entries.
<code>ddr_phy_4lanes</code>	This module is the parameterizable 4-lane PHY in an I/O bank.
<code>ddr_phy_ck_addr_cmd_delay</code>	This module contains the logic to provide the required delay on the address and control signals.
<code>ddr_phy_dqs_delay</code>	This module contains the DQS to DQ phase offset logic.
<code>ddr_phy_dqs_found_cal</code>	This module contains the Read leveling calibration logic (PHASER_IN DQSFOUND calibration logic).
<code>ddr_phy_init</code>	This module contains the memory initialization and overall master state control during initialization and calibration.
<code>ddr_phy_rdlvl</code>	This module contains the Read leveling Stage1 calibration logic (Window detection with PRBS pattern).
<code>ddr_phy_top</code>	This is the top-level module for the physical layer.
<code>ddr_phy_wrcal</code>	This module contains the write calibration logic.

**Table 1-8: Modules in user\_design/rtl/phy Directory (Cont'd)**

Name	Description
ddr_phy_wrlvl	This module contains the write leveling logic.
ddr_prbs_gen	This PRBS module uses a many-to-one feedback mechanism for 2n sequence generation.

### **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 1-9](#)).

**Table 1-9: 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.
ui_top.v/vhd	This is the top level of the memory controller user interface.

### **<component\_name>/user\_design/ucf**

[Table 1-10](#) lists the modules in the user\_design/ucf directory.

**Table 1-10: Modules in user\_design/ucf Directory**

Name	Description
<component_name>.ucf	This is the UCF for the core and the user design.

## Verify Pin Changes and Update Design

This feature verifies the input UCF for bank selections, byte selections, and pin allocation. It also generates errors and warnings in a separate dialog when the user clicks on the **Validate** button on the page. This feature is useful to verify the UCF for any pinout changes made 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, and the UCF that needs to be verified. In the CORE Generator tool, the recustomization option should be selected to reload the project. The design is allowed to generate only when the MIG DRC is met. Ignore warnings about validating the pinout, which is the intent. Just validating the UCF is not sufficient; it is mandatory to proceed with design generation to get the UCF with updated clock and phaser related constraints and RTL top-level module for various updated Map parameters.

Here are the rules verified from the input UCF:

- If a pin is allocated to more than one signal, the tool reports an error. Further verification is not done if the UCF does not adhere to the uniqueness property.
- Verified common rules:
  - The interface can span across a maximum of three consecutive banks.
  - Interface banks should reside in the same column of the FPGA.
  - Interface banks should be either High Performance (HP) or High Range (HR). HP banks are used for the high frequencies.

- The chosen interface banks should have the same SLR region if the chosen device is of stacked silicon interconnect technology.
- VREF I/Os should be used as GPIOs when an internal VREF is used or if there are no inout and input ports in a bank.
- The I/O standard of each signal is verified as per the configuration chosen.
- The VCCAUX I/O of each signal is verified and provides a warning message if the provided VCCAUX I/O is not valid.
- Verified data pin rules:
  - Pins related to one strobe set should reside in the same byte group.
  - The strobe pair (DQS) should be allocated to the DQSCC I/O pair.
  - An FPGA byte lane should not contain pins related to two different strobe sets.
  - VREF I/O can be used only when the internal VREF is chosen.
- Verified address pin rules:
  - Address signals cannot mix with data bytes except for the ddr3\_reset\_n signal for DDR3 SDRAM interfaces.
  - Address signals cannot mix with data bytes except for the ddr2\_reset\_n signal for DDR2 SDRAM interfaces. The ddr2\_reset\_n port exists for RDIMMs only.
  - It can use any number of isolated byte lanes
- Verified system pin rules:
  - System clock:
    - These pins should be allocated to either SR/MR CC I/O pair.
    - These pins must be allocated in the Memory banks column.
    - If the selected system clock type is single-ended, need to check whether the reference voltage pins are unallocated in the bank or internal VREF is used
  - Reference clock:
    - These pins should be allocated to either SR/MR CC I/O pair.
    - If the selected system clock type is single-ended, need to check whether the reference voltage pins are unallocated in the bank or internal VREF is used.
  - Status signals:
    - The sys\_rst signal should be allocated in the bank where the VREF I/O is unallocated or internal VREF is used.
    - These signals should be allocated in the non-memory banks because the I/O standard is not compatible. The I/O standard type should be LVCMOS with at least 1.8V.
    - These signals can be allocated in any of the columns (there is no hard requirement because these signals should reside in a memory column); however, it is better to allocate closer to the chosen memory banks.

## 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. See this file for the recommended build options for the design.

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

The MIG tool provides a synthesizable test bench to generate various traffic data patterns to the memory controller (MC). 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 1-24](#).

`ddr2_sim_tb_Top` or `Ddr3_sim_tb_Top`

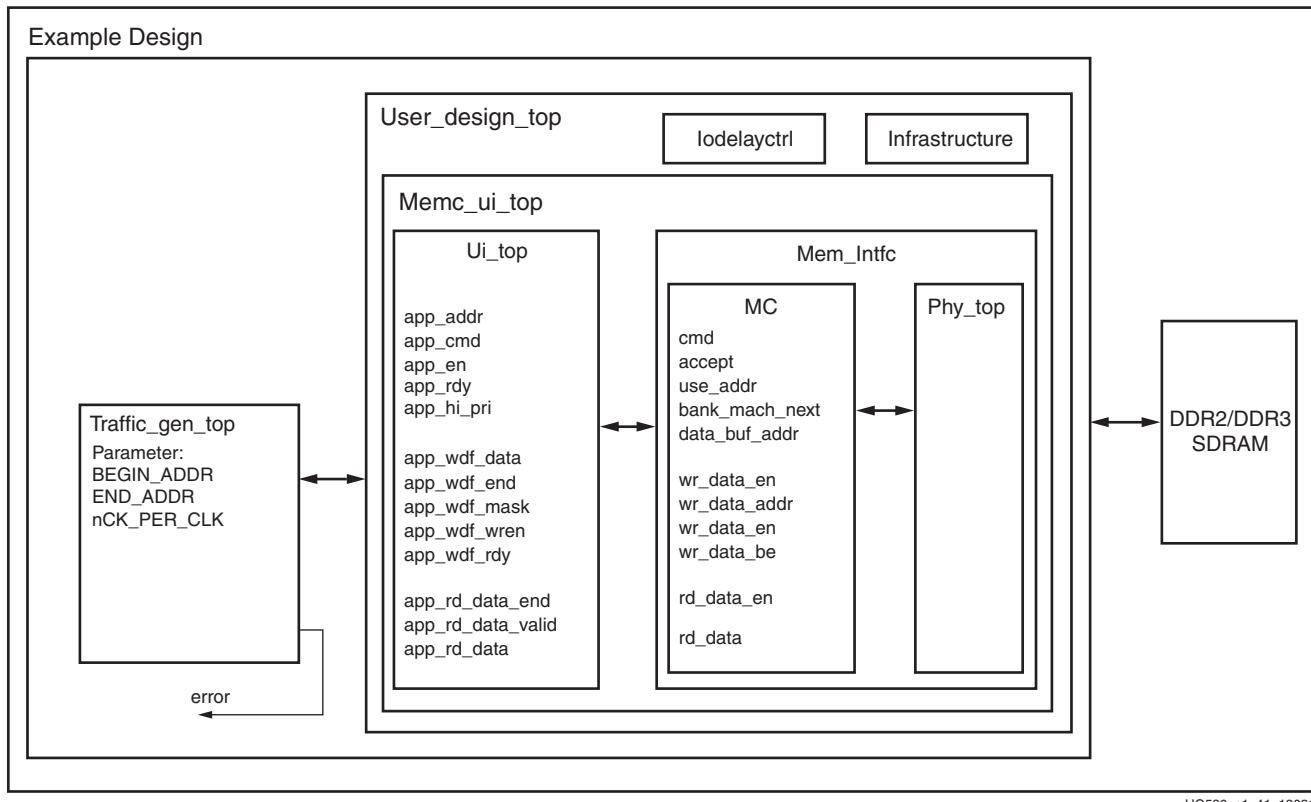


Figure 1-24: Synthesizable Example Design Block Diagram

Figure 1-25 shows the simulation result of a simple read and write transaction between the tb\_top and memc\_intfc modules.

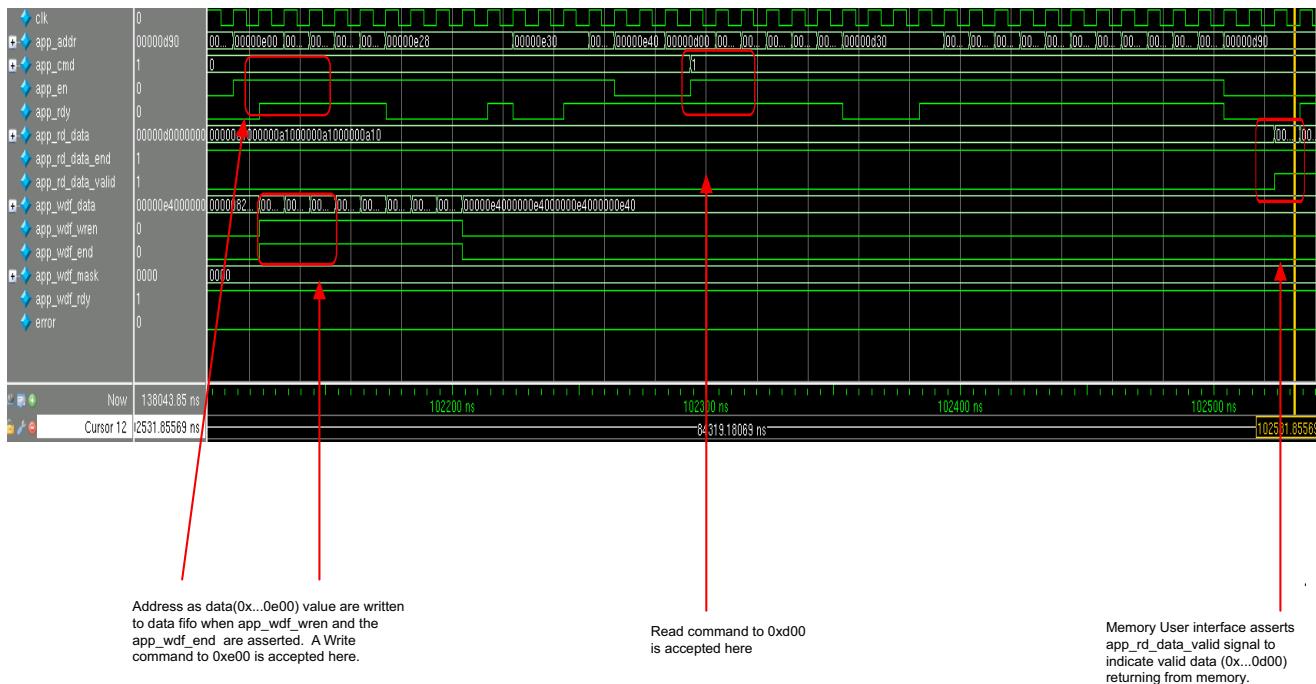


Figure 1-25: User Interface Read and Write Cycle

### 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.

## 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 1-11](#) describes these parameters.

**Table 1-11: 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 "VIRTEX7".
MEMORY_TYPE	Indicate the memory controller type.	"DDR2", "DDR3"
nCK_PER_CLK	This is the memory controller clock to DRAM clock ratio.	This must be set to 4.
NUM_DQ_PINS	The is the total 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.
MEM_BURST_LEN	This is the memory data burst length.	This must be set to 8.
MEM_COL_WIDTH	This is the number of memory column address bits.	This option is based on the selected memory device.
DATA_WIDTH	This is the user interface data bus width.	For nCK_PER_CLK = 4, DATA_WIDTH = NUM_DQ_PINS * 8.
ADDR_WIDTH	This is the memory address bus width. It is equal to RANK_WIDTH + BANK_WIDTH + ROW_WIDTH + COL_WIDTH.	
MASK_SIZE	This parameter specifies the mask width in the user interface data bus.	
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_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 1-11: 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.	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• CGEN_FIXED: The address, burst length, and instruction are taken directly from the fixed_addr_i, fixed_bl_i, and fixed_instr_i inputs.</li> <li>• CGEN_SEQUENTIAL: The address is incremented sequentially, and the increment is determined by the data port size.</li> <li>• 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.</li> <li>• CGEN_ALL (default): This option turns on all of the options above and allows addr_mode_i, instr_mode_i, and bl_mode_i to select the type of generation during run time.</li> </ul>

Table 1-11: Traffic Generator Parameters Set in the example\_top Module (Cont'd)

Parameter	Parameter Description	Parameter Value
DATA_PATTERN	This parameter sets the data pattern circuits to be generated through rtl logic. For larger devices, the DATA_PATTERN can be set to "DGEN_ALL", enabling all supported data pattern circuits to be generated. In hardware, the data pattern is selected and/or changed using vio_data_value_mode. The pattern can only be changed when DATA_PATTERN is set to DGEN_ALL.	<p>Valid settings for this parameter are:</p> <ul style="list-style-type: none"> <li>• ADDR (default): The address is used as a data pattern.</li> <li>• 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.</li> <li>• WALKING1: Walking 1s are on the DQ pins and the starting position of 1 depends on the address value.</li> <li>• WALKING0: Walking 0s are on the DQ pins and the starting position of 0 depends on the address value.</li> <li>• NEIGHBOR: The Hammer pattern is on all DQ pins except one. The address determines the exception pin location.</li> <li>• PRBS: A 32-stage LFSR generates random data and is seeded by the starting address.</li> <li>• DGEN_ALL: This option turns on all available options:           <ul style="list-style-type: none"> <li>0x1: FIXED - 32 bits of fixed_data.</li> <li>0x2: ADDRESS - 32 bits address as data.</li> <li>0x3: HAMMER</li> <li>0x4: SIMPLE8 - Simple 8 data pattern that repeats every 8 words.</li> <li>0x5: WALKING1s - Walking 1s are on the DQ pins.</li> <li>0x6: WALKING0s - Walking 0s are on the DQ pins.</li> <li>0x7: PRBS - A 32-stage LFSR generates random data.</li> <li>0x9: SLOW HAMMER - This is the slow MHz hammer data pattern.</li> <li>0xF: PHY_CALIB pattern - 0xFF, 00, AA, 55, 55, AA, 99, 66. This mode only generates READ commands at address zero.</li> </ul> </li> </ul>
CMDS_GAP_DELAY	This parameter allows pause delay between each user burst command.	Valid values: 0 to 32.
SEL_VICTIM_LINE	Select a victim DQ line whose state is always at logic High.	<p>This parameter only applies to the Hammer pattern. Valid settings for this parameter are 0 to NUM_DQ_PINS.</p> <p>When value = NUM_DQ_PINS, all DQ pins have the same Hammer pattern.</p>
EYE_TEST	Force the traffic generator to only generate writes to a single location, and no read transactions are generated.	<p>Valid settings for this parameter are "TRUE" and "FALSE".</p> <p>When set to "TRUE", any settings in vio_instr_mode_value are overridden.</p>

**Note:** The traffic generator might support more options than are available in the 7 series memory controller. The settings must match supported values in the memory controller.

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\_ctrl 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 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.
    - 4: 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.
    - 5: DGEN\_WALKING1. Walking 1s are on the DQ pins. The starting position of 1 depends on the address value.
    - 6: DGEN\_WALKING0. Walking 0s 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. This data mode only works with PRBS address mode or Sequential address mode.

## Modifying Port Address Space

The address space for a port can be 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 1-12](#) shows some examples of setting the two mask parameters.

*Table 1-12: Example Settings for Address Space and PRBS Masks*

SADDR	EADDR	PRBS_SADDR_MASK_POS	PRBS_EADDR_MASK_POS
0x1000	0xFFFF	0x00001000	0xFFFFF0000
0x2000	0xFFFF	0x00002000	0xFFFFF0000
0x3000	0xFFFF	0x00003000	0xFFFFF0000
0x4000	0xFFFF	0x00004000	0xFFFFF0000
0x5000	0xFFFF	0x00005000	0xFFFFF0000
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 1-13](#).

**Table 1-13: Traffic Generator Signal Descriptions**

Signal Name	Direction	Description
clk_i	Input	This signal is the clock input.
memc_init_done	Input	This is the input status signal from the memory controller to indicate that it is ready accept traffic.
manual_clear_error	Input	Input signal to clear error flag.
memc_cmd_addr_o[31:0]	Output	Start address for current transaction.
memc_cmd_en_o	Output	This active-High signal is the write-enable signal for the Command FIFO.
memc_cmd_full_i	Input	This connects to inversion of app_rdy of memory controller. When this input signal is asserted, TG continues to assert the memc_cmd_en_o, memc_cmd_addr_o value and memc_cmd_instr until the memc_cmd_full_i is deasserted.
memc_cmd_instr[2:0]	Output	Command code for current instruction. Command Write: 3'b000 Command Read: 3'b001
memc_rd_data_i[DWIDTH-1:0]	Input	Read data value returning from memory.
memc_rd_empty_i	Input	This active-High signal is the empty flag for the Read Data FIFO in memory controller. It indicates there is no valid data in the FIFO.
memc_rd_en_o	Output	This signal is only used in MCB-like interface.
memc_wr_data_o[DWIDTH-1:0]	Output	Write data value to be loaded into Write Data FIFO in memory controller.
memc_wr_en_o	Output	This active-High signal is the write enable for the Write Data FIFO. It indicates that the value on memc_wr_data is valid.
memc_wr_full_i	Input	This active-High signal is the full flag for the Write Data FIFO from memory controller. When this signal is High, TG holds the write data value and keeps assertion of memc_wr_en until the memc_wr_full_i goes Low.
qdr_wr_cmd_o	Output	This signal is only used to send write commands to the QDRII+ user interface.
vio_modify_enable	Input	Allow vio_xxxx_mode_value to alter traffic pattern.

Table 1-13: Traffic Generator Signal Descriptions (Cont'd)

Signal Name	Direction	Description
vio_data_mode_value[3:0]	Input	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• 0x0: Reserved.</li> <li>• 0x1: FIXED - 32 bits of fixed_data as defined through fixed_data_i inputs.</li> <li>• 0x2: ADDRESS - 32 bits address as data.</li> <li>• 0x3: HAMMER - All 1s are on DQ pins during the rising edge of DQS, and all 0s are on the DQ pins during the falling edge of DQS, except the VICTIM line as defined in the parameter "SEL_VICTIM_LINE". This option is only valid if parameter DATA_PATTERN = "DGEN_HAMMER" or "DGEN_ALL".</li> <li>• 0x4: SIMPLE8 - Simple 8 data pattern that repeats every 8 words. The patterns can be defined by the "simple_datax" inputs.</li> <li>• 0x5: WALKING1s - Walking 1s are on the DQ pins. The starting position of 1 depends on the address value. This option is only valid if the parameter DATA_PATTERN = "DGEN_WALKING" or "DGEN_ALL".</li> <li>• 0x6: WALKING0s - Walking 0s are on the DQ pins. The starting position of 0 depends on the address value. This option is only valid if the parameter DATA_PATTERN = "DGEN_WALKING0" or "DGEN_ALL".</li> <li>• 0x7: PRBS - A 32-stage LFSR generates random data and is seeded by the starting address. This option is only valid if the parameter DATA_PATTERN = "DGEN_PRBS" or "DGEN_ALL".</li> <li>• 0x9: SLOW HAMMER - This is the slow MHz hammer data pattern.</li> <li>• 0xF: PHY_CALIB pattern - 0xFF, 00, AA, 55, 55, AA, 99, 66. This mode only generates READ commands at address zero. This is only valid in the Virtex®-7 family.</li> </ul>
vio_addr_mode_value[2:0]	Input	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• 0x1: FIXED address mode. The address comes from the fixed_addr_i input bus.</li> <li>• 0x2: 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.</li> <li>• 0x3: SEQUENTIAL address mode. The address is generated from the internal address counter. The increment is determined by the user interface port width.</li> </ul>
vio_instr_mode_value[3:0]	Input	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• 0x1: Command type (read/write) as defined by fixed_instr_i.</li> <li>• 0x2: Random read/write commands.</li> <li>• 0xE: Write only at address zero.</li> <li>• 0xF: Read only at address zero.</li> </ul>
vio_bl_mode_value[3:0]	Input	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• 0x1: Fixed burst length as defined in the fixed_bl_i inputs.</li> <li>• 0x2: The user burst length is generated from the internal PRBS generator. Each burst value defines the number of back-to-back commands that are generated.</li> </ul>

Table 1-13: Traffic Generator Signal Descriptions (Cont'd)

Signal Name	Direction	Description
vio_fixed_instr_value	Input	Valid settings are: <ul style="list-style-type: none"><li>• 0x0: Write instruction</li><li>• 0x1: Read instruction</li></ul>
vio_fixed_bl_value	Input	Valid settings are 1 to 256.
vio_pause_traffic	Input	Pause traffic generation on the fly.
vio_data_mask_gen	Input	This mode is only used if the data mode pattern is <i>address as data</i> . If this is enabled, a random memc_wr_mask is generated after the memory pattern has been filled in memory. The write data byte lane is jammed with 8'hFF if the corresponding memc_write_mask is asserted.
cmp_data[DWIDTH - 1:0]	Output	Expected data to be compared with read back data from memory.
cmp_data_valid	Output	Compare data valid signal.
cmp_error	Output	This compare error flag asserts whenever cmp_data is not the same as the readback data from memory.
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: <ul style="list-style-type: none"><li>• [31:0]: Read start address</li><li>• [37:32]: Read burst length</li><li>• [39:38]: Reserved</li><li>• [40]: mcb_cmd_full</li><li>• [41]: mcb_wr_full</li><li>• [42]: mcb_rd_empty</li><li>• [64 + (DWIDTH - 1):64]: expected_cmp_data</li><li>• [64 + (2*DWIDTH - 1):64 + DWIDTH]: read_data</li></ul>
simple_data0[31:0]	Input	User-defined simple data 0 for simple 8 repeat data pattern.
simple_data1[31:0]	Input	User-defined simple data 1 for simple 8 repeat data pattern.
simple_data2[31:0]	Input	User-defined simple data 2 for simple 8 repeat data pattern.
simple_data3[31:0]	Input	User-defined simple data 3 for simple 8 repeat data pattern.
simple_data4[31:0]	Input	User-defined simple data 4 for simple 8 repeat data pattern.
simple_data5[31:0]	Input	User-defined simple data 5 for simple 8 repeat data pattern.
simple_data6[31:0]	Input	User-defined simple data 6 for simple 8 repeat data pattern.
simple_data7[31:0]	Input	User-defined simple data 7 for simple 8 repeat data pattern.
fixed_data_i[31:0]	Input	User-defined fixed data pattern.
fixed_instr_i[2:0]	Input	User-defined fixed command pattern. 000: Write command 001: Read command
fixed_bl_i[5:0]	Input	User-defined fixed burst length. Each burst value defines the number of back to back commands that are generated.

## 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 1-13, page 44](#).

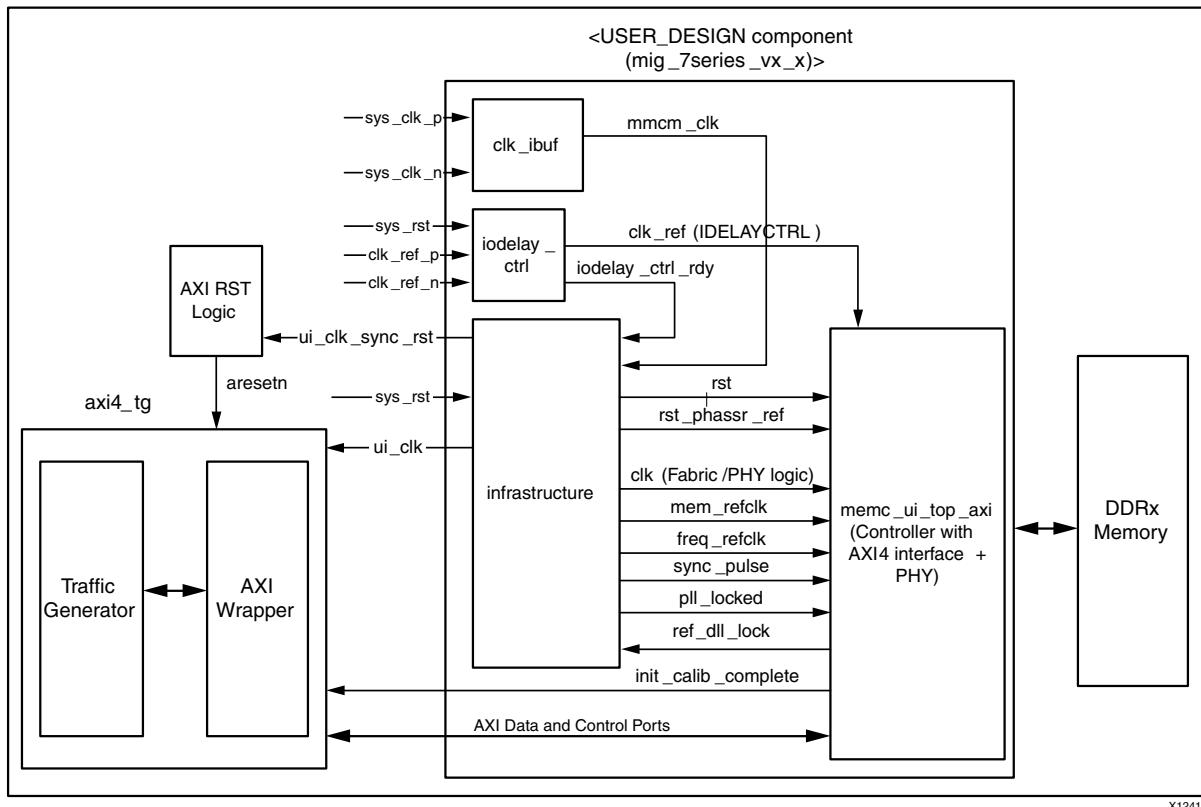
With some modifications, the example design can be changed to allow addr\_mode\_i, instr\_mode\_i, and bl\_mode\_i to be changed dynamically when run\_traffic\_i is deasserted. However, after changing the setting, the memory initialization steps need to be repeated to ensure that the proper pattern is loaded into the memory space.

#### **Note:**

- When the chip select option is disabled, the simulation test bench always ties the memory model chip select bit(s) to zero for proper operation.
- 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.

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

The MIG tool provides a synthesizable AXI4 test bench to generate various traffic patterns to the memory controller. This test bench consists of an instance of user design (memory controller) with AXI4 interface, a traffic\_generator (axi4\_tg) that generates traffic patterns through the AXI4 interface of the controller as shown in [Figure 1-26](#). The infrastructure block inside the user design provides clock resources to both the controller and the traffic generator. [Figure 1-26](#) shows a block diagram of the example design test bench. The details of the clocks in [Figure 1-26](#) are provided in [Clocking Architecture, page 69](#).



*Figure 1-26: Synthesizable Example Design Block for AXI4 Interface*

[Figure 1-27](#) shows the simple write transaction being performed on the AXI4 interface. This transaction consists of a command phase, a data phase, and a response phase. This follows the standard AXI4 protocol.

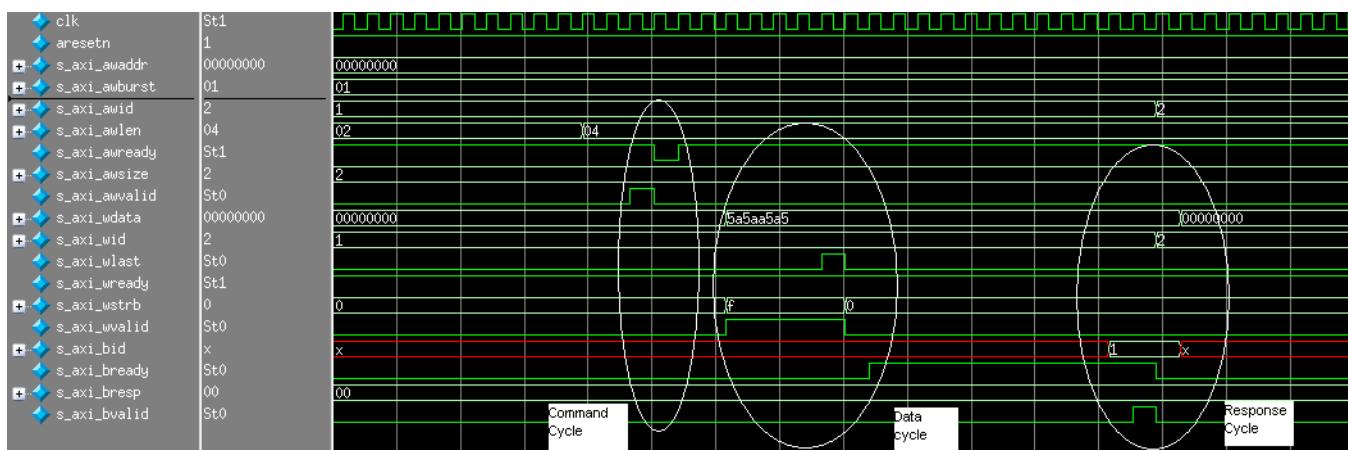


Figure 1-27: AXI4 Interface Write Cycle

Figure 1-28 shows a simple read transaction being performed on the AXI4 interface. This transaction consists of a command phase and data phase. This follows the standard AXI4 protocol.

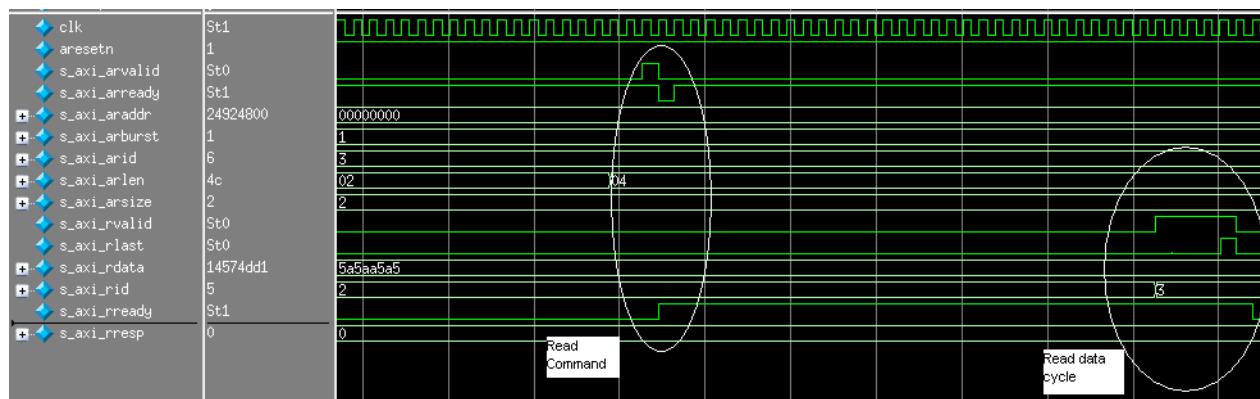


Figure 1-28: AXI4 Interface Read Cycle

The example design generated when the AXI4 interface is selected as the user interface is different compared to the standard traffic generator user interface. The intent of this synthesizable test bench is to verify the basic 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 1-14 shows the signals of interest during verification of the AXI4 test bench. These signals can be found in the example\_top module.

Table 1-14: Signals of Interest During Simulation for the AXI4 Test Bench

Signal	Description
test_cmptd	When asserted, this signal indicates that the current round of tests with random reads and writes is 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 is completed.

Table 1-14: Signals of Interest During Simulation for the AXI4 Test Bench

Signal	Description
cmd_err	When asserted, this signal indicates that the command phase of the AXI4 transaction (read or write) has an error.
write_err	When asserted, this signal indicates that the write transaction to memory resulted in an 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 has the status of the write transaction. The details of the status are given in Table 1-15.
read_cmptd	This signal is asserted for one clock indicating that the current read transaction is completed.
read_err	When asserted, this signal indicates that the read transaction to the memory resulted in an 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 has the status of the read transaction. The details of the status are given in Table 1-16.

The initialization and the calibration sequence remain the same as that indicated in [Simulating the Example Design \(for Designs with the Standard User Interface\)](#), page 37. The status that is generated for a write transaction can be found in [Figure 1-29](#).

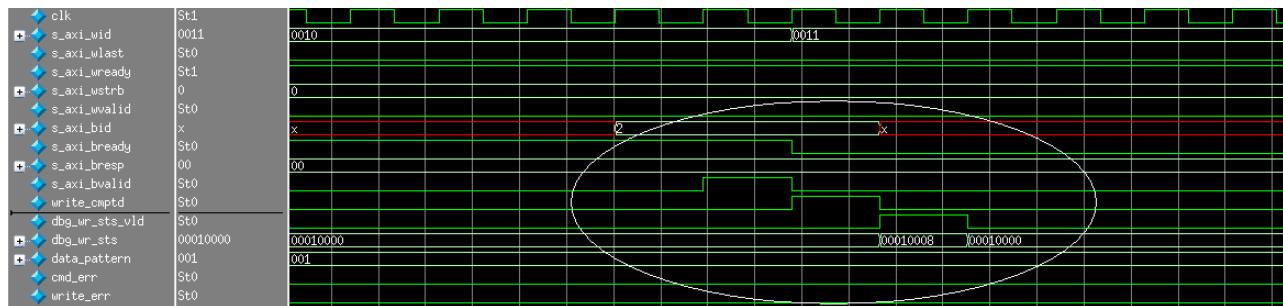


Figure 1-29: Status for the Write Transaction

Table 1-15: Debug Status for the Write Transaction

Bits	Status Description
1:0	Write response received for AXI
5:2	Response ID for the write response

Table 1-15: Debug Status for the Write Transaction (Cont'd)

Bits	Status Description
8:6	AXI wrapper write FSM state when timeout (watchdog timer should be enabled) occurs: <ul style="list-style-type: none"><li>• 3 'b001: Data write transaction</li><li>• 3 'b010: Waiting for acknowledgment for written data</li><li>• 3 'b011: Dummy data write transaction</li><li>• 3 'b100: Waiting for response from the response channel</li></ul>
15:9	Reserved
16	Command error occurred during a write transaction.
17	Write error occurred. The write transaction could not be completed.
20:18	Data pattern used for the current transaction: <ul style="list-style-type: none"><li>• 000: 5A and A5</li><li>• 001: PRBS pattern</li><li>• 010: Walking zeros</li><li>• 011: Walking ones</li><li>• 100: All ones</li><li>• 101: All zeros</li></ul>
31:21	Reserved

The status generated for a read transaction is shown in Figure 1-30.

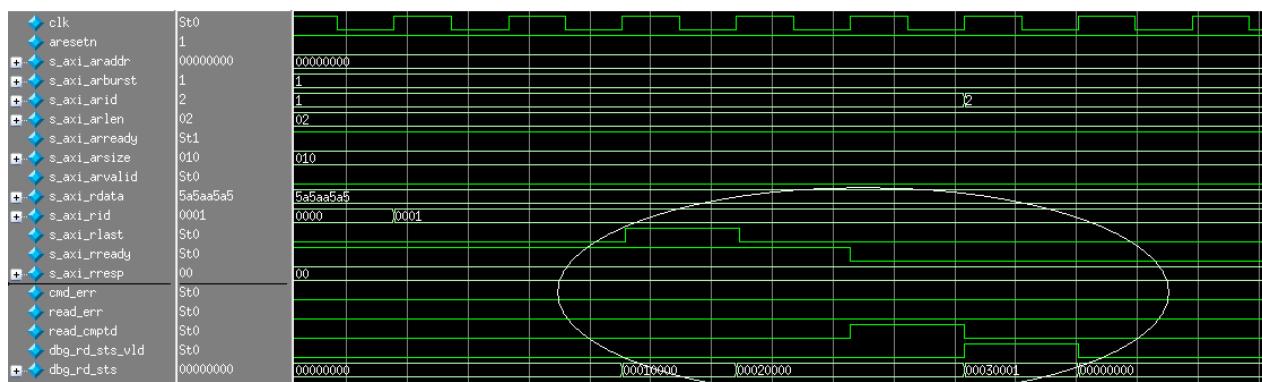


Figure 1-30: Status for the Read Transaction

Table 1-16: 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 timeout (watchdog timer should be enabled) occurs: <ul style="list-style-type: none"><li>• 2 'b01: Read command transaction</li><li>• 2 'b10: Data read transaction</li></ul>

Table 1-16: Debug Status for the Read Transaction (Cont'd)

Bits	Status Description
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: • 000: 5A and A5 • 001: PRBS pattern • 010: Walking zeros • 011: Walking ones • 100: All ones • 101: All zeros
31:30	Reserved

Calibration and other DDR data read and write transactions are similar to what is described in [Simulating the Example Design \(for Designs with the Standard User Interface\), page 37](#). The AXI4 write and read transactions are started only after the init\_calib\_complete signal is asserted.

## 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>/<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.

## Getting Started with EDK

EDK provides an alternative package to the RTL created by the MIG tool in the CORE Generator tool. The IP catalog in XPS contains the IP core `axi_7series_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_7series_ddrx` pcore only provides an AXI4 slave interface to DDR2 or DDR3 SDRAM in Verilog.

The simplest way to get started with the `axi_7series_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_7series_ddrx` memory controller. If the memory controller is selected, an already configured, connected, and tested `axi_7series_ddrx` controller is provided for a particular reference board

such as the KC705 board. For more information on BSB, see chapter 2 of *EDK Concepts, Tools, and Techniques*. [\[Ref 5\]](#)

When starting with a new project, the `axi_7series_ddrx` IP can be added to the design by dragging the memory controller into the project from the IP catalog. The `axi_7series_ddrx` IP is configured using the same MIG tool used in the CORE Generator tool and therefore the GUI flow is as described in [Getting Started with the CORE Generator Tool, page 7](#). 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 is able to generate the correct constraints and parameter values for itself during the XPS Platform Generator (PlatGen) tool. 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. 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 more information about EDK and XPS, see *EDK Concepts, Tools, and Techniques* [\[Ref 5\]](#) and the *Embedded System Tools Reference Manual* [\[Ref 6\]](#).

## 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 69](#). These clocking signals are used:

- freq\_refclk: either 1x memory clock with fine phase shift of 45 degrees enabled for memory frequencies greater than or equal to 400 MHz or 2x memory clock frequency for frequencies between 200–400 MHz. Ensure that this clock meets the 400–933 MHz requirement.
- mem\_refclk: 1x memory clock frequency
- sync\_pulse: 0.0625 x memory clock. This signal must have a duty cycle of 1/16 or 6.25%.
- clk\_ref: IDELAYCTRL reference clock, usually 200 MHz.
- clk: 0.25x memory clock in 4:1 mode and 0.5x the memory clock in 2:1 mode.

Here is an example portion of an EDK clock generator instantiation with associated `axi_7series_ddrx` port connections:

```
BEGIN clock_generator
PARAMETER INSTANCE = clock_generator_0
PARAMETER HW_VER = 4.03.a
PARAMETER C_CLKIN_FREQ = 200000000
PARAMETER C_CLKOUT0_FREQ = 400000000
PARAMETER C_CLKOUT0_GROUP = PLLE0
PARAMETER C_CLKOUT0_BUF = FALSE
PARAMETER C_CLKOUT0_PHASE = 45
PARAMETER C_CLKOUT1_FREQ = 400000000
PARAMETER C_CLKOUT1_GROUP = PLLE0
PARAMETER C_CLKOUT1_BUF = FALSE
PARAMETER C_CLKOUT2_FREQ = 250000000
PARAMETER C_CLKOUT2_GROUP = PLLE0
PARAMETER C_CLKOUT2_BUF = FALSE
```

```

PARAMETER C_CLKOUT2_PHASE = 10
PARAMETER C_CLKOUT2_DUTY_CYCLE = 0.0625
PARAMETER C_CLKOUT3_FREQ = 100000000
PARAMETER C_CLKOUT3_GROUP = PLLE0
PARAMETER C_CLKOUT3_BUF = TRUE
PARAMETER C_CLKOUT4_FREQ = 200000000
PARAMETER C_CLKOUT4_GROUP = NONE
PARAMETER C_CLKOUT4_BUF = TRUE
PARAMETER C_CLKFBOUT_BUF = FALSE
PARAMETER C_CLKOUT5_FREQ = 50000000
PARAMETER C_CLKOUT5_GROUP = PLLE0
PARAMETER C_CLKOUT5_BUF = TRUE
PORT CLKOUT0 = freq_refclk
PORT CLKOUT1 = mem_refclk
PORT CLKOUT2 = sync_pulse
PORT CLKOUT3 = sys_clk_s
PORT CLKOUT4 = clk_ref
PORT CLKOUT5 = sys_clk_axilite_s
PORT CLKIN = dcm_clk_s
PORT LOCKED = proc_sys_reset_0_Dcm_locked
PORT RST = sys_rst_s
END

```

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 uppercase and prefixed with “C\_.” For example, behavioral simulation run time can be improved by adding this MHS parameter to the axi\_7series\_ddrx instantiation:

```
PARAMETER C_SIM_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. The primary AXI4 interface is called S\_AXI.

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 8x the width of the memory width in 4:1 mode and 4x the width in 2:1 mode.

## 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, see *EDK Concepts, Tools, and Techniques* [Ref 5] and the *Embedded System Tools Reference Manual* [Ref 6].

## Simulation Considerations

The simplest approach to simulate a design using axi\_7series\_ddrx is to use the test bench generator called SimGen in the EDK XPS. SimGen automatically connects stimulus to the top-level clock and resets. It also connects the correct memory model to the axi\_7series\_ddrx external ports. If an ELF file is provided, SimGen also loads the ELF file into memory.

**Note:** axi\_7series\_ddrx does not support structural simulation because it is not a supported flow for the underlying MIG PHYs. Structural simulation is therefore not recommended.

axi\_7series\_ddrx simulation should be performed in the behavioral/functional level. It 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, mem\_refclk, freq\_refclk, and sync\_pulse ports of axi\_7series\_ddrx must be completely phase-aligned.

The initialization and calibration sequences are automatically shortened in behavioral simulation to greatly reduce simulation time. To simulate full initialization and calibration, edit the MHS to add this line to the axi\_7series\_ddrx pcore section:

```
parameter C_SIM_BYPASS_INIT_CAL = SIM_INIT_CAL_FULL
```

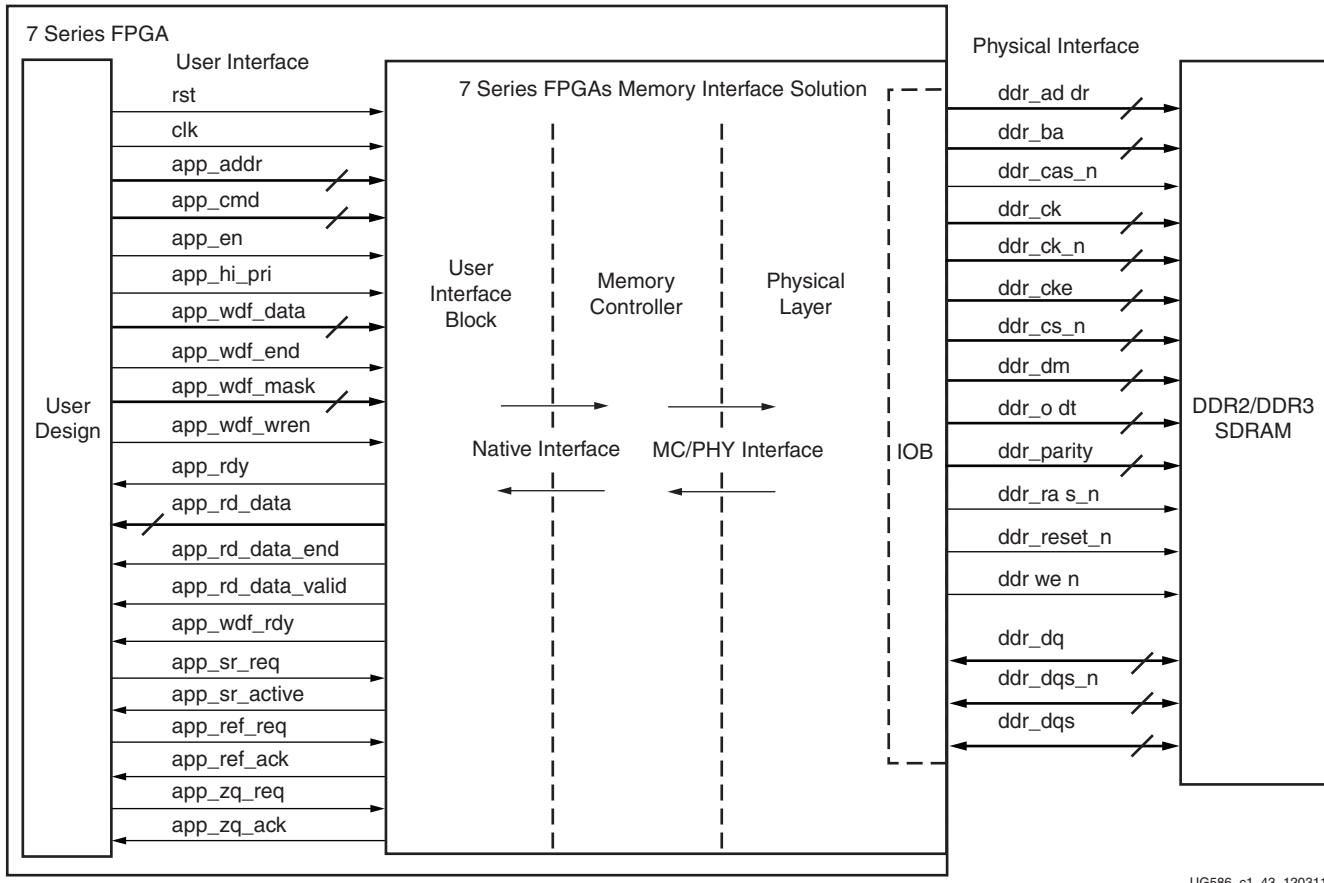
This requires you to run simulations into the millisecond range.

## Core Architecture

This section describes the architecture of the 7 series FPGAs memory interface solutions core, providing an overview of the core modules and interfaces.

### Overview

The 7 series FPGAs memory interface solutions core is shown in [Figure 1-31](#).



UG586\_c1\_43\_120311

Figure 1-31: 7 Series FPGAs Memory Interface Solution

## User Design

The user design block shown in Figure 1-31 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 memory controller connects to the physical interface and handles all the interface requirements to that module. The memory controller 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 memory controller. 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 IDELAYs. IDELAYs are associated with the data group (DQ). 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/.vhdl` module). Based on this attribute, the ISE tool properly replicates IODELAYCTRLs as needed within the design.

The IDELAYCTRL reference frequency should be set to 200 MHz. Based on the IODELAY\_GROUP attribute that is set, the ISE tool replicates the IODELAYCTRLs for each region where the IDELAY blocks exist. When a user creates a multi-controller design on their own, 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 1-17](#) and connects to an FPGA user design to allow access to an external memory device.

*Table 1-17: 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.
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.

Table 1-17: User Interface (*Cont'd*)

Signal	Direction	Description
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	Input	This input is reserved and should be tied to 0.
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_i	Input	When asserted, this active-High signal corrects single bit data errors. This input is valid only when ECC is enabled in the GUI. In the example design, this signal is always tied to 1.
app_sr_req	Input	This input is reserved and should be tied to 0.
app_sr_active	Output	This output is reserved.
app_ref_req	Input	This active-High input requests that a refresh command be issued to the DRAM.
app_ref_ack	Output	This active-High output indicates that the memory controller has sent the requested refresh command to the PHY interface.
app_zq_req	Input	This active-High input requests that a ZQ calibration command be issued to the DRAM.
app_zq_ack	Output	This active-High output indicates that the memory controller has sent the requested ZQ calibration command to the PHY interface.
clk	Input	This UI clock must be a quarter of the DRAM clock.
clk_mem	Input	This is a full-frequency memory clock.
init_calib_complete	Output	PHY asserts init_calib_complete when calibration is finished.

**Table 1-17: User Interface (Cont'd)**

<b>Signal</b>	<b>Direction</b>	<b>Description</b>
app_ecc_multiple_err[7:0] <sup>(1)</sup>	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 memc\_ui\_top module level only. This signal should only be used when ECC is enabled.

**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 1-18](#).

**Table 1-18: Commands for app\_cmd[2:0]**

<b>Operation</b>	<b>app_cmd[2:0] Code</b>
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], 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\_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 ad app\_wdf\_wren are asserted.

**app\_ref\_req**

When asserted, this active-High input requests that the memory controller send a refresh command to the DRAM. It must be pulsed for a single cycle to make the request and then deasserted at least until the app\_ref\_ack signal is asserted to acknowledge the request and indicate that it has been sent.

**app\_ref\_ack**

When asserted, this active-High input acknowledges a refresh request and indicates that the command has been sent from the memory controller to the PHY.

**app\_zq\_req**

When asserted, this active-High input requests that the memory controller send a ZQ calibration command to the DRAM. It must be pulsed for a single cycle to make the request and then deasserted at least until the app\_zq\_ack signal is asserted to acknowledge the request and indicate that it has been sent.

**app\_zq\_ack**

When asserted, this active-High input acknowledges a ZQ calibration request and indicates that the command has been sent from the memory controller to the PHY.

**rst**

This is the reset input for the UI.

**clk**

This is the input clock for the UI. It must be a quarter 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.

**init\_calib\_complete**

PHY asserts init\_calib\_complete when calibration is finished. The application has no need to wait for init\_calib\_complete 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\_7series\_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 3].

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 simple round-robin arbiter 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.

## AXI4 Slave Interface Parameters

[Table 1-19](#) lists the AXI4 slave interface parameters.

**Table 1-19: 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 8x the memory data width. The width can be smaller, but not greater than 8x 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_RD_WR_ARB_ALGORITHM	"RD_PRI_REG"	"TDM", "ROUND_ROBIN", "RD_PRI_REG", "RD_PRI_REG_STARVE_LIMIT" "	This parameter indicates the Arbitration algorithm scheme. See <a href="#">Arbitration in AXI Shim, page 64</a> for more information.
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.

## AXI4 Slave Interface Signals

**Table 1-20** lists the AXI4 slave interface specific signal. Clock/reset to the interface is provided from the memory controller.

**Table 1-20: 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 1-20: 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_rrresp	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 AXI Shim

The AXI4 protocol calls for independent read and write address channels. The memory controller has one address channel. 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 alternate every clock cycle. The read or write requests 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 AXI master. The grant to the read and write address channels alternate 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 then the grant is given to the other address channel with the pending request.

## Read Priority (RD\_PRI\_REG)

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.

## 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 1-21](#).

**Table 1-21: 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.
data_buf_addr[7:0]	Input	This input indicates the data buffer address where the memory controller: <ul style="list-style-type: none"> <li>• Locates data while processing write commands.</li> <li>• Places data while processing read commands.</li> </ul>

**Table 1-21: Native Interface Command Signals (Cont'd)**

<b>Signal</b>	<b>Direction</b>	<b>Description</b>
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.
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 1-22](#).

**Table 1-22: Memory Interface Commands**

<b>Operation</b>	<b>cmd[2:0] Code</b>
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 1-23](#)). These signals connect to the control, address, and data signals of a buffer in the user design.

**Table 1-23: Native Interface Write Command Signals**

<b>Signal</b>	<b>Direction</b>	<b>Description</b>
wr_data[ $2 \times nCK\_PER\_CLK \times PAYLOAD\_WIDTH - 1:0$ ]	Input	This is the input data for write commands.
wr_data_addr [DATA_BUF_ADDR_WIDTH - 1:0]	Output	This output provides the base address for the source data buffer for write commands.
wr_data_mask[ $2 \times nCK\_PER\_CLK \times DATA\_WIDTH/8 - 1:0$ ]	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\_mask**

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\_mask 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 1-24). 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 1-24: Native Interface Read Command Signals**

Signal	Direction	Description
rd_data[ $2 \times nCK\_PER\_CLK \times PAYLOAD\_WIDTH - 1:0$ ]	Output	This is the output data from read commands.
rd_data_addr [DATA_BUF_ADDR_WIDTH - 1:0]	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[1:0]	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.

**Native Interface Maintenance Command Signals**

**Table 1-25** lists the native interface maintenance command signals.

**Table 1-25: Native Interface Maintenance Command Signals**

Signal	Direction	Description
app_sr_req	Input	This input is reserved and should be tied to 0.
app_sr_active	Output	This output is reserved.
app_ref_req	Input	This active-High input requests that a refresh command be issued to the DRAM.
app_ref_ack	Output	This active-High output indicates that the memory controller has sent the requested refresh command to the PHY interface.

**Table 1-25: Native Interface Maintenance Command Signals (Cont'd)**

Signal	Direction	Description
app_zq_req	Input	This active-High input requests that a ZQ calibration command be issued to the DRAM.
app_zq_ack	Output	This active-High output indicates that the memory controller has sent the requested ZQ calibration command to the PHY interface.

**app\_ref\_req**

When asserted, this active-High input requests that the memory controller send a refresh command to the DRAM. It must be pulsed for a single cycle to make the request and then deasserted at least until the app\_ref\_ack signal is asserted to acknowledge the request and indicate that it has been sent.

**app\_ref\_ack**

When asserted, this active-High input acknowledges a refresh request and indicates that the command has been sent from the memory controller to the PHY.

**app\_zq\_req**

When asserted, this active-High input requests that the memory controller send a ZQ calibration command to the DRAM. It must be pulsed for a single cycle to make the request and then deasserted at least until the app\_zq\_ack signal is asserted to acknowledge the request and indicate that it has been sent.

**app\_zq\_ack**

When asserted, this active-High input acknowledges a ZQ calibration request and indicates that the command has been sent from the memory controller to the PHY.

## Clocking Architecture

The PHY design requires that a PLL module be 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
- IDELAY reference clock (200 MHz)

One PLL is required for the PHY. The PLL is used to generate the clocks for most of the internal logic, the frequency reference clocks to the phasers, and a synchronization pulse required for keeping PHY control blocks synchronized in multi-I/O bank implementations.

For DDR3 SDRAM clock frequencies between 400 MHz and 933 MHz, both the phaser frequency reference clocks have the same frequency as the memory clock frequency. For DDR2 or DDR3 SDRAM clock frequencies below 400 MHz, one of the phaser frequency reference clocks runs at the same frequency as the memory clock and the second frequency reference clock must be either 2x or 4x the memory clock frequency such that it meets the

range requirement of 400 MHz to 933 MHz. The two phaser frequency reference clocks must be generated by the same PLL so they are in phase with each other. The block diagram of the clocking architecture is shown in [Figure 1-32](#).

The default setting for the PLL multiply (M) and divide (D) values is for the system clock input frequency to be equal to the memory clock frequency. This one to one ratio is not required. The PLL input divider (D) can be any value listed in the *7 Series FPGAs Clocking Resources User Guide* [Ref 18] as long as the PLLE2 operating conditions are met and the other constraints listed here are observed. The PLL multiply (M) value must be between 1 and 16 inclusive. The PLL output divider (O) for the memory clock must be 2 for 800 Mb/s and above, and 4 for 400 to 800 Mb/s. The PLL VCO frequency range must be kept in the range specified in the silicon data sheet. The sync\_pulse must be 1/16 of the mem\_refclk frequency and must have a duty cycle of 1/16 or 6.25%. For information on physical placement of the PLL and the System Clock CCIO input, see [Design Guidelines, page 126](#).

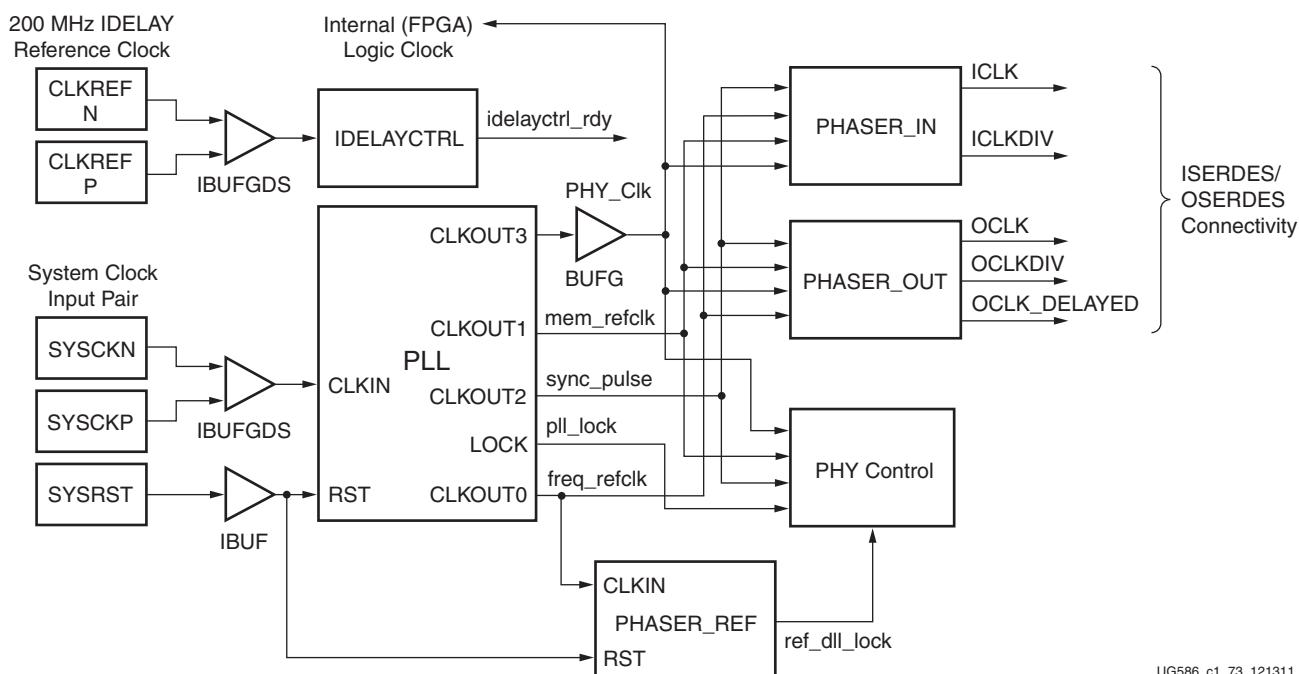


Figure 1-32: **Clocking Architecture**

The details of the ISERDES/OSERDES connectivity are shown in [Figure 1-38, page 88](#) and [Figure 1-40, page 90](#).

### Internal (FPGA) Logic Clock

The internal FPGA logic is clocked by a global clocking resource at a quarter frequency of the DDR2 or DDR3 SDRAM clock frequency. This PLL also outputs the high-speed DDR2 or DDR3 memory clock.

### Write Path (Output) I/O Logic Clock

The output path comprising both data and controls is clocked by PHASER\_OUT. The PHASER\_OUT provides synchronized clocks for each byte group to the OUT\_FIFOs and to the OSERDES/ODDR. The PHASER\_OUT generates a byte clock (OCLK), a divided byte clock (OCLKDIV), and a delayed byte clock (OCLK\_DELAYED) for its associated byte group. These clocks are generated directly from the Frequency Reference clock and are in

phase with each other. The byte clock is the same frequency as the Frequency Reference clock and the divided byte clock is half the frequency of the Frequency Reference clock. OCLK\_DELAYED is used to clock the DQS ODDR to achieve the required 90-degree phase offset between the write DQS and its associated DQ bits. The PHASER\_OUT also drives the signalling required to generate DQS during writes, the DQS and DQ 3-state associated with the data byte group, and the Read Enable for the OUT\_FIFO of the byte group. The clocking details of the address/control and the write paths using PHASER\_OUT are shown in [Figure 1-38](#) and [Figure 1-40](#).

### Read Path (Input) I/O Logic Clock

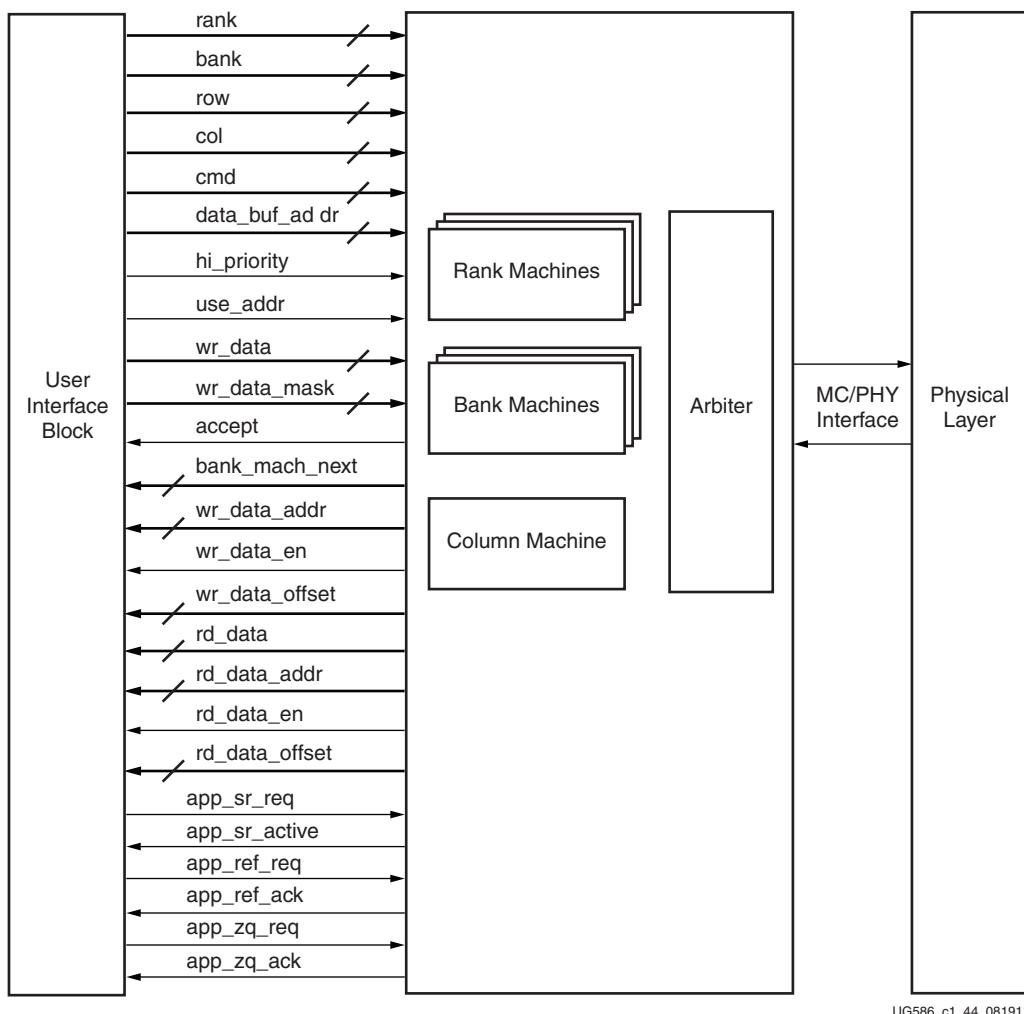
The input read datapath is clocked by the PHASER\_IN block. The PHASER\_IN block provides synchronized clocks for each byte group to the IN\_FIFOs and to the IDDR/ISERDES. The PHASER\_IN block receives the DQS signal for the associated byte group and generates two delayed clocks for DDR2 or DDR3 SDRAM data captures: read byte clock (ICLK) and read divided byte clock (ICLKDIV). ICLK is the delayed version of the frequency reference clock that is phase-aligned with its associated DQS. ICLKDIV is used to capture data into the first rank of flip-flops in the ISERDES. ICLKDIV is aligned to ICLK and is the parallel transfer clock for the last rank of flip-flops in the ISERDES. ICLKDIV is also used as the write clock for the IN\_FIFO associated with the byte group. The PHASER\_IN block also drives the write enable (WrEnable) for the IN\_FIFO of the byte group. The clocking details of the read path using PHASER\_IN is shown in [Figure 1-40](#).

### IDELAY Reference Clock

A 200 MHz IDELAY clock must be supplied to the IDELAYCTRL module. The IDELAYCTRL module continuously calibrates the IDELAY elements in the I/O region to account for varying environmental conditions. The IP 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.

## 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 1-33](#).



*Figure 1-33: 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 memory controller 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 memory controller 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. This is discussed in greater detail in the [Precharge Policy](#) section below.

The duration of a bank machine's assignment to a particular DRAM bank is coupled to user requests rather than the state of the target DRAM bank. 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.

The following simplified example illustrates this concept. Consider the case when the memory controller and DRAM are idle when a single request arrives. The bank machine at the head of the pool:

1. Accepts the user request
2. Activates the target row
3. Issues the column (read or write) command
4. Precharges the target row
5. Returns to the idle pool of bank machines

Similar functionality applies when multiple requests arrive targeting different rows or banks.

Now consider the case when a request arrives targeting an open DRAM bank, managed by an already active bank machine. The already active bank machine recognizes that the new request targets the same DRAM bank and skips the precharge step ([step 4](#)). The bank machine at the head of the idle pool accepts the new user request and skips the activate step ([step 2](#)).

Finally, when a request arrives in between both a previous and subsequent request all to the same target DRAM bank, the controller skips both the activate ([step 2](#)) and precharge ([step 4](#)) operations.

A bank machine precharges a DRAM bank as soon as possible unless another pending request targets the same bank. This is discussed in greater detail in the [Precharge Policy](#) section.

Column commands can be reordered for the purpose of optimizing memory interface throughput. The ordering algorithm nominally ensures data coherence. The reordering feature is explained in greater detail in the [Reordering](#) section.

## 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 memory controller 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.

## Precharge Policy

The controller implements an aggressive precharge policy. The controller examines the input queue of requests as each transaction completes. If no requests are in the queue for a currently open bank/row, the controller closes it to minimize latency for requests to other rows in the bank. Because the queue depth is equal to the number of bank machines, greater efficiency can be obtained by increasing the number of bank machines

(nBANK\_MACHS). As this number is increased, fabric timing becomes more challenging. In some situations, the overall system efficiency can be greater with an increased number of bank machines and a lower memory clock frequency. Simulations should be performed with the target design's command behavior to determine the optimum setting.

## Error Correcting Code

The memory controller optionally implements an Error Correcting Code (ECC). 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 errors of two bits are detected. Errors of more than two bits might or might not be detected.

[Figure 1-34](#) shows the ECC block diagram. These blocks are instantiated in the memory controller (`mc.v`) module.

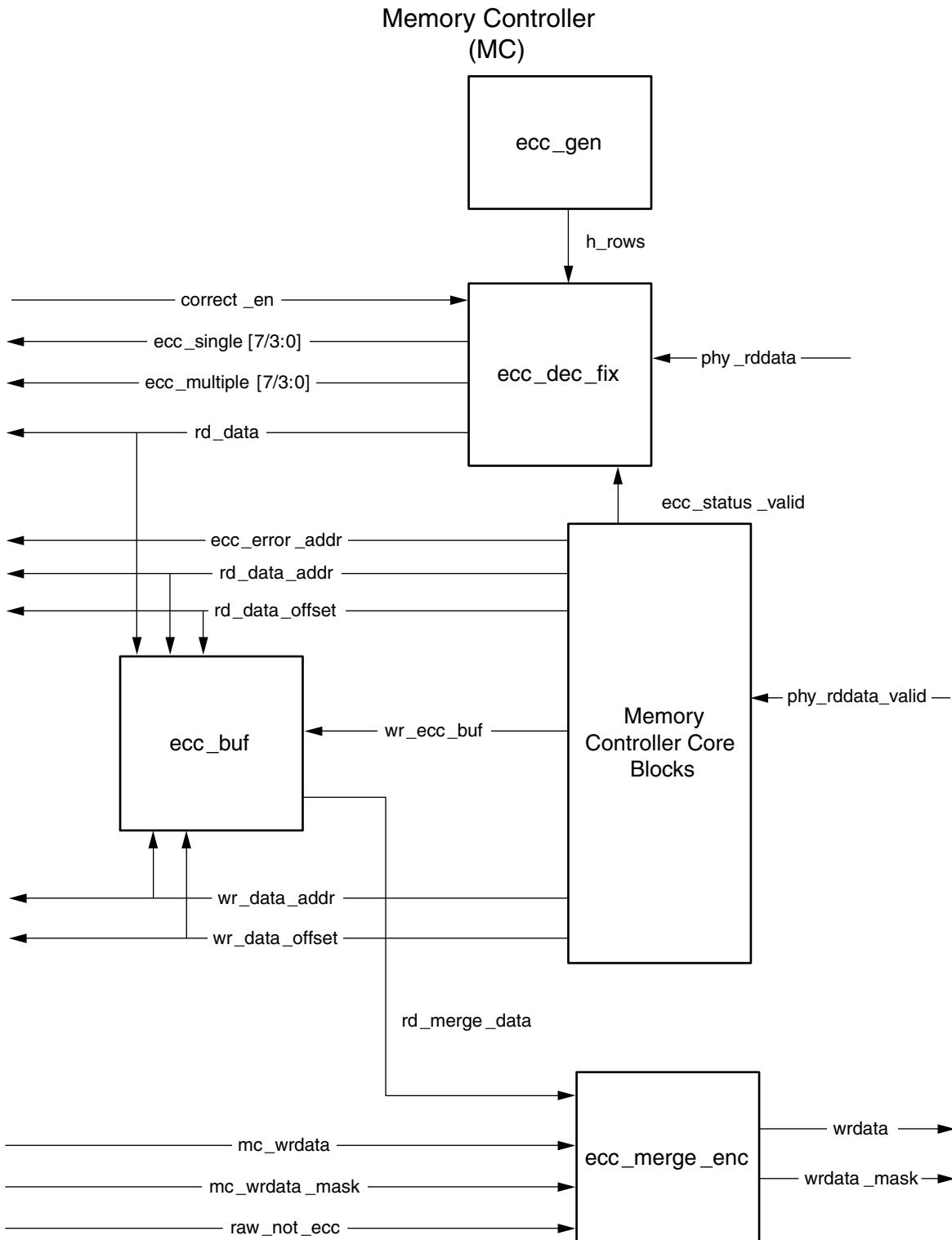


Figure 1-34: ECC Block Diagram

The ECC mode is optional and supported only for a 72-bit data width. 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. A top-level parameter called ECC controls the addition of ECC logic. When this parameter is set to “ON”, ECC is enabled, and when the parameter is set to “OFF”, ECC is disabled.

The ECC functionality is implemented as three functional blocks. A 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 matrices 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 phy\_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 timing signal ecc\_status\_valid 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 given, 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. To write partially into memory, app\_wdf\_mask needs to be driven along with the wr\_bytes command for ECC enabled designs. [Table 1-26](#) shows the available commands when ECC mode is enabled.

**Table 1-26: 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 given, 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, 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 1-27** provides the details of ECC ports at the User Interface.

**Table 1-27: 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 in 2:1 mode.
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.
app_wdf_mask [APP_MASK_WIDTH – 1:0]	Input	This signal provides the mask for app_wdf_data[].

## 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 system self-test because there is no way to test the bits in the DRAM array corresponding to the ECC bits. There is also no way to send errors 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. app\_raw\_not\_ecc\_i is 7 bits wide (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 the app\_correct\_en\_i should be set to zero to disable correction.

With the above two features, 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.

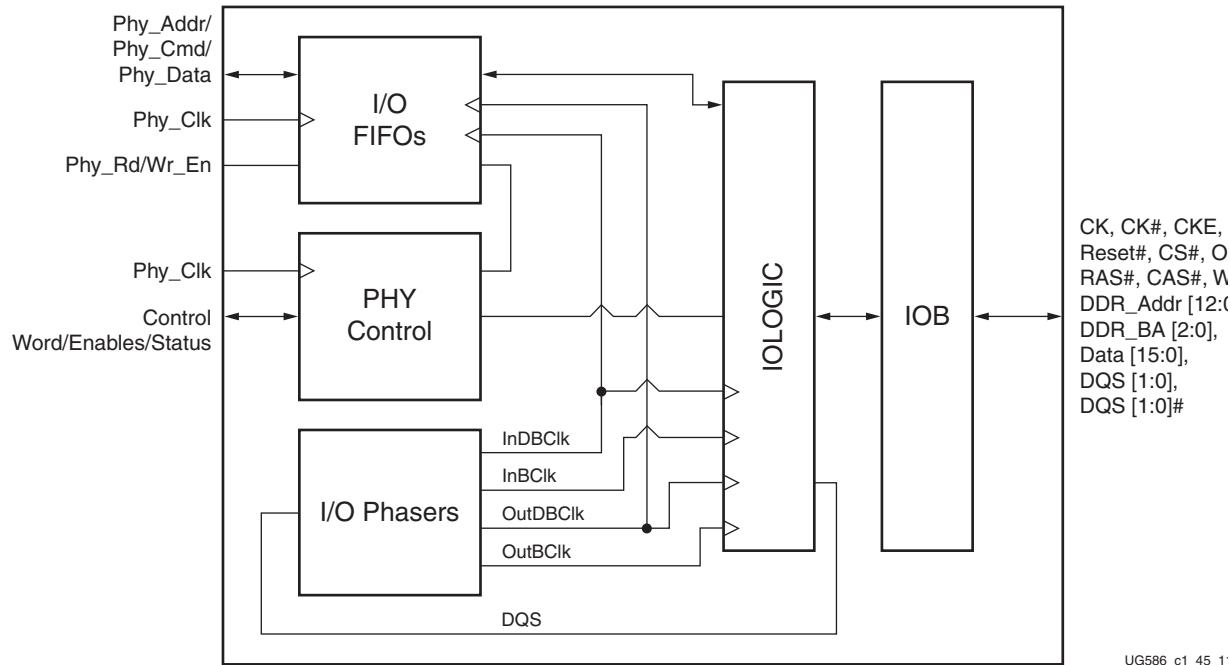
The PHY is provided as a single HDL codebase for DDR2 and DDR3 SDRAMs. The MIG tool customizes the SDRAM type and numerous other design-specific parameters through top-level HDL parameters and constraints contained in a user constraints file (UCF).

### Overall PHY Architecture

The 7 series FPGA PHY is composed of dedicated blocks and soft calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high-performance physical layers. Dedicated clock structures within an I/O bank referred to as byte group clocks help minimize the number of loads driven by the byte group clock drivers. Byte group clocks are driven by phaser blocks. The phaser blocks (PHASER\_IN and PHASER\_OUT) are multi-stage programmable delay line loops that can dynamically track DQS signal variation and provide precision phase adjustment.

Each 7 series FPGA I/O bank has dedicated blocks comprising a PHY control block, four PHASER\_IN and PHASER\_OUT blocks, four IN/OUT\_FIFOs, IOLOGIC (ISERDES, OSERDES, ODDR, IDELAY), and IOBs. Four byte groups exist in an I/O bank, and each byte group contains the PHASER\_IN and PHASER\_OUT, IN\_FIFO and OUT\_FIFO, and twelve IOLOGIC and IOB blocks. Ten of the twelve IOIs in a byte group are used for DQ and DM bits, and the other two IOIs are used to implement differential DQS signals.

Figure 1-35 shows the dedicated blocks available in a single I/O bank. A single PHY control block communicates with all four PHASER\_IN and PHASER\_OUT blocks within the I/O bank.



UG586\_c1\_45\_1

Figure 1-35: Single Bank DDR2/DDR3 PHY Block Diagram

The memory controller and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is either a divided by 4 or divided by 2 version of the DDR2 or DDR3 memory clock. A block diagram of the PHY design is shown in [Figure 1-36](#).

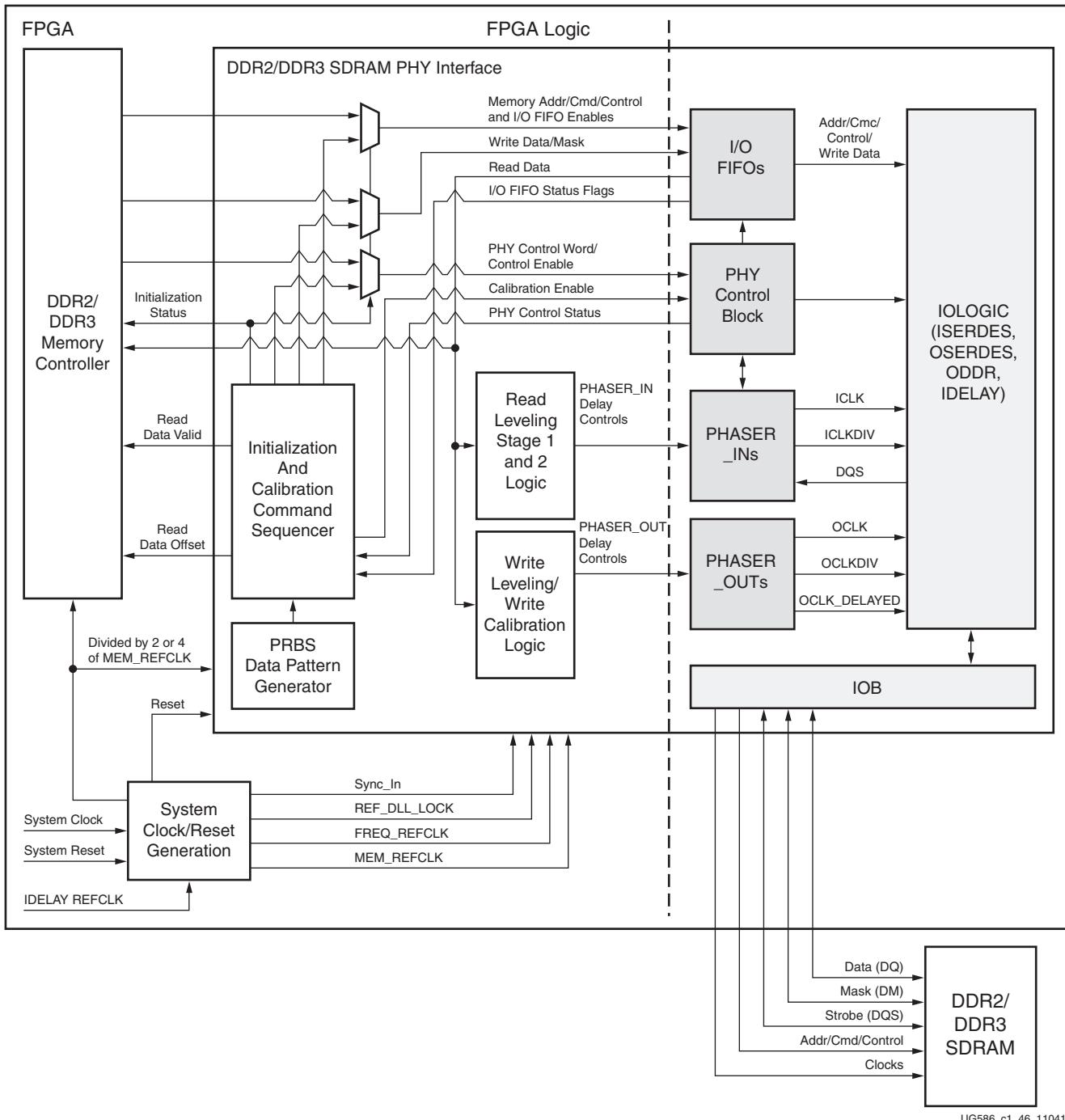
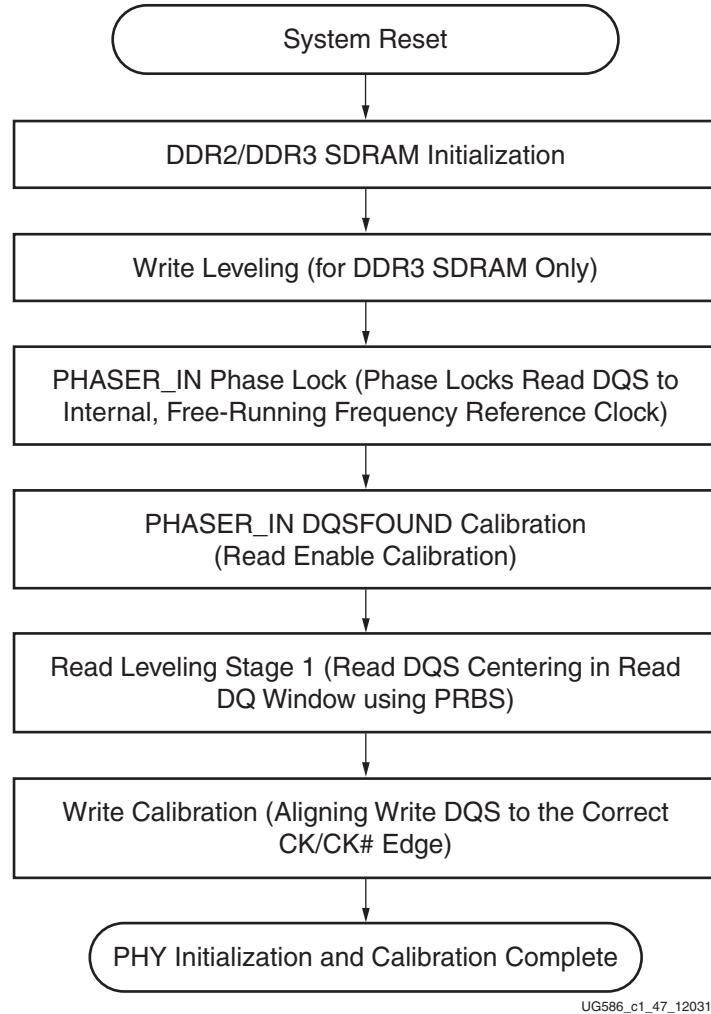


Figure 1-36: PHY Block Diagram

### Memory Initialization and Calibration 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 1-37](#) shows the overall flow of memory initialization and the different stages of calibration.



UG586\_c1\_47\_120311

[Figure 1-37: PHY Overall Initialization and Calibration Sequence](#)

The calibration stages in [Figure 1-37](#) correspond to these sections:

- [Memory Initialization, page 91](#).
- [Write Leveling, page 91](#).
- [PHASER\\_IN Phase Lock, page 94](#).
- [PHASER\\_IN DQSFOUND Calibration, page 94](#).
- [Read Leveling, page 95](#).
- [Write Calibration, page 97](#).

## I/O Architecture

Each 7 series FPGA I/O bank has dedicated blocks comprising a PHY control block, four PHASER\_IN and PHASER\_OUT blocks, four IN/OUT\_FIFOs, ISERDES, OSERDES, ODDR, IDELAY, and IOBs. A single PHY control block communicates with all four PHASER\_IN and PHASER\_OUT blocks within the I/O bank.

## PHY Control Block

The PHY control block is the central control block that manages the flow of data and control information between the FPGA logic and the dedicated PHY. This includes control over the flow of address, command, and data between the IN/OUT\_FIFOs and ISERDES/OSERDES, and control of the PHASER\_IN and PHASER\_OUT blocks. The PHY control block receives control words from the calibration logic or the memory controller at the slow frequency (1/4 the frequency of the DDR2 or DDR3 SDRAM clock) PHY\_Clk rate and processes the control words at the DDR2 or DDR3 SDRAM clock rate (CK frequency).

The calibration logic or the memory controller initiates a DDR2 or DDR3 SDRAM command sequence by writing address, command, and data (for write commands) into the IN/OUT\_FIFOs and simultaneously or subsequently writes the PHY control word to the PHY control block. The PHY control word defines a set of actions that the PHY control block does to initiate the execution of a DDR2 or DDR3 SDRAM command.

The PHY control block provides the control interfaces to the byte group blocks within its I/O bank. When multi-I/O bank implementations are required, each PHY control block within a given I/O bank controls the byte group elements in that bank. This requires that the PHY control blocks stay in phase with their adjacent PHY control blocks. The center PHY control block is configured to be the master controller for a three I/O bank implementation. For two bank implementations, either PHY control block can be designated the master.

The PHY control interface is used by the calibration logic or the memory controller to write PHY control words to the PHY. The signals in this interface are synchronous to the PHY\_Clk and are listed in [Table 1-28](#). This is a basic FIFO style interface. Control words are written into the control word FIFO on the rising edge of PHY\_Clk when PHY\_Ctl\_WrEn is High and PHY\_Ctl\_Full is Low. For multi-I/O bank PHYs, the same control word must be written into each PHY control block for proper operation.

*Table 1-28: PHY Control Interface*

Signal	Direction	Signal Description
PHY_Clk	Input	This is the PHY interface clock for the control word FIFO. PHY control word signals are captured on the rising edge of this clock.
PHY_Ctl_Wr_N	Input	This active-Low signal is the write enable signal for the control word FIFO. A control word is written into the control word FIFO on the rising edge of PHY_Clk, when this signal is active.
PHY_Ctl_Wd[31:0]	Input	This is the PHY control word described in <a href="#">Table 1-29</a> .
PHY_Ctl_Full	Output	This active-High output is the full flag for the control word FIFO. It indicates that the FIFO cannot accept any more control words and blocks writes to the control word FIFO.
PHY_Ctl_AlmostFull	Output	This active-High output is the almost full flag for the control word FIFO. It indicates that the FIFO can accept no more than one additional control word as long as the PHY_Ctl_Full signal is inactive.
PHY_Ctl_Ready	Output	This active-High output becomes set when the PHY control block is ready to start receiving commands.

The PHY control word is broken down into several fields, as shown in [Table 1-29](#).

**Table 1-29: PHY Control Word**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Act	Event	CAS					Seq								Data Offset		Rank	Low		Aux_Out				Control Offset		PHY						

- **PHY Command:** This field defines the actions undertaken by the PHY control block to manage command and data flow through the dedicated PHY. The PHY commands are:
  - Write (Wr - 0x01): This command instructs the PHY control block to read the address, command, and data OUT\_FIFOs and transfer the data read from those FIFOs to their associated IOIs.
  - Read (Rd - 0x03): This command instructs the PHY control block to read the address, command OUT\_FIFOs, and transfer the data read from those FIFOs to their associated IOIs. In addition, data read from the memory is transferred after its arrival from the data IOIs to the Data IN\_FIFO.
  - Non-Data (ND - 0x04): This command instructs the PHY control block to read the address and command OUT\_FIFOs and transfer the data read from those FIFOs to their associated IOIs.
- **Control Offset:** This field is used to control when the address and command IN/OUT\_FIFOs are read and transferred to the IOIs. The control offset is in units of the DDR2 or DDR3 SDRAM clock cycle.
- **Auxiliary Output:** This field is used to control when the auxiliary output signals (Aux\_Output[3:0]) are used. Auxiliary outputs can be configured to activate during read and write commands. The timing offset and duration are controlled by the attributes described in [Table 1-30, page 85](#). The ODT and CKE signals of the DDR2 or DDR3 SDRAM are output by the PHY via the auxiliary outputs.
- **Low Index (Bank):** The dedicated PHY has internal counters that require this field to specify which of the eight DDR2 or DDR3 SDRAM banks to use for the data command. The MIG IP core does not use these internal counters; therefore, this field should be all zeros.
- **Rank Count:** This field is used to specify which of the two DDR2 or DDR3 SDRAM ranks to use for the data command. This information is passed to the PHASER\_IN blocks to select the correct delay values.
- **Data Offset:** This field is used to control when the data IN/OUT\_FIFOs are read or written based on the PHY command. The data offset is in units of the DDR2 or DDR3 SDRAM clock cycle.
- **Seq:** This field contains a sequence number used in combination with the Sync\_In control signal from the PLL to keep two or more PHY control blocks executing the commands read from their respective control queues in sync. Commands with a given seq value must be executed by the command parser within the PHY control block during the specific phase indicated by the Seq field.
- **CAS Slot:** The slot number being used by the memory controller for write/read (CAS) commands.
- **Event Delay:** The dedicated PHY has internal counters that require this field to specify the delay values loaded into these counters. The event delay is in units of DDR2 or DDR3 SDRAM clock cycles. The MIG IP core does not use these internal counters; therefore, this field should be all zeros.

- **Activate Precharge:** The dedicated PHY has internal counters that require this field to specify the type of DDR2 or DDR3 command related to the event delay counter. Valid values are:
  - 00: No action
  - 01: Activate
  - 10: Precharge
  - 11: Precharge/Activate.

The MIG IP core does not use these internal counters; therefore, this field should be all zeros.

**Table 1-30: Auxiliary Output Attributes**

Attribute	Type	Description
MC_AO_WRLVL_EN	Vector[3:0]	This attribute specifies whether or not the related Aux_Output is active during write leveling as specified by the PC_Enable_Calib[1] signal. For example, this attribute specifies whether ODT is active during write leveling.
WR_CMD_OFFSET_0	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated write command is executed that the auxiliary output becomes active. For example, this attribute ensures that the ODT signal is asserted at the correct clock cycle to meet the JEDEC ODTLon and ODTLoff specifications.
WR_DURATION_0	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a write command. For example, this attribute ensures that the ODT signal is asserted at the correct clock cycle to meet the JEDEC ODTLon and ODTLoff specifications.
RD_CMD_OFFSET_0	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated read command is executed that the auxiliary output becomes active.
RD_DURATION_0	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a read command.
WR_CMD_OFFSET_1	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated write command is executed that the auxiliary output becomes active.
WR_DURATION_1	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a write command.
RD_CMD_OFFSET_1	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated read command is executed that the auxiliary output becomes active.

Table 1-30: Auxiliary Output Attributes (Cont'd)

Attribute	Type	Description
RD_DURATION_1	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a read command.
WR_CMD_OFFSET_2	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated write command is executed that the auxiliary output becomes active.
WR_DURATION_2	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a write command.
RD_CMD_OFFSET_2	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated read command is executed that the auxiliary output becomes active.
RD_DURATION_2	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a read command.
WR_CMD_OFFSET_3	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated write command is executed that the auxiliary output becomes active.
WR_DURATION_3	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a write command.
RD_CMD_OFFSET_3	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated read command is executed that the auxiliary output becomes active.
RD_DURATION_3	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles the auxiliary output remains active for a read command.
CMD_OFFSET	Vector[5:0]	This attribute specifies how long in DDR2 or DDR3 SDRAM clock cycles after the associated command is executed that the auxiliary output defined by AO_TOGGLE toggles.
AO_TOGGLE	Vector[3:0]	This attribute specifies which auxiliary outputs are in toggle mode. An auxiliary output in toggle mode is inverted when its associated AO bit is set in the PHY control word after the CMD_OFFSET has expired.

The PHY control block has a number of counters that are not enabled because the synchronous mode is used where PHY\_Clk is either 1/4 or 1/2 the frequency of the DDR2 or DDR3 SDRAM clock frequency.

At every rising edge of PHY\_Clk, a PHY control word is sent to the PHY control block with information for four memory clock cycles worth of commands and a 2-bit Seq count value.

The write enable to the control FIFO is always asserted and no operation (NOP) commands are issued between valid commands in the synchronous mode of operation. The Seq count must be incremented with every command sequence of four. The Seq field is used to synchronize PHY control blocks across multiple I/O banks.

The DDR3 SDRAM RESET\_N signal is directly controlled by the FPGA logic, not the PHY control word. The DDR2 SDRAM RESET\_N signal for RDIMM interfaces is directly controlled by the FPGA logic, not the PHY control word. The ODT and CKE signals are controlled by the PHY control block based on the auxiliary output field in the PHY control word. The assertion of ODT with respect to the write command and the duration for which it should be asserted can be specified using attributes associated with each auxiliary output bit described in [Table 1-30, page 85](#). The calibration logic or memory controller needs to assert the desired number of auxiliary output bits and set the associated attributes with every write and read command request. The PHY control block, in conjunction with the PHASER\_OUT, generates the write DQS and the DQ/DQS 3-state control signals during read and write commands.

The PHY cmd field is set based on whether the sequence of four commands has either a write, a read, or neither. The PHY cmd field is set to write if there is a write request in the command sequence. It is set to read if there is a read request in the command sequence, and it is set to non-data if there is neither a write nor a read request in the command sequence. A write and a read request cannot be issued within a sequence of four commands. The control offset field in the PHY control word defines when the command OUT\_FIFOs is read out and transferred to the IOLOGIC. The data offset defines when the data OUT\_FIFOs are read out with respect to the command OUT\_FIFOs being read. For read commands, the data offset is determined during calibration. The PHY control block assumes that valid data associated with a write command is already available in the DQ OUT\_FIFO when it is required to be read out.

### Command Path

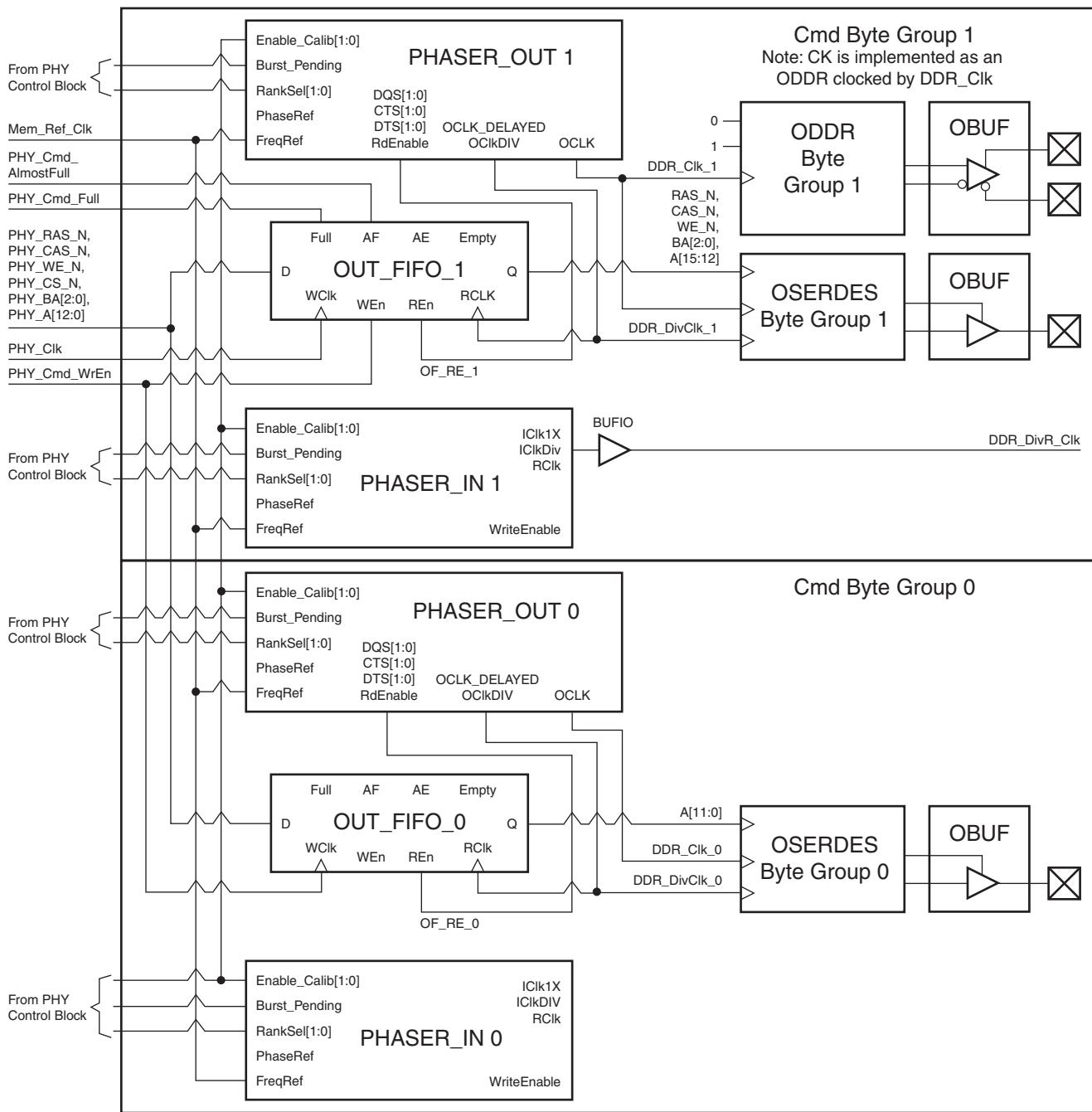
A command requested by the calibration logic or memory controller is sent out as a PHY control word to the PHY control block and a simultaneous input to the address/control/command OUT\_FIFOs. Each of the address/control/command signals must have values for four memory clock cycles because each PHY\_Clk cycle entails four memory clock cycles.

There are three types of commands:

- Write commands including write and write with auto precharge. The PHY command values in the PHY control word for both these write commands are the same (0x01). The difference is the address value input to the OUT\_FIFO. Address bit A10 is 1 for writes with auto precharge in the address OUT\_FIFOs.
- Read commands including read and read with auto precharge. The PHY command values in the PHY control word for both these read commands are the same (0x11). The difference is the address value input to the OUT\_FIFO. Address bit A10 is 1 for reads with auto precharge in the address OUT\_FIFOs.
- Non-Data commands including Mode Register Set, Refresh, Precharge, Precharge All Banks, Activate, No Operation, Deselect, ZQ Calibration Long, and ZQ Calibration Short. The PHY command values in the PHY control word for all these commands are the same (0x100). The ras\_n, cas\_n, we\_n, bank address, and address values input to the OUT\_FIFOs associated with these commands differ.

[Figure 1-38](#) shows the block diagram of the address/control/command path. The OSERDES is used in single data rate (SDR) mode because address/control/commands are SDR signals. A PHY control word is qualified with the Phy\_Ctl\_Wr\_N signal and an entry

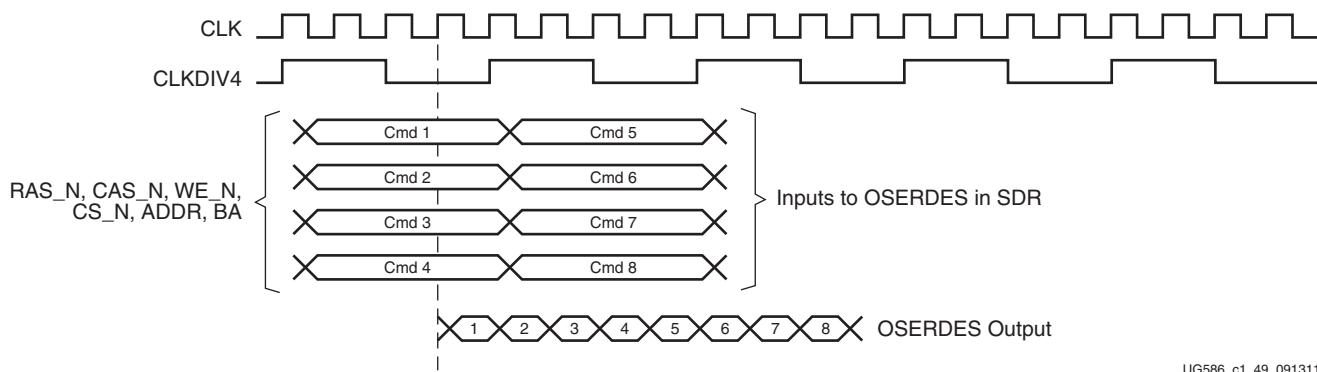
to the OUT\_FIFOs is qualified with the PHY\_Cmd\_WrEn signal. The FPGA logic need not issue NOP commands during long wait times between valid commands to the PHY control block because the default in the dedicated PHY for address/commands can be set to 0 or 1 as needed.



UG586\_c1\_48\_111110

Figure 1-38: Address/Command Path Block Diagram

The timing diagram of the address/command path from the output of the OUT\_FIFO to the FPGA pins is shown in [Figure 1-39](#).



UG586\_c1\_49\_091311

*Figure 1-39: Address/Command Timing Diagram*

### Datapath

The datapath comprises the write and read datapaths. The datapath in the 7 series FPGA is completely implemented in dedicated logic with IN/OUT\_FIFOs interfacing the FPGA logic. The IN/OUT\_FIFOs provide datapath serialization/deserialization in addition to clock domain crossing, thereby allowing the FPGA logic to operate at low frequencies up to 1/4 the frequency of the DDR2 or DDR3 SDRAM clock. [Figure 1-40](#) shows the block diagram of the datapath.

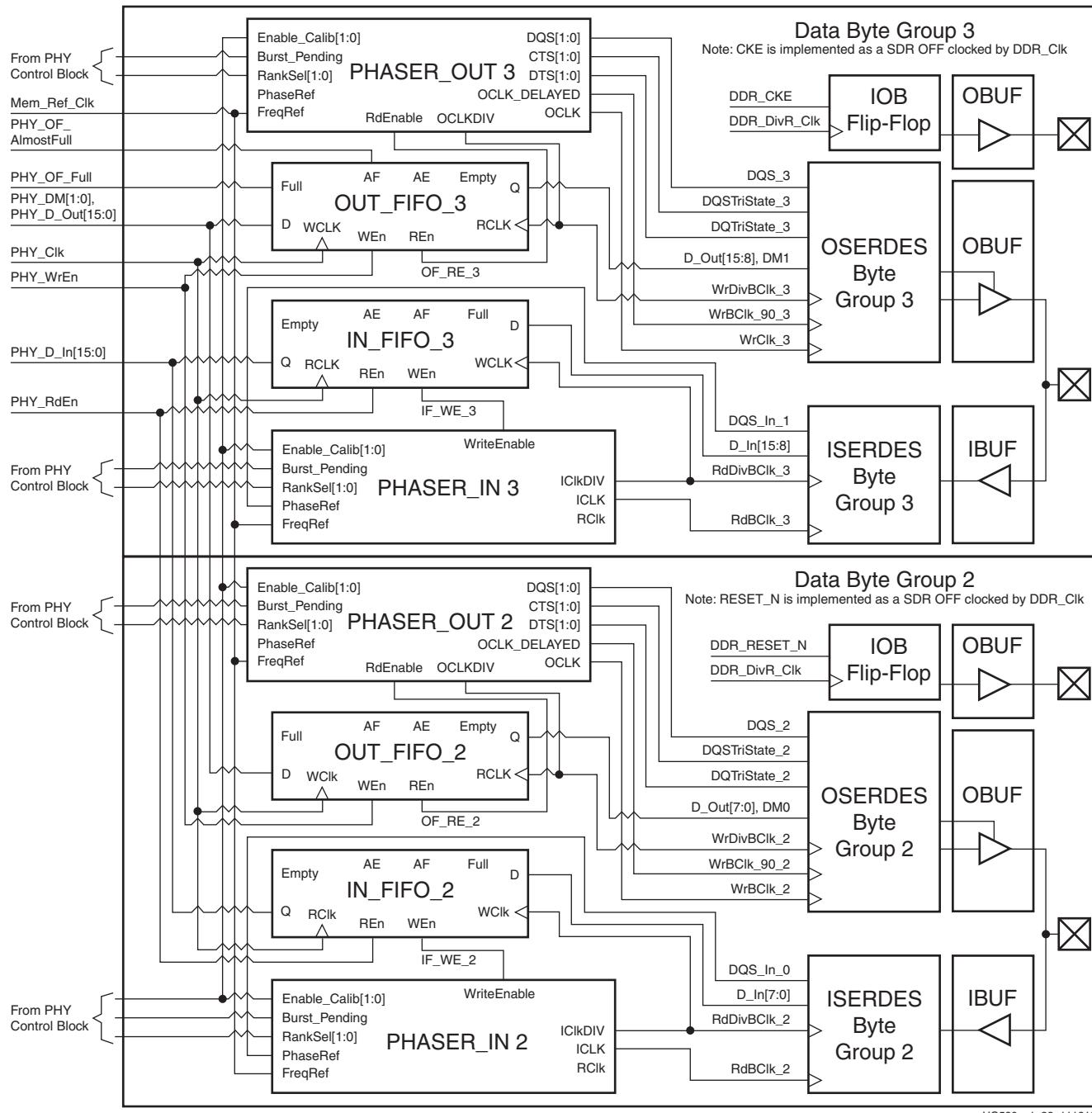


Figure 1-40: Datapath Block Diagram

Each IN/OUT\_FIFO has a storage array of memory elements arranged as 10 groups 8 bits wide and 8 entries deep. During a write, the OUT\_FIFO receives 8 bits of data for each DQ bit from the calibration logic or memory controller and writes the data into the storage array in the PHY\_Clk clock domain, which is 1/4 the frequency of the DDR2 or DDR3 SDRAM clock. The OUT\_FIFO serializes from 8 bits to 4 bits and outputs the 4-bit data to the OSERDES in the OCLKDIV domain that is half the frequency of the DDR2 or DDR3 SDRAM clock. The OSERDES further serializes the 4-bit data to a serial DDR data stream in the OCLK domain. The PHASER\_OUT clock output OCLK is used to clock DQ bits.

whereas the OCLK\_DELAYED output is used to clock DQS to achieve the 90-degree phase offset between DQS and its associated DQ bits during writes. During write leveling, both OCLK and OCLK\_DELAYED are shifted together to align DQS with CK at each DDR2 or DDR3 component.

The IN\_FIFO shown in [Figure 1-39](#) receives 4-bit data from each DQ bit ISERDES in a given byte group and writes them into the storage array. The IN\_FIFO is used to further deserialize the data by writing two of the 4-bit datagrams into each 8-bit memory element. This 8-bit parallel data is output in the PHY\_Clk clock domain which is 1/4 the frequency of the DDR2 or DDR3 SDRAM clock. Each read cycle from the IN\_FIFO contains all the byte data read during a burst length 8 memory read transaction. The data bus width input to the dedicated PHY is 8X that of the DDR2 or DDR3 SDRAM when running the FPGA logic at 1/4 the frequency of the DDR2 or DDR3 SDRAM clock.

## Calibration and Initialization Stages

### 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 like BURST\_MODE (BL), BURST\_TYPE, CAS latency (CL), CAS write latency (CWL), additive latency (AL), write recovery for auto precharge (tWR), on-die termination resistor values (RTT\_NOM and RTT\_WR), and output driver strength (OUTPUT\_DRV).

### Write Leveling

Write leveling, which is a feature available in DDR3 SDRAM, is performed in this stage of calibration. 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. Write leveling, a new feature in DDR3 SDRAMs, 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 t<sub>DQSS</sub> 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 has the capability to delay DQS until a 0-to-1 transition is detected on DQ. Write leveling is performed once after power up. The calibration logic ORs the DQ bits in a byte to determine the transition because different memory vendors use different bits in a byte as feedback. The DQS delay can be achieved with the PHASER\_OUT fine and coarse delay adjustment in the 7 series FPGAs.

[Figure 1-41](#) shows the write leveling block diagram.

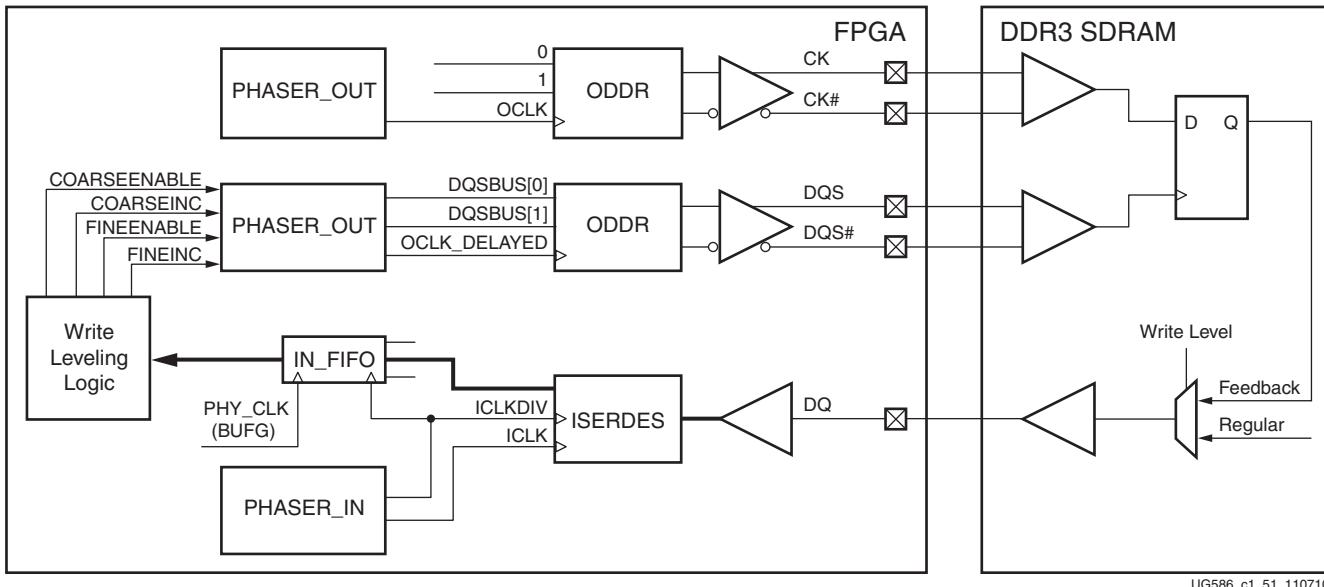


Figure 1-41: Write Leveling Block Diagram

The timing diagram for write leveling is shown in Figure 1-42. Periodic DQS pulses are output by the FPGA memory interface to detect the level of the CK clock at the DDR3 SDRAM device. The interval between DQS pulses is specified as a minimum of 16 clock cycles. DQS is delayed using the PHASER\_OUT fine and coarse delay 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  $t_{DQSS}$  specification.

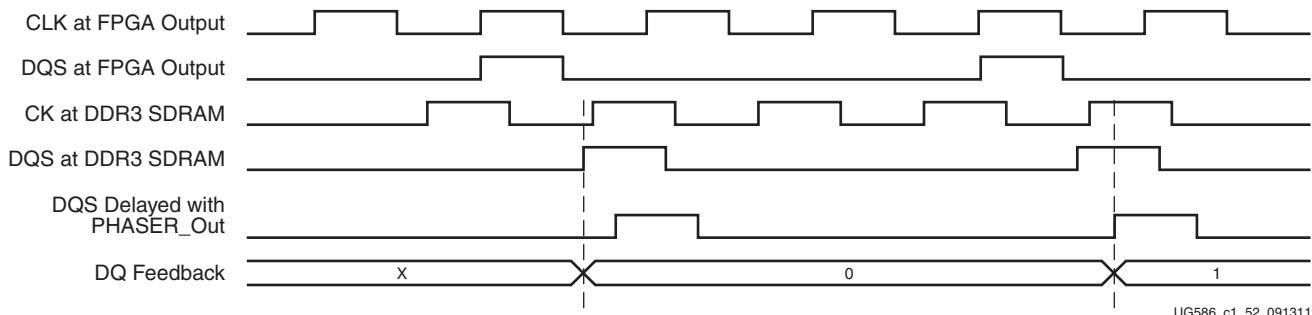
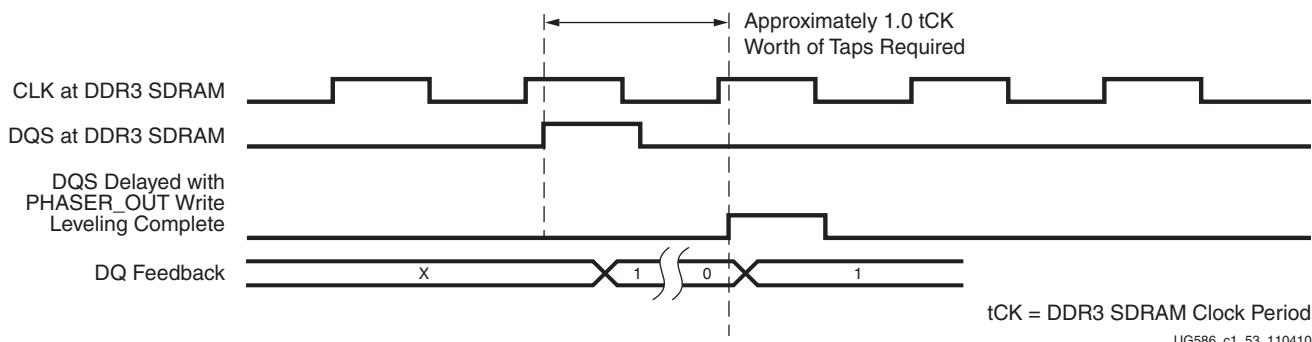


Figure 1-42: Write Leveling Timing Diagram

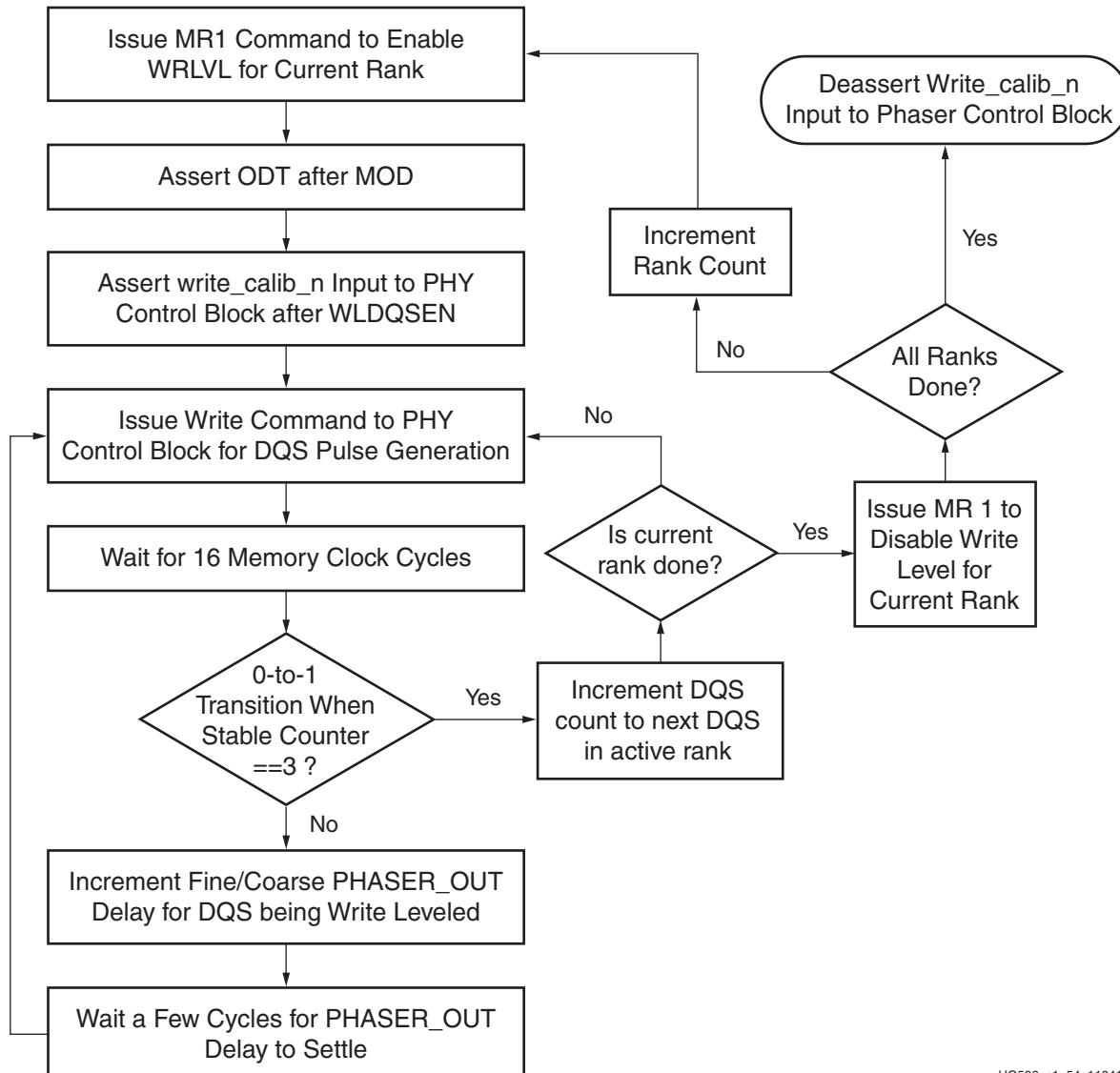
**Figure 1-43** shows that the worst-case delay required during write leveling can be one tCK (DDR3 SDRAM clock period).



**Figure 1-43: Write Leveling Taps Requirement**

### Implementation Details

The Write\_Calib\_N signal indicating the start of write leveling mode is input to the PHY control block after tWLDQSEN to ensure that DQS is driven Low after ODT is asserted. In this mode, periodic write requests must be issued to the PHY control block to generate periodic DQS pulses for write leveling. During write leveling, PHASER\_IN outputs a free-running clock used to capture the DQ feedback to the DQ IN\_FIFOs. During write leveling, the data byte group IN\_FIFOs is in flow-through mode. [Figure 1-44](#) shows the flow diagram of the sequence of commands during write leveling. The PHASER\_OUT fine phase shift taps are incremented one tap at a time to observe a 0-to-1 transition on the feedback DQ. A stable counter is implemented in the write leveling logic to mitigate the risk of finding a false edge in the jitter region. A counter value of 3 means that the sampled data value was constant for 3 consecutive tap increments and DQS is considered to be in a stable region with respect to CK. The counter value is reset to 0 whenever a value different from the previous value is detected. Edge detection is inhibited when the stable counter value is less than 3. The write\_calib\_n signal is deasserted when write leveling is performed on all DQSSs in all ranks.



UG586\_c1\_54\_110410

Figure 1-44: Write Leveling Flow Diagram

### PHASER\_IN Phase Lock

PHASER\_IN is placed in the read calibration mode to phase align its free-running frequency reference clock to the associated read DQS. The calibration logic issues back-to-back read commands to provide the PHASER\_IN block with a continuous stream of DQS pulses for it to achieve lock. A continuous stream of DQS pulses is required for the PHASER\_IN block to phase align the free-running frequency reference clock to the associated read DQS. Each DQS has a PHASER\_IN block associated with it. When the PHASER\_IN lock signal (pi\_phase\_locked) of all the DQS PHASER\_INs are asserted, the calibration logic deasserts the read calibration signal to put the PHASER\_INs in normal operation mode.

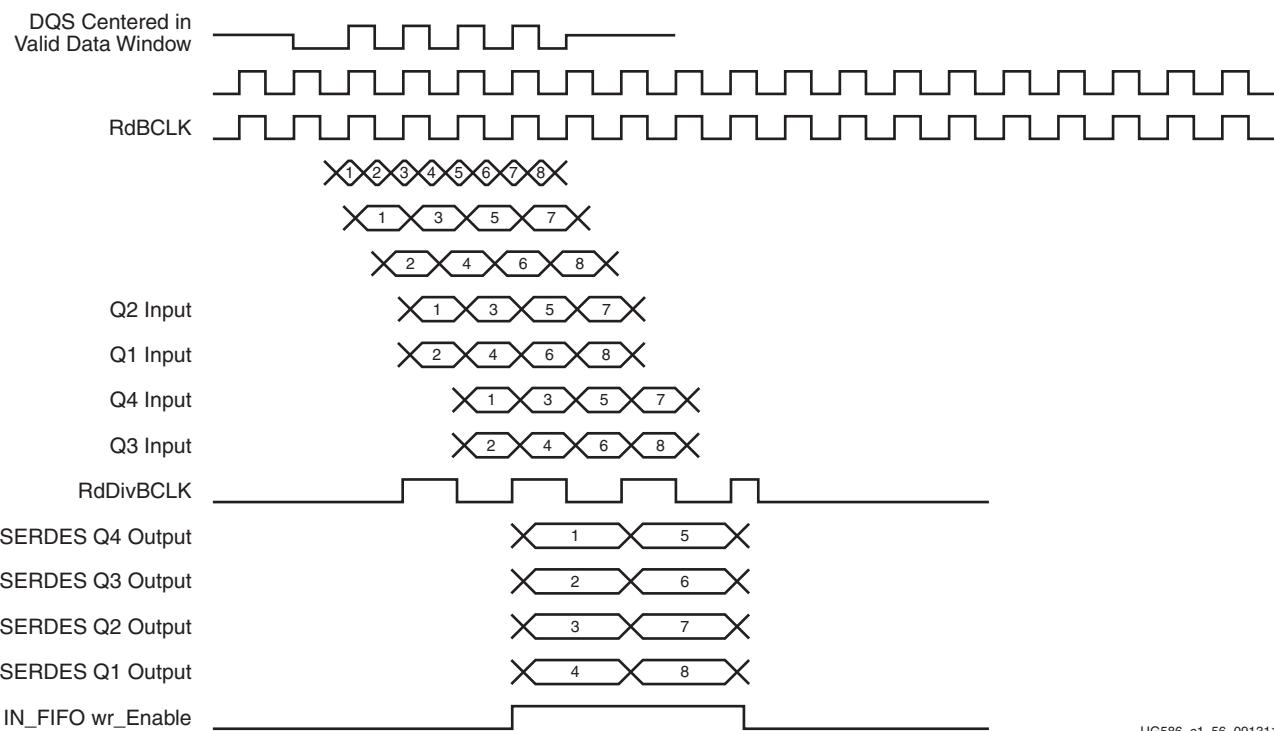
### PHASER\_IN DQSFOUND Calibration

This calibration stage is required to align the different DQS groups to the same PHY\_Clk clock edge. Different DQS groups have different skews with respect to each other because

of clock (CK) fly-by routing differences to each DDR2 or DDR3 component, and delay differences in each component. This calibration stage is required to determine the optimal position of read data\_offset with respect to the read command for the entire interface.

In this stage of calibration, the PHASER\_IN block is in normal operation mode and the calibration logic issues a set of four back-to-back read commands with gaps in between. The data\_offset associated with the first read command is not accurate because the round-trip delays are unknown. The data\_offset for the first set of read commands is set to CL+7. The data\_offset value for the subsequent set of reads is decremented one memory clock cycle at a time until the DQSFOUND output from the PHASER\_IN block is asserted. When the DQSFOUND signal is asserted for all the bytes, this calibration stage is complete.

Each byte group can be read out of the IN\_FIFO on different PHY\_Clk cycles due to fly-by routing differences and delay differences within each group. Therefore, the IN\_FIFO Not Empty flags for all the byte groups are ANDed together and used as the read enable for all data IN\_FIFOs. [Figure 1-45](#) shows the read data capture timing diagram.



UG586\_c1\_56\_091311

*Figure 1-45: Read Data Capture Timing Diagram*

### Read Leveling

Read leveling stage 1 is required to center align the read strobe in the read valid data window for the first stage of capture. In strobe-based memory interfaces like DDR2 or DDR3 SDRAM, the second stage transfer requires an additional pulse which in 7 series FPGAs is provided by the PHASER\_IN block. This stage of calibration uses the PHASER\_IN stage 2 fine delay line to center the capture clock in the valid DQ window. The capture clock is the free-running FREQ\_REF clock that is phase aligned to read DQS in the PHASER\_IN phase locked stage. A PHASER\_IN provides two clock outputs namely ICLK and ICLKDIV. ICLK is the stage 2 delay output and ICLKDIV is the rising edge aligned divided by 2 version of ICLK.

The ICLK and ICLKDIV outputs of one PHASER\_IN block are used to clock all the DQ ISERDES associated with one byte. The ICLKDIV is also the write clock for the read DQ IN\_FIFOs. One PHASER\_IN block is associated with a group of 12 I/Os. Each I/O bank in the 7 series FPGA has four PHASER\_IN blocks, and hence four bytes for DDR2 or DDR3 SDRAM can be placed in a bank.

### Implementation Details

This stage of read leveling is performed one byte at a time where each DQS is center aligned to its valid byte window. At the start of this stage, a write command is issued to a specified DDR2 or DDR3 SDRAM address location with a predefined data pattern. This write command is followed by back-to-back read commands to continuously read data back from the same address location that was written to.

The algorithm first increments the IDELAY taps for all DQ bits in a byte simultaneously until an edge is detected. At the end of the IDELAY increments, DQS is at or before the left edge of the window.

The calibration logic reads data out of the IN\_FIFO and records it for comparison. The data pattern sequence is important for this stage of calibration. No assumption is made about the initial relationship between DQS and the data window at tap 0 of the fine delay line. The algorithm then delays DQS using the PHASER\_IN fine delay line until a DQ window edge is detected.

An averaging algorithm is used for data window detection where data is read back over multiple cycles at the same tap value. The number of sampling cycles is set to 214. In addition to averaging, there is also a counter to track whether DQS is positioned in the unstable jitter region. A counter value of 3 means that the sampled data value was constant for three consecutive tap increments and DQS is considered to be in a stable region. The counter value is reset to 0 whenever a value different from the previous value is detected. The next step is to increment the fine phase shift delay line of the DQS PHASER\_IN block one tap at a time until a data mismatch is detected. The data read out of IN\_FIFO after the required settling time is then compared with the recorded data at the previous tap value. This is repeated until a data mismatch is found, indicating the detection of a valid data window edge. A valid window is the number of PHASER\_IN fine phase shift taps for which the stable counter value is a constant 3. This algorithm mitigates the risk of detecting a false valid edge in the unstable jitter regions.

There are three possible scenarios for the initial DQS position with respect to the data window. The first valid rising edge of DQS could either be in the previous data window, in the left noise region of the current data window, or just past the left noise region inside the current data window. The PHASER\_IN fine delay line has 64 taps. (A bit time worth of taps. Tap resolution therefore changes with frequency.) The first two scenarios would result in the left data window edge being detected with a tap count less than 1/2 the bit time and the second window edge might or might not be detected, depending on the frequency and the width of the noise region. The third scenario results in the right window edge being detected with a tap count close to a bit time. When both edges are detected, the final DQS tap value is computed as:

$$\text{first\_edge\_taps} + (\text{second\_edge\_taps} - \text{first\_edge\_taps})/2.$$

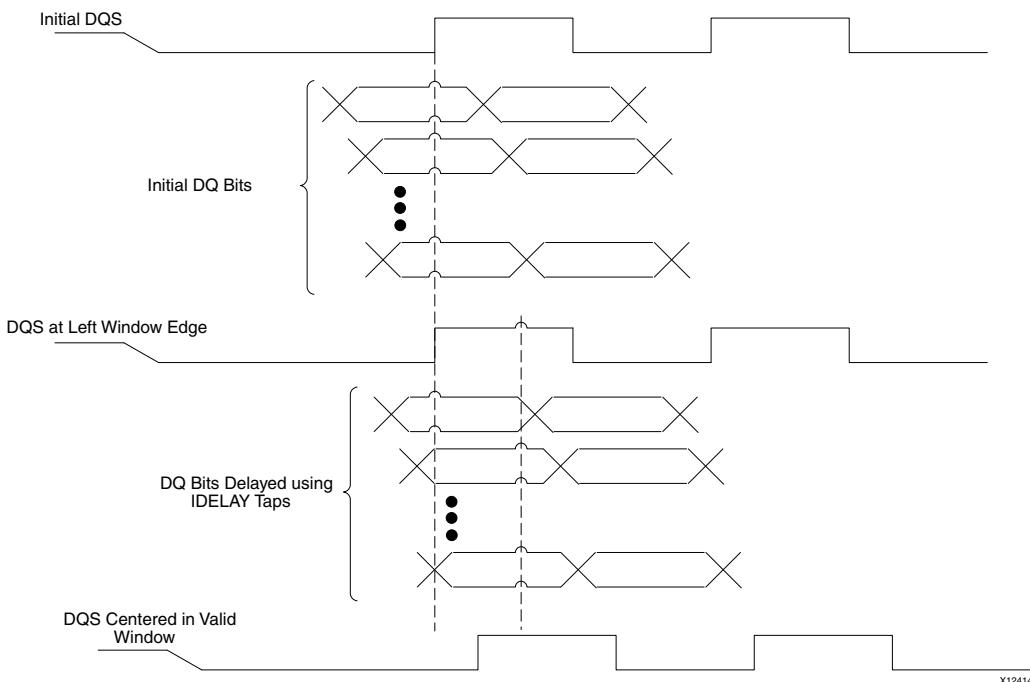
When only one edge is detected and the tap value of the detected edge is less than 1/2 of a bit time, the final DQS tap value is computed as:

$$(\text{first\_edge\_taps} + (63 - \text{first\_edge\_taps})/2)$$

When only one edge is detected and the tap value of the detected edge is almost a bit time, the final DQS tap value is computed as:

$$(63 - ((63 - \text{first\_edge\_taps})/2))$$

[Figure 1-46](#) shows the timing diagram for DQS center alignment in the data valid window.



[Figure 1-46: Read Leveling Stage 1 Timing Diagram](#)

### Write Calibration

Write calibration is performed after both stages of read leveling because correct data pattern sequence detection is necessary for this stage of calibration. Write calibration is required to align DQS to the correct CK edge. During write leveling, DQS is aligned to the nearest rising edge of CK. However, this might not be the edge that captures the write command. Depending on the interface type (UDIMM, RDIMM, or component), the DQS could either be one CK cycle earlier than, one CK cycle later than, or aligned to the CK edge that captures the write command. [Figure 1-47](#) shows several different scenarios based on the initial phase relationship between DQS and CK for a UDIMM or RDIMM interface. [Figure 1-48](#) shows an initial DQS to CK alignment case for component interfaces. The assumption is that component interfaces also use the fly-by topology, thereby requiring write leveling.

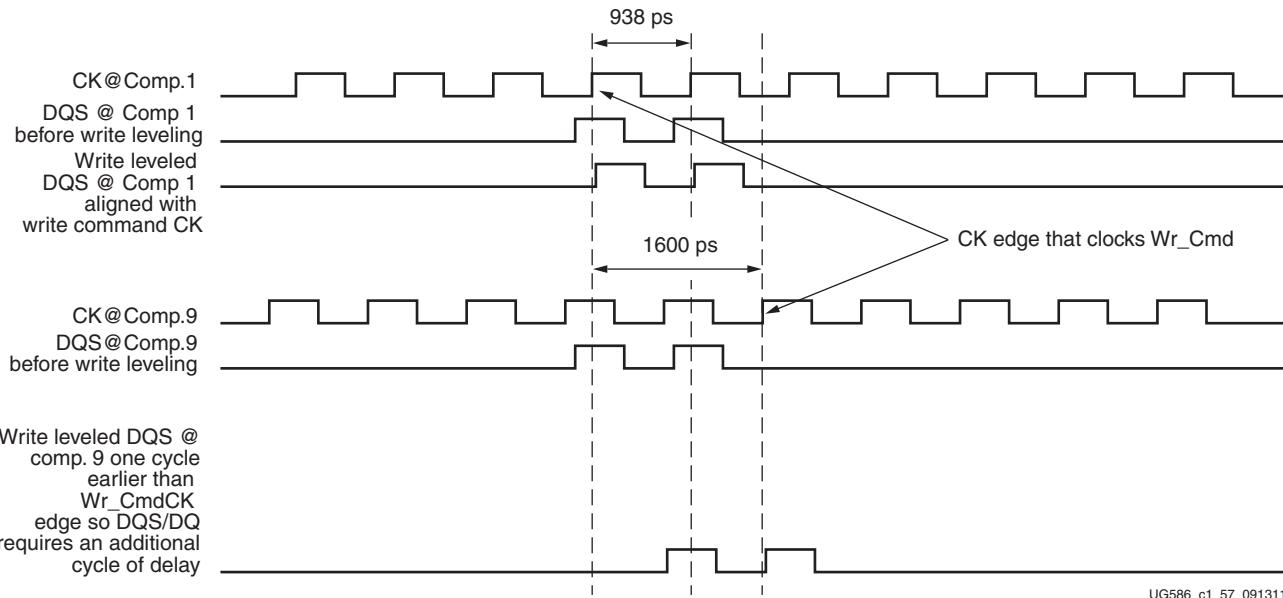


Figure 1-47: UDIMM/RDIMM DQS-to-CK Initial Alignment

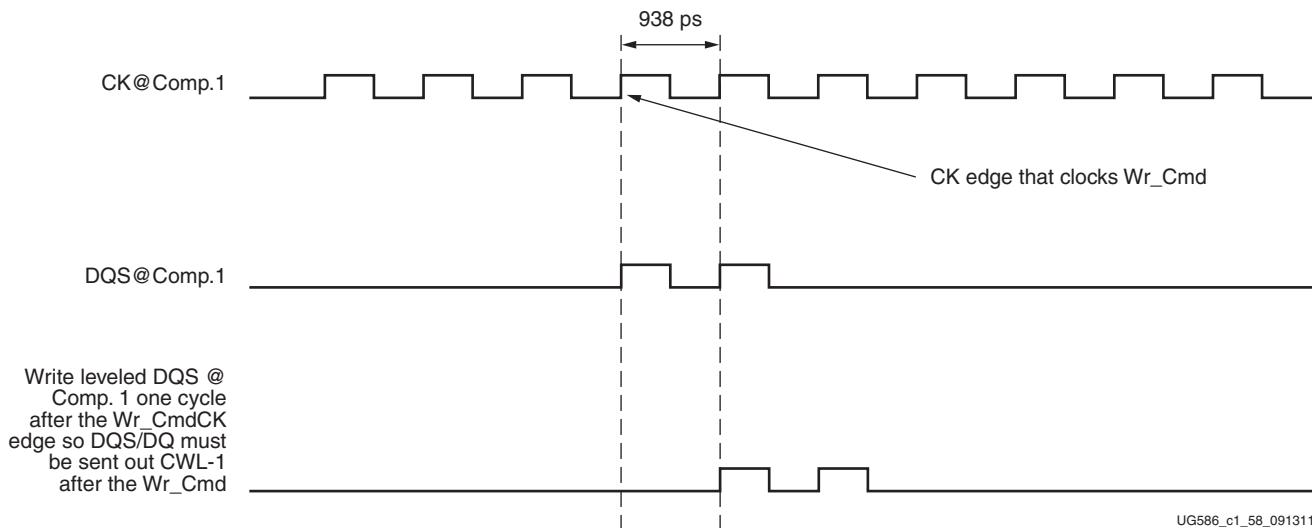


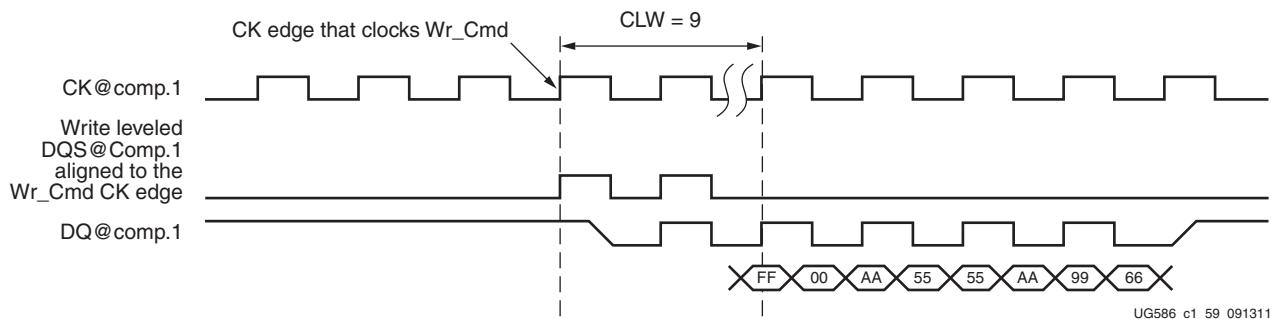
Figure 1-48: Component DQS-to-CK Initial Alignment

The PHASER\_OUT fine and coarse delay provides 1 tCK worth of delay for write leveling. The additional clock cycle of delay required to align to the correct CK edge is achieved using the coarse delay line. If the total delay required is over one clock cycle, the div\_cycle\_delay input to the PHASER\_OUT block need not be asserted because a circular buffer was added to the PHASER\_OUT block.

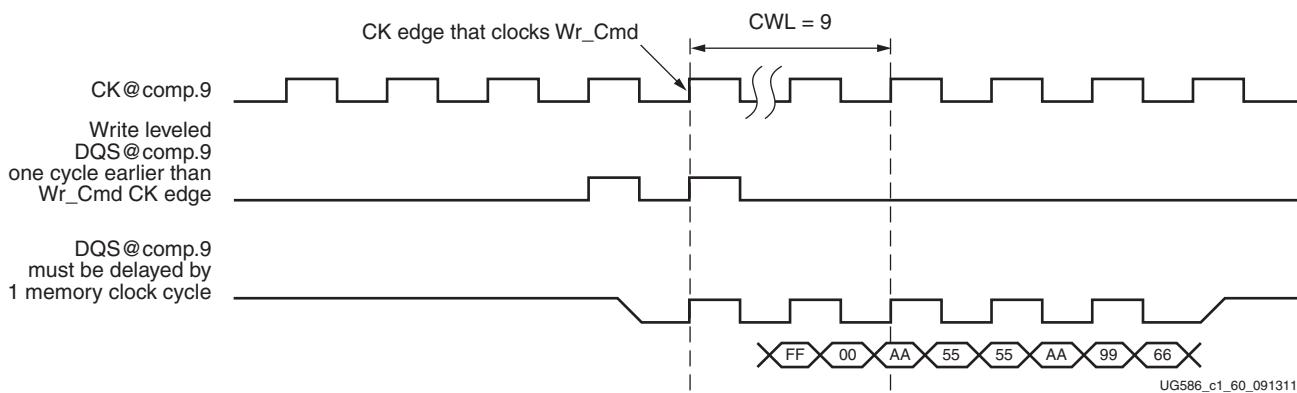
### Implementation Details

A write command is issued with a known write data pattern (FF 00 AA 55 55 AA 99 66) to a specific location. This is followed by a read command to the same location. The data read back out of the IN\_FIFO is compared with the expected data pattern on a byte basis. If the data read out matches the expected pattern, no further changes are required in

the write path for that byte, as shown in [Figure 1-49](#). If the first two data words read back match the second set of data words in the expected pattern, the DQS and DQ 3-state signal must be delayed by one memory clock. This scenario is shown in [Figure 1-50](#). After all the bytes are calibrated, the calibration logic asserts the init\_calib\_complete signal indicating the completion of the initialization and calibration sequence. The memory controller can now drive the address, command, and data buses.



**Figure 1-49: DQS Aligned to the Correct CK Edge - No Change in Write Path**



**Figure 1-50: DQS Aligned to Incorrect CK Edge - Delay DQS/DQ by One Memory Clock Cycle**

### Memory Controller to PHY Interface

The calibration logic module constructs the PHY control word before sending it to the PHY control block during calibration. After calibration is complete, the init\_calib\_complete signal is asserted and sent to the memory controller to indicate that normal operation can begin. To avoid latency increase, the memory controller must send commands in the format required by the dedicated PHY block. As a result, the address, command, control, and data buses are multiplexed before being sent to the PHY control block. These buses are driven by the calibration module during the memory initialization and calibration stages and by the memory controller during normal operation. [Table 1-31](#) describes the memory controller to PHY interface signals. These signals are synchronous to the fabric clock.

Table 1-31: Memory Controller to Calibration Logic Interface Signals

Signal Name	Width	I/O To/From PHY	Type	Description
rst	1	Input		The rstdiv0 output from the infrastructure module synchronized to the PHY_Clk domain.
PHY_Clk	1	Input		This clock signal is 1/4 the frequency of the DDR2 or DDR3 clock.
mem_refclk	1	Input		This is the DDR2 or DDR3 frequency clock.
freq_refclk	1	Input		This signal is the same frequency as mem_refclk between 400 MHz to 933 MHz, and 1/2 or 1/4 of mem_refclk for frequencies below 400 MHz.
sync_pulse	1	Input		This is the synchronization pulse output by the PLL.
pll_lock	1	Input		The LOCKED output of the PLL instantiated in the infrastructure module.
mc_ras_n	[nCK_PER_CLK0 – 1:0]	Input	Active Low	mc_xxx_n[0] is the first cmd in the sequence of four.
mc_cas_n	[nCK_PER_CLK – 1:0]	Input	Active Low	mc_xxx_n[0] is the first cmd in the sequence of four.
mc_we_n	[nCK_PER_CLK – 1:0]	Input	Active Low	mc_xxx_n[0] is the first cmd in the sequence of four.
mc_address	[ROW_WIDTH × nCK_PER_CLK – 1:0]	Input		mc_address[ROW_WIDTH – 1:0] is the first command address in the sequence of four.
mc_bank	[BANK_WIDTH × nCK_PER_CLK – 1:0]	Input		mc_bank[BANK_WIDTH – 1:0] is the first command bank address in the sequence of four.
mc_cs_n	[CS_WIDTH × nCS_PER_RANK × nCK_PER_CLK – 1:0]	Input		mc_cs_n [CS_WIDTH – 1:0] is the cs_n associated with the first command in the sequence.
mc_reset_n	1	Input	Active Low	mc_reset_n is input directly to the IOLOGIC without an OUT_FIFO.
mc_wrdata	[2 × nCK_PER_CLK × DQ_WIDTH – 1:0]	Input		This is the write data to the dedicated PHY. It is 8x the memory DQ width for a 4:1 clock ratio.
mc_wrdata_mask	[2 × nCK_PER_CLK × (DQ_WIDTH/8) – 1:0]	Input		This is the write data mask to the dedicated PHY. It is 8x the memory DM width for a 4:1 clock ratio.
mc_wrdata_en	1	Input	Active High	This signal is the WREN input to the DQ OUT_FIFO.
mc_cmd_wren	1	Input	Active Low	This signal is the write enable input of the address/command OUT_FIFOs.

Table 1-31: Memory Controller to Calibration Logic Interface Signals (Cont'd)

Signal Name	Width	I/O To/From PHY	Type	Description
mc_ctl_wren	1	Input	Active Low	This signal is the write enable input to the PHY control word FIFO in the dedicated PHY block.
mc_cmd	[2:0]	Input		This signal is used for PHY_Ctl_Wd configuration: 0x04: Non-data command (No column command in the sequence of commands) 0x01: Write command 0x03: Read command
mc_data_offset	[5:0]	Input		This signal is used for PHY_Ctl_Wd configuration: 0x00: Non-data command (No column command in the sequence of commands) CWL + COL cmd position: Write command calib_rd_data_offset+COL cmd position - 1: Read command
mc_aux_out0	[3:0]	Input	Active High	This is the auxiliary outputs field in the PHY control word used to control ODT and CKE assertion.
mc_aux_out1	[3:0]	Input	Active High	This is the auxiliary outputs field in the PHY control word used to control ODT and CKE assertion for four-rank interfaces.
mc_rank_cnt	[1:0]	Input		This is the rank accessed by the command sequence in the PHY control word.
phy_mc_ctl_full	1	Output	Active High	Bitwise AND of all the Almost FULL flags of all the PHY Control FIFOs. The Almost FULL flag is asserted when the FIFO is one entry away from being FULL.
phy_mc_cmd_full	1	Output	Active High	Bitwise OR of all the Almost FULL flags of all the command OUT_FIFOs. The Almost FULL flag is asserted when the FIFO is one entry away from being FULL.
phy_mc_data_full	1	Output	Active High	Bitwise OR of all the Almost FULL flags of all the write data OUT_FIFOs. The Almost FULL flag is asserted when the FIFO is one entry away from being FULL.
phy_rd_data	[2 × nCK_PER_CLK × DQ_WIDTH – 1:0]	Output		This is the read data from the dedicated PHY. It is 8x the memory DQ width for a 4:1 clock ratio.
phy_rddata_valid	1	Output	Active High	This signal is asserted when valid read data is available.
calib_rd_data_offset	[6 × RANKS – 1:0]	Output		This signal is the calibrated read data offset value with respect to command 0 in the sequence of four commands.

Table 1-31: Memory Controller to Calibration Logic Interface Signals (Cont'd)

Signal Name	Width	I/O To/From PHY	Type	Description
init_calib_complete	1	Output	Active High	This signal is asserted after memory initialization and calibration are completed.

**Notes:**

1. The parameter nCK\_PER\_CLK defines the number of DDR2 or DDR3 SDRAM clock cycles per PHY\_Clk cycle.
2. The parameter ROW\_WIDTH is the number of DDR2 or DDR3 SDRAM ranks.
3. The parameter BANK\_WIDTH is the number of DDR2 or DDR3 SDRAM banks.
4. The parameter CS\_WIDTH is the number of DDR2 or DDR3 SDRAM cs\_n signals.
5. The parameter CKE\_WIDTH is the number of DDR2 or DDR3 SDRAM CKE signals.
6. The parameter DQ\_WIDTH is the width of the DDR2 or DDR3 SDRAM DQ bus.

## 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 117](#) 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 an 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. See this specification [\[Ref 3\]](#) 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 AXSIZE. The address at the input of the memory controller user interface from the AXI shim can be configured in two modes. See [User Interface, page 57](#) for more details.

The normalization process remaps the address based on the data width in the AXI shim. [Table 1-32](#) shows the first step in normalization done based on user interface data widths. The value of the user interface data width is equal to  $2 * \text{nCK\_PER\_CLK} * \text{PAYLOAD\_WIDTH}$ .

The PAYLOAD\_WIDTH parameter, in the user design top, specifies the width of the DQ bus used for the user data. See [Table 1-35](#) for more information on parameters. The shift operation is performed to define the ratio between DRAM width and the UI width.

**Table 1-32: Shift operation for AXI Address Based on Data Width**

UI Data Width	Right Shift (Applied to Address Bits for 2:1 Memory Controller to DRAM Clock Ratio)	Right Shift (Applied to Address Bits for 4:1 Memory Controller to DRAM Clock Ratio)
32	No shift	Not applicable
64	Shift by 1	No shift
128	Shift by 2	Shift by 1
256	Shift by 3	Shift by 2
512	Shift by 4	Shift by 3
1024	Shift by 5	Shift by 4

The resultant address is then masked using the following equation for generating the mask bits.

$$\text{BURST\_MASK} = \{\text{ADDR\_WIDTH}\{1'b1\}\} \wedge \{(\text{BURST\_LEN} + \text{nCK\_PER\_CLK}/2)\{1'b1\}\};$$

Where:

1. ADDR\_WIDTH is the memory controller address width
2. BURST\_LEN is equal to "1" if nCK\_PER\_CLK is "4" and equal to "2" for BL8 and nCK\_PER\_CLK is "2"
3. nCK\_PER\_CLK is the memory controller clock to DRAM clock ratio.

## Example 1

With a 2:1 memory controller to DRAM clock ratio, the AXI data width is 64 bits and the address applied for the given transaction on the AXI interface is 0092\_4920 (Hex). The following steps indicate the address value in each normalization step:

1. After the first shift operation, the resulting address is 0049\_2490 (Hex).
2. Assuming the address width to be 28 and burst mode to be BL8, the mask value is 0FFF\_FFF8 (Hex).
3. The resulting address is 32'h0049\_2490.
4. The address driven to the DRAM interface if the memory address order is "ROW\_BANK\_COLUMN" is ROW = 0249 (Hex), BANK = 1 (Hex) and COLUMN = 090 (Hex).

## Example 2

With a 4:1 memory controller to DRAM clock ratio, the AXI data width is 64 bits and the address applied for the given transaction on the AXI interface is 0000\_2490 (Hex). The following steps indicate the address value in each normalization step:

1. Because there is no shift operation the resulting address will be 0000\_2490 (Hex)
2. Assuming the address width to be 28 and burst mode to be BL8, the mask value is 0FFF\_FFF8 (Hex), nCK\_PER\_CLK value is 4, and BURST\_LEN value is 1.

3. The resulting address is 32'h0000\_2490.
4. This address driven to the DRAM interface if the memory address order is “BANK\_ROW\_COLUMN” is ROW = 0009 (Hex), BANK = 0 (Hex) and COLUMN = 090 (Hex).

The address increments are performed as defined by [Table 1-33](#) for AXI burst transactions.

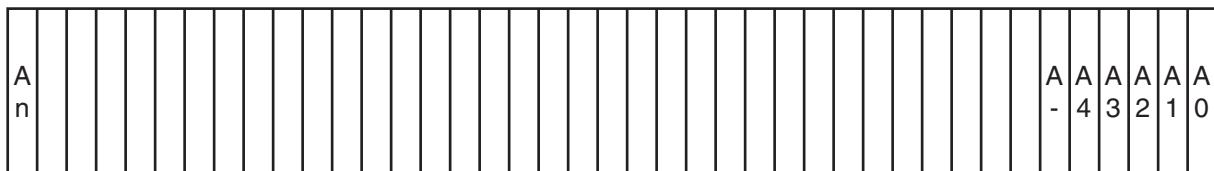
**Table 1-33: Address Increments for Multiple AXI Bursts**

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
1024	1	128
1024	2	256

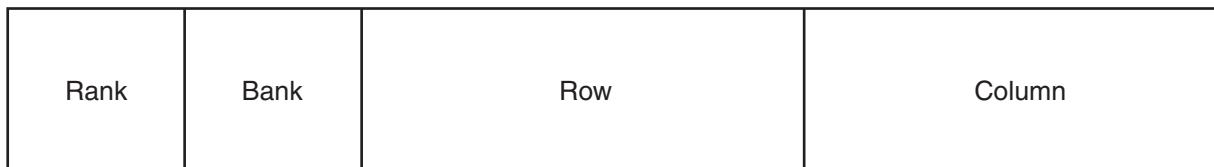
## User Interface

The mapping between the User Interface address bus and the physical memory row, bank and column can be configured. Depending on how the application data is organized, addressing scheme Bank- Row-Column or Row-Bank-Column can be chosen to optimize controller efficiency. These addressing schemes are shown in [Figure 1-51](#) and [Figure 1-52](#).

User Address



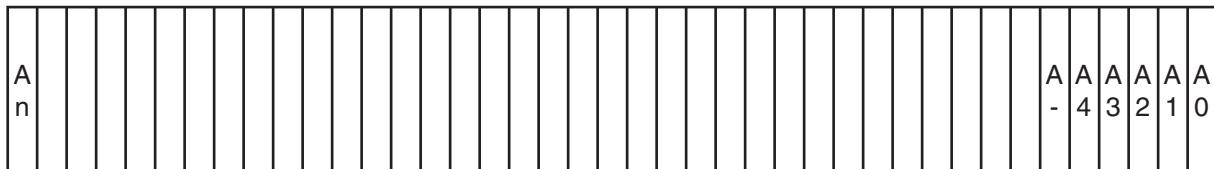
Memory



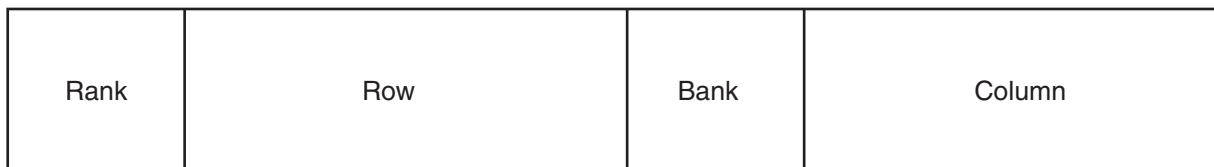
UG586\_c1\_61\_091410

**Figure 1-51: Memory Address Mapping for Bank-Row-Column Mode in UI Module**

User Address



Memory

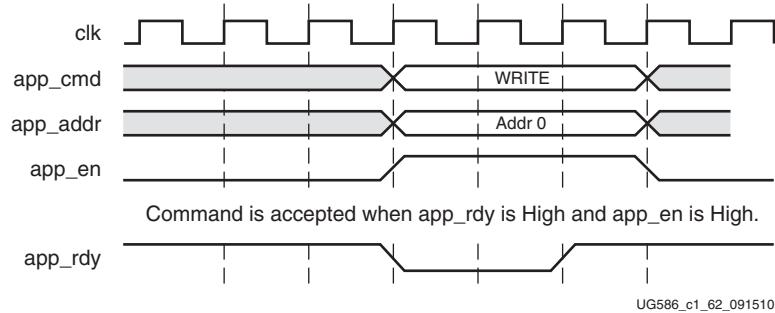


UG586\_c1\_61a\_012411

**Figure 1-52: Memory Address Mapping for Row-Bank-Column Mode in 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 1-53](#).



**Figure 1-53: UI Command Timing Diagram with app\_rdy Asserted**

A non back-to-back write command can be issued as shown in [Figure 1-54](#). This figure depicts three scenarios for the app\_wdf\_data, app\_wdf\_wren, and app\_wdf\_end signals, as follows:

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.

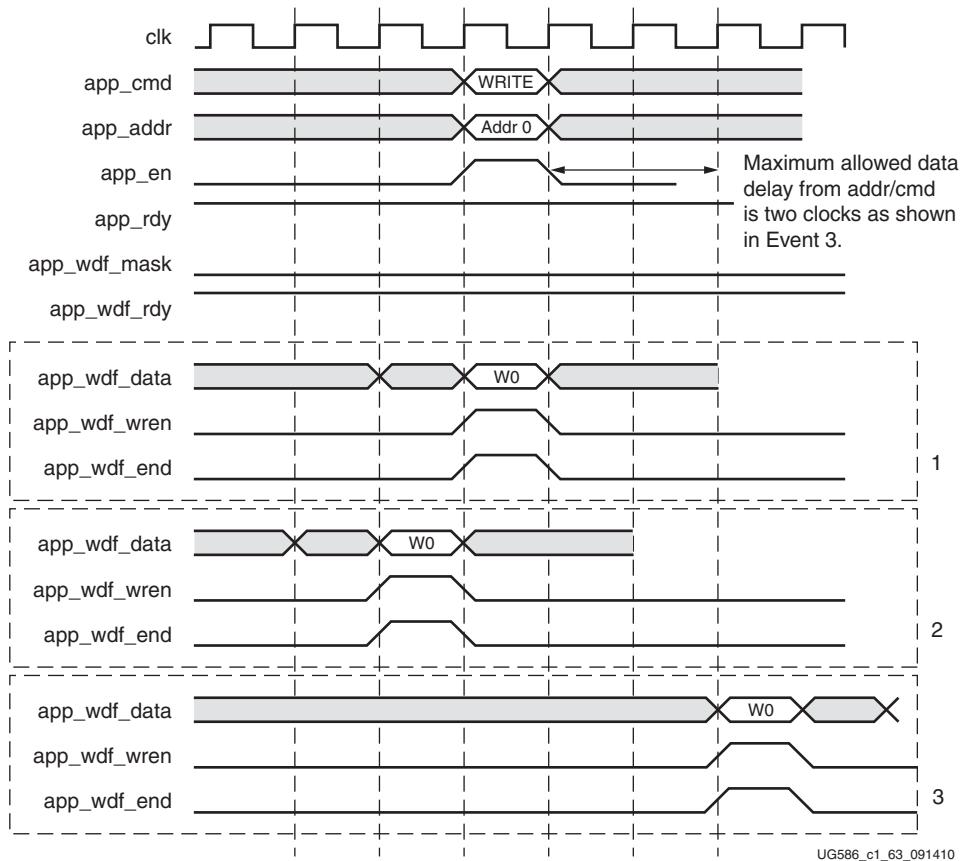


Figure 1-54: 4:1 Mode UI Interface Write Timing Diagram  
(Memory Burst Type = BL8)

### Write Path

The write data is registered in the write FIFO when **app\_wdf\_wren** is asserted and **app\_wdf\_rdy** is High (Figure 1-55). 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.

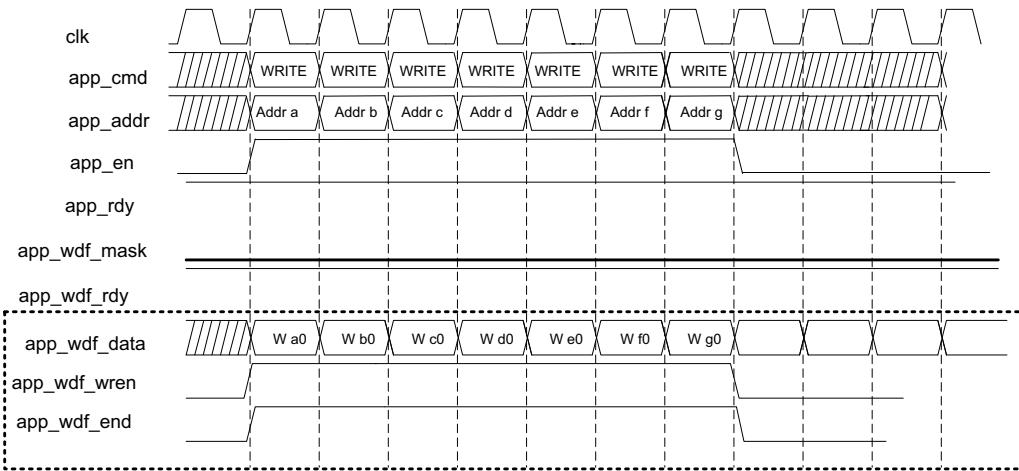


Figure 1-55: 4:1 Mode UI Interface Back-To-Back Write Commands Timing Diagram  
(Memory Burst Type = BL8)

As shown in Figure 1-53, page 106, 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 1-56.

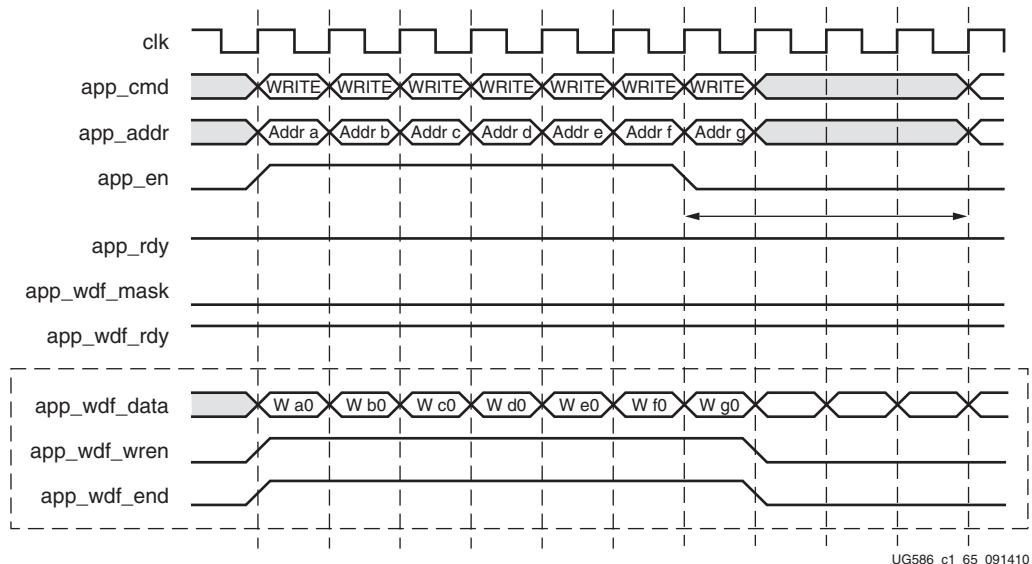
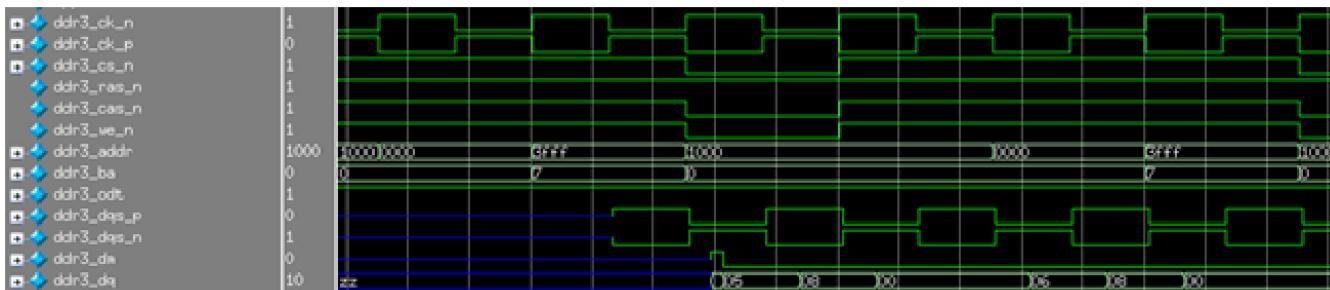


Figure 1-56: 4:1 Mode UI Interface Back-to-Back Write Commands Timing Diagram  
(Memory Burst Type = BL8)

The **app\_wdf\_end** signal must be used to indicate the end of a memory write burst. For memory burst types of eight, the **app\_wdf\_end** signal must be asserted on the second write data word.

The map of the application interface data to the DRAM output data can be explained with an example.

For a 4:1 memory controller to DRAM clock ratio with an 8-bit memory, at the application interface, if the 64-bit data driven is 0000\_0806\_0000\_0805 (Hex), the data at the DRAM interface is as shown in Figure 1-57. This is for a BL8 (Burst Length 8) transaction.



*Figure 1-57: Data at the DRAM Interface for 4:1 Mode*

The data values at different clock edges are as shown in Table 1-34.

**Table 1-34: Data Values at Different Clock Edges**

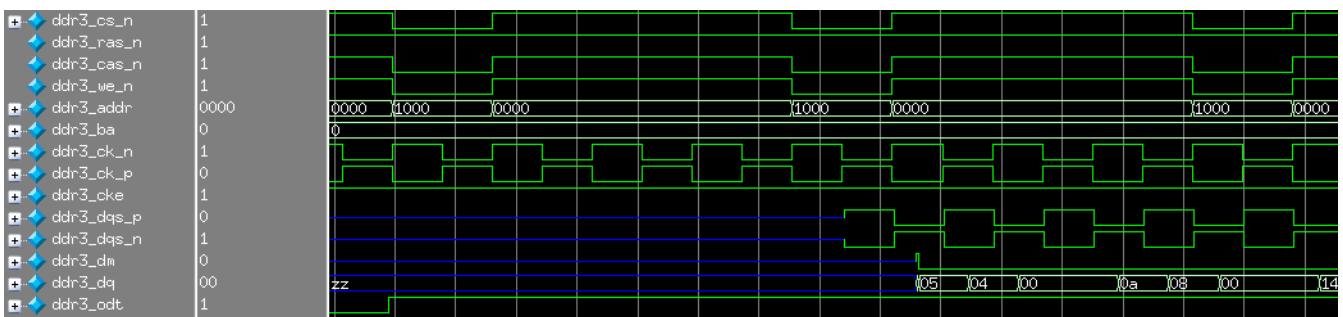
Rise0	Fall0	Rise1	Fall1	Rise2	Fall2	Rise3	Fall3
05	08	00	00	06	08	00	00

For a 2:1 memory controller to DRAM clock ratio, the application data width is 32 bits. Hence for BL8 transactions, 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 1-58. 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 1-58: Data at the Application Interface for 2:1 Mode*

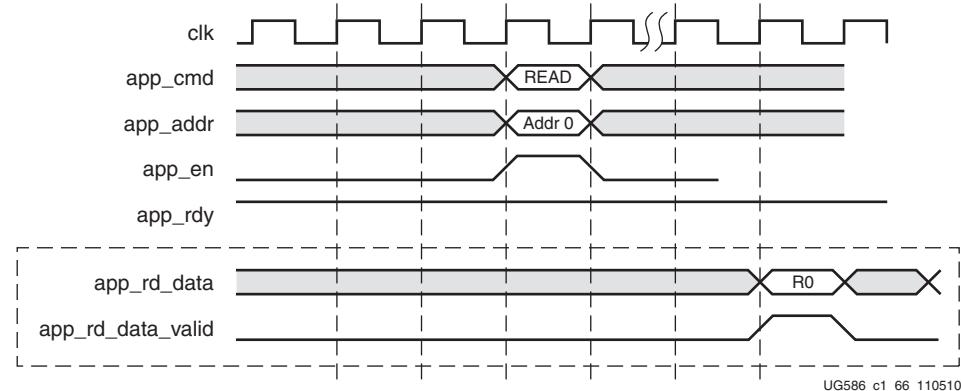
Figure 1-59 shows the corresponding data at the DRAM interface.



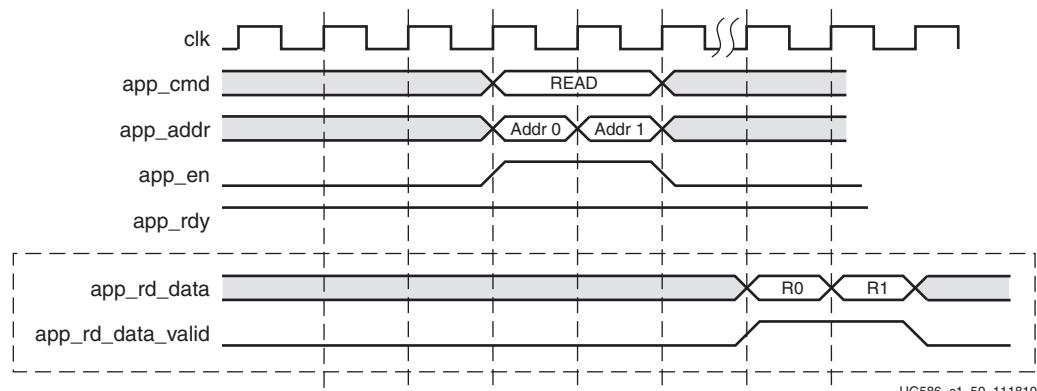
*Figure 1-59: Data at the DRAM Interface for 2:1 Mode*

## 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 1-60 and Figure 1-61). The app\_rd\_data\_end signal indicates the end of each read command burst and is not needed in user logic.



**Figure 1-60:** 4:1 Mode UI Interface Read Timing Diagram  
(Memory Burst Type = BL8)



**Figure 1-61:** 4:1 Mode UI Interface Read Timing Diagram  
(Memory Burst Type = BL4 or BL8)

In Figure 1-61, the read data returned is always in the same order as the requests made on the address/control bus.

## User Refresh

For user-controlled refresh, the memory controller managed maintenance should be disabled by setting the USER\_REFRESH parameter to "ON."

To request a REF command, app\_ref\_req is strobed for one cycle. When the memory controller sends the command to the PHY, it strobes app\_ref\_ack for one cycle, after which another request can be sent. Figure 1-62 illustrates the interface.

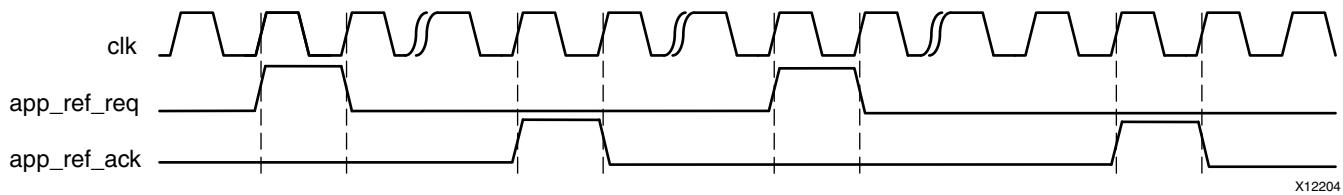


Figure 1-62: User-Refresh Interface

A user-refresh operation can be performed any time provided the handshake defined above is followed. There are no additional interfacing requirements with respect to other commands. However, pending requests affect when the operation goes out. The memory controller fulfills all pending data requests before issuing the refresh command. Timing parameters must be considered for each pending request when determining when to strobe app\_ref\_req to avoid a tREFI violation. To account for the worst case, subtract tRCD, CL, the data transit time, and tRP for each bank machine to ensure that all transactions can complete before tREFI expires. [Equation 1-1](#) shows the REF request interval maximum.

$$(tREFI - (tRCD + ((CL + 4) \times tCK) + tRP) \times nBANK\_MACHS) \quad \text{Equation 1-1}$$

A user REF should be issued immediately following calibration to establish a time baseline for determining when to send subsequent requests.

### User ZQ

For user-controlled ZQ calibration, the memory controller managed maintenance should be disabled by setting the tZQI parameter to 0.

To request a ZQ command, app\_zq\_req is strobed for one cycle. When the memory controller sends the command to the PHY, it strobes app\_zq\_ack for one cycle, after which another request can be sent. [Figure 1-63](#) illustrates the interface.

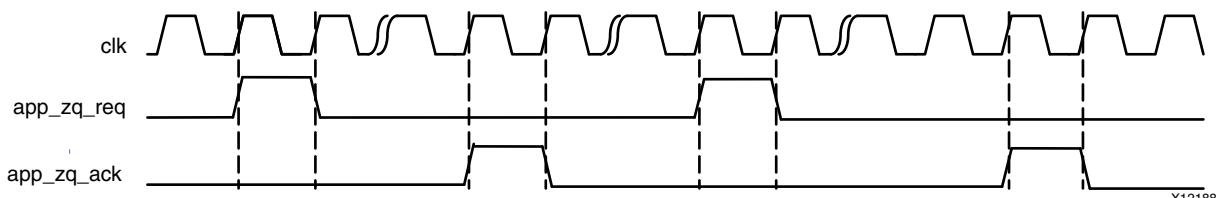


Figure 1-63: User ZQ Interface

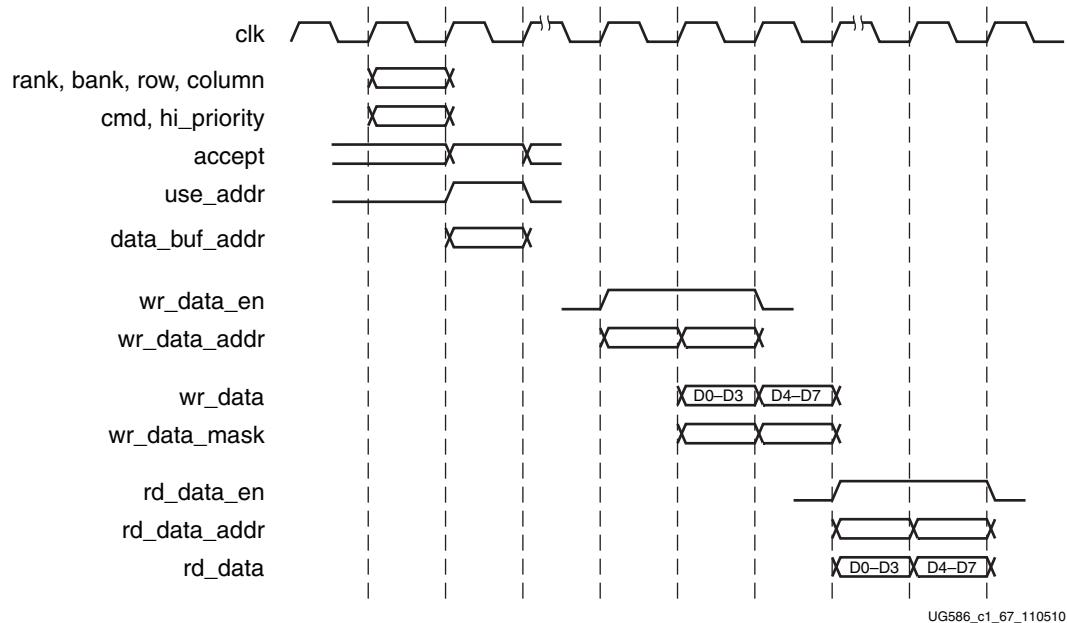
A user ZQ operation can be performed any time provided the handshake defined above is followed. There are no additional interfacing requirements with respect to other commands. However, pending requests affect when the operation goes out. The memory controller fulfills all pending data requests before issuing the ZQ command. Timing parameters must be considered for each pending request when determining when to strobe app\_zq\_req to achieve the desired interval if precision timing is desired. To account for the worst case, subtract tRCD, CL, the data transit time and tRP for each bank machine to ensure that all transactions can complete before the target tZQI expires. [Equation 1-2](#) shows the ZQ request interval maximum.

$$(tZQI - (tRCD + ((CL + 4) \times tCK) + tRP) \times nBANK\_MACHS) \quad \text{Equation 1-2}$$

A user ZQ should be issued immediately following calibration to establish a time baseline for determining when to send subsequent requests.

## Native Interface

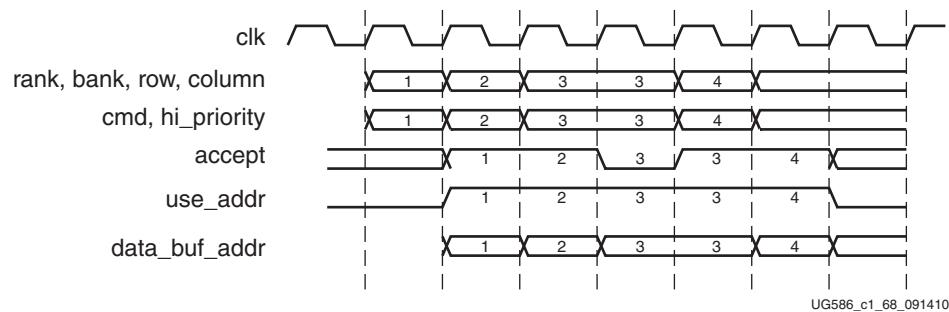
The native interface protocol is shown in [Figure 1-64](#).



**Figure 1-64: 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.

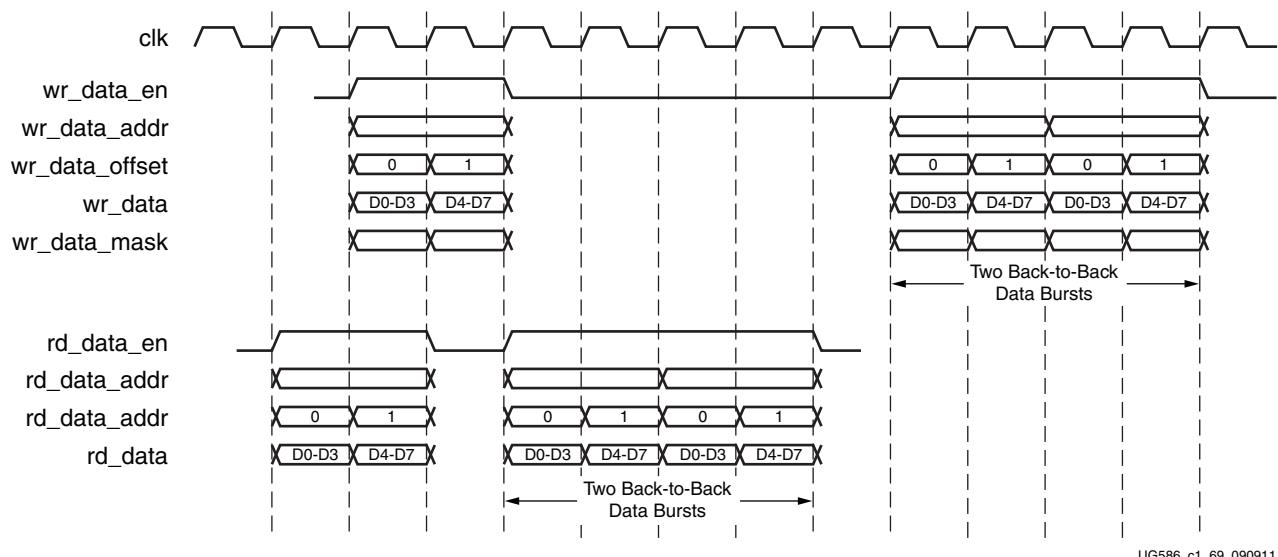
The address and command are presented to the native interface one state before they are validated with the use\_addr signal. The memory interface indicates that it can accept the request by asserting the accept signal. Requests are confirmed as accepted when use\_addr and accept are both asserted in 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 1-65](#).



**Figure 1-65: Native Interface Flow Control**

In [Figure 1-65](#), 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 1-66](#). Write data must be supplied in the same clock cycle that wr\_data\_en is asserted.



**Figure 1-66: 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 1-66](#).

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.

## User Refresh

See [User Refresh](#) for the UI. The feature is identical in the native interface.

## User ZQ

See [User ZQ](#) for the UI. The feature is identical in the native interface.

## Physical Layer Interface (Non-Memory Controller Design)

The MIG Physical Layer, or PHY, can be used without the memory controller. The PHY files are located in the `user_design/rtl/phy` directory generated by the MIG tool. Also needed are the infrastructure files located in `user_design/rtl/clocking`. The MIG memory controller can be used as an example of how to interface to the PHY. The `user_design/rtl/ip_top/mem_intf.v` file shows a sample instantiation of the memory controller and the PHY.

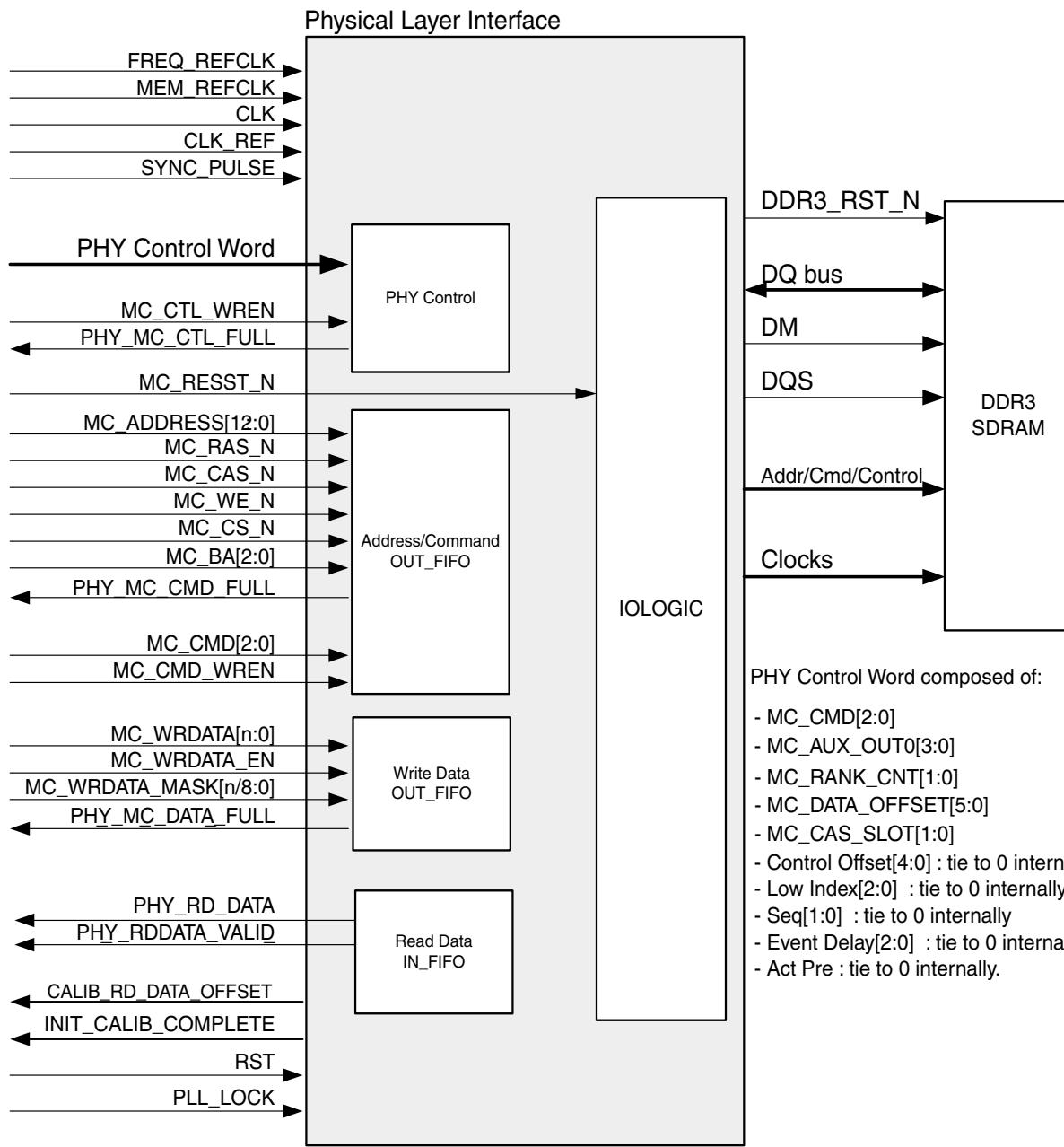
The PHY provides a physical interface to an external DDR2 or DDR3 SDRAM. The PHY generates signal timing and sequencing required to interface to the memory device. It contains clock-, address-, and control-generation logic, write and read datapaths, and state logic for initializing the SDRAM after power-up. 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. At the end of calibration the PHY asserts the `init_calib_complete` signal output to the memory controller. The assertion of this signal indicates that the memory controller can begin normal memory transactions.

A detailed description of the PHY architecture and the various stages of calibration are provided in [PHY, page 79](#). The signals required for the memory controller to interface to the PHY are listed in [Table 1-31](#).

For clocking requirements see [Clocking Architecture, page 69](#). The user can choose to use the infrastructure, `iodelay_ctrl`, and `clk_ibuf` modules provided in the `clocking` directory output by the MIG tool or instantiate the primitives in these modules in their system design.

The PHY Control FIFO, command OUT\_FIFOs, and write data OUT\_FIFOs are all in asynchronous operation mode. The read clock and the write clock to these FIFOs differ in frequency and phase. Therefore all three OUT\_FIFO FULL flags (`phy_mc_ctl_full`, `phy_mc_cmd_full`, and `phy_mc_data_full`) described in [Table 1-31, page 100](#) must be monitored by the controller to prevent overflow of the PHY Control FIFO and the OUT\_FIFOs, leading to loss of command and data.

Memory commands and data can be sent directly through the PHY interface. Different command types are sent through different slots. The CAS Write Latency (CWL) command dictates the slot number to use for write/read commands. For an odd CWL value, CAS slot numbers 1 or 3 can be used; for an even CWL value, CAS slot numbers 0 or 2 can be used for the write/read commands. In [Figure 1-67](#), the Control Offset, Low Index, Event Delay, Seq, and Act Pre fields of PHY Control words are tied Low internally inside the `phy_top` module and are not used.



X12189

Figure 1-67: **PHY Interface Example**

The data offset field (MC\_DATA\_OFFSET) in the PHY control word for read commands is determined during PHASER\_IN\_DQSFOUND calibration. It is provided by the PHY through the PHY interface. The memory controller must add the slot number being used to this read data offset value provided by the PHY. PHY control inside the PHY needs to know when to read data from IN\_FIFO after a READ command has been issued to memory.

$$\text{Read data offset} = \text{Calibrated PHY read data offset} + \text{slot number}$$

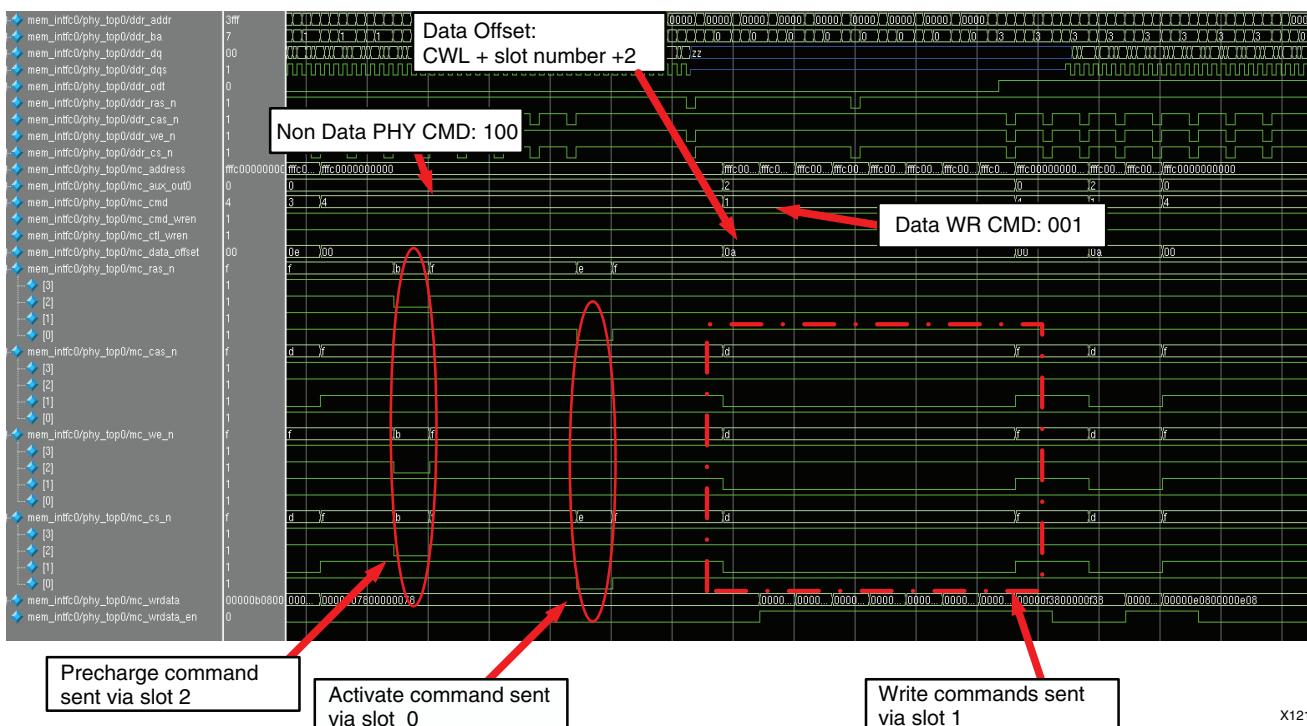
The data offset field in the PHY control word for write commands must be set based on the slot number being used, CWL, and the nCK\_PER\_CLK parameter value as shown in the following equations:

- For nCK\_PER\_CLK = 4  
Write data offset = CWL + 2 + slot number
- For nCK\_PER\_CLK = 2  
Write data offset = CWL - 2 + slot number

The write waveform shown in [Figure 1-68](#) illustrates an example with DDR3 SDRAM CWL = 7 and nCK\_PER\_CLK = 4. The selected slot number can be 1 or 3.

$$\text{Write data offset} = \text{CWL} + \text{slot number} + 2$$

$$= 7 + 1 + 2 = 10$$



[Figure 1-68: Sending Write Commands in the PHY Interface](#)

**Note:** Bits 3 through 7, bits 12 through 14, bits 23 through 24, and bits 27 through 31 in [Table 1-29](#), [page 84](#) are not used in this example.

The write waveform shown in Figure 1-69 illustrates an example with calibrated PHY read data offset = 10. For a selected slot number of 1, nCK\_PER\_CLK of 4, the read data offset is:

$$\text{Read data offset} = \text{Calibrated PHY read data offset} + \text{slot number}$$

$$= 10 + 1 = 11$$

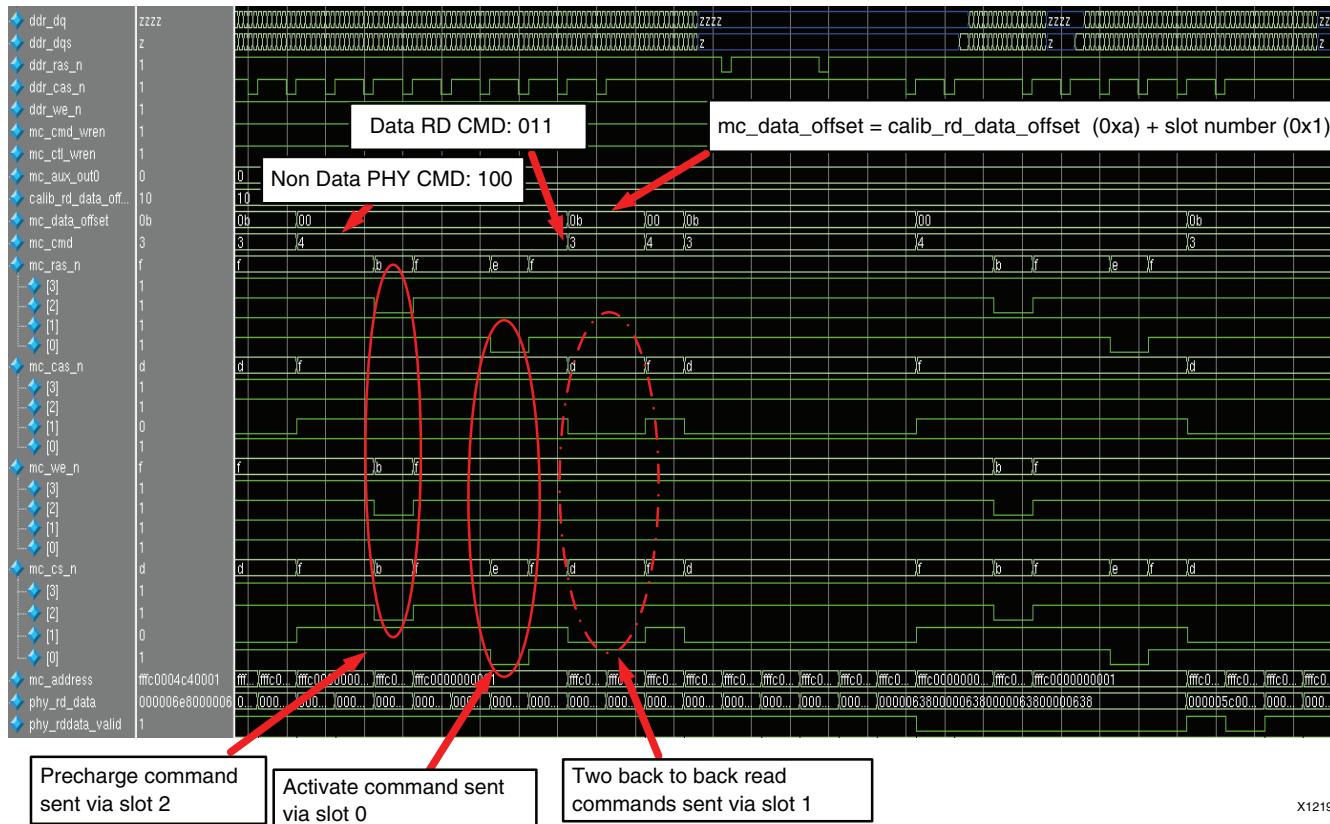


Figure 1-69: Sending Read Commands in the PHY Interface

**Note:** Bits 3 through 7, bits 12 through 14, bits 23 through 24, and bits 27 through 31 in Table 1-29, page 84 are not used in this example.

## Customizing the Core

The 7 series FPGAs memory interface solution supports several configurations for DDR2 or 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 1-35, Table 1-36, and Table 1-37.

Table 1-35: 7 Series FPGA Memory Solution Configuration Parameters

Parameter	Description	Options
REFCLK_FREQ <sup>(1)</sup>	This is the reference clock frequency for IODELAYCTRLs. This can be set to 200.0 for any speed grade device. For more information, see the IODELAYE1 Attribute Summary table in the 7 Series FPGAs SelectIO Resources User Guide [Ref 1].	200.0
SIM_BYPASS_INIT_CAL <sup>(2)</sup>	This is the calibration procedure for simulation. "OFF" is not supported in simulation. "OFF" must be used for hardware implementations. "FAST" enables a fast version of read and write leveling. "SIM_FULL" enables full calibration but skips the power up initialization delay. "SIM_INIT_CAL_FULL" enables full calibration including the power-up delays.	"OFF" "SIM_INIT_CAL_FULL" "FAST" "SIM_FULL"
nCK_PER_CLK	This is the number of memory clocks per clock.	4
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 $\text{ceil}(\log_2(\text{DQS\_WIDTH}))$ .	
ADDR_WIDTH	This is the memory address bus width. It is equal to RANK_WIDTH + BANK_WIDTH + ROW_WIDTH + COL_WIDTH.	
BANK_WIDTH	This is the number of memory bank address bits.	This option is based on the selected memory device.
CS_WIDTH	This is the number of unique CS outputs to memory.	This option is based on the selected MIG tool configuration.
CK_WIDTH	This is the number of CK/CK# outputs to memory.	This option is based on the selected MIG tool configuration.
CKE_WIDTH	This is the number of CKE outputs to memory.	This option is based on the selected MIG tool configuration.
ODT_WIDTH	This is the number of ODT outputs to memory.	This option is based on the selected MIG tool configuration.
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 $\text{ceil}(\log_2(\text{RANKS}))$ .	This option is based on the selected memory device.
ROW_WIDTH	This is the DRAM component address bus width.	This option is based on the selected memory device.
DM_WIDTH	This is the number of data mask bits.	DQ_WIDTH/8

Table 1-35: 7 Series FPGA Memory Solution Configuration Parameters (*Cont'd*)

Parameter	Description	Options
DQ_WIDTH	This is the memory DQ bus width.	This parameter supports DQ widths from 8 to a maximum of 72 in increments of 8. The available maximum DQ width is frequency dependent on the selected memory device.
DQS_WIDTH	This is the memory DQS bus width.	DQ_WIDTH/8
BURST_MODE	This is the memory data burst length.	DDR3: "8" DDR2: "8"
BM_CNT_WIDTH	This is the number of bits required to index a bank machine and is given by $\text{ceil}(\log_2(\text{nBANK\_MACHS}))$ .	
ADDR_CMD_MODE	This parameter is used by the controller to calculate timing on the memory addr/cmd bus.	"1T"
ORDERING <sup>(3)</sup>	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"
RTT_NOM	This is the nominal ODT value.	DDR3_SDRAM: "120": RZQ/2 "60": RZQ/4 "40": RZ/6 DDR2_SDRAM: "150": 150 Ω "75": 75 Ω "50": 50 Ω
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 1-35: 7 Series FPGA Memory Solution Configuration Parameters (Cont'd)

Parameter	Description	Options
OUTPUT_DRV	This is the DRAM reduced output drive option.	"HIGH" "LOW"
REG_CTRL	This is the option for DIMM or unbuffered DIMM selection.	"ON": Registered DIMM "OFF": Components, SODIMMs, UDIMMs.
IODELAY_GRP	This is an ASCII character string to define an IDELAY group used in a memory design. This is used by the ISE tools to group all instantiated IDELAYs into the same bank. Unique names must be assigned when multiple IP cores are implemented on the same FPGA.	Default: "IODELAY_MIG"
ECC_TEST	This option, when set to "ON," allows the entire DRAM bus width to be accessible through 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"
TCQ	This is the clock-to-Q delay for simulation purposes.	(The value is in picoseconds.)
tCK	This is the memory tCK clock period (ps).	The value, in picoseconds, is based on the selected frequency in the MIG tool.
DIFF_TERM_SYSCLK	"TRUE", "FALSE"	Differential termination for system clock input pins.
DIFF_TERM_REFCLK	"TRUE", "FALSE"	Differential termination for IDELAY reference clock input pins.

**Notes:**

1. The lower limit (maximum frequency) is pending characterization.
2. Core initialization during simulation can be greatly reduced by using SIM\_BYPASS\_INIT\_CAL. Three simulation modes are supported. Setting SIM\_BYPASS\_INIT\_CAL to FAST causes write leveling and read calibration to occur on only one bit per memory device. This is then used across the remaining data bits. When SIM\_BYPASS\_INIT\_CAL is set to SKIP, no read calibration occurs, and the incoming clocks and data are assumed to be aligned. Setting SIM\_BYPASS\_INIT\_CAL to SIM\_INIT\_CAL\_FULL causes complete memory initialization and calibration sequence occurs on all byte groups. SIM\_BYPASS\_INIT\_CAL should be set to SIM\_INIT\_CAL\_FULL for simulations only. SIM\_BYPASS\_INIT\_CAL should be set to OFF for implementation, or the core does not function properly.
3. 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 1-36 depend on the selected memory clock frequency, memory device, memory configuration, and FPGA speed grade. The values for these parameters are embedded in the memc\_ui\_top IP core and should not be modified in the top level. Xilinx strongly recommends that the MIG tool be rerun for different configurations.

Table 1-36: Embedded 7 Series FPGAs Memory Solution Configuration Parameters

Parameter	Description	Options
tFAW	This is the minimum interval of four active commands.	This value, in picoseconds, is based on the device selection in the MIG tool.
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	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	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	This is the average periodic refresh interval for memory.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRFC	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	This is the PRECHARGE command period.	This value, in picoseconds, is based on the device selection in the MIG tool.
tRTP	This is the READ-to-PRECHARGE command delay.	This value, in picoseconds, is based on the device selection in the MIG tool.
tWTR	This is the WRITE-to-READ command delay.	This value, in picoseconds, is based on the device selection in the MIG tool.
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. Set to 0, if the user manages this function.
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	This is the read CAS latency. The available option is frequency dependent in the MIG tool.	DDR3: 5, 6, 7, 8, 9 DDR2: 3, 4, 5, 6
CWL	This is the write CAS latency. The available option is frequency dependent in the MIG tool.	DDR3: 5, 6, 7, 8
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	This option enables or disables the IDELAY high-performance mode.	"ON" "OFF"

Table 1-36: Embedded 7 Series FPGAs Memory Solution Configuration Parameters (Cont'd)

Parameter	Description	Options
SYSCLK_TYPE	DIFFERENTIAL SINGLE_ENDED	This parameter indicates whether the system uses single-ended or differential system 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_i must be used.
REFCLK_TYPE	DIFFERENTIAL SINGLE_ENDED	This parameter indicates whether the system uses single-ended or differential reference clocks. Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, ref_clk_p/ref_clk_n must be used. For single-ended clocks, ref_clk_i must be used.
USE_DM_PORT	This is the enable data mask option used during memory write operations.	1: Enable 0: Disable
CK_WIDTH	This is the number of CK/CK# outputs to memory.	
DQ_CNT_WIDTH	This is ceil(log2(DQ_WIDTH)).	
DRAM_TYPE	This is the supported memory standard for the memory controller.	"DDR3", "DDR2"
DRAM_WIDTH	This is the DQ bus width per DRAM component.	
AL	This is the additive latency.	0
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. This parameter is set to 5 for 4:1 mode and 4 for 2:1 mode.	5, 4
SLOT_0_CONFIG	This is the rank mapping.	Single-rank setting: 8'b0000_0001 Dual-rank setting: 8'b0000_0011
ECC	This is the error correction code, available in 72-bit data width configurations. ECC is not currently available.	72
RANKS	This is the number of ranks.	

**Table 1-36: Embedded 7 Series FPGAs Memory Solution Configuration Parameters (Cont'd)**

Parameter	Description	Options
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.	
USER_REFRESH	This parameter indicates if the user manages refresh commands. Can be set for either the User or Native interface.	"ON", "OFF"

[Table 1-37](#) contains parameters set up by the MIG tool based on the pinout selected. When making pinout changes, it is recommended to rerun the MIG tool to set up the parameters properly. See [Bank and Pin Selection Guides for DDR3 Designs, page 126](#) and [Bank and Pin Selection Guides for DDR2 Designs, page 133](#).

Mistakes to the pinout parameters can result in non-functional simulation, an unrouteable design, and/or trouble meeting timing. These parameters are used to set up the PHY and route all the necessary signals to and from it. The following parameters are calculated based on selected Data and Address/Control byte groups. These parameters do not consider the system signals selection (that is, system clock, reference clock and status signals).

**Table 1-37: DDR2/DDR3 SDRAM Memory Interface Solution Pinout Parameters**

Parameter	Description	Example
BYTE_LANES_B0, BYTE_LANES_B1, BYTE_LANES_B2	Defines the byte lanes being used in a given I/O bank. A "1" in a bit position indicates a byte lane is used, and a "0" indicates unused.	Ordering of bits from MSB to LSB is T0, T1, T2, and T3 byte groups. 4'b1101: For a given bank, three byte lanes are used and one byte lane is not used.
DATA_CTL_B0, DATA_CTL_B1, DATA_CTL_B2	Defines mode of use of byte lanes in a given I/O bank. A "1" in a bit position indicates a byte lane is used for data, and a "0" indicates it is used for address/control.	4'b1100: With respect to the BYTE_LANE example, two byte lanes are used for Data and one for Address/Control.

Table 1-37: DDR2/DDR3 SDRAM Memory Interface Solution Pinout Parameters (Cont'd)

Parameter	Description	Example
PHY_0_BITLANES, PHY_1_BITLANES, PHY_2_BITLANES	12-bit parameter per byte lane used to determine which I/O locations are used to generate the necessary PHY structures. This parameter is provided as per bank. Except CKE, ODT, and RESET pins, all Data and Address/Control pins are considered for this parameter generation.	This parameter denotes for all byte groups of a selected bank. All 12 bits are denoted for a byte lane. For example, this parameter is 48'hFFE_FFF_000_DF6 for one bank. 12'hDF6 (12'b1101_1111_0110): bit lines 0, 3, and 9 are not used, the rest of the bits are used.
CK_BYTEmap	Bank and byte lane location information for the CK/CK#. An 8-bit parameter is provided per pair of signals. <ul style="list-style-type: none"> <li>[7:4] - Bank position. Values of 0, 1, or 2 are supported</li> <li>[3:0] - Byte lane position within a bank. Values of 0, 1, 2, and 3 are supported.</li> </ul>	Upper-most Data or Address/Control byte group selected bank is referred to as Bank 0 in parameters notation. Numbering of banks is 0, 1, and 2 from top to bottom. Byte groups T0, T1, T2, and T3 are numbered in parameters as 3, 2, 1 and 0, respectively. 144'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_03: This parameter is denoted for 18 clock pairs with 8 bits for each clock pin. In this case, only one clock pair is used. Ordering of parameters is from MSB to LSB (that is, CK[0]/ CK#[0] corresponds to LSB 8 bits of the parameter). 8'h13: CK/CK# placed in bank 1, byte lane 3. 8'h20: CK/CK# placed in bank 2, byte lane 0.
ADDR_map	Bank and byte lane position information for the address. 12-bit parameter provided per pin. <ul style="list-style-type: none"> <li>[11:8] - Bank position. Values of 0, 1, or 2 are supported</li> <li>[7:4] - Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> <li>[3:0] - Bit position within a byte lane. Values of [0, 1, 2, ..., A, B] are supported.</li> </ul>	Upper-most Data or Address/Control byte group selected bank is referred to as Bank 0 in parameters notation. Numbering of banks is 0, 1, and 2 from top to bottom. Byte groups T0, T1, T2, and T3 are numbered in parameters as 3, 2, 1 and 0, respectively. Bottom-most pin in a byte group is referred as "0" in MAP parameters. Numbering is counted from 0 to 9 from bottom-most pin to top pin in a byte group by excluding DQSCC I/Os. DQSCC_N and DQSCC_P pins of the byte group are numbered as A and B, respectively. 192'h000_000_039_038_037_036_035_034_033_032_031_029_028_027_026_02B: This parameter is denoted for Address width of 16 with 12 bits for each pin. In this case the Address width is 14 bits. Ordering of parameters is from MSB to LSB (that is, ADDR[0] corresponds to the 12 LSBs of the parameter). 12'h02B: Address pin placed in bank 0, byte lane 2, at location B. 12'h235: Address pin placed in bank 2, byte lane 3, at location 5.
BANK_map	Bank and byte lane position information for the bank address. See the <a href="#">ADDR_map</a> description.	See <a href="#">ADDR_map</a> example.
CAS_map	Bank and byte lane position information for the CAS command. See the <a href="#">ADDR_map</a> description.	See the <a href="#">ADDR_map</a> example

Table 1-37: DDR2/DDR3 SDRAM Memory Interface Solution Pinout Parameters (Cont'd)

Parameter	Description	Example
CKE_ODT_BYTE_MAP	Bank and byte lane position information for the CKE/ODT phaser constraint. This parameter is referred to as one of the Address/Control byte groups. See <a href="#">CK_BYTE_MAP</a> description.	See <a href="#">CK_BYTE_MAP</a> example
CS_MAP	Bank and byte lane position information for the chip select. See the <a href="#">ADDR_MAP</a> description.	See the <a href="#">ADDR_MAP</a> example
PARITY_MAP	Bank and byte lane position information for the parity bit. Parity bit exists for RDIMMs only. See the <a href="#">ADDR_MAP</a> description.	See the <a href="#">ADDR_MAP</a> example
RAS_MAP	Bank and byte lane position information for the RAS command. See the <a href="#">ADDR_MAP</a> description.	See the <a href="#">ADDR_MAP</a> example
WE_MAP	Bank and byte lane position information for the WE command. See the <a href="#">ADDR_MAP</a> description.	See the <a href="#">ADDR_MAP</a> example
DQS_BYTE_MAP	Bank and byte lane position information for the strobe. See the <a href="#">CK_BYTE_MAP</a> description.	See <a href="#">CK_BYTE_MAP</a> example
DATA0_MAP, DATA1_MAP, DATA2_MAP, DATA3_MAP, DATA4_MAP, DATA5_MAP, DATA6_MAP, DATA7_MAP, DATA8_MAP	Bank and byte lane position information for the data bus. See the <a href="#">ADDR_MAP</a> description.	See the <a href="#">ADDR_MAP</a> example
MASK0_MAP, MASK1_MAP	Bank and byte lane position information for the data mask. See the <a href="#">ADDR_MAP</a> description.	See the <a href="#">ADDR_MAP</a> example

# Design Guidelines

Guidelines for DDR2 and DDR3 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 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

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the DDR3 SDRAM physical layer. Xilinx 7 series FPGAs have dedicated logic for each DQS byte group. Four DQS byte groups are available in each 50-pin bank. Each byte group consists of a clock-capable I/O pair for the DQS and ten associated I/Os. In a typical DDR3 configuration, eight of these ten I/Os are used for the DQs, one is used for the data mask (DM), and one is left over for other signals in the memory interface. Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, DDR3 memory interfaces must be arranged in the banks vertically and not horizontally. In addition, the maximum height is three banks.

The MIG tool, when available, should be used to generate a pinout for a 7 series DDR3 interface. The MIG tool follows these rules:

- DQS signals for a byte group must be connected to a designated DQS CC pair in the bank.
- DQ signals must be connected to the byte group pins associated with the corresponding DQS.
- Control (RAS\_N, CAS\_N, WE\_N, CS\_N, CKE, ODT) and address lines must be connected to byte groups not used for the data byte groups.
- The VRN/VRP pins can be used for an address/control pin, if the following conditions are met:

- DCI cascade is used or the bank does not need the VRN/VRP pins, as in the case of only outputs.
- The adjacent byte group (T0/T3) is used as an address/control byte group.
- A used pin exists in the adjacent byte group (T0/T3) or the CK output is contained in the adjacent byte group.
- All address/control byte groups must be in the same I/O bank. Address/control byte groups cannot be split between banks.
- The address/control byte groups must be in the middle I/O bank of interfaces that span three I/O banks.
- CK must be connected to a p-n pair in one of the control byte groups. Any p-n pair in the group is acceptable, including SRCC, MRCC, and DQSCC pins.
- RESET\_N can be connected to any available pin in the interface banks, including the VRN/VRP pins if DCI cascade is used.
- VRN and VRP are used for the digitally controlled impedance (DCI) reference for banks that support DCI. DCI cascade is permitted.
- The interface must be arranged vertically.
- No more than three banks can be used for a single interface.
- All address/control byte groups must be in the same bank.
- The system clock input must be in the same column as the memory interface. The system clock input is recommended to be in the address/control bank, when possible.
- Devices implemented with SSI technology have SLRs. Memory interfaces cannot span across SLRs. Ensure that this rule is followed for the part chosen and for any other pin-compatible parts that can also be used.

### Bank Sharing Among Controllers

No unused part of a bank used in a memory interface is permitted to be shared with another memory interface. The dedicated logic that controls all the FIFOs and phasers in a bank is designed to only operate with a single memory interface and cannot be shared with other memory interfaces.

### Pin Swapping

- Pins can be freely swapped within each byte group (data and address/control), except for the DQS pair which must be on a clock-capable DQS pair and the CK which must be on a p-n pair.
- Byte groups (data and address/control) can be freely swapped with each other.
- Pins in the address/control byte groups can be freely swapped within and between their byte groups.
- No other pin swapping is permitted.

### Internal Vref

Internal Vref can only be used for data rates of 800 Mb/s or below.

### System Clock, PLL Location, and Constraints

The PLL is required to be in the bank that supplies the clock to the memory to meet the specified interface performance. The system clock input is also strongly recommended to be in this bank. The MIG tool follows these two rules whenever possible. The exception is

a 16-bit interface in a single bank where there might not be pins available for the clock input. In this case, the clock input needs to come from an adjacent bank through the frequency backbone to the PLL. The system clock input to the PLL must come from clock capable I/O.

The system clock input can only be used for an interface in the same column. The system clock input cannot be driven from another column. The additional PLL or MMCM and clock routing required for this induces too much additional jitter.

Unused outputs from the PLL can be used as clock outputs. Only the settings for these outputs can be changed. Settings related to the overall PLL behavior and the used outputs must not be disturbed.

A PLL cannot be shared among interfaces.

See [Clocking Architecture, page 69](#) for information on allowed PLL parameters.

## 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 IDELAY 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 LVCMOS15, SSTL15, SSTL15\_T\_DCI, DIFF\_SSTL15, 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.
- init\_calib\_complete: This signal indicates when the internal calibration is done and that the interface is ready for use.
- tg\_compare\_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.

16-bit wide interfaces might need to have the system clock in a bank above or below the bank with the address/control and data. In this case, MIG puts an additional constraint in the UCF. An example is shown below:

```
NET "sys_clk_p" CLOCK_DEDICATED_ROUTE = BACKBONE;
PIN "*"/u_ddr3_infrastructure/plle2_i.CLKIN1" CLOCK_DEDICATED_ROUTE =
BACKBONE;
```

This should only be used in MIG generated memory interface designs. This results in a warning listed below during PAR. This warning can be ignored.

WARNING: Place:1402 - A clock IOB / PLL clock component pair have been found that are not placed at an optimal clock IOB / PLL site pair. The

clock IOB component <sys\_clk\_p> is placed at site <IOB\_X1Y76>. The corresponding PLL component <u\_backb16/u\_ddr3\_infrastructure/plle2\_i> is placed at site <PLLE2\_ADV\_X1Y2>. The clock IO can use the fast path between the IOB and the PLL if the IOB is placed on a Clock Capable IOB site that has dedicated fast path to PLL sites within the same clock region. You may want to analyze why this problem exists and correct it. This is normally an ERROR but the CLOCK\_DEDICATED\_ROUTE constraint was applied on COMP.PIN <sys\_clk\_p.PAD> allowing your design to continue. This constraint disables all clock placer rules related to the specified COMP.PIN. The use of this override is highly discouraged as it may lead to very poor timing results. It is recommended that this error condition be corrected in the design.

Do not drive user clocks via the I/O clocking backbone from the region(s) containing the MIG generated memory interface to CMT blocks in adjacent regions due to resource limitations. Consult the *7 Series FPGAs Clocking Resources User Guide* for more information.

The MIG tool sets the Vccaux\_io constraint based on the data rate and voltage input selected. The generated UCF has additional constraints as needed. For example:

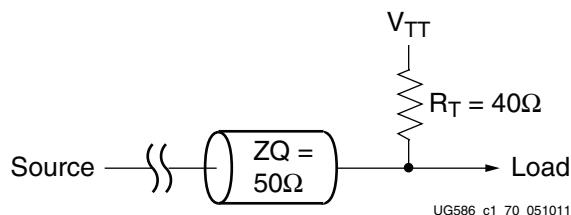
```
NET "ddr3_dq[0]" LOC = "E16" | IOSTANDARD = SSTL15_T_DCI | VCCAUX_IO = HIGH ; # Bank: 15 - Byte: T2
NET "ddr3_dq[1]" LOC = "D17" | IOSTANDARD = SSTL15_T_DCI
| VCCAUX_IO = HIGH ; # Bank: 15 - Byte: T2
```

Consult the Constraints Guide for more information.

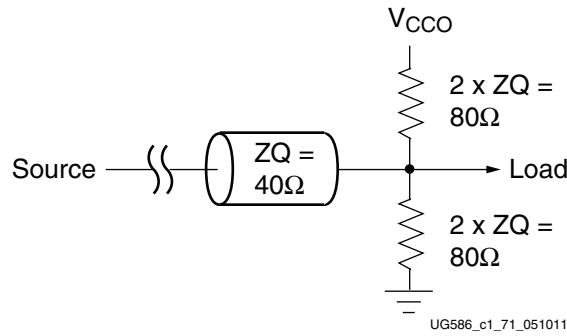
## Termination

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.
- $40\Omega$  traces and termination are required for operation at 1333 Mb/s and higher.  $50\Omega$  is acceptable below 1333 Mb/s. [Figure 1-70](#) and [Figure 1-71](#) are for 1333 Mb/s and higher.
- Unidirectional signals are to be terminated with the memory device's internal termination or a pull-up of  $40\Omega$  to  $V_{TT}$  at the load ([Figure 1-70](#)). A split  $80\Omega$  termination to  $V_{CCO}$  and a  $80\Omega$  termination to GND can be used ([Figure 1-71](#)), but takes more power. For bidirectional signals, the termination is needed at both ends of the signal. ODT should be used on the memory side. For best performance in HP banks, DCI should be used. For best performance in HR banks, IN\_TERM (internal termination) should be used. As noted above,  $50\Omega$  termination is acceptable below 1333 Mb/s.



*Figure 1-70:  $40\Omega$  Termination to  $V_{TT}$*

Figure 1-71: 80Ω Split Termination to V<sub>CCO</sub> and GND

- Differential signals should be terminated with the memory device's internal termination or an 80Ω differential termination at the load (Figure 1-72). For bidirectional signals, termination is needed at both ends of the signal. ODT should be used on the memory side. For best performance in HP banks, DCI should be used. For best performance in HR banks, IN\_TERM (internal termination) should be used. 100Ω termination is acceptable below 1333 Mb/s.

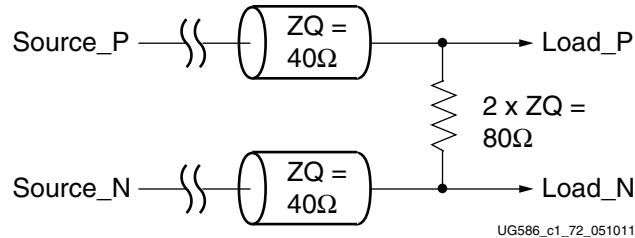


Figure 1-72: 80Ω Differential Termination

- 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 (HR banks) or IN\_TERM (HR banks) is required at the FPGA to meet the specified performance.
- The RESET signal is not terminated. This signal should be pulled down during memory initialization with a 4.7 kΩ resistor connected to GND.
- ODT, which terminates a signal at the memory, is required. The MIG tool should be used to specify the configuration of the memory system for setting the mode register properly. See Micron technical note TN-47-01 [Ref 7] 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 (data mask not used or data mask disabled scenarios). The default settings for the ODT for the supported configurations are listed in Table 1-38 and Table 1-39.

**Table 1-38: Default Settings for ODT Supported Configurations - Writes**

		Write To	FPGA DCI or IN_TERM	Slot 1		Slot 2	
Slot 1	Slot 2			Rank 1	Rank 2	Rank 1	Rank 2
SR	SR	Slot 1	off	120Ω <sup>(1)</sup>	na	40Ω	na
		Slot 2	off	40Ω	na	120Ω <sup>(1)</sup>	na
DR	na (Single Slot)	Slot 1	off	120Ω	ODT off	na	na
SR	na (Single Slot)	Slot 1	off	120Ω	na	na	na

**Notes:**

1. Uses Dynamic ODT

**Table 1-39: Default Settings for ODT Supported Configurations - Reads**

		Write To	FPGA DCI or IN_TERM	Slot 1		Slot 2	
Slot 1	Slot 2			Rank 1	Rank 2	Rank 1	Rank 2
SR	SR	Slot 1	40Ω	ODT off	na	40Ω	na
		Slot 2	40Ω	40Ω	na	ODT off	na
DR	na (Single Slot)	Slot 1	40Ω	ODT off	ODT off	na	na
SR	na (Single Slot)	Slot 1	40Ω	ODT off	na	na	na

## 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) in the High-Performance banks. In the High-Range banks, the tool uses the SSTL15 and DIFF\_SSTL15 standard with the internal termination (IN\_TERM) attribute chosen in the GUI.
- The SSTL15 and DIFF\_SSTL15 standards are used for unidirectional outputs, such as control/address, and forward memory clocks.
- LVCMOS15 is used for the RESET\_N signal driven to the DDR3 memory.

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. The package delay should be included when determining the effective trace length. Note that different parts in the same package have different internal package skew values. De-rate the minimum period appropriately in the MIG Controller Options page when different parts in the same package are used.

One 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  $(L \times C)$ .

Another method is to use PARTGen. The PARTGen utility generates a PKG file that contains the package delay values for every pin of the device under consideration. For example, to obtain the package delay information for the 7 series FPGA XC7K160T-FF676, this command should be issued:

```
partgen -v xc7k160t-ff676
```

This generates a file named `xc7k160tff676.pkg` in the current directory with package trace delay information for each pin. While applying specific trace-matching guidelines for the DDR3 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 electrical delays between DDR3 SDRAM signals:

- The maximum electrical delay between any DQ and its associated DQS/DQS# should be  $\pm 5$  ps.
- The maximum electrical delay between any address and control signals and the corresponding CK/CK# should be  $\pm 25$  ps.
- The maximum electrical delay of any DQS/DQS# should be less than that of CK/CK#.

The specified DQ to DQS skew limit can be increased if the memory interface is not operated at the maximum frequency. [Table 1-40](#) indicates the relaxed skew limit (+/-) for these cases. The vertical axis is the bit rate in Mb/s. The horizontal axis is the DDR3 SDRAM component speed rating. The top portion of the chart is for skew changes relative to the 1867 limit of the -3 FPGA speed grade while the lower portion is for the 1600 limit of the -1 and -2 FPGA speed grades.

**Table 1-40: DQ to DQS Skew Limit**

	Memory Component Rating				
	1867	1600	1333	1066	800
From 1867 to:					
1600	49.6	31.4			
1333	112.1	93.9	66.4		
1066	150.0	150.0	150.0	125.2	
800	150.0	150.0	150.0	150.0	150.0
From 1600 to:					
1333		67.5	40.0		
1066		150.0	133.8	98.8	
800		150.0	150.0	150.0	150.0

For example, if the -3 FPGA operates at 1600 Mb/s with a 1600 rated DDR3 component, the DQ to DQS skew limit is  $\pm 31.4$  ps. If the interface operates at 1066 with a 1333 rated DDR3 component, the skew limit is  $\pm 150$  ps.

Similarly, the specified CK to address/control skew limit can be increased if the memory interface is not operated at the maximum frequency. [Table 1-41](#) indicates the relaxed skew limit (+/-) for these cases. The vertical axis is the bit rate in Mb/s. The horizontal axis is the DDR3 SDRAM component speed rating. The top portion of the chart is for skew changes

relative to the 1867 limit of the -3 FPGA speed grade, while the lower portion is for the 1600 limit of the -1 and -2 FPGA speed grade.

**Table 1-41: CK to Address/Control Skew Limit**

	<b>Memory Component Rating</b>				
	<b>1867</b>	<b>1600</b>	<b>1333</b>	<b>1066</b>	<b>800</b>
From 1867 to:					
1600	114.3	94.3			
1333	150.0	150.0	150.0		
1066	150.0	150.0	150.0	150.0	
800	150.0	150.0	150.0	150.0	150.0
From 1600 to:					
1333		150.0	130.0		
1066		150.0	150.0	150.0	
800		150.0	150.0	150.0	150.0

For example, if the -3 FPGA operates at 1600 Mb/s with a 1600 rated DDR3 component, the CK to address/control skew limit is  $\pm 94.3$  ps. If a -1 FPGA operates at 1066 with a 1333 rated DDR3 component, the skew limit is  $\pm 150$  ps.

Write leveling is required for both DIMMs and components. Designs using multiple components must 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. The skew between bytes in an I/O bank must be 1 ns or less.

## 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 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

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the DDR2 SDRAM physical layer. Xilinx 7 series FPGAs have dedicated logic for each DQS byte group. Four DQS byte groups are available

in each 50-pin bank. Each byte group consists of a clock-capable I/O pair for the DQS and 10 associated I/Os. In a typical DDR2 configuration, 8 of these 10 I/Os are used for the DQs: one is used for the data mask (DM), and one remains for other signals in the memory interface. Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, DDR2 memory interfaces must be arranged in the banks vertically and not horizontally. In addition, the maximum height is three banks.

The MIG tool, when available, should be used to generate a pinout for a 7 series DDR2 interface. The MIG tool follows these rules:

- DQS signals for a byte group must be connected to a designated DQS CC pair in the bank.
- DQ signals and a DM signal must be connected to the byte group pins associated with the corresponding DQS.
- Control (RAS\_N, CAS\_N, WE\_N, CS\_N, CKE, ODT) and address lines must be connected to byte groups not used for the data byte groups.
- The VRN/VRP pins can be used for an address/control pin, if the following conditions are met:
  - DCI cascade is used or the bank does not need the VRN/VRP pins, as in the case of only outputs.
  - The adjacent byte group (T0/T3) is used as an address/control byte group.
  - A used pin exists in the adjacent byte group (T0/T3) or the CK output is contained in the adjacent byte group.
- All address/control byte groups must be in the same I/O bank. Address/control byte groups cannot be split between banks.
- The address/control byte groups must be in the middle I/O bank of interfaces that span three I/O banks.
- CK must be connected to a p-n pair in one of the control byte groups. Any p-n pair in the group is acceptable, including SRCC, MRCC, and DQSCC pins. These pins are generated for each component and a maximum of four ports/pairs only are allowed due to I/O pin limitations. Only one CK pair must be connected for one byte group.
- CS\_N pins are generated for each component and a maximum of four ports/pairs only are allowed due to I/O pin limitations.
- RESET\_N can be connected to any available pin in the interface banks, including the VRN/VRP pins, if DCI cascade is used.
- For single rank components and DIMMs, only one CKE port is generated.
- For single rank components and DIMMs, the ODT port is repeated based on the number of components. The maximum number of allowed ports is 3.
- For data widths of 16 with a x8 part, only one set of CK/CK#, CS, ODT ports is generated to fit the design in a single bank.
- VRN and VRP are used for the digitally controlled impedance (DCI) reference for banks that support DCI. DCI cascade is permitted.
- The interface must be arranged vertically.
- No more than three banks can be used for a single interface. All the banks chosen must be consequent.
- The system clock input must be in the same column as the memory interface. The system clock input is recommended to be in the address/control bank, when possible

- Devices implemented with SSI technology have SLRs. Memory interfaces cannot span across SLRs. Ensure that this rule is followed for the part chosen and for any other pin-compatible parts that can also be used.

### Bank Sharing Among Controllers

No unused part of a bank used in a memory interface is permitted to be shared with another memory interface. The dedicated logic that controls all the FIFOs and phasers in a bank is designed to only operate with a single memory interface and cannot be shared with other memory interfaces.

### Pin Swapping

- Pins can be freely swapped within each byte group (data and address/control), except for the DQS pair which must be on a clock-capable DQS pair and the CK, which must be on a p-n pair.
- Byte groups (data and address/control) can be freely swapped with each other.
- Pins in the address/control byte groups can be freely swapped within and between their byte groups.
- No other pin swapping is permitted.

### Internal Vref

Internal Vref can only be used for data rates of 800 Mb/s or below.

### System Clock, PLL Location, and Constraints

The PLL is required to be in the bank that supplies the clock to the memory to meet the specified interface performance. The system clock input is also strongly recommended to be in this bank. The MIG tool follows these two rules whenever possible. The exception is a 16-bit interface in a single bank where there might not be pins available for the clock input. In this case, the clock input needs to come from an adjacent bank through the frequency backbone to the PLL. The system clock input to the PLL must come from clock capable I/O.

The system clock input can only be used for an interface in the same column. The system clock input cannot be driven from another column. The additional PLL or MMCM and clock routing required for this induces too much additional jitter.

Unused outputs from the PLL can be used as clock outputs. Only the settings for these outputs can be changed. Settings related to the overall PLL behavior and the used outputs must not be disturbed.

A PLL cannot be shared among interfaces.

See [Clocking Architecture, page 69](#) for information on allowed PLL parameters.

### 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 IDELAY 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 LVCMOS18, SSTL18\_II, SSTL18\_II\_T\_DCI, DIFF\_SSTL18\_II, or DIFF\_SSTL18\_II\_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.
- init\_calib\_complete: This signal indicates when the internal calibration is done and that the interface is ready for use.
- tg\_compare\_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.

A 16-bit wide interface might need to have the system clock in a bank above or below the bank with the address/control and data. In this case, the MIG tool puts an additional constraint in the UCF. An example is shown below:

```
NET "sys_clk_p" CLOCK_DEDICATED_ROUTE = BACKBONE;
PIN "* /u_ddr2_infrastructure/plle2_i.CLKIN1" CLOCK_DEDICATED_ROUTE =
BACKBONE;
```

This should only be used in MIG generated memory interface designs. This results in a warning listed below during PAR. This warning can be ignored.

*WARNING:Place:1402 - A clock IOB/PLL clock component pair have been found that are not placed at an optimal clock IOB/PLL site pair. The clock IOB component <sys\_clk\_p> is placed at site <IOB\_X1Y76>. The corresponding PLL component <u\_backb16/u\_ddr2\_infrastructure/plle2\_i> is placed at site <PLLE2\_ADV\_X1Y2>. The clock IO can use the fast path between the IOB and the PLL if the IOB is placed on a Clock Capable IOB site that has dedicated fast path to PLL sites within the same clock region. You may want to analyze why this problem exists and correct it. This is normally an ERROR but the CLOCK\_DEDICATED\_ROUTE constraint was applied on COMP.PIN <sys\_clk\_p.PAD> allowing your design to continue. This constraint disables all clock placer rules related to the specified COMP.PIN. The use of this override is highly discouraged as it may lead to very poor timing results. It is recommended that this error condition be corrected in the design.*

Do not drive user clocks via the I/O clocking backbone from the region(s) containing the MIG generated memory interface to CMT blocks in adjacent regions due to resource limitations. Consult the *7 Series FPGAs Clocking Resources User Guide* [Ref 18] for more information.

The MIG tool sets the Vccaux\_io constraint based on the data rate and voltage input selected. The generated UCF has additional constraints as needed. For example:

```
NET "ddr2_dq[0]" LOC = "E16" | IOSTANDARD = SSTL15_T_DCI | VCCAUX_IO =
HIGH ; # Bank: 15 - Byte: T2
NET "ddr2_dq[1]" LOC = "D17" | IOSTANDARD = SSTL15_T_DCI
| VCCAUX_IO = HIGH ; # Bank: 15 - Byte: T2
```

Consult the *Xilinx Timing Constraints User Guide* [Ref 13] for more information.

## 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 depends on various factors, such as speed requirements, termination topology, use of unbuffered DIMMs, and multiple rank DIMMs. Loading 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  $40\ \Omega$  to VTT at the load ([Figure 1-70](#)). A split  $80\ \Omega$  termination to  $V_{CCO}$  and an  $80\ \Omega$  termination to GND can be used ([Figure 1-71](#)), but takes more power. For bidirectional signals, the termination is needed at both ends of the signal. ODT should be used on the memory side. For best performance in HP banks, DCI should be used. For best performance in HR banks, IN\_TERM (internal termination) should be used.
- Differential signals should be terminated with the memory device's internal termination or a  $80\ \Omega$  differential termination at the load ([Figure 1-72](#)). For bidirectional signals, termination is needed at both ends of the signal. ODT should be used on the memory side. For best performance in HP banks, DCI should be used. For best performance in HR banks, IN\_TERM (internal termination) should be used.
- 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 (HR banks) or IN\_TERM (HR banks) is required at the FPGA to meet the specified performance.
- The RESET signal is not terminated. This signal should be pulled down during memory initialization with a  $4.7\ k\Omega$  resistor connected to GND.
- ODT, which terminates a signal at the memory, is required. The MIG tool should be used to specify the configuration of the memory system for setting the mode register properly. See Micron technical note TN-47-01 [[Ref 7](#)] 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 (data mask not used or data mask disabled scenarios).
- DM should be pulled to GND if ODT is used but DM is not driven by the FPGA (for scenarios where the data mask is not used or is disabled).

## I/O Standards

These rules apply to the I/O standard selection for DDR2 SDRAMs:

- Designs generated by the MIG tool use the SSTL15\_T\_DC1 and DIFF\_SSTL15\_T\_DC1 standards for all bidirectional I/O (DQ, DQS) in the High-Performance banks. In the High-Range banks, the tool uses the SSTL15 and DIFF\_SSTL15 standard with the internal termination (IN\_TERM) attribute chosen in the GUI.
- The SSTL15 and DIFF\_SSTL15 standards are used for unidirectional outputs, such as control/address and forward memory clocks.
- LVCMOS15 is used for the RESET\_N signal driven to the DDR2 memory. The MIG tool creates the UCF using the appropriate standard based on input from the GUI.

## Trace Lengths

The trace lengths described in this section are for high-speed operation. The package delay should be included when determining the effective trace length. Different parts in the same package have different internal package skew values. Derate the minimum period appropriately in the MIG Controller Options page when different parts in the same package are used.

One method to determine 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  $(L \times C)$ .

Another method is to use PARTGen. The PARTGen utility generates a PKG file that contains the package delay values for every pin of the device under consideration. For example, to obtain the package delay information for the 7 series FPGAs XC7K160T-FF676, this command should be issued:

```
partgen -v xc7k160tffg676
```

This generates a file named `xc7k160tffg676.pkg` in the current directory with package trace delay information for each pin. 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 to decrease the maximum possible performance for the target device.

These rules indicate the maximum electrical delays between DDR2 SDRAM signals:

- The maximum electrical delay between any DQ and its associated DQS/DQS# should be  $\pm 5$  ps.
- The maximum electrical delay between any address and control signals and the corresponding CK/CK# should be  $\pm 25$  ps.
- The maximum electrical delay of any DQS/DQS# should be less than that of CK/CK#.

## Pinout Examples

**Table 1-42** shows an example of a 16-bit DDR3 interface contained within one bank. This example is for a component interface using a 1 Gb x16 part. If x8 components are used or a higher density part is needed that would require more address pins, these options are possible:

- An additional bank can be used.
- RESET\_N can be moved to another bank as long as timing is met. External timing for this signal is not critical and a level shifter can be used.
- DCI cascade can be used to free up the VRN/VRP pins if another bank is available for the DCI master.

Internal Vref is used in this example.

**Table 1-42: 16-Bit DDR3 Interface Contained in One Bank**

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	VRP	N/A	SE	49	
1	DQ15	D_11	P	48	

Table 1-42: 16-Bit DDR3 Interface Contained in One Bank (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	DQ14	D_10	N	47	
1	DQ13	D_09	P	46	
1	DQ12	D_08	N	45	
1	DQS1_P	D_07	P	44	DQSCC-P
1	DQS1_N	D_06	N	43	DQSCC-N
1	DQ11	D_05	P	42	
1	DQ10	D_04	N	41	
1	DQ9	D_03	P	40	
1	DQ8	D_02	N	39	
1	DM1	D_01	P	38	
1		D_00	N	37	
1	DQ7	C_11	P	36	
1	DQ6	C_10	N	35	
1	DQ5	C_09	P	34	
1	DQ4	C_08	N	33	
1	DQS0_P	C_07	P	32	DQSCC-P
1	DQS0_N	C_06	N	31	DQSCC-N
1	DQ3	C_05	P	30	
1	DQ2	C_04	N	29	
1	DQ1	C_03	P	28	CCIO-P
1	DQ0	C_02	N	27	CCIO-N
1	DM0	C_01	P	26	CCIO-P
1	RESET_N	C_00	N	25	CCIO-N
1	RAS_N	B_11	P	24	CCIO-P
1	CAS_N	B_10	N	23	CCIO-N
1	WE_N	B_09	P	22	CCIO-P
1	BA2	B_08	N	21	CCIO-N
1	CK_P	B_07	P	20	DQSCC-P
1	CK_N	B_06	N	19	DQSCC-N
1	BA1	B_05	P	18	
1	BA0	B_04	N	17	
1	CS_N	B_03	P	16	

Table 1-42: 16-Bit DDR3 Interface Contained in One Bank (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	ODT	B_02	N	15	
1	CKE	B_01	P	14	
1	A12	B_00	N	13	
1	A11	A_11	P	12	
1	A10	A_10	N	11	
1	A9	A_09	P	10	
1	A8	A_08	N	9	
1	A7	A_07	P	8	DQSCC-P
1	A6	A_06	N	7	DQSCC-N
1	A5	A_05	P	6	
1	A4	A_04	N	5	
1	A3	A_03	P	4	
1	A2	A_02	N	3	
1	A1	A_01	P	2	
1	A0	A_00	N	1	
1	VRN	N/A	SE	0	

Table 1-43 shows an example of a 32-bit DDR3 interface contained within two banks. This example uses 2 Gb x8 components.

Table 1-43: 32-Bit DDR3 Interface Contained in Two Banks

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	VRP	N/A	SE	49	
1		D_11	P	48	
1		D_10	N	47	
1		D_09	P	46	
1		D_08	N	45	
1		D_07	P	44	DQSCC-P
1		D_06	N	43	DQSCC-N
1		D_05	P	42	
1		D_04	N	41	
1		D_03	P	40	
1		D_02	N	39	
1		D_01	P	38	

Table 1-43: 32-Bit DDR3 Interface Contained in Two Banks (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1		D_00	N	37	
1		C_11	P	36	
1		C_10	N	35	
1		C_09	P	34	
1		C_08	N	33	
1		C_07	P	32	DQSCC-P
1		C_06	N	31	DQSCC-N
1		C_05	P	30	
1		C_04	N	29	
1		C_03	P	28	CCIO-P
1		C_02	N	27	CCIO-N
1	CKE	C_01	P	26	CCIO-P
1	ODT	C_00	N	25	CCIO-N
1	RAS_N	B_11	P	24	CCIO-P
1	CAS_N	B_10	N	23	CCIO-N
1	WE_N	B_09	P	22	CCIO-P
1	BA2	B_08	N	21	CCIO-N
1	CK_P	B_07	P	20	DQSCC-P
1	CK_N	B_06	N	19	DQSCC-N
1	BA1	B_05	P	18	
1	BA0	B_04	N	17	
1	CS_N	B_03	P	16	
1	A14	B_02	N	15	
1	A13	B_01	P	14	
1	A12	B_00	N	13	
1	A11	A_11	P	12	
1	A10	A_10	N	11	
1	A9	A_09	P	10	
1	A8	A_08	N	9	
1	A7	A_07	P	8	DQSCC-P
1	A6	A_06	N	7	DQSCC-N
1	A5	A_05	P	6	

Table 1-43: 32-Bit DDR3 Interface Contained in Two Banks (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	A4	A_04	N	5	
1	A3	A_03	P	4	
1	A2	A_02	N	3	
1	A1	A_01	P	2	
1	A0	A_00	N	1	
1	VRN	N/A	SE	0	
2	VRP	N/A	SE	49	
2	DQ31	D_11	P	48	
2	DQ30	D_10	N	47	
2	DQ29	D_09	P	46	
2	DQ28	D_08	N	45	
2	DQS3_P	D_07	P	44	DQSCC-P
2	DQS3_N	D_06	N	43	DQSCC-N
2	DQ27	D_05	P	42	
2	DQ26	D_04	N	41	
2	DQ25	D_03	P	40	
2	DQ24	D_02	N	39	
2	DM3	D_01	P	38	
2		D_00	N	37	
2	DQ23	C_11	P	36	
2	DQ22	C_10	N	35	
2	DQ21	C_09	P	34	
2	DQ20	C_08	N	33	
2	DQS2_P	C_07	P	32	DQSCC-P
2	DQS2_N	C_06	N	31	DQSCC-N
2	DQ19	C_05	P	30	
2	DQ18	C_04	N	29	
2	DQ17	C_03	P	28	CCIO-P
2	DQ16	C_02	N	27	CCIO-N
2	DM2	C_01	P	26	CCIO-P
2		C_00	N	25	CCIO-N
2	DQ15	B_11	P	24	CCIO-P

**Table 1-43: 32-Bit DDR3 Interface Contained in Two Banks (Cont'd)**

<b>Bank</b>	<b>Signal Name</b>	<b>Byte Group</b>	<b>I/O Type</b>	<b>I/O Number</b>	<b>Special Designation</b>
2	DQ14	B_10	N	23	CCIO-N
2	DQ13	B_09	P	22	CCIO-P
2	DQ12	B_08	N	21	CCIO-N
2	DQS1_P	B_07	P	20	DQSCC-P
2	DQS1_N	B_06	N	19	DQSCC-N
2	DQ11	B_05	P	18	
2	DQ10	B_04	N	17	
2	DQ9	B_03	P	16	
2	DQ8	B_02	N	15	
2	DM1	B_01	P	14	
2		B_00	N	13	
2	DQ7	A_11	P	12	
2	DQ6	A_10	N	11	
2	DQ5	A_09	P	10	
2	DQ4	A_08	N	9	
2	DQS0_P	A_07	P	8	DQSCC-P
2	DQS0_N	A_06	N	7	DQSCC-N
2	DQ3	A_05	P	6	
2	DQ2	A_04	N	5	
2	DQ1	A_03	P	4	
2	DQ0	A_02	N	3	
2	DM0	A_01	P	2	
2	RESET_N	A_00	N	1	
2	VRN	N/A	SE	0	

**Table 1-44** shows an example of a 64-bit DDR3 interface contained within three banks. This example uses four 2 Gb x16 components.

**Table 1-44: 64-Bit DDR3 Interface in Three Banks**

<b>Bank</b>	<b>Signal Name</b>	<b>Byte Group</b>	<b>I/O Type</b>	<b>I/O Number</b>	<b>Special Designation</b>
1	VRP	N/A	SE	49	
1	DQ63	D_11	P	48	
1	DQ62	D_10	N	47	
1	DQ61	D_09	P	46	

Table 1-44: 64-Bit DDR3 Interface in Three Banks (Cont'd)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	DQ60	D_08	N	45	
1	DQS7_P	D_07	P	44	DQSCC-P
1	DQS7_N	D_06	N	43	DQSCC-N
1	DQ59	D_05	P	42	
1	DQ58	D_04	N	41	
1	DQ57	D_03	P	40	
1	DQ56	D_02	N	39	
1	DM7	D_01	P	38	
1		D_00	N	37	
1	DQ55	C_11	P	36	
1	DQ54	C_10	N	35	
1	DQ53	C_09	P	34	
1	DQ52	C_08	N	33	
1	DQS6_P	C_07	P	32	DQSCC-P
1	DQS6_N	C_06	N	31	DQSCC-N
1	DQ51	C_05	P	30	
1	DQ50	C_04	N	29	
1	DQ49	C_03	P	28	CCIO-P
1	DQ48	C_02	N	27	CCIO-N
1	DM6	C_01	P	26	CCIO-P
1		C_00	N	25	CCIO-N
1	DQ47	B_11	P	24	CCIO-P
1	DQ46	B_10	N	23	CCIO-N
1	DQ45	B_09	P	22	CCIO-P
1	DQ44	B_08	N	21	CCIO-N
1	DQS5_P	B_07	P	20	DQSCC-P
1	DQS5_N	B_06	N	19	DQSCC-N
1	DQ43	B_05	P	18	
1	DQ42	B_04	N	17	
1	DQ41	B_03	P	16	
1	DQ40	B_02	N	15	
1	DM5	B_01	P	14	

Table 1-44: 64-Bit DDR3 Interface in Three Banks (Cont'd)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1		B_00	N	13	
1	DQ39	A_11	P	12	
1	DQ38	A_10	N	11	
1	DQ37	A_09	P	10	
1	DQ36	A_08	N	9	
1	DQS4_P	A_07	P	8	DQSCC-P
1	DQS4_N	A_06	N	7	DQSCC-N
1	DQ35	A_05	P	6	
1	DQ34	A_04	N	5	
1	DQ33	A_03	P	4	
1	DQ32	A_02	N	3	
1	DM4	A_01	P	2	
1		A_00	N	1	
1	VRN	N/A	SE	0	
2	VRP	N/A	SE	49	
2		D_11	P	48	
2		D_10	N	47	
2		D_09	P	46	
2		D_08	N	45	
2		D_07	P	44	DQSCC-P
2		D_06	N	43	DQSCC-N
2		D_05	P	42	
2		D_04	N	41	
2		D_03	P	40	
2		D_02	N	39	
2		D_01	P	38	
2		D_00	N	37	
2		C_11	P	36	
2		C_10	N	35	
2		C_09	P	34	
2		C_08	N	33	
2		C_07	P	32	DQSCC-P

Table 1-44: 64-Bit DDR3 Interface in Three Banks (Cont'd)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
2		C_06	N	31	DQSCC-N
2		C_05	P	30	
2		C_04	N	29	
2		C_03	P	28	CCIO-P
2		C_02	N	27	CCIO-N
2		C_01	P	26	CCIO-P
2	ODT	C_00	N	25	CCIO-N
2	RAS_N	B_11	P	24	CCIO-P
2	CAS_N	B_10	N	23	CCIO-N
2	WE_N	B_09	P	22	CCIO-P
2	BA2	B_08	N	21	CCIO-N
2	CK_P	B_07	P	20	DQSCC-P
2	CK_N	B_06	N	19	DQSCC-N
2	BA1	B_05	P	18	
2	BA0	B_04	N	17	
2	CS_N	B_03	P	16	
2	CKE	B_02	N	15	
2	A13	B_01	P	14	
2	A12	B_00	N	13	
2	A11	A_11	P	12	
2	A10	A_10	N	11	
2	A9	A_09	P	10	
2	A8	A_08	N	9	
2	A7	A_07	P	8	DQSCC-P
2	A6	A_06	N	7	DQSCC-N
2	A5	A_05	P	6	
2	A4	A_04	N	5	
2	A3	A_03	P	4	
2	A2	A_02	N	3	
2	A1	A_01	P	2	
2	A0	A_00	N	1	
2	VRN	N/A	SE	0	

Table 1-44: 64-Bit DDR3 Interface in Three Banks (Cont'd)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
3	VRP	N/A	SE	49	
3	DQ31	D_11	P	48	
3	DQ30	D_10	N	47	
3	DQ29	D_09	P	46	
3	DQ28	D_08	N	45	
3	DQS3_P	D_07	P	44	DQSCC-P
3	DQS3_N	D_06	N	43	DQSCC-N
3	DQ27	D_05	P	42	
3	DQ26	D_04	N	41	
3	DQ25	D_03	P	40	
3	DQ24	D_02	N	39	
3	DM3	D_01	P	38	
3		D_00	N	37	
3	DQ23	C_11	P	36	
3	DQ22	C_10	N	35	
3	DQ21	C_09	P	34	
3	DQ20	C_08	N	33	
3	DQS2_P	C_07	P	32	DQSCC-P
3	DQS2_N	C_06	N	31	DQSCC-N
3	DQ19	C_05	P	30	
3	DQ18	C_04	N	29	
3	DQ17	C_03	P	28	CCIO-P
3	DQ16	C_02	N	27	CCIO-N
3	DM2	C_01	P	26	CCIO-P
3		C_00	N	25	CCIO-N
3	DQ15	B_11	P	24	CCIO-P
3	DQ14	B_10	N	23	CCIO-N
3	DQ13	B_09	P	22	CCIO-P
3	DQ12	B_08	N	21	CCIO-N
3	DQS1_P	B_07	P	20	DQSCC-P
3	DQS1_N	B_06	N	19	DQSCC-N
3	DQ11	B_05	P	18	

Table 1-44: 64-Bit DDR3 Interface in Three Banks (Cont'd)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
3	DQ10	B_04	N	17	
3	DQ9	B_03	P	16	
3	DQ8	B_02	N	15	
3	DM1	B_01	P	14	
3		B_00	N	13	
3	DQ7	A_11	P	12	
3	DQ6	A_10	N	11	
3	DQ5	A_09	P	10	
3	DQ4	A_08	N	9	
3	DQS0_P	A_07	P	8	DQSCC-P
3	DQS0_N	A_06	N	7	DQSCC-N
3	DQ3	A_05	P	6	
3	DQ2	A_04	N	5	
3	DQ1	A_03	P	4	
3	DQ0	A_02	N	3	
3	DM0	A_01	P	2	
3	RESET_N	A_00	N	1	
3	VRN	N/A	SE	0	

Table 1-45 shows an example of a 72-bit DDR3 interface contained within three banks. This example is for a 4 GB UDIMM using nine 4 Gb x8 components. The serial presence detect (SPD) pins are not used here. CB[7:0] is represented as DQ[71:64] and S0# as CS\_N for consistency with the component design examples in Table 1-42, page 138, Table 1-43, page 140, and Table 1-44, page 143.

Table 1-45: 72-Bit DDR3 UDIMM Interface in Three Banks

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	VRP	N/A	SE	49	
1	DQ63	D_11	P	48	
1	DQ62	D_10	N	47	
1	DQ61	D_09	P	46	
1	DQ60	D_08	N	45	
1	DQS7_P	D_07	P	44	DQSCC-P
1	DQS7_N	D_06	N	43	DQSCC-N
1	DQ59	D_05	P	42	

**Table 1-45: 72-Bit DDR3 UDIMM Interface in Three Banks (*Cont'd*)**

<b>Bank</b>	<b>Signal Name</b>	<b>Byte Group</b>	<b>I/O Type</b>	<b>I/O Number</b>	<b>Special Designation</b>
1	DQ58	D_04	N	41	
1	DQ57	D_03	P	40	
1	DQ56	D_02	N	39	
1	DM7	D_01	P	38	
1		D_00	N	37	
1	DQ55	C_11	P	36	
1	DQ54	C_10	N	35	
1	DQ53	C_09	P	34	
1	DQ52	C_08	N	33	
1	DQS6_P	C_07	P	32	DQSCC-P
1	DQS6_N	C_06	N	31	DQSCC-N
1	DQ51	C_05	P	30	
1	DQ50	C_04	N	29	
1	DQ49	C_03	P	28	CCIO-P
1	DQ48	C_02	N	27	CCIO-N
1	DM6	C_01	P	26	CCIO-P
1		C_00	N	25	CCIO-N
1	DQ47	B_11	P	24	CCIO-P
1	DQ46	B_10	N	23	CCIO-N
1	DQ45	B_09	P	22	CCIO-P
1	DQ44	B_08	N	21	CCIO-N
1	DQS5_P	B_07	P	20	DQSCC-P
1	DQS5_N	B_06	N	19	DQSCC-N
1	DQ43	B_05	P	18	
1	DQ42	B_04	N	17	
1	DQ41	B_03	P	16	
1	DQ40	B_02	N	15	
1	DM5	B_01	P	14	
1		B_00	N	13	
1	DQ39	A_11	P	12	
1	DQ38	A_10	N	11	
1	DQ37	A_09	P	10	

Table 1-45: 72-Bit DDR3 UDIMM Interface in Three Banks (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
1	DQ36	A_08	N	9	
1	DQS4_P	A_07	P	8	DQSCC-P
1	DQS4_N	A_06	N	7	DQSCC-N
1	DQ35	A_05	P	6	
1	DQ34	A_04	N	5	
1	DQ33	A_03	P	4	
1	DQ32	A_02	N	3	
1	DM4	A_01	P	2	
1		A_00	N	1	
1	VRN	N/A	SE	0	
2	VRP	N/A	SE	49	
2	DQ71	D_11	P	48	
2	DQ70	D_10	N	47	
2	DQ69	D_09	P	46	
2	DQ68	D_08	N	45	
2	DQS8_P	D_07	P	44	DQSCC-P
2	DQS8_N	D_06	N	43	DQSCC-N
2	DQ67	D_05	P	42	
2	DQ66	D_04	N	41	
2	DQ65	D_03	P	40	
2	DQ64	D_02	N	39	
2	DM8	D_01	P	38	
2		D_00	N	37	
2		C_11	P	36	
2		C_10	N	35	
2		C_09	P	34	
2		C_08	N	33	
2		C_07	P	32	DQSCC-P
2		C_06	N	31	DQSCC-N
2		C_05	P	30	
2		C_04	N	29	
2		C_03	P	28	CCIO-P

Table 1-45: 72-Bit DDR3 UDIMM Interface in Three Banks (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
2	ODT0	C_02	N	27	CCIO-N
2	CKE0	C_01	P	26	CCIO-P
2	CS_N0	C_00	N	25	CCIO-N
2	RAS_N	B_11	P	24	CCIO-P
2	CAS_N	B_10	N	23	CCIO-N
2	WE_N	B_09	P	22	CCIO-P
2	BA2	B_08	N	21	CCIO-N
2	CK_P	B_07	P	20	DQSCC-P
2	CK_N	B_06	N	19	DQSCC-N
2	BA1	B_05	P	18	
2	BA0	B_04	N	17	
2	A15	B_03	P	16	
2	A14	B_02	N	15	
2	A13	B_01	P	14	
2	A12	B_00	N	13	
2	A11	A_11	P	12	
2	A10	A_10	N	11	
2	A9	A_09	P	10	
2	A8	A_08	N	9	
2	A7	A_07	P	8	DQSCC-P
2	A6	A_06	N	7	DQSCC-N
2	A5	A_05	P	6	
2	A4	A_04	N	5	
2	A3	A_03	P	4	
2	A2	A_02	N	3	
2	A1	A_01	P	2	
2	A0	A_00	N	1	
2	VRN	N/A	SE	0	
3	VRP	N/A	SE	49	
3	DQ31	D_11	P	48	
3	DQ30	D_10	N	47	
3	DQ29	D_09	P	46	

Table 1-45: 72-Bit DDR3 UDIMM Interface in Three Banks (*Cont'd*)

Bank	Signal Name	Byte Group	I/O Type	I/O Number	Special Designation
3	DQ28	D_08	N	45	
3	DQS3_P	D_07	P	44	DQSCC-P
3	DQS3_N	D_06	N	43	DQSCC-N
3	DQ27	D_05	P	42	
3	DQ26	D_04	N	41	
3	DQ25	D_03	P	40	
3	DQ24	D_02	N	39	
3	DM3	D_01	P	38	
3		D_00	N	37	
3	DQ23	C_11	P	36	
3	DQ22	C_10	N	35	
3	DQ21	C_09	P	34	
3	DQ20	C_08	N	33	
3	DQS2_P	C_07	P	32	DQSCC-P
3	DQS2_N	C_06	N	31	DQSCC-N
3	DQ19	C_05	P	30	
3	DQ18	C_04	N	29	
3	DQ17	C_03	P	28	CCIO-P
3	DQ16	C_02	N	27	CCIO-N
3	DM2	C_01	P	26	CCIO-P
3		C_00	N	25	CCIO-N
3	DQ15	B_11	P	24	CCIO-P
3	DQ14	B_10	N	23	CCIO-N
3	DQ13	B_09	P	22	CCIO-P
3	DQ12	B_08	N	21	CCIO-N
3	DQS1_P	B_07	P	20	DQSCC-P
3	DQS1_N	B_06	N	19	DQSCC-N
3	DQ11	B_05	P	18	
3	DQ10	B_04	N	17	
3	DQ9	B_03	P	16	
3	DQ8	B_02	N	15	
3	DM1	B_01	P	14	

**Table 1-45: 72-Bit DDR3 UDIMM Interface in Three Banks (Cont'd)**

<b>Bank</b>	<b>Signal Name</b>	<b>Byte Group</b>	<b>I/O Type</b>	<b>I/O Number</b>	<b>Special Designation</b>
3		B_00	N	13	
3	DQ7	A_11	P	12	
3	DQ6	A_10	N	11	
3	DQ5	A_09	P	10	
3	DQ4	A_08	N	9	
3	DQS0_P	A_07	P	8	DQSCC-P
3	DQS0_N	A_06	N	7	DQSCC-N
3	DQ3	A_05	P	6	
3	DQ2	A_04	N	5	
3	DQ1	A_03	P	4	
3	DQ0	A_02	N	3	
3	DM0	A_01	P	2	
3	RESET_N	A_00	N	1	
3	VRN	N/A	SE	0	

## Supported Devices for 7 Series 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 1-46](#) lists memory devices supported by the tool for 7 series FPGA DDR3 designs.

**Table 1-46: Supported Components for DDR3 SDRAM**

Components	Packages
MT41J128M8XX-15E	JP, HA, HX, JT
MT41J128M16XX-15E	JP, HA, HX, JT
MT41J128M16XX-187E	JP, HA, HX, JT
MT41J128M8XX-125	JP, HA, HX, JT
MT41J256M8XX-187E	JP, HA, HX, JT
MT41J256M8XX-15E	JP, HA, HX, JT
MT41J128M8XX-125E	JP, HA, HX, JT
MT41J128M8XX-15E	JP, HA, HX, JT
MT41J128M16-15E	JP, HA, HX, JT
MT41J128M16-187E	JP, HA, HX, JT
MT41J128M16-125	JP, HA, HX, JT
MT41J64M16-15E	JP, HA, HX, JT

# *QDRII+ Memory Interface Solution*

---

## Introduction

The QDRII+ SRAM memory interface solution is a physical layer for interfacing Xilinx® 7 series FPGAs 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 7 series 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 7 series FPGAs. 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 202](#).

**Note:** QDRII+ SRAM designs currently do not support memory-mapped AXI4 interfaces.

For detailed information and updates about the 7 series FPGAs QDRII+ SRAM memory interface core, see the Xilinx 7 series FPGA data sheets [\[Ref 15\]](#).

## Getting Started with the CORE Generator Tool

This section is a step-by-step guide for using the CORE Generator™ tool to generate a QDRII+ SRAM memory interface in a 7 series FPGA, run the design through implementation with the Xilinx tools, and simulate the example design using the synthesizable test bench provided.

### System Requirements

- ISE® Design Suite, version 13.4

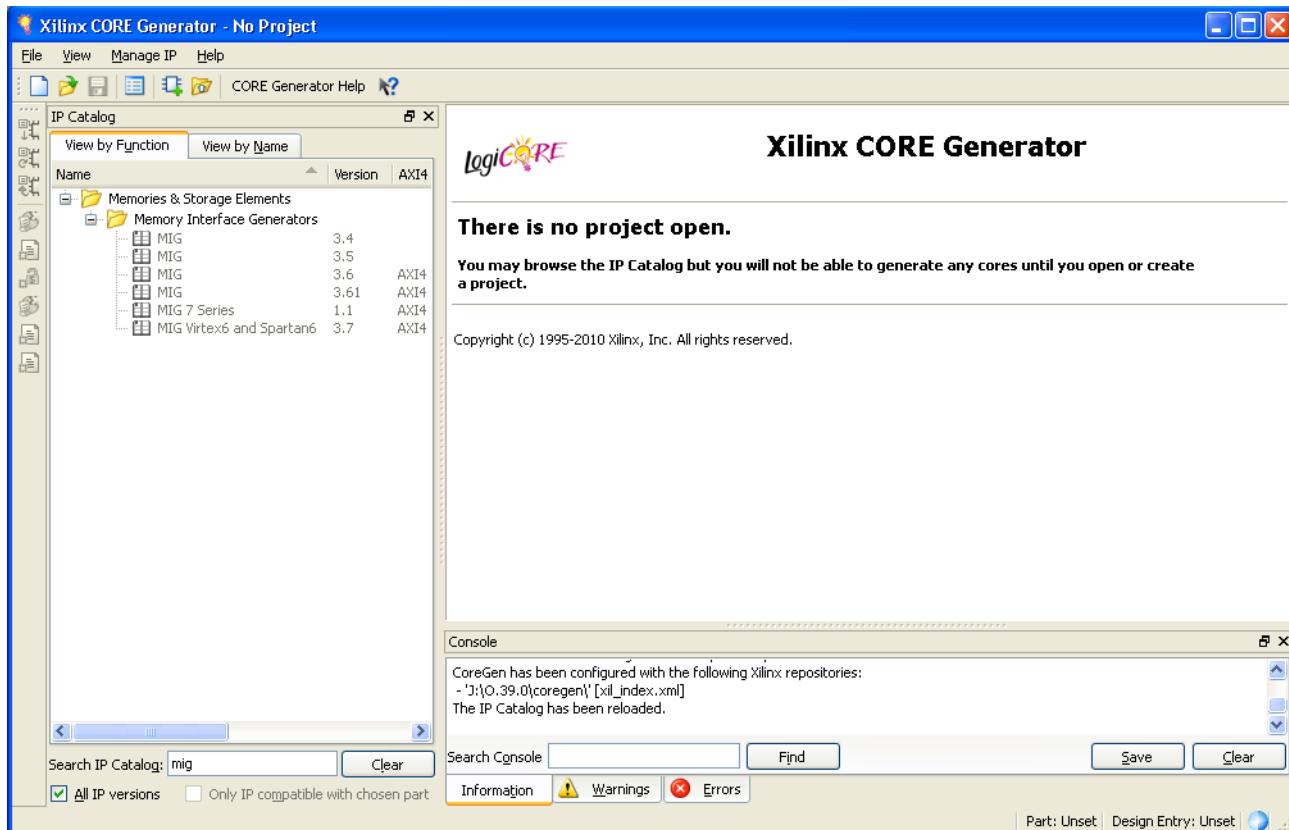
## Customizing and Generating the Core

### Generation through Graphical User Interface

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

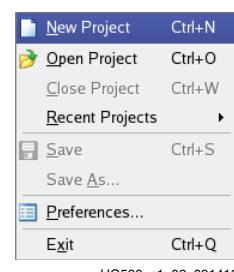
These steps should be followed to generate a 7 series FPGAs QDRII+ SRAM design:

1. To launch the MIG tool from the CORE Generator tool, type **mig** in the search IP catalog box ([Figure 2-1](#)).



*Figure 2-1: Xilinx CORE Generator Tool*

2. Choose **File** → **New Project** to open the New Project dialog box. Create a new project named **7Series\_MIG\_Example\_Design** ([Figure 2-2](#)).



*Figure 2-2: New CORE Generator Tool Project*

3. Enter a project name and location. Click **Save** (Figure 2-3).

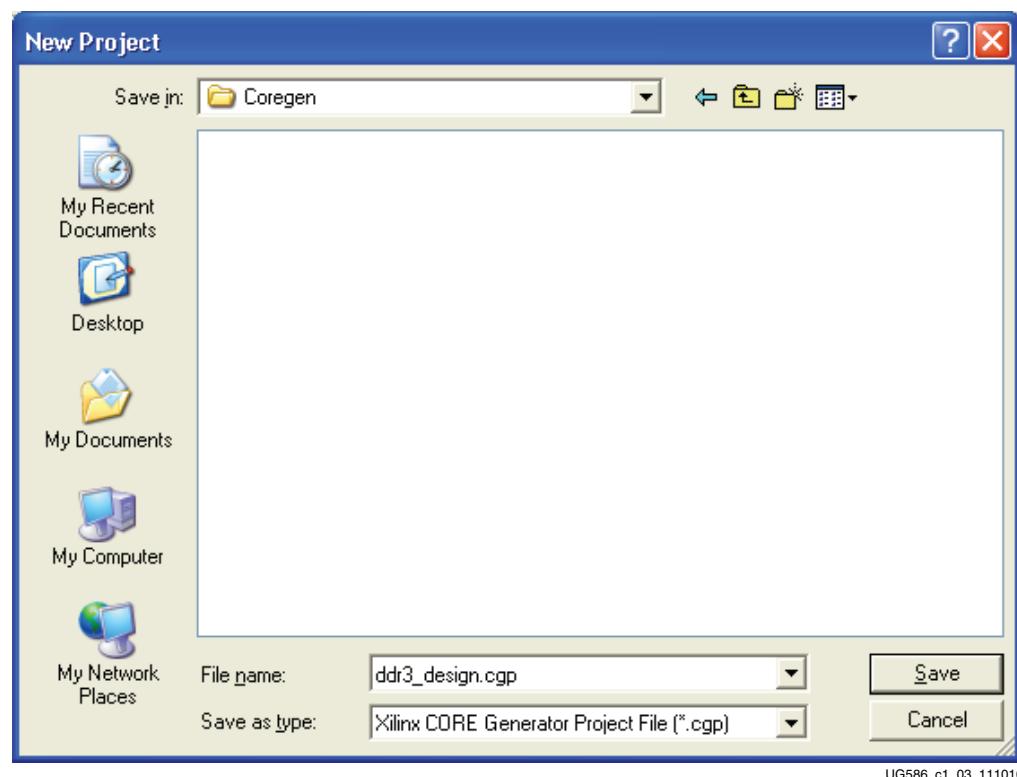


Figure 2-3: New Project Menu

4. Select these project options for the part (Figure 2-4):
- Select the target Kintex™-7 or Virtex®-7 device.

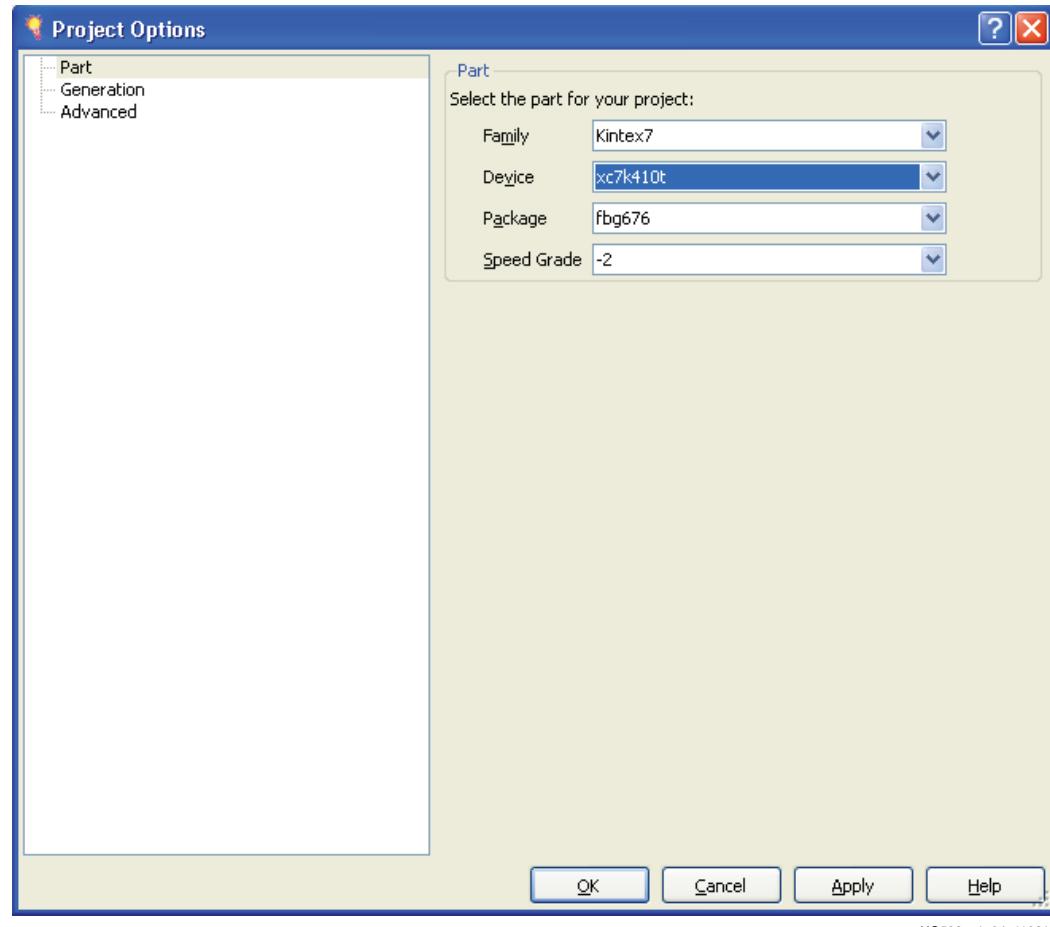


Figure 2-4: CORE Generator Tool Device Selection Page

5. Select **Verilog** as the Design Entry Option and **ISE** for the Vendor Flow Setting. Click **OK** to finish the Project Options setup ([Figure 2-5](#)).

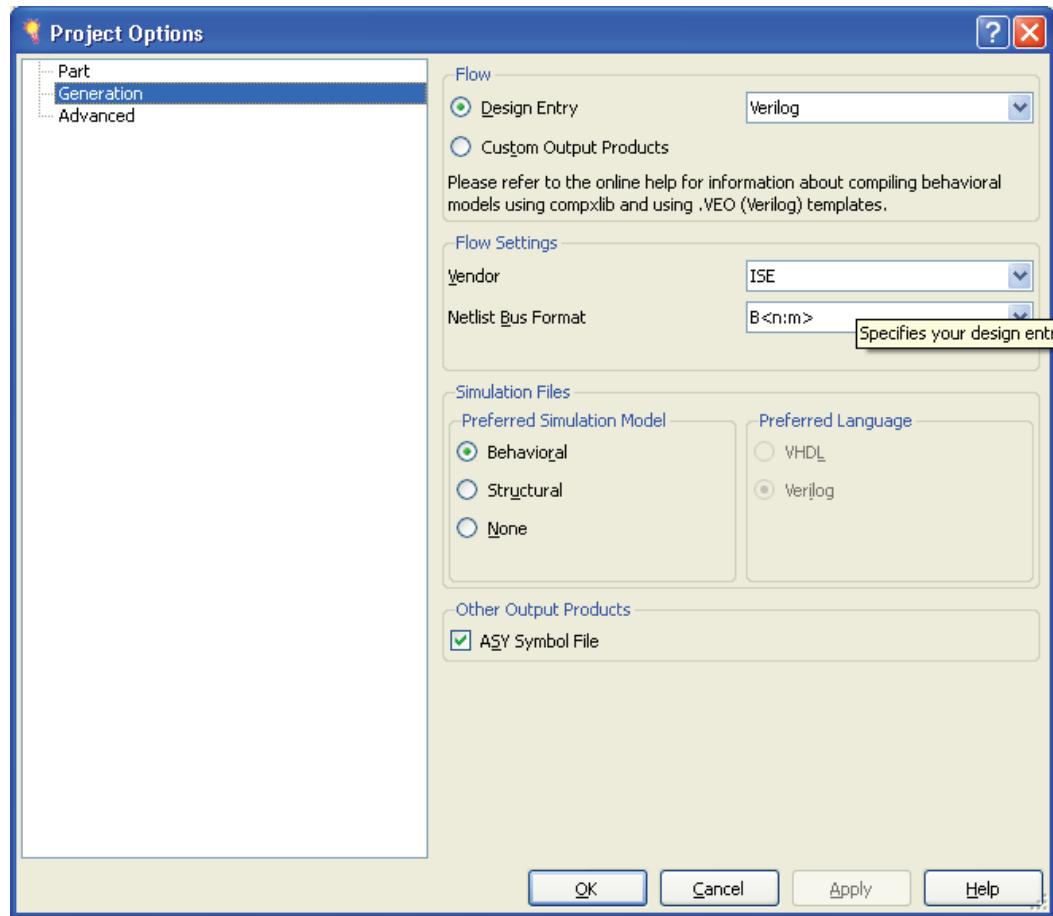


Figure 2-5: CORE Generator Tool Design Flow Setting Page

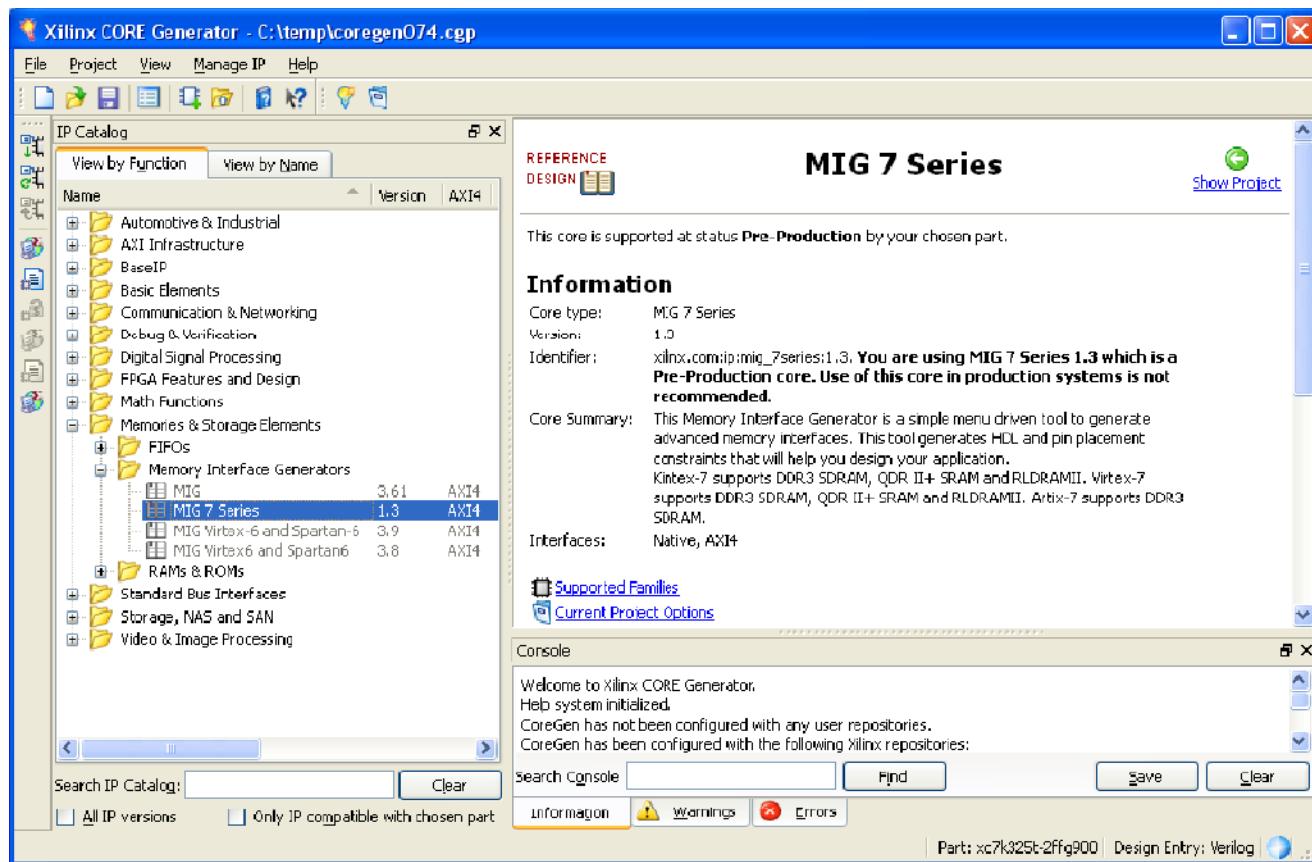
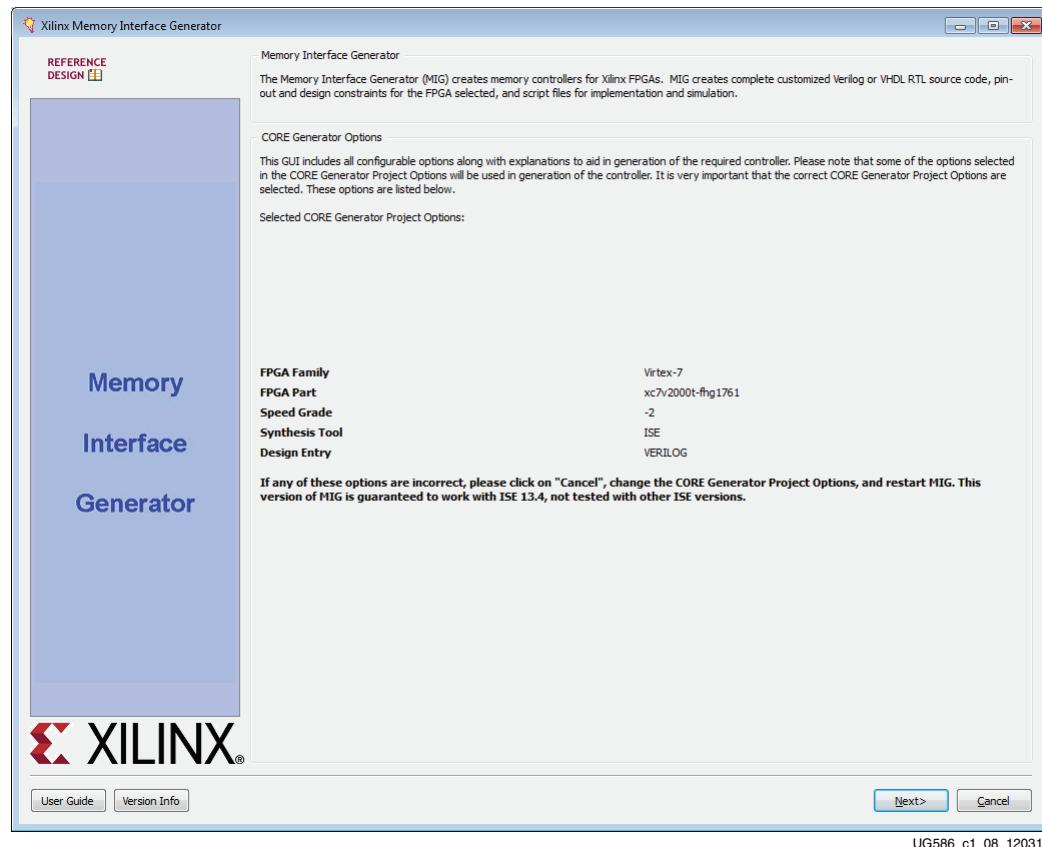
6. Select **MIG 7 Series 1.4** (Figure 2-6).

Figure 2-6: 7 Series\_MIG Design Project Page

7. The options screen in the CORE Generator tool displays the details of the selected CORE Generator tool options that are selected before invoking the MIG tool ([Figure 2-7](#)).

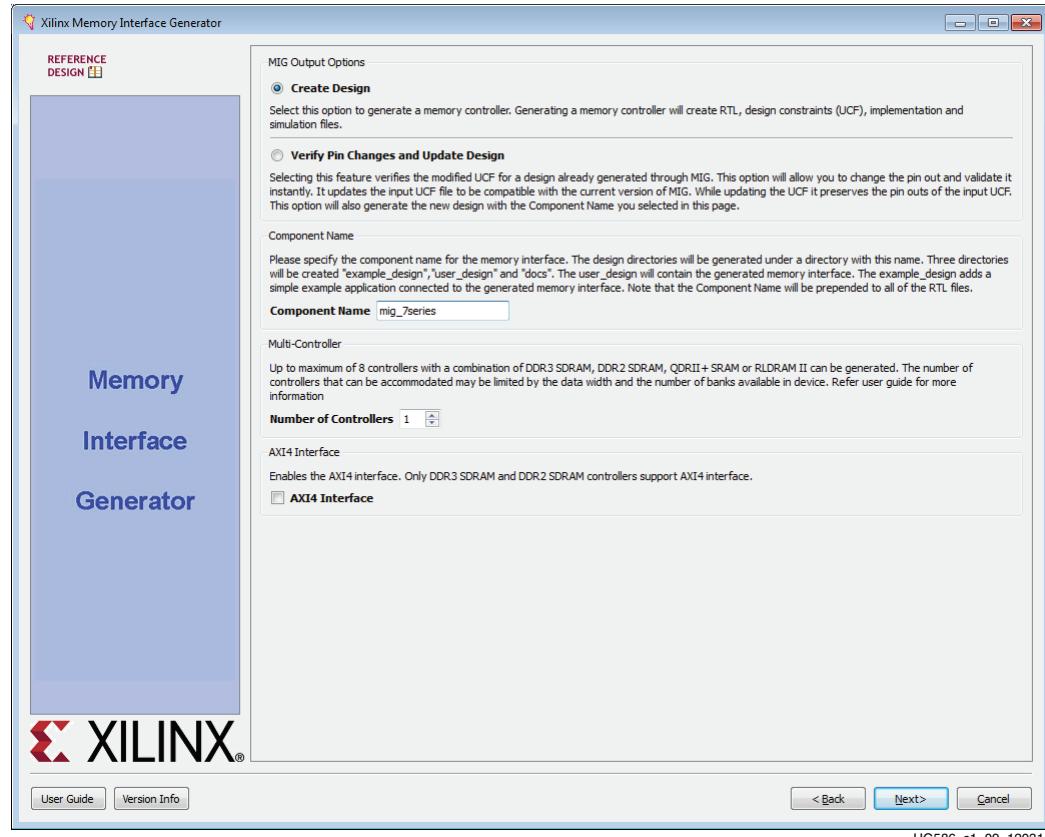


*Figure 2-7: 7 Series FPGAs Memory Interface Generator Front Page*

8. Click **Next** to display the **Output Options** page.

## 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 2-8](#)).
2. Choose the number of controllers to be generated. This selection determines the replication of further pages.



*Figure 2-8: 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, the component name is corrected to be the IP instance name from XPS.

3. Click **Next** to display the **Pin Compatible FPGAs** page.

## Pin Compatible FPGAs

The **Pin Compatible FPGAs** page 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 2-9).

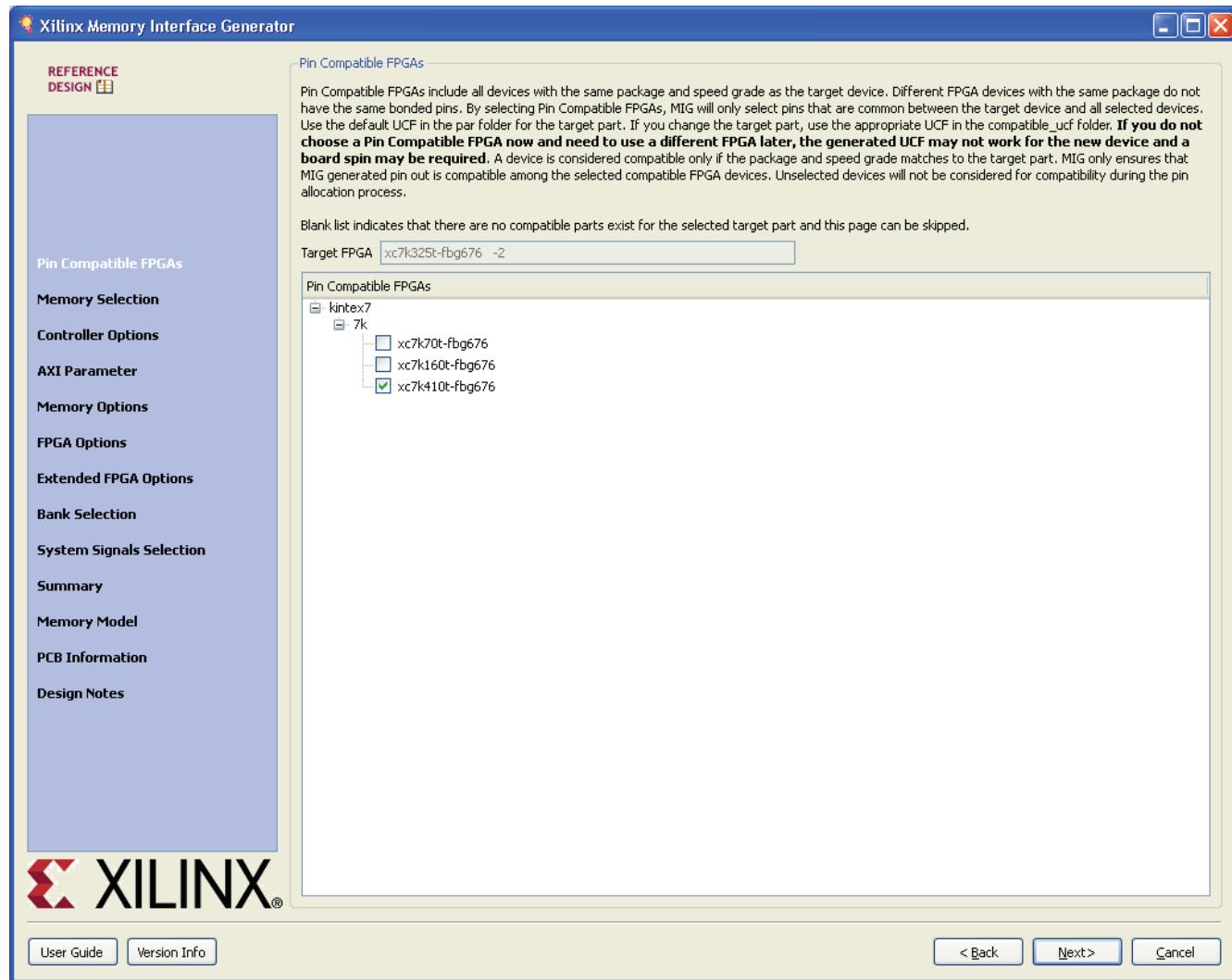


Figure 2-9: Pin-Compatible 7 Series FPGAs

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** page.

## Creating the 7 Series 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** page

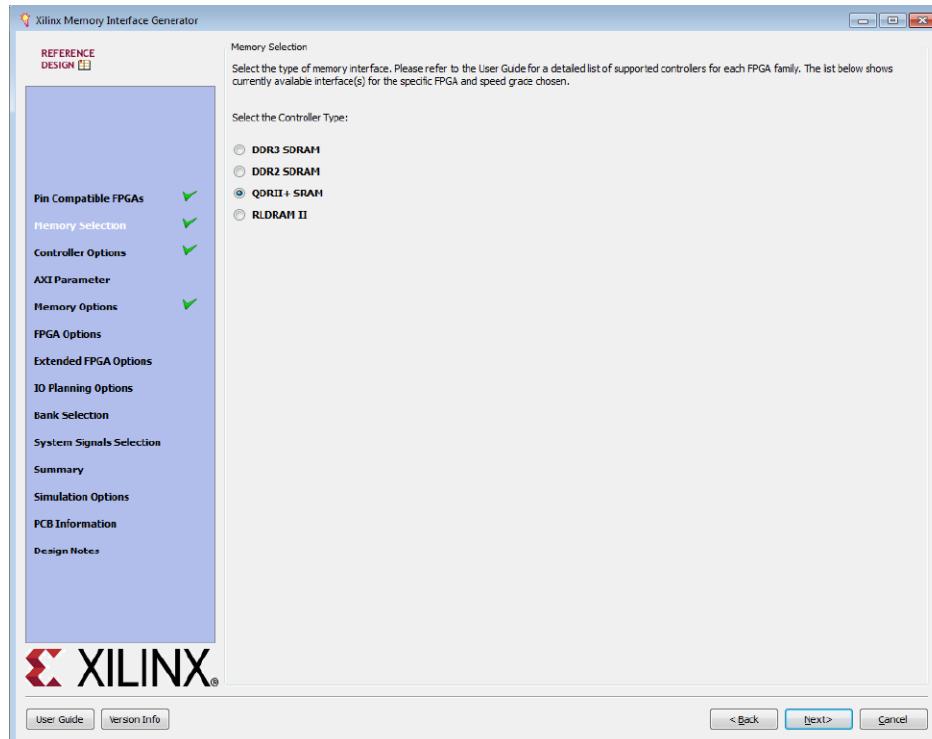


Figure 2-10: Memory Selection Page

QDRII+ SRAM designs do not support memory-mapped AXI4 interfaces.

## Controller Options

This page shows the various controller options that can be selected.

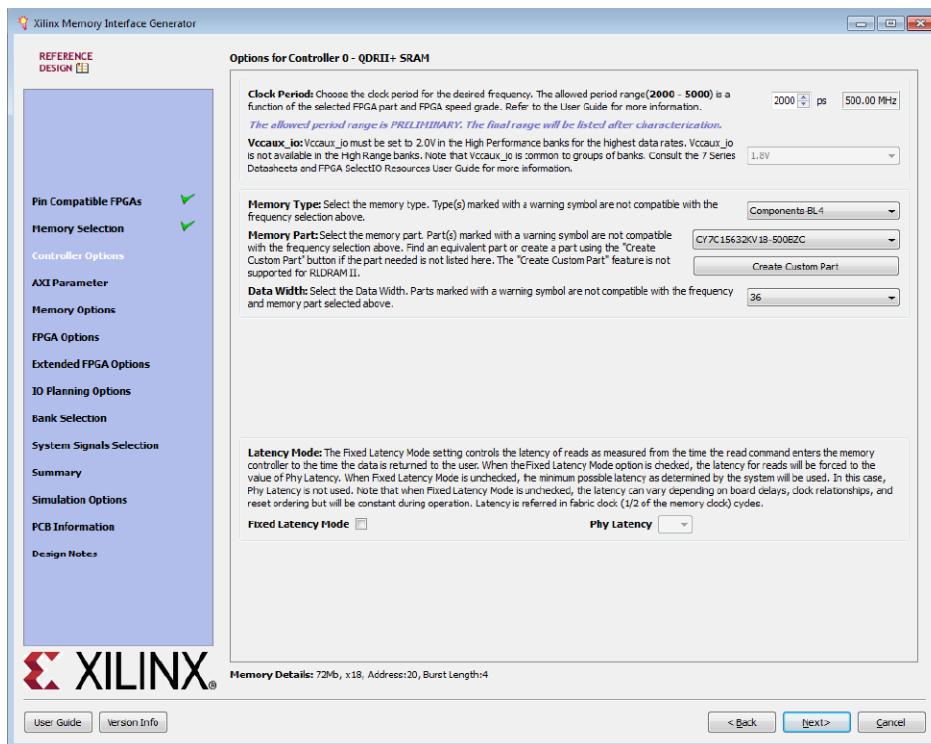


Figure 2-11: Controller Options Page

- **Frequency:** This feature indicates the operating frequency for all the controllers. The frequency block is limited by factors such as the selected FPGA and device speed grade.
- **Vccaux\_io:** Vccaux\_io is set based on the period/frequency setting. 2.0V is required at the highest frequency settings in the High Performance column. The MIG tool automatically selects 2.0V when required. Either 1.8 or 2.0V can be used at lower frequencies. Groups of banks share the Vccaux\_io supply. See the *7 Series FPGAs SelectIO Resources User Guide* [Ref 1] for more information.
- **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 (Create Custom Part). 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.
- **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.
- **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 21 to 30 cycles. Based on actual hardware conditions, if the latency seen through the system is higher, this value needs to be modified accordingly by the user in the top level RTL file. If **Fixed Latency Mode** is not used, the core uses the minimum number of cycles through the system.

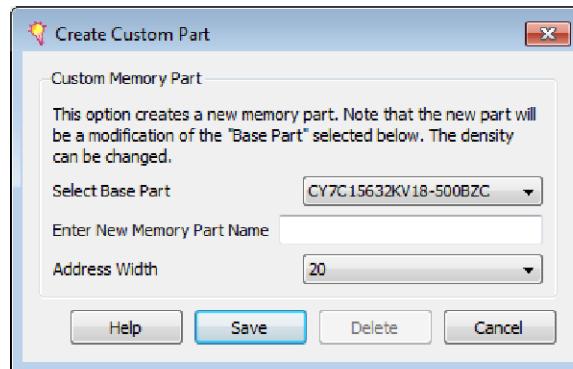
- **Memory Details:** The bottom of the **Controller Options** page. [Figure 2-12](#) displays the details for the selected memory configuration.

**Memory Details:** 72Mb, x18, Address:20, Burst Length:4

*Figure 2-12: Selected Memory Configuration Details*

### Create Custom Part

1. On the **Controller Options** page select the appropriate frequency. 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. 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 page appears, as shown in [Figure 2-13](#).



*Figure 2-13: Create Custom Part Page*

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

1. Enter the appropriate **Memory Part Name** in the text box.
2. Select the suitable base part from the **Select Base Part** list.
3. Select a suitable value for the Row Address.
4. 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. It is also saved into the database for reuse and to produce the design.
5. Click **Next** to display the **FPGA Options** page.

## Memory Options

Figure 2-14 shows the Memory Options page.

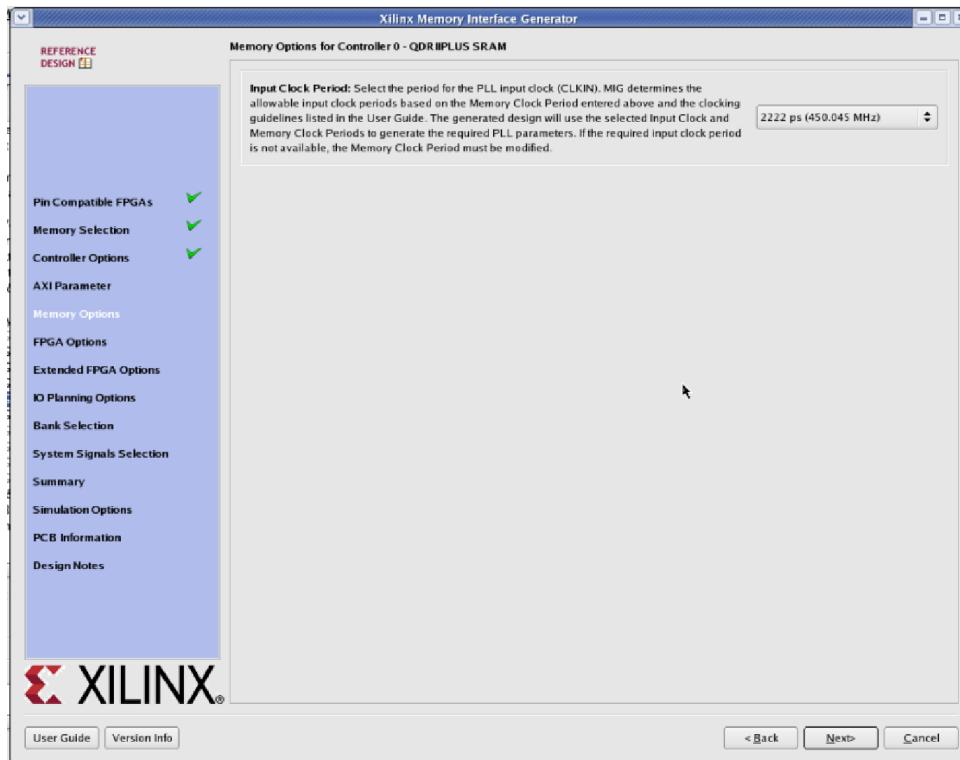


Figure 2-14: Memory Options Page

- **Input Clock Period:** The desired input clock period is selected from the list. These values are determined by the chosen memory clock period and the allowable limits of the PLL parameters. See [Clocking Architecture, page 186](#) for more information on the PLL parameter limits.

## FPGA Options

Figure 2-15 shows the FPGA Options page.

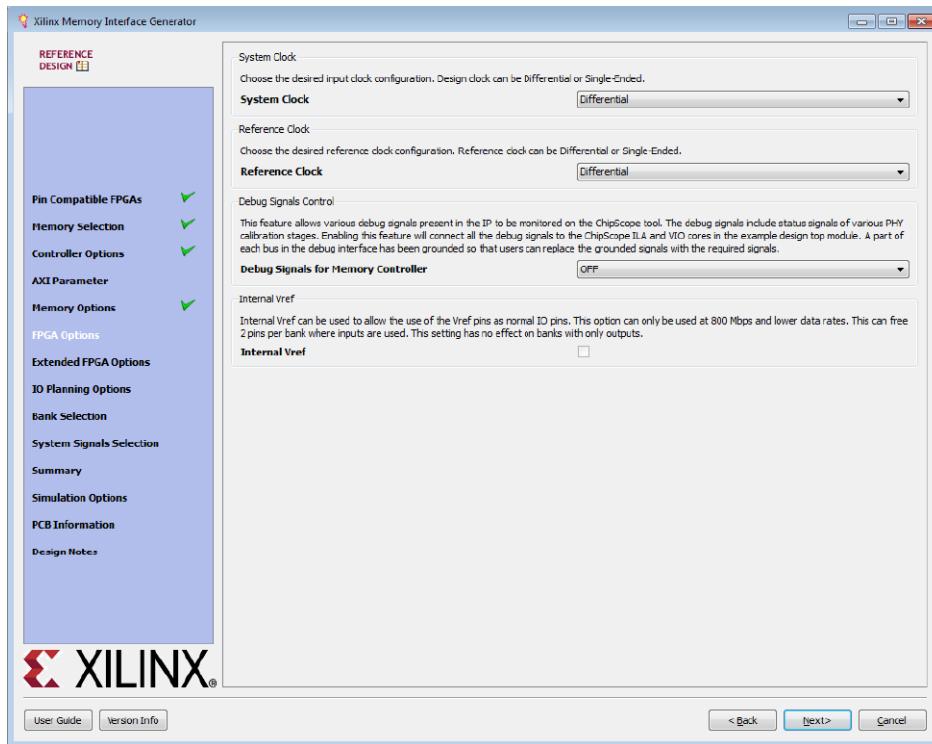


Figure 2-15: **FPGA Options Page**

- **System Clock.** This option selects the clock type (Single-Ended or Differential) for the sys\_clk signal pair.
- **Reference Clock.** This option selects the clock type (Single-Ended or Differential) for the ref\_clk signal pair.
- **Debug Signals Control.** This option enables calibration status and user port signals to be port mapped to the ChipScope™ analyzer modules in the design\_top module. 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™ tool 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 tool. The debug port is always disabled for functional simulations.
- **Internal Vref Selection.** Internal Vref can be used for data group bytes to allow the use of the V<sub>REF</sub> pins for normal I/O usage. Internal Vref should only be used for data rates of 800 Mb/s or below.

Click **Next** to display the Extended FPGA Options page.

## Extended FPGA Options

Figure 2-16 shows the Extended FPGA Options page.

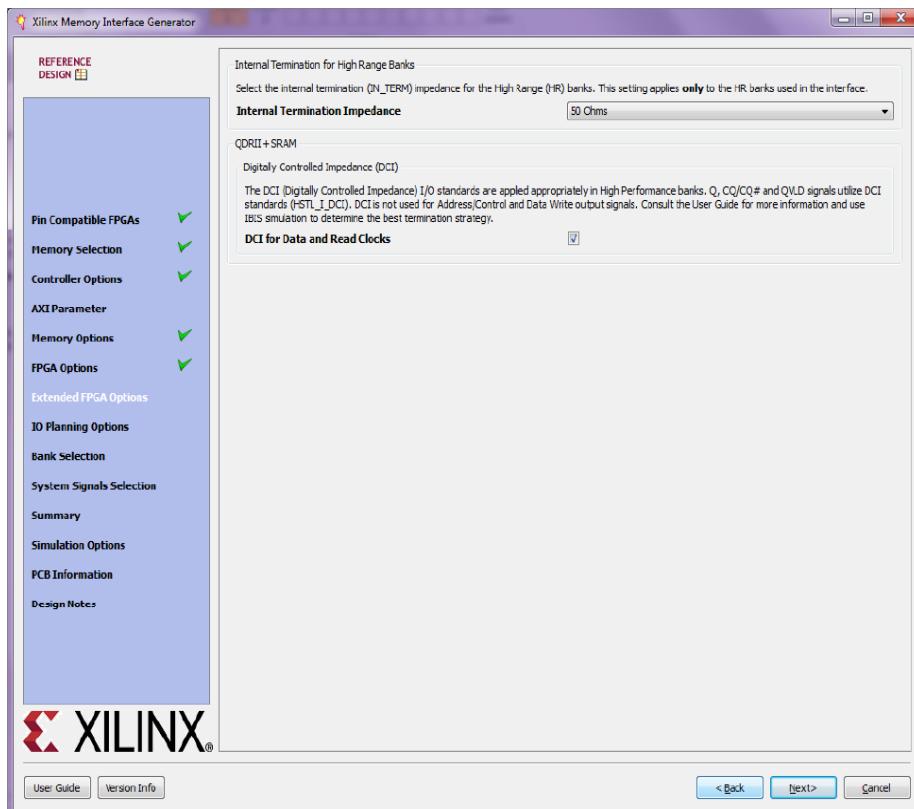


Figure 2-16: Extended FPGA Options Page

- **Digitally Controlled Impedance (DCI):** When selected, this option internally terminates the signals from the QDRII+ SRAM read path. DCI is available in the High Performance Banks.
- **Internal Termination for High Range Banks.** The internal termination option can be set to 40, 50, or 60Ω or disabled. This termination is for the read datapath from the QDRII+ SRAM. This selection is only for High Range banks.

## I/O Planning Options

Figure 2-17 shows the I/O Planning Options page.

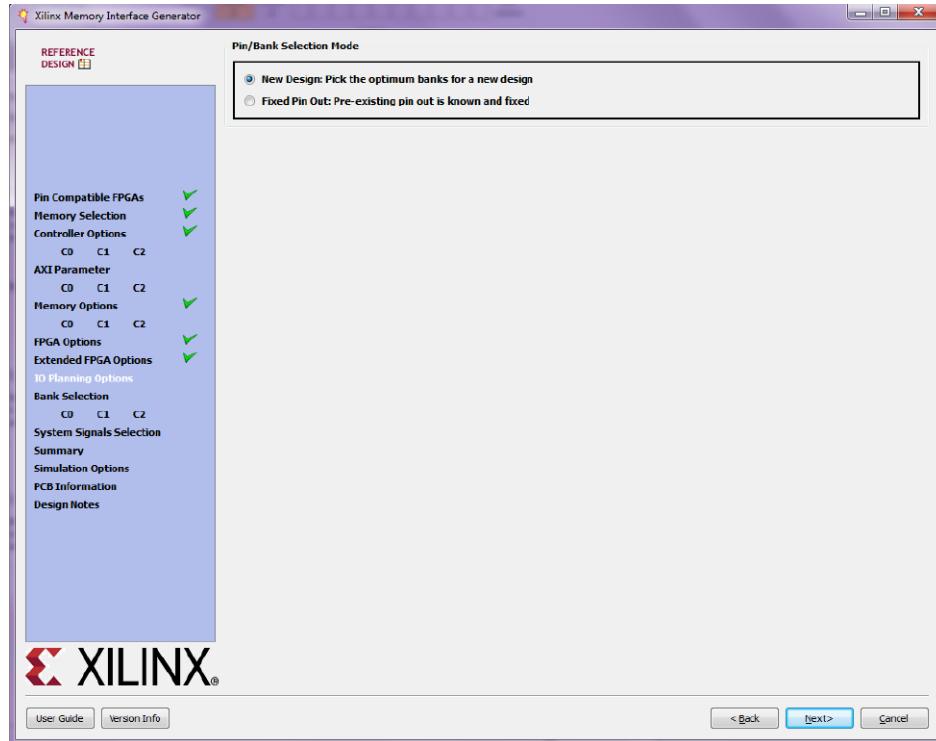


Figure 2-17: I/O Planning Options Page

- **Pin/Bank Selection Mode:** This allows the user to specify an existing pinout and generate the RTL for this pinout or pick banks for a new design. Figure 2-18 shows the options for using an existing pinout. The user must assign the appropriate pins for each signal. A choice of each bank 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 against the MIG pinout rules. One cannot proceed until the MIG DRC has been validated by clicking the **Validate** button.

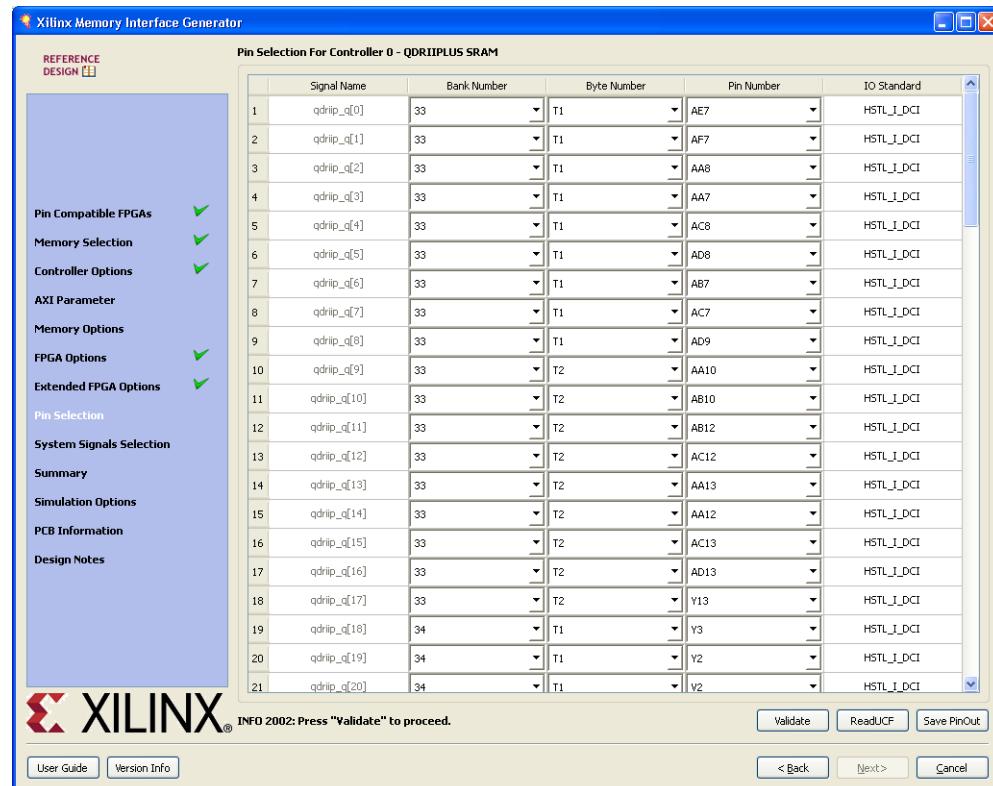


Figure 2-18: Pin/Bank Selection Mode

## Bank Selection

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

- Address and control signals
- Data Read signals
- Data Write signals

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. To unselect the banks that are selected, click the **Deselect Banks** button. To restore the defaults, click the **Restore Defaults** button. Vccaux\_io groups are shown for HP banks in devices with these groups using dashed lines. Vccaux\_io is common to all banks in these groups. The memory interface must have the same Vccaux\_io for all banks used in the interface. MIG automatically sets the VCCAUX\_IO constraint appropriately for the data rate requested.

For devices implemented with SSI technology, the SLRs are indicated by a number in the header in each bank, for example, *SLR 1*. Interfaces cannot span across Super Logic Regions. Not all devices have Super Logic Regions.

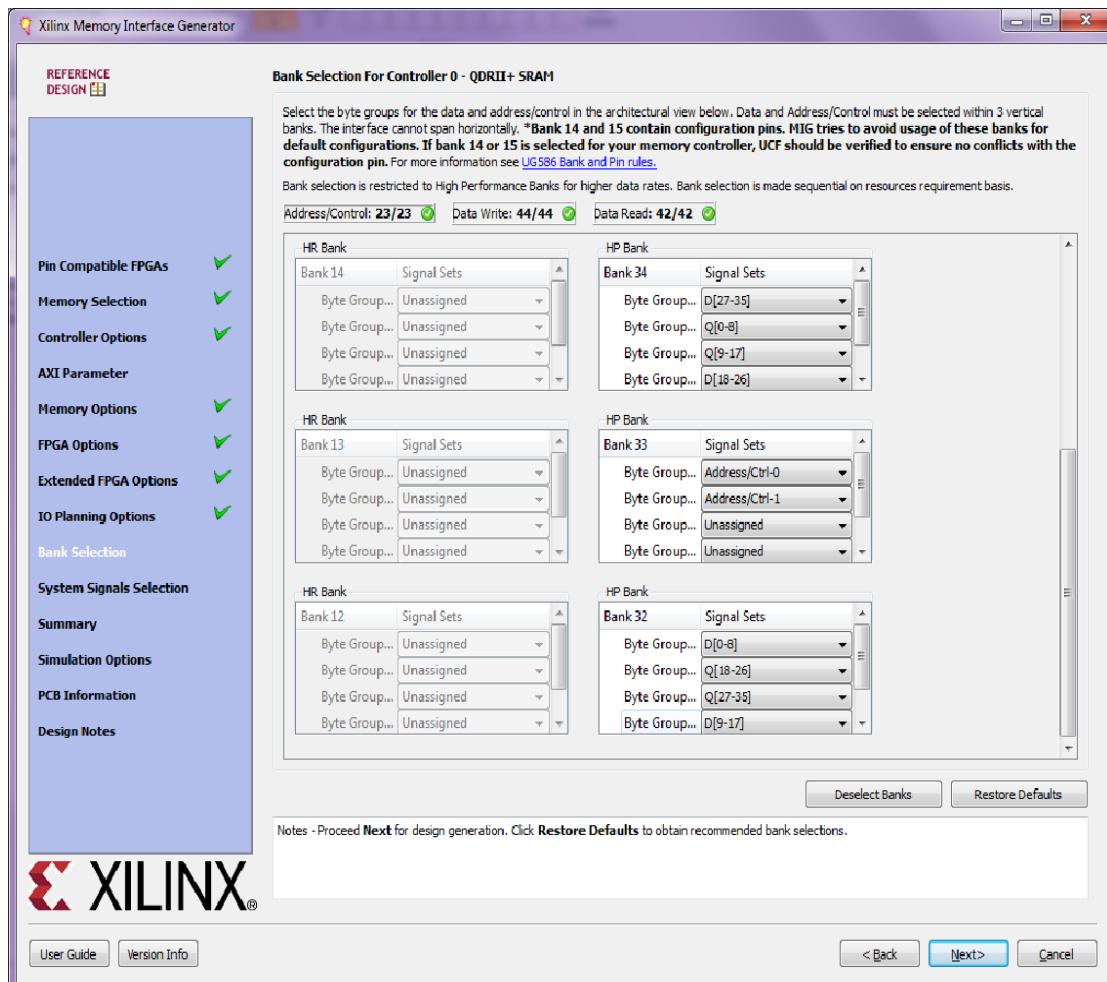


Figure 2-19: Bank Selection Page

## System Pins Selection

Select the pins for the system signals on this page. The MIG tool allows the selection of either external pins or internal connections, as desired.

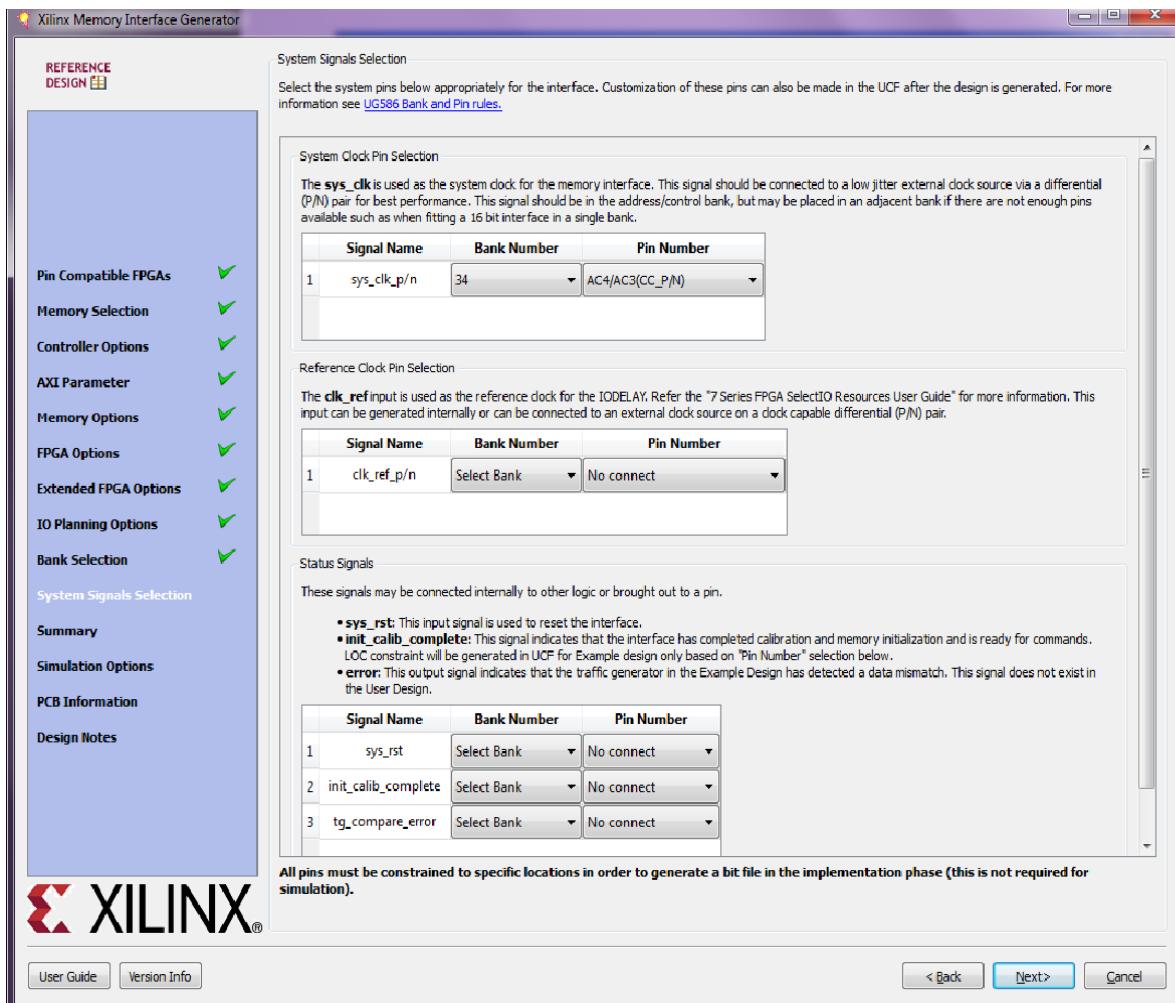


Figure 2-20: System Pins Selection Page

- **sys\_clk:** This is the system clock input for the memory interface and is typically connected to a low-jitter external clock source. Either a single input or a differential pair can be selected based on the **System Clock** selection in the **FPGA Options** page (Figure 2-15). The sys\_clk input must be in the same column as the memory interface. If this pin is connected in the same banks as the memory interface, the MIG tool selects an I/O standard compatible with the interface, such as DIFF\_SSTL15 or SSTL15. If sys\_clk is not connected in a memory interface bank, the MIG tool selects an appropriate standard such as LVCMOS18 or LVDS. The UCF can be modified as desired after generation.
- **clk\_ref:** This is the reference frequency input for the IDELAY control. This is a 200 MHz input. The clk\_ref input can be generated internally or connected to an external source. A single input or a differential pair can be selected based on the **System Clock** selection in the **FPGA Options** page (Figure 2-15). The I/O standard is selected in a similar way as sys\_clk above.

- **sys\_rst**: This is the system reset input that can be generated internally or driven from a pin. The MIG tool selects an appropriate I/O standard for the input such as SSTL15 if the input is within the interface banks, and LVCMOS18 if it is not.
- **init\_calib\_complete**: This output indicates that the memory initialization and calibration is complete and that the interface is ready to use. The init\_calib\_complete signal is normally only used internally, but can be brought out to a pin if desired.
- **tg\_compare\_error**: This output indicates that the traffic generator in the example design has detected a data compare error. This signal is only generated in the example design and is not part of the user design. This signal is not typically brought out to a pin but can be, if desired.

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

## Summary

This page provides the complete details about the 7 series FPGA memory core selection, interface parameters, CORE Generator tool options, and FPGA options of the active project.

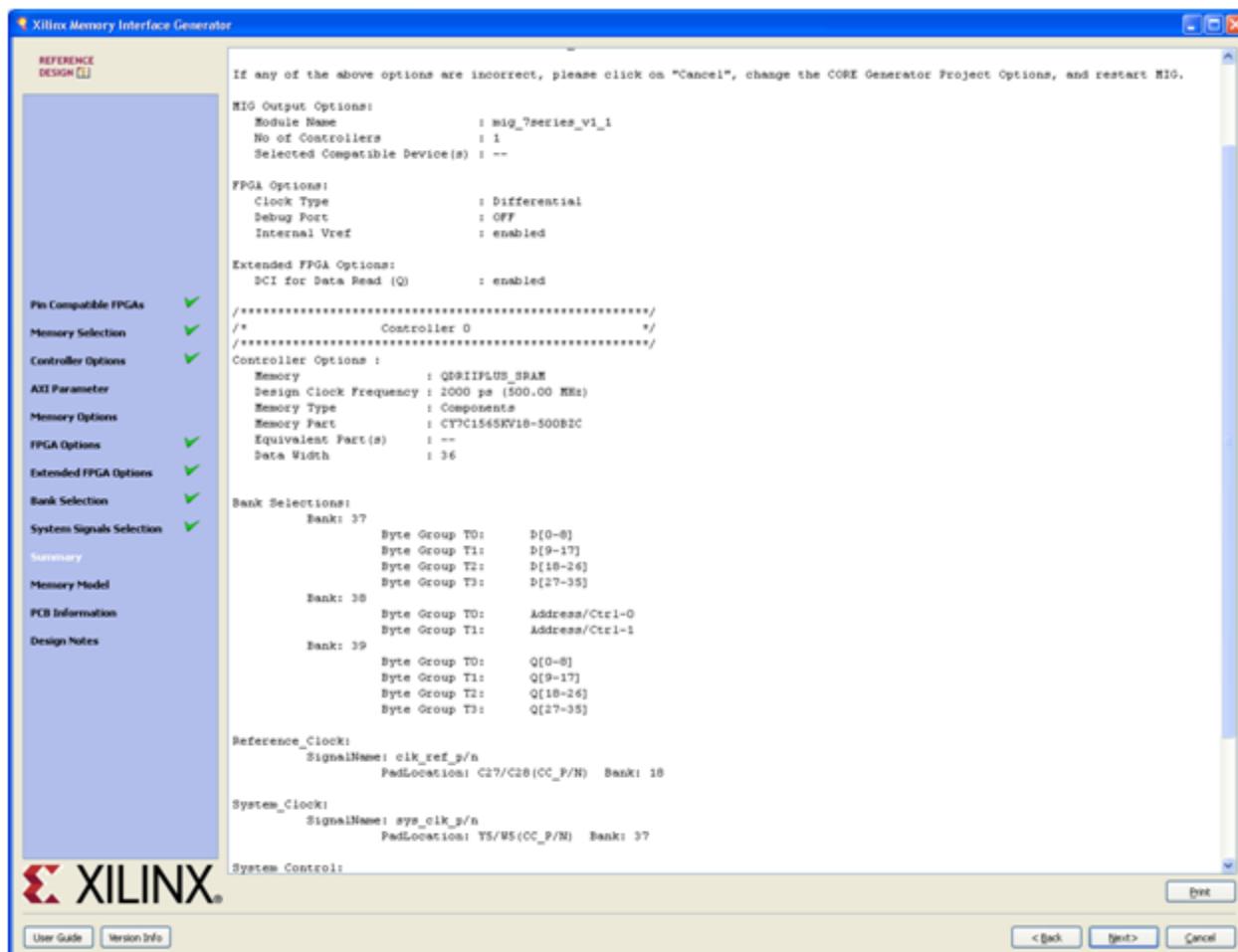


Figure 2-21: Summary Page

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 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.

## Finish

After the design is generated, a README page is displayed with additional useful information.

Click **Close** to complete the MIG tool flow.

## MIG Directory Structure and File Descriptions

This section explains the MIG tool directory structure and provides detailed output file descriptions.

### Output Directory Structure

The MIG tool places all output files and directories in a folder called `<component_name>`, where `<component_name>` was specified on the [MIG Output Options, page 162](#) of the MIG design creation flow.

[Figure 2-22](#) shows the output directory structure for the memory controller design. There are three folders created within the `<component_name>` directory:

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

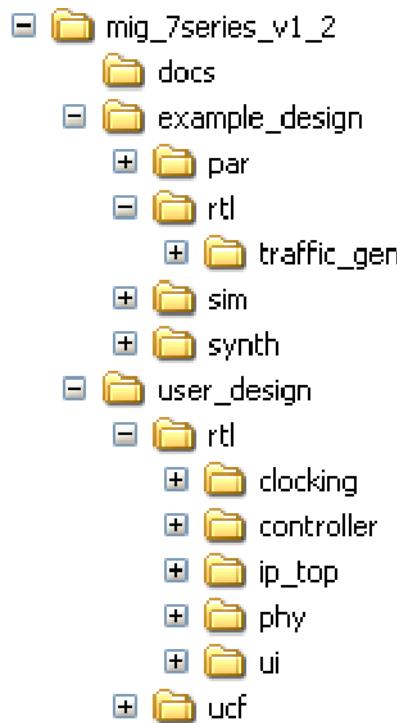


Figure 2-22: MIG Directory Structure

## Directory and File Contents

The 7 series FPGAs 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 directory structure contains all necessary RTL, constraints, and script files for simulation and implementation of the complete MIG example design with a test bench. The optional ChipScope™ tool module is also included in this directory structure.

[Table 2-1](#) lists the files in the example\_design/rtl directory.

**Table 2-1: Files in example\_design/rtl Directory**

Name	Description
example_top.v	This top-level module serves as an example for connecting the user design to the 7 series FPGA memory interface core.

**Table 2-2** lists the files in the example\_design/rtl/traffic\_gen directory.

**Table 2-2: Modules in example\_design/rtl/traffic\_gen Directory**

Name	Description
memc_traffic_gen.v/vhd	This is the top-level module 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 pseudo-random binary sequence (PRBS) generator generates PRBS commands, addresses, and burst lengths.
memc_flow_vcontrol.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. 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_vcontrol.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 writes and ready signals to mcb_flow_vcontrol.v/vhd.
s7ven_data_gen.v/vhd	This module generates different data patterns.
a_fifo.v/vhd	This is a synchronous FIFO using LUT RAMs.
data_prbs_gen.v/vhd	This 32-bit linear feedback shift register (LFSR) generates PRBS data patterns.
init_mem_pattern_ctr.v/vhd	This module generates flow control logic for the traffic generator.
traffic_gen_top.v/vhd	This module is the top level of the traffic generator and comprises the memc_traffic_gen and init_mem_pattern_ctr modules.
tg_prbs_gen.v/vhd	This PRBS uses one too many feedback mechanisms because it always has a single level XOR (XNOR) for feedback. The TAP is chosen from the table listed in XAPP052, <i>Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators</i> . The TAPS position can be defined in a parameter.
tg_status.v/vhd	This module compares the memory read data against compare data generated from the data_gen module. The error signal is asserted if the comparison is not equal.
vio_init_pattern_bram.v/vhd	This module takes external defined data inputs as its block RAM init pattern. It allows users to change simple test data pattern without recompilation.

**Table 2-3** lists the files in the example\_design/sim directory.

**Table 2-3: Files in example\_design/sim Directory**

Name	Description
sim.do	This is the ModelSim simulator script file.
sim_tb_top.v	This file is the simulation top-level file.

**Table 2-4** lists the files in the example\_design/par directory.

**Table 2-4: Files in the example\_design/par Directory**

Name	Description
example_top.ucf	This file is the UCF for the core of the example design.
create_ise.bat	The user double-clicks this file to create an ISE tool project. The generated ISE tool project contains the recommended build options for the design. To run the project in GUI mode, the user double-clicks the ISE tool project file to open up the ISE tool in GUI mode with all project settings.
ise_flow.bat	This script file runs the design through synthesis, build, map, and par. It sets all the required options. Users should refer to this file for the recommended build options for the design.
rem_files.bat	Removes all the implementation files generated during implementation
set_ise_prop.tcl	List of properties to ISE tool
xst_options.txt	List of properties to synthesis tool
ila_cg.xco, icon_cg.xco, vio_cg.xco	XCO files for ChipScope modules to be generated when debug is enabled

**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.

**Table 2-5** lists the files in the example\_design/synth directory.

**Table 2-5: Files in the example\_design/synth Directory**

Name	Description
example_top.prj	Lists all the RTL files to be compiled by XST tool

#### <component name>/user\_design/

This directory contains the memory controller RTL files that are instantiated in the example design and a UCF.

**Table 2-6** lists the files in the user\_design/rtl directory

**Table 2-6: Files in the user\_design/rtl Directory**

Name	Description
<component_name>.v	This top level module servers as an example for connecting the user design to the 7 series FPGA QDRII+ SRAM memory interface core.

**Table 2-7** lists the files in the user\_design/rtl/clocking directory.

**Table 2-7: Files in the user\_design/rtl/clocking Directory**

Name	Description
infrastructure.v	This module helps in clock generation and distribution.
clk_ibuf.v	This module instantiates the system clock input buffers.
iodelay_ctrl.v	This module instantiates the IDELAYCTRL primitive needed for IODELAY use.

**Table 2-8** lists the files in the user\_design/rtl/phy directory:

**Table 2-8: Files in the user\_design/rtl/phy**

Name	Description
qdr_phy_top.v	This is the top-level module for the physical layer.
qdr_phy_write_top.v	This is the top-level wrapper for the write path.
qdr_rld_phy_read_top.v	This is the top-level of the read path.
qdr_rld_mc_phy.v	This module is a parameterizable wrapper instantiating up to three I/O banks each with 4-lane PHY primitives.
qdr_phy_write_init_sm.v	This module contains the logic for the initialization state machine.
qdr_phy_write_control_io.v	This module contains the logic for the control signals going to the memory.
qdr_phy_write_data_io.v	This module contains the logic for the data and byte writes going to the memory.
qdr_rld_prbs_gen.v	This PRBS module uses a many-to-one feedback mechanism for 2n sequence generation.
qdr_rld_phy_ck_addr_cmd_delay.v	This module contains the logic to provide the required delay on the address and control signals
qdr_rld_phy_rdlvl.v	This module contains the logic for stage 1 calibration.
qdr_rld_phy_read_stage2_cal.v	This module contains the logic for stage 2 calibration.
qdr_rld_phy_read_data_align.v	This module realigns the incoming data.
qdr_rld_phy_read_vld_gen.v	This module contains the logic to generate the valid signal for the read data returned on the user interface.
qdr_phy_byte_lane_map.v	This wrapper file handles the vector remapping between the mc_phy module ports and the user's memory ports.
qdr_rld_phy_4lanes.v	This module is the parameterizable 4-lane PHY in an I/O bank.
qdr_rld_byte_lane.v	This module contains the primitive instantiations required within an output or input byte lane.
qdr_rld_byte_group_io.v	This module contains the parameterizable I/O Logic instantiations and the I/O terminations for a single byte lane.

**Table 2-9** lists the files in the user\_design/ucf directory.

**Table 2-9: Files in the user\_design/ucf Directory**

Name	Description
<component name>.ucf	This file is the UCF for the core of the user design.

## Verify Pin Changes and Update Design

This feature verifies the input UCF for bank selections, byte selections, and pin allocation. It also generates errors and warnings in a separate dialog when the user clicks on the **Validate** button on the page. This feature is useful to verify the UCF for any pinout changes made 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. In the CORE Generator tool, the recustomization option should be selected to reload the project. The design is allowed to generate only when the MIG DRC is met. Ignore warnings about validating the pinout, which is the intent. Just validating the UCF is not sufficient; it is mandatory to proceed with design generation to get the UCF with updated clock and phaser-related constraints and RTL top-level module for various updated Map parameters.

Here are the rules verified from the input UCF:

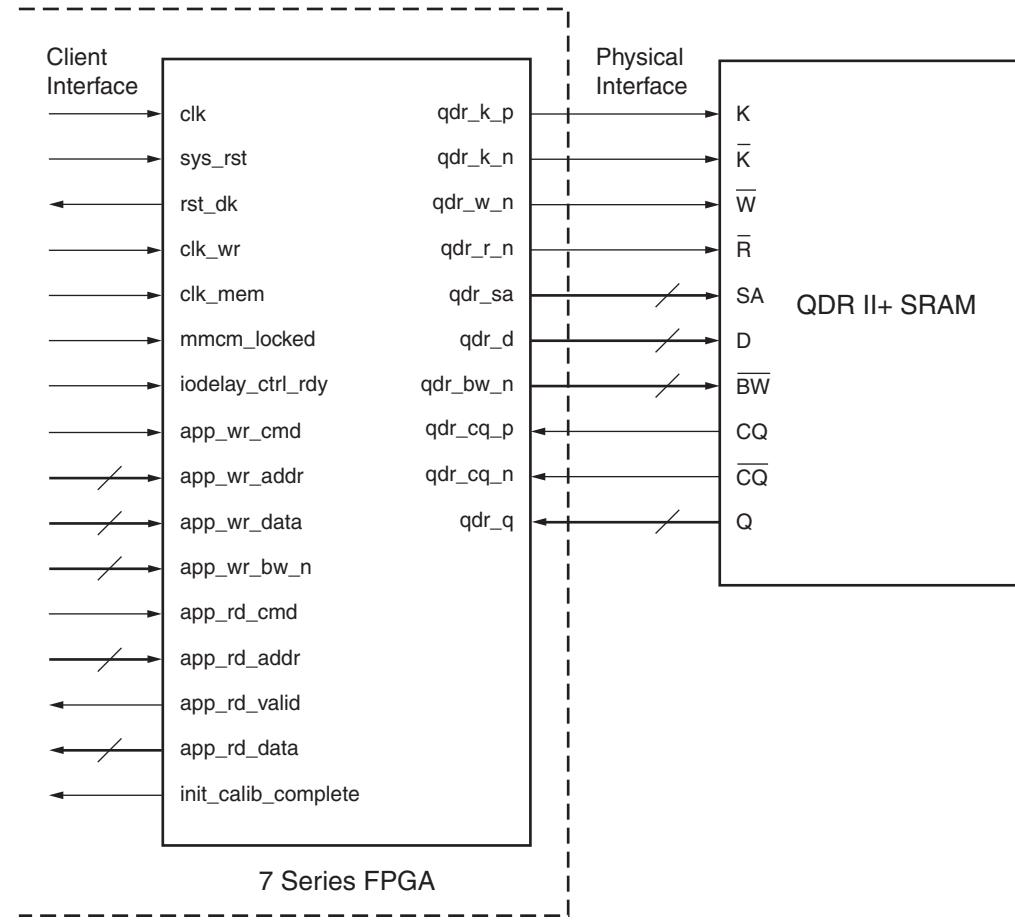
- If a pin is allocated to more than one signal, the tool reports an error. Further verification is not done if the UCF does not adhere to the uniqueness property.
- Verified common rules:
  - The interface can span across a maximum of three consecutive banks.
  - Interface banks should reside in the same column of the FPGA.
  - Interface banks should be either High Performance (HP) or High Range (HR). HP banks are used for the high frequencies.
  - The chosen interface banks should have the same SLR region if the chosen device is of stacked silicon interconnect technology.
  - V<sub>REF</sub> I/Os should be used as GPIOs when an internal V<sub>REF</sub> is used or if there are no inout and input ports in a bank.
  - The I/O standard of each signal is verified as per the configuration chosen.
  - The VCCAUX I/O of each signal is verified and provides a warning message if the provided VCCAUX I/O is not valid.
- Verified data read pin rules:
  - Pins related to one component should be allocated in one bank only.
  - The strobe pair (CQ) should be allocated to either the MRCC P or the MRCC N pin.
  - Read data pins cannot span more than the required byte lanes. For example, an 18-bit component should occupy only 2 byte lanes.
  - A byte lane should contain pins of only one read byte, for example, Q[0-8] or Q[9-17].
  - A byte lane should not contain pins of more than one component.
  - An FPGA byte lane should not contain pins related to two different strobe sets.
  - V<sub>REF</sub> I/O can be used only when the internal V<sub>REF</sub> is chosen.

- Verified data write pin rules:
  - Pins related to one component should be allocated in only one bank.
  - Write clocks (K/K#) pairs should be allocated to the DQS CC I/Os.
  - Write data pins cannot span more than the required byte lanes. For example, an 18-bit component should occupy only 2 byte lanes.
  - A byte lane should not contain pins of more than one component.
  - A byte lane should contain pins of only one write byte, for example, D[0-8] or D[9-17].
  - Irrespective of internal Vref usage, VREF pins can be used as GPIOs unless the bank contains other input signals.
- Verified address pin rules:
  - Address signals cannot mix with data bytes except for the qdriip\_dll\_off\_n signal.
  - It can use any number of isolated byte lanes
- Verified system pin rules:
  - System clock:
    - These pins should be allocated to either SR/MR CC I/O pair.
    - These pins must be allocated in the Memory banks column.
    - If the selected system clock type is single-ended, need to check whether the reference voltage pins are unallocated in the bank or internal VREF is used
  - Reference clock:
    - These pins should be allocated to either SR/MR CC I/O pair.
    - If the selected system clock type is single-ended, need to check whether the reference voltage pins are unallocated in the bank or internal VREF is used.
  - Status signals:
    - The sys\_rst signal should be allocated in the bank where the VREF I/O is unallocated or internal VREF is used.
    - These signals should be allocated in the non-memory banks because the I/O standard is not compatible. The I/O standard type should be LVCMOS with the I/O voltage at 1.8V.
    - These signals can be allocated in any of the columns (there is no hard requirement because these signals should reside in a memory column); however, it is better to allocate closer to the chosen memory banks.

## Core Architecture

### Overview

Figure 2-23 shows a high-level block diagram of the 7 series 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.

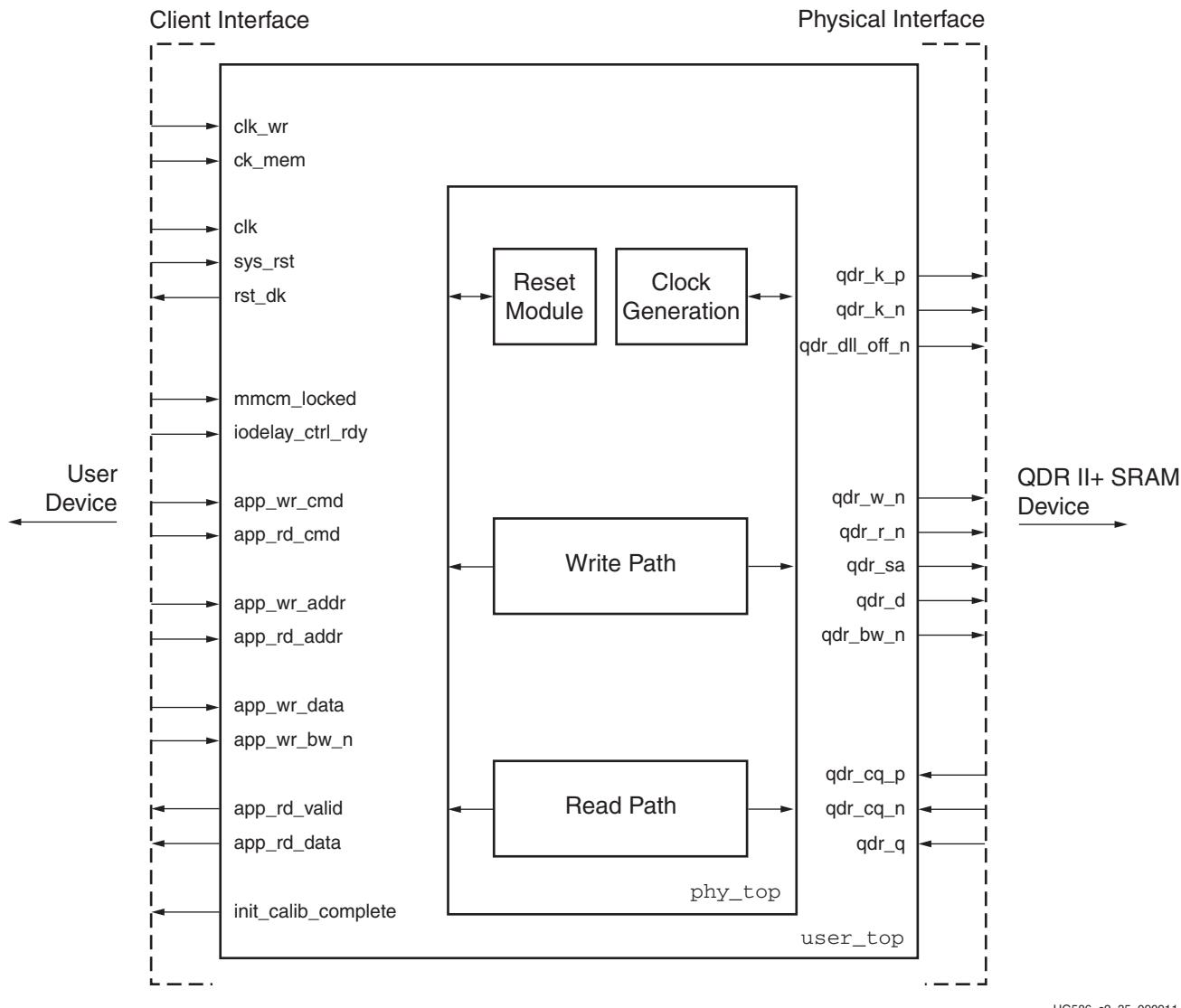


UG586\_c2\_34\_090911

Figure 2-23: High-Level Block Diagram of QDRII+ Interface Solution

The PHY is composed of these elements, as shown in [Figure 2-24](#):

- User interface
- Physical interface
  - a. Write path
  - b. Read datapath



UG586\_c2\_35\_090911

**Figure 2-24: 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. See [User Interface](#) for more details about this protocol. The physical interface generating the proper timing relationships and DDR signaling to communicate with the external memory device, while conforming to QDR II+ protocol and timing requirements. See [Physical Interface](#) for more details.

Within the PHY, logic is broken up into read and write paths. The write path generates the QDR II+ signaling for generating read and write requests. This includes 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. See [Calibration](#) for more details about this process.

## User Interface

The client interface connects the 7 series 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 2-10](#). To accommodate for burst length 4 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.

**Table 2-10: Client Interface Request Signals**

Signal	Direction	Description
init_calib_complete	Output	<b>Calibration Done.</b> 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.
app_rd_addr0[ADDR_WIDTH – 1:0]	Input	<b>Read Address.</b> This bus provides the address to use for a read request. It is valid when app_rd_cmd0 is asserted.
app_rd_cmd0	Input	<b>Read Command.</b> This signal is used to issue a read request and indicates that the address on port 0 is valid.
app_rd_data0[DATA_WIDTH × BURST_LEN – 1:0]	Output	<b>Read Data.</b> This bus carries the data read back from the read command issued on app_rd_cmd0.
app_rd_valid0	Output	<b>Read Valid.</b> This signal indicates that data read back from memory is now available on app_rd_data0 and should be sampled.
app_rd_addr1[ADDR_WIDTH – 1:0]	Input	<b>Read Address.</b> This bus provides the address to use for a read request. It is valid when app_rd_cmd1 is asserted.
app_rd_cmd1	Input	<b>Read Command.</b> This signal is used to issue a read request and indicates that the address on port 1 is valid.
app_rd_data1[DATA_WIDTH × 2 – 1:0]	Output	<b>Read Data.</b> This bus carries the data read back from the read command issued on app_rd_cmd1.
app_rd_valid1	Output	<b>Read Valid.</b> This signal indicates that data read back from memory is now available on app_rd_data1 and should be sampled.
app_wr_addr0[ADDR_WIDTH – 1:0]	Input	<b>Write Address.</b> This bus provides the address for a write request. It is valid when app_wr_cmd0 is asserted.

Table 2-10: Client Interface Request Signals (Cont'd)

Signal	Direction	Description
app_wr_bw_n0[BW_WIDTH × BURST_LEN – 1:0]	Input	<b>Write Byte Writes.</b> This bus provides the byte writes to use for a write request. It is valid when app_wr_cmd0 is asserted. These enables are active Low.
app_wr_cmd0	Input	<b>Write Command.</b> This signal is used to issue a write request and indicates that the corresponding sideband signals on write port 0 are valid.
app_wr_data0[DATA_WIDTH × BURST_LEN – 1:0]	Input	<b>Write Data.</b> This bus provides the data to use for a write request. It is valid when app_wr_cmd0 is asserted.
app_wr_addr1[ADDR_WIDTH – 1:0]	Input	<b>Write Address.</b> This bus provides the address for a write request. It is valid when app_wr_cmd1 is asserted.
app_wr_bw_n1[BW_WIDTH × 2 – 1:0]	Input	<b>Write Byte Writes.</b> This bus provides the byte writes to use for a write request. It is valid when app_wr_cmd1 is asserted. These enables are active Low.
app_wr_cmd1	Input	<b>Write Command.</b> This signal is used to issue a write request and indicates that the corresponding sideband signals on write port 1 are valid.
app_wr_data1[DATA_WIDTH × 2 – 1:0]	Input	<b>Write Data.</b> This bus provides the data to use for a write request. It is valid when app_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 2-25](#).

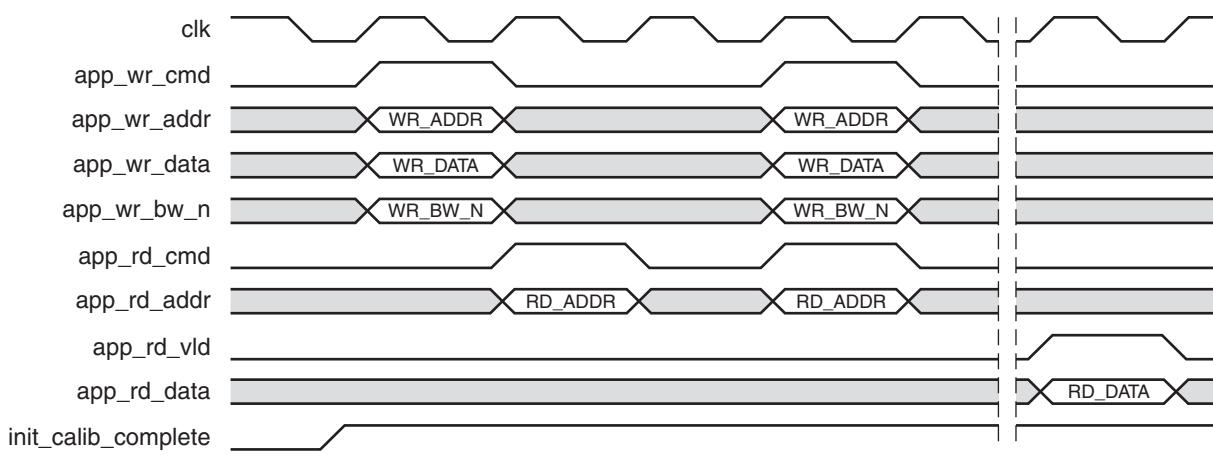


Figure 2-25: Components of the QDRII+ SRAM Memory Interface Solution

Before any requests can be made, the init\_calib\_complete signal must be asserted High, as shown in [Figure 2-25](#), no read or write requests can take place, and the assertion of

app\_wr\_cmd or app\_rd\_cmd on the client interface is ignored. A write request is issued by asserting app\_wr\_cmd as a single cycle pulse. At this time, the app\_wr\_addr, app\_wr\_data, and app\_wr\_bw\_n signals must be valid. On the following cycle, a read request is issued by asserting app\_rd\_cmd for a single cycle pulse. At this time, app\_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 2-25 also shows data returning from the memory device to the user design. The app\_rd\_vld signal is asserted, indicating that app\_rd\_data is now valid. This should be sampled on the same cycle that app\_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 2-25 and is solely to demonstrate protocol.

## Clocking Architecture

The PHY design requires that a PLL module be used to generate various clocks. 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) logic
- Read path (input) and delay logic
- IDELAY reference clock (200 MHz)

One PLL is required for the PHY. The PLL is used to generate the clocks for most of the internal logic, the input clocks to the phasers, and a synchronization pulse required to keep the PHASER blocks synchronized in a multi-I/O bank implementation.

The PHASER blocks require three clocks, a memory reference clock, a frequency reference clock and a phase reference clock from the PLL. The memory reference clock is required to be at the same frequency as that of the QDRII+ memory interface clock. The frequency reference clock must be either 2x or 4x the memory clock frequency such that it meets the frequency range requirement of 400 MHz to 1066 MHz. The phase reference clock is used in the read banks, and is generated using the memory read clock (CQ/CQ#) routed internally and provided to the Phaser logic to assist with data capture.

The internal fabric clock generated by the PLL is clocked by a global clocking resource at half the frequency of the QDII+ memory frequency.

A 200 MHz IDELAY reference clock must be supplied to the IDELAYCTRL module. The IDELAYCTRL module continuously calibrates the IDELAY elements in the I/O region to account for varying environmental conditions. The IP 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 module. This ensures that the clock is stable before being used.

Table 2-11 lists the signals used in the infrastructure module that provides the necessary clocks and reset signals required in the design.

**Table 2-11: Infrastructure Clocking and Reset Signals**

<b>Signal</b>	<b>Direction</b>	<b>Description</b>
mmcm_clk	Input	System clock input.
sys_rst	Input	Core reset from user application.
iodelay_ctrl_rdy	Input	IDELAYCTRL lock status.
clk	Output	Half frequency fabric clock.
mem_refclk	Output	PLL output clock at same frequency as the memory clock.
freq_refclk	Output	PLL output clock to provide the FREQREFCLK input to the Phaser. The freq_refclk is generated such that its frequency in the range of 400 MHz - 1066 MHz.
sync_pulse	Output	PLL output generated at 1/16 of mem_Refclk and is a synchronization signal sent to the PHY hard blocks that are used in a multi-bank implementation.
pll_locked	Output	Locked output from PLLE2_ADV.
rstdiv0	Output	Reset output synchronized to internal fabric half frequency clock.

## 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 2-12](#). These signals can be directly connected to the corresponding signals on the QDRII+ SRAM device.

**Table 2-12: Physical Interface Signals**

<b>Signal</b>	<b>Direction</b>	<b>Description</b>
qdr_cq_n	Input	<b>QDR CQ#.</b> This is the echo clock returned from the memory derived from qdr_k_n.
qdr_cq_p	Input	<b>QDR CQ.</b> This is the echo clock returned from the memory derived from qdr_k_p.
qdr_d	Output	<b>QDR Data.</b> This is the write data from the PHY to the QDR II+ memory device.
qdr_dll_off_n	Output	<b>QDR DLL Off.</b> This signal turns off the DLL in the memory device.
qdr_bw_n	Output	<b>QDR Byte Write.</b> This is the byte write signal from the PHY to the QDRII+ SRAM device.
qdr_k_n	Output	<b>QDR Clock K#.</b> This is the inverted input clock to the memory device.
qdr_k_p	Output	<b>QDR Clock K.</b> This is the input clock to the memory device.
qdr_q	Input	<b>QDR Data Q.</b> This is the data returned from reads to memory.

Table 2-12: Physical Interface Signals (Cont'd)

Signal	Direction	Description
qdr_qvld	Input	<b>QDR Q Valid.</b> This signal indicates that the data on qdr_q is valid. It is only present in QDRII+ SRAM devices.
qdr_sa	Output	<b>QDR Address.</b> This is the address supplied for memory operations.
qdr_w_n	Output	<b>QDR Write.</b> This is the write command to memory.
qdr_r_n	Output	<b>QDR Read.</b> This is the read command to memory.

### Interfacing with the Memory Device

Figure 2-26 shows the physical interface protocol for a four-word memory device.

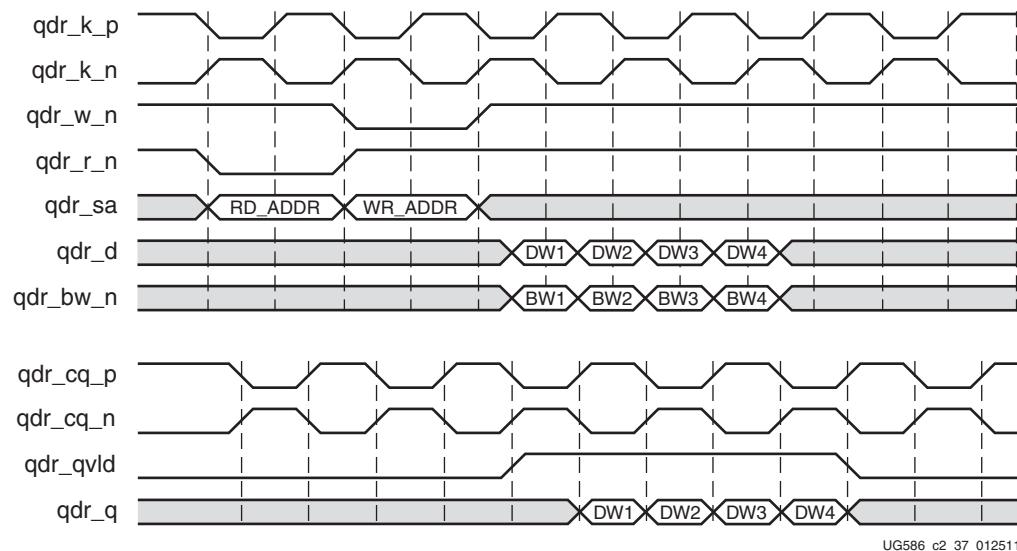


Figure 2-26: Four-Word Burst Length Memory Device Protocol

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

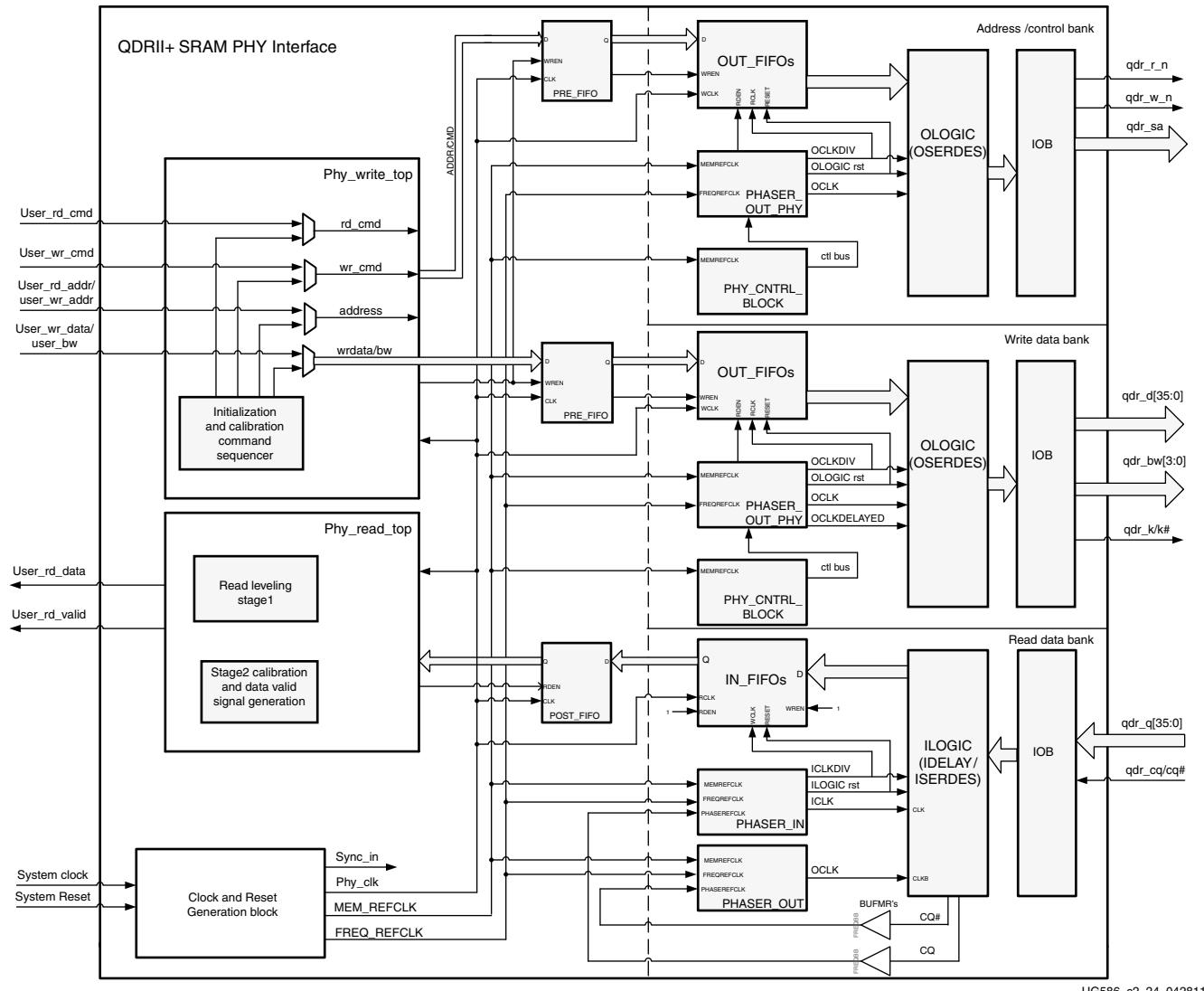
## PHY Architecture

The 7 series FPGA PHY is composed of dedicated blocks and soft calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high-performance physical layers.

Some of the dedicated blocks that are used in the QDRII+ SRAM PHY and their features are described below:

- I/Os available within each 7 series bank are grouped into four byte groups, where each byte group consists of up to 12 I/Os.
- PHASER\_IN/PHASER\_OUT blocks are available in each byte group and are multi-stage programmable delay line loops that can provide precision phase adjustment of the clocks. Dedicated clock within an I/O bank referred to as byte group clocks generated by the PHASERs help minimize the number of loads driven by the byte group clock drivers.
- OUT\_FIFO and IN\_FIFO are shallow 8-deep FIFOs available in each byte group and serve to transfer data from the fabric domain to the I/O clock domain. OUT\_FIFOs are used to store output data and address/controls that need to sent to the memory while IN\_FIFOs are used to store captured read data before transfer to the FPGA fabric.

The [Pinout Requirements](#) section explains the rules that need to be followed while placing the memory interface signals inside the byte groups.



UG586\_c2\_24\_042811

Figure 2-27: High-Level PHY Block Diagram for a 36-Bit QDRII+ Interface

## 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 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 2-28](#) shows a high-level block diagram of the write path and its submodules.

## Output Architecture

The output path of the QDRII+ interface solution uses OUT\_FIFOs, PHASER\_OUT\_PHY, and OSERDES primitives available in the 7 series FPGAs. These blocks are used for clocking all outputs of the PHY to the memory device.

The PHASER\_OUT provides the clocks required to clock out the outputs to the memory. It provides synchronized clocks for each byte group, to the OUT\_FIFOs and to the

OSERDES/ODDR. The PHASER\_OUT generates the byte clock (OCLK), the divided byte clock (OCLKDIV), and a delayed byte clock (OCLK\_DELAYED) for its associated byte group. The byte clock (OCLK) is the same frequency as the memory interface clock and the divided byte clock (OCLKDIV) is half the frequency of the memory interface clock. The byte clock (OCLK) is used to clock the Write data (D) and Byte write (BW) signals to the memory from the OSERDES. The PHASER\_OUT output, OCLK\_DELAYED, is a 90 degree phase shifted output with respect to the byte clock (OCLK) and is used to generate the write clock (K/K#) to the memory.

The OUT\_FIFOs serve as a temporary buffer to convert the write data from the fabric domain to the PHASER clock domain, which clocks out the output data from the I/O logic. The fabric logic writes into the OUT\_FIFOs in the fabric half-frequency clock based on the FULL flag output from the OUT\_FIFO. The clocks required for operating the OUT\_FIFOs and OSERDES are provided by the PHASER\_OUT.

The clocking details of the write paths using PHASER\_OUT are shown in [Figure 2-28](#).

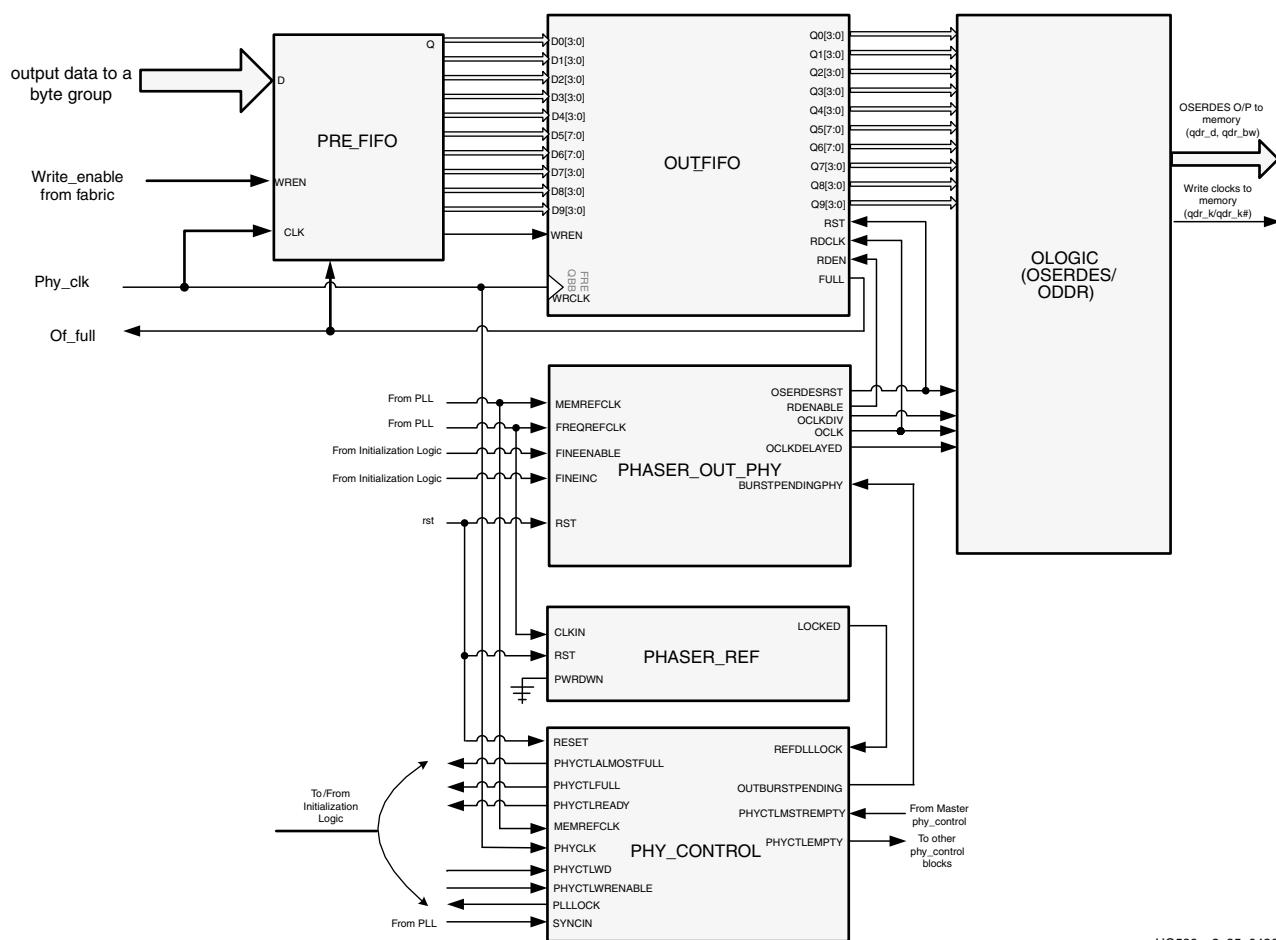
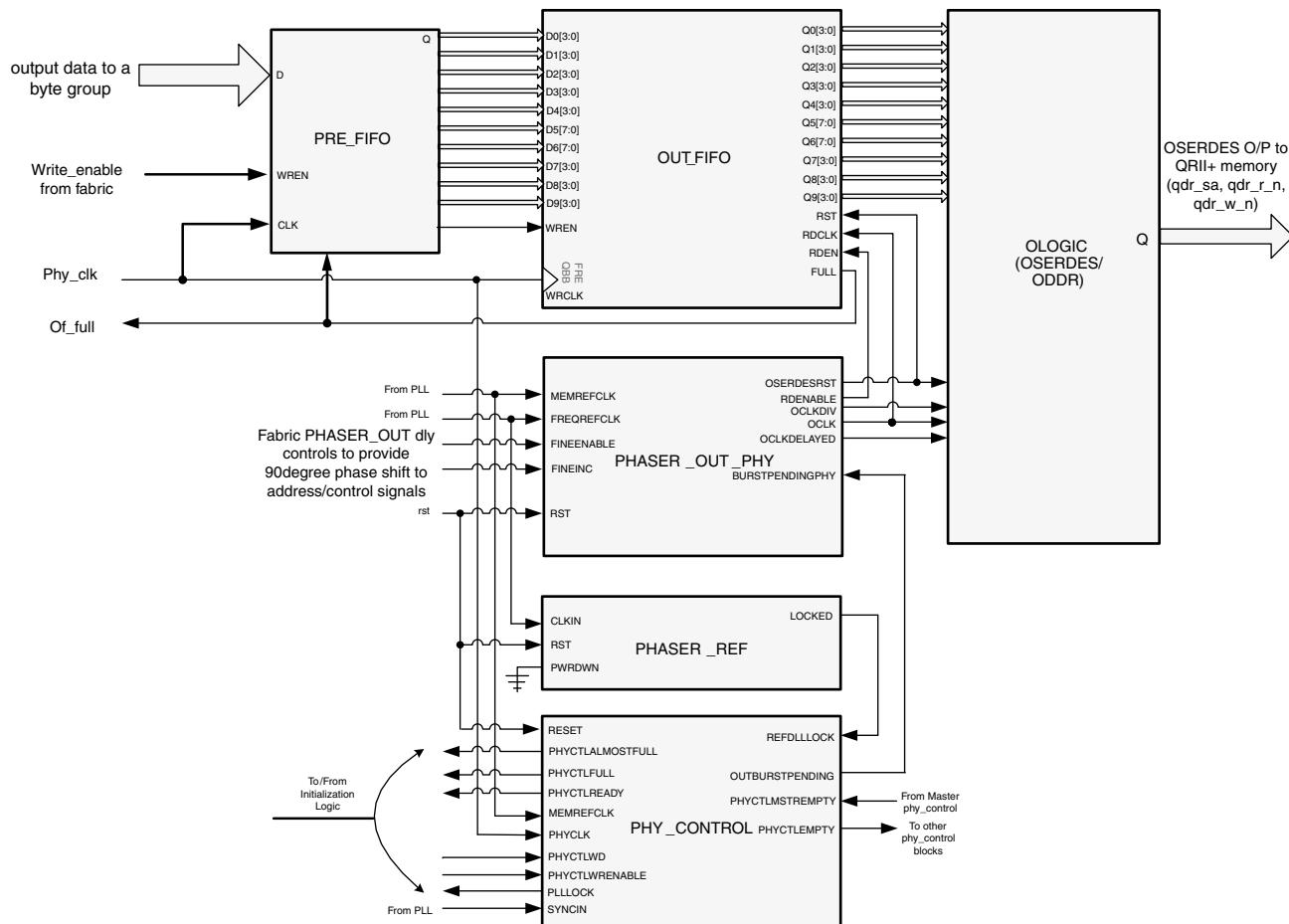


Figure 2-28: Write Path

UG586\_c2\_25\_042611

The clocking details of the address/control using PHASER\_OUT are shown in Figure 2-29.



UG586\_c2\_26\_042611

Figure 2-29: Address Path

## Output Path

Because the address/command and write data are provided by the user backend, the QDR PHY transfers the signals from the fabric domain to their internal PHASER clock domain and provides them from the OSERDES to the memory. The OUT\_FIFOS are used mainly as domain transfer elements in the design, and therefore the write and read enables of the OUT\_FIFO need to be constantly enabled. The PHY Control block helps with this requirement.

### PHY Control Block

The QDR PHY uses the PHY Control block to interface to the OUT\_FIFOS and PHASER\_OUT\_PHY. The PHY Control block helps to prevent the condition where one or more of the OUT\_FIFOS are operating close to the EMPTY condition of the OUT\_FIFO, which could potentially make the OUT\_FIFO go EMPTY (based on how the WRCLK and RDCLK are aligned at the OUT\_FIFO over voltage-temperature variations) thereby causing the OUT\_FIFO to stall. The PHY Control block helps the OUT\_FIFO to operate closer to the FULL condition of the OUT\_FIFO.

The steps required for the initialization are as follows:

- After Phy\_control\_Ready is asserted, PHY\_CONTROL is programmed with a *large* delay into the pc\_phy\_counters. The Control word format is shown in [Figure 2-29](#) and [Figure 2-30](#).

Bits	35:32	31	30	29:25	24:23	22:17	16:15	14:12	11:8	7:3	2	1	0
Field	AO1	Major OP	Minor OP	Event Delay	Seq	Data Offset	IndexHi (Rank)	IndexLo (Bank)	AO0	Command Offset	Non-Data	Read	Data

Figure 2-29: Control Word Format

MajorOP	MinorOP	EventDelay	IndexHi	IndexLo	Registers
0-REGPRE	0 - REG	Register Data[4:0]	IndexHi[16] = Register Data[5] IndexHi[15] = Register Addr[3]	Register Address Bits [2:0]	4'b0000 - 4'b0011: Reserved 4'b0100: CTLCORR 4'b0101: RRDCNTR 4'b0110: REF2ACT 4'b0111: TFAW 4'b1000: A2ARD 4'b1001: A2AWR 4'b1010: PRE2ACT 4'b1011: ACT2PRE 4'b1100: RDA2ACT 4'b1101: RD2PRE 4'b1110: WRA2ACT 4'b1111: WR2PRE
1 - PRE	1 - PRE	5'b000xx - STALL	DC	DC	The STALL operation delays the issue of the Ready signal from pc_phy_counters to the sequencing state machines.
		5'b010xx - REF	Rank	DC	
		5'b100xx - PREBANK	Rank	Bank	
		5'b110xx - PREALL	Rank	DC	
		All others - NOP	DC	DC	
1-ACTRDWR	ACT	29:28: ACT Slot 27: AP 26:25: RDWR Slot	Rank	Bank	

Figure 2-29: Control Word Decode

The delay counter is used to delay the PHY Control block from fetching the next command from the PHY Control Word FIFO, and allows time for it to be filled to capacity. This FIFO needs to be prevented from going empty, because that stalls the PHY\_CONTROL, and in turn leads to gaps in the read enable assertion for the OUT\_FIFOs, which should be avoided.

The OUT\_FIFO is used in ASYNC\_MODE and in the 4x4 mode.

The PHY control word has these assignments:

- Control word [31:30] is set to 01.
- Control word [29:25] is set to 5'b11111, which is the large delay programmed into the pc\_phy\_Counters.
- A non-data command is issued by asserting control word[2].
- Command and data offset are set to 0.
- Phy\_ctl\_wr is set to 1 as long as the PHY Control Word FIFO (phy\_ctl\_fifo) is not FULL.

2. Entries are written into the OUT\_FIFO (for command/address, and for write data); these entries are NOPs until the FULL condition is reached.
3. After the FULL flag goes High with the ninth write, all writes to the FIFO are stopped until the FULL flag is deasserted (see [step 4](#)).
4. Eventually, the PHY\_CONTROL asserts RDENABLE for the OUT\_FIFO (after the *large* delay has expired)
5. After reads begin, the FULL flag is deasserted.
6. Two clock cycles after FULL deassertion, begin writing again to the OUT\_FIFO. Continue to provide Data commands to the PHY Control block: Control word[2:0] is set to 001.
7. At this point, both WRENABLE and RDENABLE are constantly asserted.

## Pre-FIFO

When the OUT\_FIFO is close to the ALMOST\_FULL condition, with VT variations, it is likely that the OUT\_FIFO(s) could momentarily be FULL, based on the wr/rd clock phase alignment. A low-latency pre-FIFO is used to store the command requests/write data from the user and to help store the signals when the OUT\_FIFO indeed goes FULL.

The OSERDES blocks available in every I/O helps to 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. The OSERDES is used to clock all outputs from the PHY to the memory device. 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 PHASER\_OUT block is also used in conjunction with the OSERDES to achieve center alignment. The output clocks that drive the address, and controls are shifted such that the output signals are center aligned to the K/K# clocks at the memory.

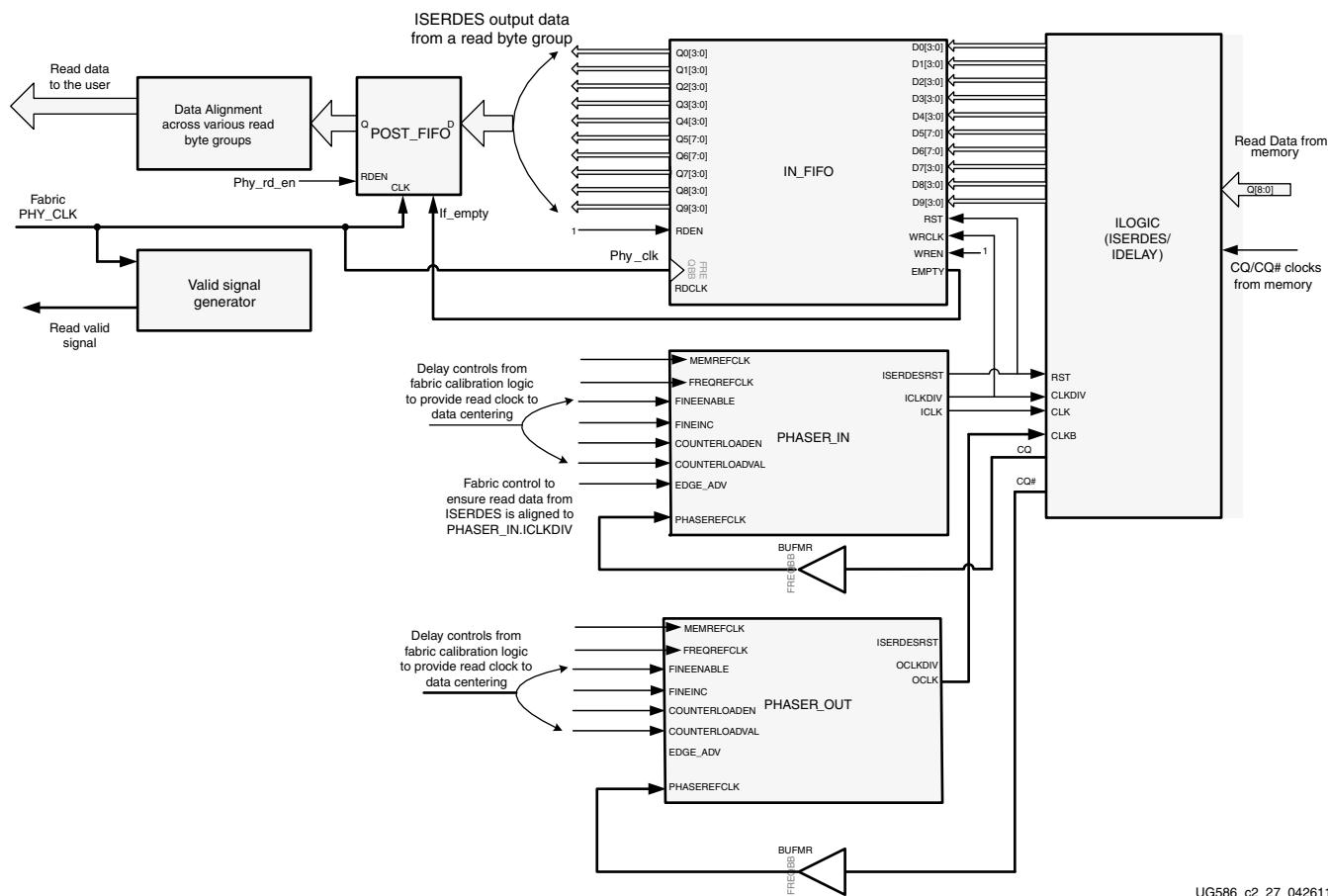
## Read Path

The read path includes data capture using the memory provided read clocks and also ensuring that the read clock is centered within the data window to ensure that good margin is available during data capture. Before any read can take place, calibration must occur. Calibration is the main function of the read path and needs to be performed before the user interface can start transactions to the memory.

## Data Capture

[Figure 2-30](#) shows a high-level block diagram of the path the read clock and the read data take from entering the FPGA until given to the user. The read clock bypasses the ILOGIC and is routed through PHASERs within each byte group through multi-region BUFMRs. The BUFMR output can drive the PHASERREFCLK inputs of PHASERs in the immediate bank and also the PHASERs available in the bank above and below the current bank. The PHASER generated byte group clocks (ICLK, OCLK, and ICLKDIV) are then used to capture the read data (Q) available within the byte group using the ISERDES block. The calibration logic makes use of the fine delay increments available through the PHASER to ensure the byte group clocks are centered inside the read data window, ensuring maximum data capture margin.

IN\_FIFOs available in each byte group shown in [Figure 2-30](#) receive 4-bit data from each Q bit captured in the ISERDES in a given byte group and writes them into the storage array. The half-frequency PHASER\_IN generated byte group clock, ICLKDIV, that captures the data in the ISERDES is also used to write the captured read data to the IN\_FIFO. The write enables to the IN\_FIFO are always asserted to enable data to be written in continuously. A shallow, synchronous post\_fifo is used at the receiving side of the IN\_FIFO to enable captured data to be read out continuously from the FPGA logic, should a flag assertion occur in the IN\_FIFO, which could potentially stall the flow of data from the IN\_FIFO. Calibration also ensures that the read data is aligned to the rising edge of the fabric half-frequency clock and that read data from all the byte groups have the same delay. More details about the actual calibration and alignment logic is explained in [Calibration](#).



*Figure 2-30: Read Datapath*

## Calibration

The calibration logic includes providing the required amount of delay on the read clock and read data to align the clock in the center of the data valid window. The centering of the clock is done using PHASERs which provide very fine resolution delay taps on the clock. Each PHASER\_IN fine delay tap increments the clock by 1/64<sup>th</sup> of the data period.

Calibration begins after the echo clocks 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 read clock with respect to Q
2. Data alignment and valid generation

## Calibration of Read Clock and Data

The PHASER\_IN/PHASER\_OUT clocks within each byte group are used to clock all ISERDES used to capture read data (Q) associated with the corresponding byte group. ICLKDIV is also the write clock for the read data IN\_FIFOs. One PHASER\_IN block is associated with a group of 12 I/Os. Each I/O bank in the 7 series FPGA has four PHASER\_IN blocks, and hence four read data bytes can be placed in a bank.

### Implementation Details

This stage of read leveling is performed one byte at a time where the read clock is center aligned to the corresponding read data in that byte group. At the start of this stage, a write command is issued to a specified QDRII+ SRAM address location with a specific data pattern. This write command is followed by back-to-back read commands to continuously read data back from the same address location that was written to.

The calibration logic reads data out of the IN\_FIFO and records it for comparison. The calibration logic checks for the sequence of the data pattern read to determine the alignment of the clock with respect to the data. No assumption is made about the initial relationship between the capture clock and the data window at tap 0 of the fine delay line. The algorithm tries to align the rise and fall clocks to the left edge of their corresponding data window, by delaying the read data through the IDELAY element.

Next, the clocks are then delayed using the PHASER taps and centered within the corresponding data window. The PHASER\_TAP resolution is based on the FREQ\_REF\_CLK period and the per-tap resolution is equal to  $(\text{FREQ\_REFCLK\_PERIOD}/2)/64$  ps. For memory interface frequencies greater than or equal to 400 MHz, using the maximum of 64 PHASER taps can provide a delay of 1 data period or 1/2 the clock period. This enables the calibration logic to accurately center the clock within the data window.

For frequencies less than 400 MHz, because FREQ\_REF\_CLK has twice the frequency of the MEM\_REF\_CLK, the maximum delay that can be derived from the PHASER is 1/2 the data period or 1/4 the clock period. Hence for frequencies less than 400 MHz, just using the PHASER delay taps might not be sufficient to accurately center the clock in the data window. So for these frequency ranges, a combination of both data delay using IDELAY taps and PHASER taps is used. The calibration logic determines the best possible delays, based on the initial clock-data alignment.

An averaging algorithm is used for data window detection where data is read back over multiple cycles at the same tap value. The number of sampling cycles is set to 214. In addition to averaging, there is also a counter to track whether the read capture clock is positioned in the unstable jitter region. A counter value of 3 means that the sampled data value was constant for three consecutive tap increments and the read capture clock is considered to be in a stable region. The counter value is reset to 0 whenever a value different from the previous value is detected. The next step is to increment the fine phase shift delay line of the PHASER\_IN and PHASER\_OUT blocks one tap at a time until a data mismatch is detected. The data read out of IN\_FIFO after the required settling time is then compared with the recorded data at the previous tap value. This is repeated until a data mismatch is found, indicating the detection of a valid data window edge. A valid window is the number of PHASER fine phase shift taps for which the stable counter value is a

constant 3. This algorithm mitigates the risk of detecting a false valid edge in the unstable jitter regions.

## Data Alignment and Valid Generation

This phase of calibration:

- Ensures read data from all the read byte groups are aligned to the rising edge of the ISERDES CLKDIV capture clock
- 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 read valid generation logic.

After the read data capture clock centering is achieved, the calibration logic writes out a known data pattern to the QDRII+ memory and issues continuous reads back from the memory. This is done to determine whether the read data comes back aligned to the positive edge or negative edge of the ICLKDIV output of the PHASER\_IN. If the captured data from a byte group is found aligned to the negative edge, this is then made to align to the positive edge by using the EDGE\_ADV input to the PHASER\_IN, which shifts the ICLKDIV output by one fast clock cycle.

The next stage is to generate the valid signal associated with the data on the client interface. During this stage of calibration, a single write of a known data pattern 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.

## Customizing the Core

The 7 series FPGAs 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: 7 Series FPGAs QDRII+ SRAM Memory Interface Solution Configurable Parameters**

Parameter	Value	Description
MEM_TYPE	QDR2PLUS	This is the memory address bus width
CLK_PERIOD		This is the memory clock period (ps).
BURST_LEN	4	This is the memory data burst length.
DATA_WIDTH		This is the memory data bus width and can be set through the MIG tool. A maximum DATA_WIDTH of 36 is supported.

**Table 2-13: 7 Series FPGAs QDRII+ SRAM Memory Interface Solution Configurable Parameters (Cont'd)**

Parameter	Value	Description
BW_WIDTH		This must be set to DATA_WIDTH/9
NUM_DEVICES		This is the number of memory devices used.
MEM_RD_LATENCY	2.0 2.5	This specifies the number of memory clock cycles of read latency of the memory device used. This is derived from the memory vendor data sheet.
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. Only a value of 0 is supported, which provides the minimum possible latency is used.
CPT_CLK_CQ_ONLY	TRUE	This indicates only one of the read clocks provided by the memory (rise clock) is used for the data capture.
PHY_LATENCY		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.
CLK_STABLE	(see memory vendor data sheet)	This is the number of cycles to wait until the echo clocks are stable.
IODELAY_GRP		This is a unique name for the IODELAY_CTRL that is provided when multiple IP cores are used in the design.
REFCLK_FREQ	200.0 300.0	This is the reference clock frequency for IODELAYCTRLs.
RST_ACT_LOW	0,1	This is the active Low or active High reset.
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.
SYSCLK_TYPE	DIFFERENTIAL SINGLE_ENDED	This parameter indicates whether the system uses single-ended or differential system 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_i must be used.

**Table 2-13: 7 Series FPGAs QDRII+ SRAM Memory Interface Solution Configurable Parameters (Cont'd)**

Parameter	Value	Description
REFCLK_TYPE	DIFFERENTIAL SINGLE_ENDED	This parameter indicates whether the system uses single-ended or differential reference clocks. Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, ref_clk_p/ref_clk_n must be used. For single-ended clocks, ref_clk_i must be used.
DIFF_TERM	TRUE FALSE	This parameter indicates whether differential or non-differential termination is required for the system clock inputs.
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.
SIM_BYPASS_INIT_CAL	SKIP FAST OFF	This simulation only parameter is used to speed up simulations.
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 through 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 simulation.

Table 2-14 contains parameters set up by the MIG tool based on the pinout selected. When making pinout changes, it is recommended to rerun the MIG tool to set up the parameters properly. Refer to [Pinout Requirements, page 202](#). Mistakes to the pinout parameters can result in non-functional simulation, an unrouteable design, and/or trouble meeting timing. These parameters are used to set up the PHY and route all the necessary signals to and from it. The parameters are calculated based on Data Write, Data Read, and Address/Control byte groups selected. These parameters do not consider the System Signals selection (that is, system clock, reference clock, and status signals).

Table 2-14: QDRII+ SRAM Memory Interface Solution Pinout Parameters

Parameter	Description	Example
BYTE_LANES_B0, BYTE_LANES_B1, BYTE_LANES_B2	Defines the byte lanes being used in a given I/O bank. A "1" in a bit position indicates a byte lane is used, and a "0" indicates unused.	Ordering of bits from MSB to LSB is T0, T1, T2, and T3 byte groups. 4'b1101: For a given bank, three byte lanes are used, and one byte lane is not used.
DATA_CTL_B0, DATA_CTL_B1, DATA_CTL_B2	Defines mode of use of byte lanes in a given I/O bank. A "1" in a bit position indicates a byte lane is used for data, and a "0" indicates it is used for address/control.	4'b1100: With respect to the BYTE_LANE example, two byte lanes are used for Data and one for Address/Control.
PHY_0_BITLANES, PHY_1_BITLANES, PHY_2_BITLANES	12-bit parameter per byte lane used to determine which I/O locations are used to generate the necessary PHY structures. This parameter is provided per bank. Except for the QVLD and DLL_OFF_N pins, all Data Write, Data Read, and Address/Control pins are considered for this parameter generation.	This parameter denotes for all byte groups of a selected bank. All 12 bits are denoted for a byte lane. For example, this parameter is 48'hFFE_FFF_000_2FF for one bank. 12'b1101_1111_0110: Bit lines 0, 3, and 9 are not used; the rest of the bits are used.
BYTE_GROUP_TYPE_B0, BYTE_GROUP_TYPE_B1, BYTE_GROUP_TYPE_B2	Defines the byte lanes for a given I/O bank as INPUT or OUTPUT. A "1" in a bit position indicates a byte lane contains INPUT pins, and a "0" indicates byte lane contains OUTPUT pins.	4'b0110: Middle two byte lanes contain INPUT pins, and the other byte lanes contain OUTPUT pins.
K_MAP	Bank and byte lane position information for write clocks (K/K#). 8-bit parameter provided per pair of signals. <ul style="list-style-type: none"> <li>• [7:4] - Bank position. Values of 0, 1, or 2 are supported</li> <li>• [3:0] - Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> </ul>	Upper-most Data Write/Data Read or Address/Control byte group selected bank is referred as Bank 0 in parameters notation. Numbering of banks is 0, 1, and 2 from top to bottom. Byte groups T0, T1, T2, and T3 are numbered in parameters as 3, 2, 1, and 0, respectively. 48'h00_00_00_00_03_13: This parameter is denoted for 6 write clock pairs with 8 bits for each clock pin. In this case, only two write clock pairs are used. Ordering of parameters is from MSB to LSB (that is, K[0]/ K#[0] corresponds to the 8 LSBs of the parameter). 8'h13: K/K# placed in bank 1, byte lane 3. 8'h20: K/K# placed in bank 2, byte lane 0.
CQ_MAP	Bank and byte lane position information for the read clocks (CQ/CQ#). See the <a href="#">K_MAP</a> description.	See the <a href="#">K_MAP</a> example.

Table 2-14: QDRII+ SRAM Memory Interface Solution Pinout Parameters (Cont'd)

Parameter	Description	Example
ADD_MAP	<p>Bank and byte lane position information for the address. 12-bit parameter provided per pin.</p> <ul style="list-style-type: none"> <li>[11:8] - Bank position. Values of 0, 1, or 2 are supported.</li> <li>[7:4] - Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> <li>[3:0] - Bit position within a byte lane. Values of [0, 1, 2, ..., A, B] are supported.</li> </ul>	<p>Upper-most Data Write/Data Read or Address/Control byte group. The selected bank is referred to as Bank 0 in the parameters notation. Numbering of banks is 0, 1, and 2 from top to bottom.</p> <p>Byte groups T0, T1, T2, and T3 are numbered in parameters as 3, 2, 1, and 0, respectively.</p> <p>Bottom-most pin in a byte group is referred as '0' in MAP parameters.</p> <p>Bottom-most pin in a byte group is referred as '0' in the MAP parameters. Numbering is counted from 0 to 9 from the bottom-most pin to the top pin within a byte group by excluding DQSCCI/Os. DQSCC_N and DQSCC_P pins of a byte group are numbered as A and B, respectively.</p> <p>264'h000_000_239_238_237_236_23B_23A_235_234_233_232_231_230_229_228_227_226_22B_22A_225_224: This parameter is denoted for an Address width of 22 bits with 12 bits for each pin. In this example, the Address width is 20 bits. Ordering of parameters is from MSB to LSB (that is, SA[0] corresponds to the 12 LSBs of the parameter).</p> <p>12'h224: Address pin placed in bank 2, byte lane 2, at location 4.</p> <p>12'h11A: Address pin placed in bank 1, byte lane 1, at location A.</p>
RD_MAP	Bank and byte lane position information for the Read enable. See the <a href="#">ADD_MAP</a> description.	See the <a href="#">ADD_MAP</a> example
WR_MAP	Bank and byte lane position information for the Write enable. See the <a href="#">ADD_MAP</a> description.	See the <a href="#">ADD_MAP</a> example
ADDR_CTL_MAP	Bank and byte lane position information for Address byte groups. Address requires 3 byte groups and this parameters denotes the byte groups in which all 3 Address byte groups are selected. See the <a href="#">K_MAP</a> description.	See the <a href="#">K_MAP</a> example
D0_MAP, D1_MAP, D2_MAP, D3_MAP, D4_MAP, D5_MAP, D6_MAP, D7_MAP	Bank and byte lane position information for the Data Write bus. See the <a href="#">ADD_MAP</a> description.	See the <a href="#">ADD_MAP</a> example

Table 2-14: QDRII+ SRAM Memory Interface Solution Pinout Parameters (Cont'd)

Parameter	Description	Example
BW_MAP	Bank and byte lane position information for the Byte Write. See the <a href="#">ADD_MAP</a> description.	See the <a href="#">ADD_MAP</a> example
Q0_MAP, Q1_MAP, Q2_MAP, Q3_MAP, Q4_MAP, Q5_MAP, Q6_MAP, Q7_MAP	Bank and byte lane position information for the Data Read bus. See the <a href="#">ADD_MAP</a> description.	See the <a href="#">ADD_MAP</a> example

## Design Guidelines

### Design Rules

Memory types, memory parts, and data widths are restricted based on the selected FPGA, FPGA speed grade, and the design frequency. The final frequency ranges are subject to characterization results.

### 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. These internal delays can be found using the Pinout and Area Constraints Editor (PACE) tool. These rules indicate the maximum electrical delays between QDRII+ SRAM signals:

- The maximum electrical delay between any bit in the data bus, D, and its associated K/K# clocks should be  $\pm 15$  ps.
- The maximum electrical delay between any Q and its associated CQ/CQ# should be  $\pm 15$  ps.
- The maximum electrical delay between any address and control signals and the corresponding K/K# should be  $\pm 50$  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

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the QDRII+ physical layer. Xilinx 7 series FPGAs have dedicated logic for each byte group. Four byte groups are available in each 50-pin bank. Each 50-pin bank consists of four byte groups that contain one DQS Clock capable I/O pair and ten associated I/Os. Two pairs of Multi-region Clock-capable I/O (MRCC) pins are available in a bank, and are used for placing the read clocks (CQ and CQ#).

In a typical QDRII+ write bank configuration, 9 of these 10 I/Os are used for the Write data (D) and one is used for the byte write (BW). The write clocks (K/K#) use one of the DQSCCIO pairs inside the write bank. Within a read bank, the read data are placed on 9 of the 10 I/Os, QVLD placed in any of the read data byte groups and the CQ/CQ# clocks placed in the MRCC\_P pins available inside the read bank.

Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, QDRII+ memory interfaces must be arranged in the banks vertically and not horizontally. In addition, the maximum height is three banks.

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, the following rules must be taken into consideration:

- The write data bus (D) of a memory interface should be placed within a single bank. It is required to arrange the write data bus byte wise (9 bits wide) among the FPGA byte groups. All byte write (BW) signals of the interface are required to place in the same bank.
- K/K# clocks must be kept in the same bank as the write data bank. They should be placed on a DQSCCIO pin pair.
- The read data bus (Q) must be arranged byte wise (9 bits wide) among the FPGA byte groups. It is recommended to keep the complete read data bus of a memory component within a single bank.
- The read data clocks (CQ and CQ#) must be placed on the two MRCC\_P or MRCC\_N pins available in the same bank as the read data or an adjacent bank to it. It is recommended to keep the read data and read clocks in the same bank.
- All address/control signals should be placed within a single bank. The address bank should be placed adjacent to the data write (D) bank.
- The dll\_off\_n signal can be placed on any free I/O available in the banks used for the memory interface.
- It is recommended to keep the system clock pins in the data write bank.

## 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 7 series FPGAs. These standards should not be changed. [Table 2-15](#) contains a list of the ports together with the I/O standard used.

*Table 2-15: I/O Standards*

Signal <sup>(1)</sup>	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.

DCI (HR banks) or IN\_TERM (HR banks) is required at the FPGA to meet the specified performance. Designs generated by the MIG tool use the DCI standards for Data Read (Q) and Read Clock (CQ\_P and CQ\_N) in the High-Performance banks. In the High-Range banks, the MIG tool uses the HSTL\_I standard with the internal termination (IN\_TERM) attribute chosen in the GUI.

## Debugging 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-7 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:

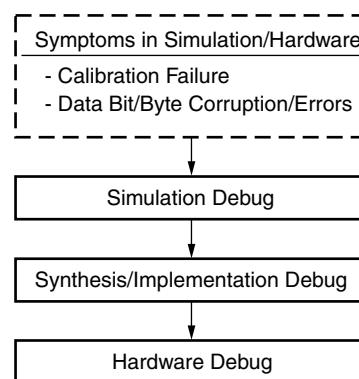
- Functional verification using the UNISIM simulation models
- Design implementation verification
- Board layout verification
- Using the QDRII+ SRAM physical layer to debug board-level issues
- General board-level debug techniques

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 2-31** shows the overall flow for debugging problems associated with these two general types of issues.



**Figure 2-31: Virtex-7 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.

### 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.

Sample debug logic by connecting the debug ports to ChipScope tool cores (that is, ILA, ICON, VIO) is provided in the example design top (example\_top) module with a Debug Signals for Memory Controller option value of “ON.” In User Design top, all debug port signals are grouped under few buses and provided in the port list. To confirm that all ChipScope tool debug ports are connected to various ChipScope tool cores, look at the reference example design top module. The debug ports generated in the User Design top module for Debug Port enable designs are “qdriip\_ila0\_data”, “qdriip\_ila0\_trig”, “qdriip\_ila1\_data”, “qdriip\_ila1\_trig”, “qdriip\_vio2\_async\_in”, and “qdriip\_vio2\_sync\_out.”

### 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 the application and the MIG tool signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool [Ref 8].

## Simulation Debug

Figure 2-32 shows the debug flow for simulation.

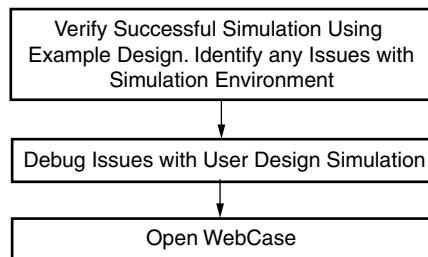


Figure 2-32: Simulation Debug Flowchart

### 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. The user must provide a QDRII+ memory model and add it to the 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 9] and the *Synthesis and Simulation Design Guide* [Ref 10]. For simulator tool support, refer to the *7 Series FPGAs Memory Interface Solutions Data Sheet* [Ref 11].

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 Traffic Generator takes control and begins executing writes and reads according to its parameterization.

**Table 2-16** shows the signals and parameters of interest, respectively, during simulation.

**Table 2-16: Signals of Interest During Simulation**

Signal Name Usage	Signal Name Usage
tg_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; it is held until the design is reset.
tg_cmp_error	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 part of the example design. This signal is asserted each time a data mismatch occurs.
app_wr_cmd	This signal indicates that the write address and write data are valid for a write command
app_wr_addr	This is the address provided for the write command
app_wr_data	This is the write data for a write command
app_wr_bw_n	This signal is the byte write control
app_rd_cmd	This signal indicates that the read address is valid for a read command
app_rd_addr	This address is provided for the read command
app_rd_data	This read data is returned from the memory device
app_rd_valid	This signal is asserted when app_rd_data is valid

### Memory Initialization

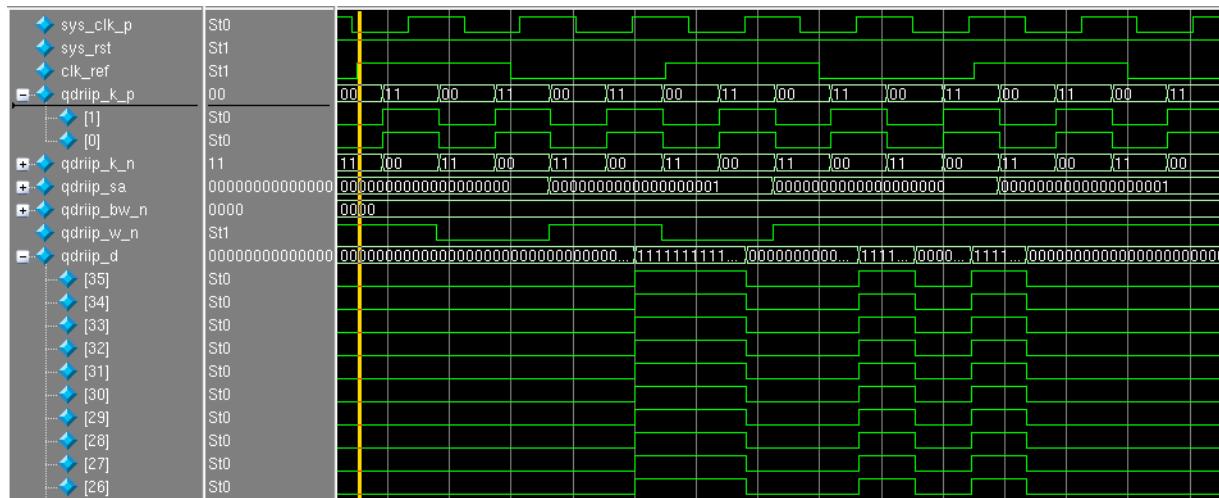
The QDRII+ memories do not require an elaborate initialization procedure. However, the user must ensure that the Doff\_n signal is provided to the memory as required by the vendor. The QDRII+ SRAM interface design provided by the MIG tool drives the Doff\_n signal from the FPGA. After the internal MMCM has locked after a wait period of 200 µs, the Doff\_n signal is asserted High. After Doff\_n is asserted 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

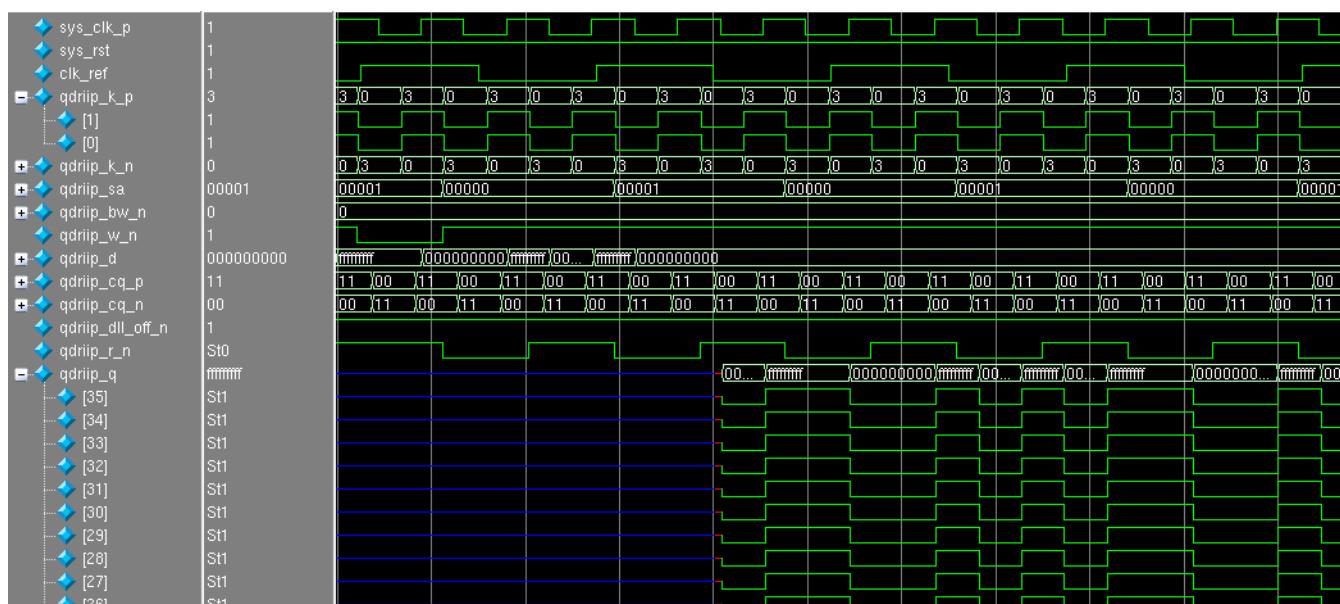
Calibration completes read leveling, write calibration, and read enable calibration. This is completed over two stages. This sequence successfully completes when the cal\_done signal is asserted. For more details, refer to [Physical Interface, page 187](#).

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 2-33](#).



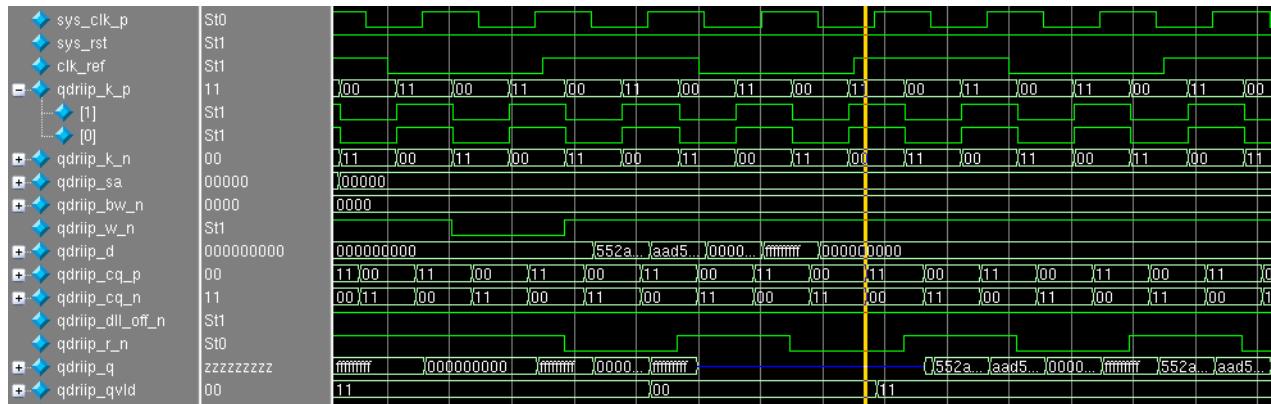
**Figure 2-33: Writes for First Stage Read Calibration**

This pattern is then continuously read back while the per-bit calibration is completed, as shown in [Figure 2-34](#).



**Figure 2-34: Reads for First Stage Read Calibration**

The second stage performs a read enable calibration. The data pattern used during this stage is ..55..AA. The data pattern is first written to the memory, and then read back for the read enable calibration, as shown in [Figure 2-35](#).



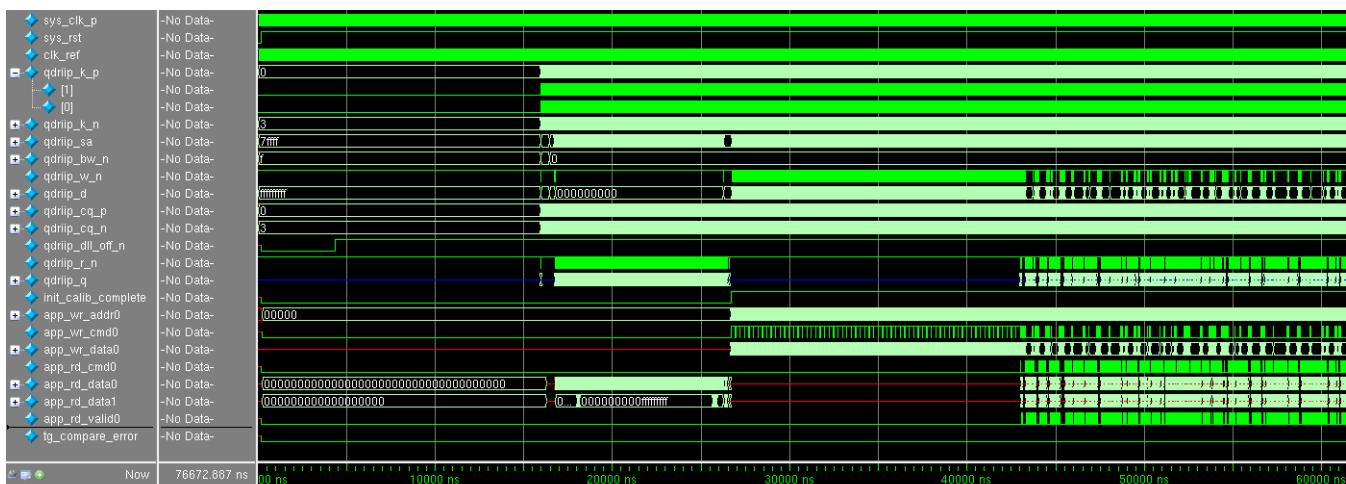
**Figure 2-35: Write and Read for Second Stage Read Calibration**

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 is asserted, signifying successful completion of the calibration process.

### Test Bench

After cal\_done is asserted, 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 2-36](#) shows a successful implementation of the test bench with no assertions on error.



**Figure 2-36: Test Bench Operation After Completion of Calibration**

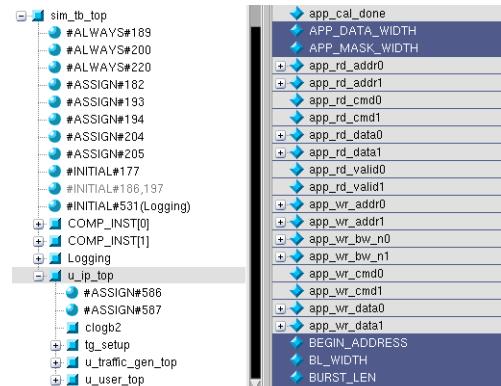
### Proper Write and Read Commands

When sending write and read commands, the user must properly assert and deassert the corresponding UI inputs. Refer to [User Interface, page 184](#) and [Interfacing with the Core](#)

through the Client Interface, page 185 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.

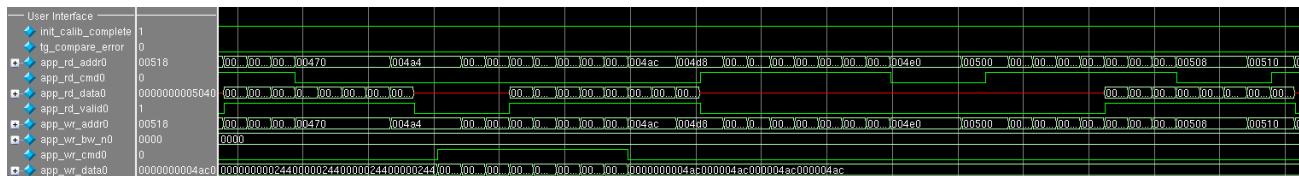
To debug data errors on the QDRII+ SRAM interface, it is necessary to pull the UI signals into the simulation waveform.

In the ModelSim Instance window, highlight **u\_ip\_top** to display the necessary UI signals in the Objects window, as shown in [Figure 2-37](#). Highlight the user interface signals noted in [Table 2-16, page 206](#), right-click, and select **Add > To Wave > Selected Signals**.

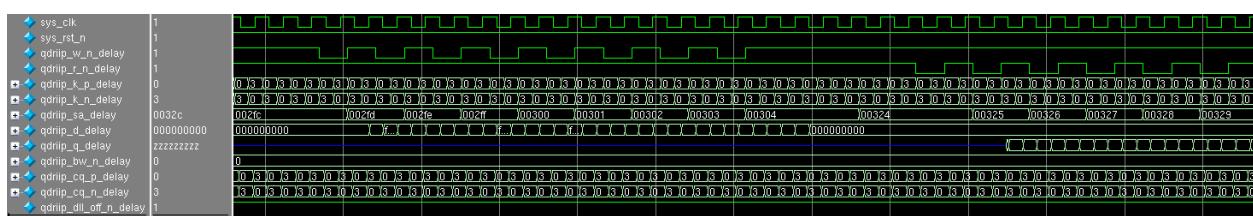


*Figure 2-37: ModelSim Instance Window*

[Figure 2-38](#) and [Figure 2-39](#) show example waveforms of a write and read on both the user interface and the QDRII+ interface.



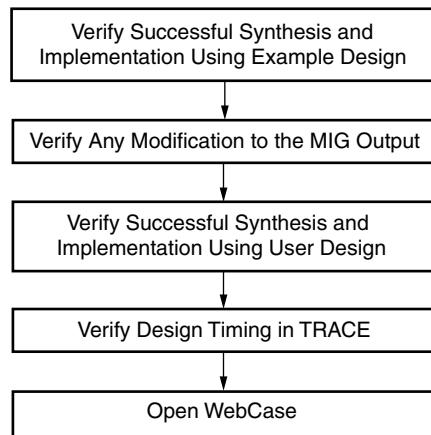
*Figure 2-38: User Interface Write and Read*



*Figure 2-39: QDRII+ Interface Write and Read*

## Synthesis and Implementation Debug

Figure 2-40 shows the debug flow for synthesis and implementation.



**Figure 2-40: 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`/`ise_flow.sh`, 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` / `.tvr`) 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 12] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 13] provides valuable information on all available Xilinx constraints.

## Hardware Debug

Figure 2-41 shows the debug flow for hardware.

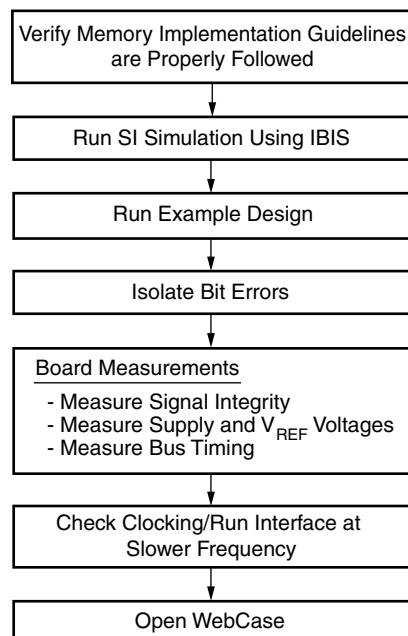


Figure 2-41: Hardware Debug Flowchart

## 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 14] 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 MIG designs with 7 series FPGAs.

## Run the Example Design

The example design provided with the MIG tool 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 8].

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 Problem Arise
dbg_phy_status[0]	rst_wr_clk	Fabric reset based on PLL lock and system input reset	If this signal stays asserted, check your clock source and system reset input
dbg_phy_status[1]	io_fifo_rden_cal_done & po_ck_addr_cmd_delay_done	I/O FIFO initialization to ensure the I/O FIFOs are in an almost full condition and the phaser out delay to provide the 90° phase shift to address/control signals are done	Check if the PHY control ready signal is asserted
dbg_phy_status[2]	init_done	QDRII+ SRAM initialization sequence is complete	N/A <sup>(1)</sup>
dbg_phy_status[3]	cal_stage1_start	Stage 1 read calibration start signal	N/A
dbg_phy_status[4]	edge_adv_cal_done	Stage 1 calibration is complete and edge_adv calibration is complete	Stage 1 calibration did not happen right. Make sure valid read data is seen during stage1 calibration.

Table 2-17: Physical Layer Simple Status Bus Description (Cont'd)

Debug Port Signal	Name	Description	If Problem Arise
dbg_phy_status[5]	cal_stage2_start	Latency calibration start signal after pi_edge_adv 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[6]	cal_stage2_start & cal_done	Latency calibration start signal	N/A
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 read calibration results are provided as part of the Debug port as various output signals. These signals can be used to capture and evaluate the read calibration results.

Read calibration uses the IODELAY to align 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-byte 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 begins with *dbg\_*. The DEBUG\_PORT parameter is always set to OFF in the sim\_tb\_top module of the sim folder, which disables the debug option for functional simulations. These signals and their associated data are described in [Table 2-18](#).

Table 2-18: DEBUG\_PORT Signal Descriptions

Signal	Direction	Description
dbg_phy_wr_cmd_n[1:0]	Output	This active-Low signal is the internal wr_cmd used for debug with the ChipScope analyzer
dbg_phy_rd_cmd_n[1:0]	Output	This active-Low signal is the internal rd_cmd used for debug with the ChipScope analyzer
dbg_phy_addr[ADDR_WIDTH × 4 – 1:0]	Output	Control address bus used for debug with the ChipScope analyzer
dbg_phy_wr_data[DATA_WIDTH × 4 – 1:0]	Output	Data being written that is used for debug with the ChipScope analyzer.
dbg_phy_init_wr_only	Input	When this input is High, the state machine in qdr_phy_write_init_sm stays at the write calibration pattern to QDRII+ memory. This verifies calibration write timing. This signal must be Low for normal operation.

Table 2-18: DEBUG\_PORT Signal Descriptions (Cont'd)

Signal	Direction	Description
dbg_phy_init_rd_only	input	When this input is High, the state machine in qdr_phy_write_init_sm stays at read calibration state from QDRII+ memory. This verifies calibration read timing and returned calibration data. This signal must be Low for normal operation.
dbg_byte_sel	Input	This input selects the corresponding byte lane whose phaser/IDELAY tap controls need to be controlled by the user
dbg_pi_f_inc	Input	This signal increments the phaser_in generated ISERDES clk that is used to capture rising data
dbg_pi_f_dec	Input	This signal decrements the phaser_in generated ISERDES clk that is used to capture rising data
dbg_po_f_inc	Input	This signal increments the phaser_out generated OSERDES clk that is used to capture falling data
dbg_po_f_dec	Input	This signal increments the phaser_out generated OSERDES clk that is used to capture falling data
dbg_phy_pi_fine_cnt	Output	This output indicates the current phaser_in tap count position
dbg_phy_po_fine_cnt	Output	This output indicates the current phaser_out tap count position
dbg_cq_num	Output	This signal indicates the current CQ/CQ# being calibrated
dbg_q_bit	Output	This signal indicates the current Q being calibrated
dbg_valid_lat[4:0]	Output	Latency in cycles of the delayed read command
dbg_q_tapcnt	Output	Current Q tap setting for each device
dbg_inc_latency	Output	This output indicates that the latency of the corresponding byte lane was increased to ensure proper alignment of the read data to the user interface.

Table 2-18: DEBUG\_PORT Signal Descriptions (Cont'd)

Signal	Direction	Description
dbg_error_max_latency	Output	This signal indicates that the latency could not be measured before the counter overflowed. Each device has one error bit.
dbg_error_adj_latency	Output	This signal indicates that the target PHY_LATENCY could not be achieved
dbg_align_rd0 [DATA_WIDTH-1:0]	Output	This bus shows the captured output of the first rising data
dbg_align_rd1 [DATA_WIDTH-1:0]	Output	This bus shows the captured output of the second rising data
dbg_align_fd0 [DATA_WIDTH-1:0]	Output	This bus shows the captured output of the first falling data
dbg_align_fd1 [DATA_WIDTH-1:0]	Output	This bus shows the captured output of the second falling data

### Write Init Debug Signals

Table 2-19 indicates the mapping between the write init debug signals on the dbg\_wr\_init bus and debug signals in the PHY. All signals are found within the qdr\_phy\_write\_init\_sm module and are all valid in the clk domain.

Table 2-19: Write Init Debug Signal Map

Bits	PHY Signal Name	Description
dbg_wr_init[0]	init_cnt_done	Initialization count is done
dbg_wr_init[1]	cq_stable	The cq clocks from memory are stable; commands can be issued
dbg_wr_init[2]	ck_addr_cmd_delay_done	90 degree shift on address/commands is done
dbg_wr_init[3]	rdlvl_stg1_start	Stage 1 calibration is started
dbg_wr_init[4]	rdlvl_stg1_done	Stage 1 calibration is completed
dbg_wr_init[5]	edge_adv_cal_done	Phase alignment has completed, proceed to latency calibration
dbg_wr_init[6]	cal_stage2_start	Latency calculation stage start
dbg_wr_init[14:7]	phy_init_cs	Initialization state machine
dbg_wr_init[15]	rdlvl_stg1_done	First stage centering is done.
dbg_wr_init[18:16]	addr_cntr	Internal counter for command generation
dbg_wr_init[19]	dbg_phy_init_wr_only	When this is '1', the state machine stays at the calibration write state

**Table 2-19: Write Init Debug Signal Map (Cont'd)**

<b>Bits</b>	<b>PHY Signal Name</b>	<b>Description</b>
dbg_wr_init[20]	dbg_phy_init_rd_only	When this is '1', the state machine stays at the calibration read state: CAL1_READ
dbg_wr_init[21]	CAL2_WRITE	The state machines stays at CAL2_WRITE
dbg_wr_init[255:22]	Reserved	N/A

### Read Stage 1 Calibration Debug Signals

**Table 2-20** indicates the mapping between bits within the dbg\_rd\_stage1\_cal bus and debug signals in the PHY. All signals are found within the qdr\_rld\_phy\_rdlv1 module and are all valid in the clk domain.

**Table 2-20: Read Stage 1 Debug Signal Map**

<b>Bits</b>	<b>PHY Signal Name</b>	<b>Description</b>
dbg_phy_rdlvl[0]	rdlvl_stg1_start	Input from initialization state machine indicating start of stage 1 calibration
dbg_phy_rdlvl[1]	rdlvl_start	Indicates when calibration logic begins
dbg_phy_rdlvl[2]	found_edge_r	Indicates a transition of data window was detected
dbg_phy_rdlvl[3]	pat0_data_match_r	Expected data pattern seen at ISERDES outputs
dbg_phy_rdlvl[4]	pat1_data_match_r	Expected data pattern is seen at ISERDES outputs; however, this is shifted due to the first read data aligning to the negative edge of the ICLKDIV
dbg_phy_rdlvl[5]	data_valid	Valid calibration data is seen
dbg_phy_rdlvl[6]	cal1_wait_r	Wait state for observing the data after the IDELAY/phaser taps have been varied
dbg_phy_rdlvl[7]	Reserved	
dbg_phy_rdlvl[8]	detect_edge_done_r	Edge detection completed
dbg_phy_rdlvl[13:9]	cal1_state_r	Calibration state machine states
dbg_phy_rdlvl[20:14]	cnt_idel_dec_cpt_r	Number of phaser taps to be decremented for centering the clock in the data window
dbg_phy_rdlvl[21]	found_first_edge_r	First edge transition detected
dbg_phy_rdlvl[22]	found_second_edge_r	Second edge transition detected
dbg_phy_rdlvl[23]	reserved	

Table 2-20: Read Stage 1 Debug Signal Map (Cont'd)

Bits	PHY Signal Name	Description
dbg_phy_rdlvl[24]	store_sr_r	Signal to store current read data prior to incrementing tap delays
dbg_phy_rdlvl[32:25]	sr_fall1_r, sr_rise1_r sr_fall0_r, sr_rise0_r	Read data stored in shift registers for comparison
dbg_phy_rdlvl[40:33]	old_sr_fall1_r, old_sr_rise1_r old_sr_fall0_r, old_sr_rise0_r	Read data stored in registers prior to tap increments
dbg_phy_rdlvl[41]	sr_valid_r	Determines when it is safe to load the ISERDES data for comparison
dbg_phy_rdlvl[42]	found_stable_eye_r	Indicates a stable eye is seen
dbg_phy_rdlvl[48:43]	tap_cnt_cpt_r	Phaser tap counter
dbg_phy_rdlvl[54:49]	first_edge_taps_r	Number of taps to detect first edge
dbg_phy_rdlvl[60:55]	second_edge_taps_r	Number of taps to detect second edge
dbg_phy_rdlvl[64:61]	cal1_cnt_cpt_r	Indicates the white byte lane is being calibrated
dbg_phy_rdlvl[65]	cal1_dlyce_cpt_r	Phaser control to enable phaser delays
dbg_phy_rdlvl[66]	cal1_dlyinc_cpt_r	Phaser control to increment phaser tap delay
dbg_phy_rdlvl[67]	found_edge_r	Indicates a transition of the data window is detected
dbg_phy_rdlvl[68]	found_stable_eye_last_r	Indicates a stable eye is seen
dbg_phy_rdlvl[74:69]	idelay_taps	Number of IDELAY taps that detect a valid data window while delaying incoming read data
dbg_phy_rdlvl[80:75]	start_win_taps	Number of IDELAY taps to be delayed to detect start of valid window
dbg_phy_rdlvl[81]	idel_tap_limit_cpt_r	Indicates the end of the IDELAY tap chain is reached
dbg_phy_rdlvl[82]	qdly_inc_done_r	Indicates valid window detection by delaying IDELAY taps is done
dbg_phy_rdlvl[83]	start_win_detect	Indicates start of window detection using IDELAY taps
dbg_phy_rdlvl[84]	detect_edge_done_r	Edge detection is completed
dbg_phy_rdlvl[90:85]	idel_tap_cnt_cpt_r	Counter that keeps track of the number of IDELAY taps used

**Table 2-20: Read Stage 1 Debug Signal Map (Cont'd)**

<b>Bits</b>	<b>PHY Signal Name</b>	<b>Description</b>
dbg_phy_rdlvl[96:91]	idelay_inc_taps_r	Number of IDELAY taps to be incremented, if needed by the calibration logic
dbg_phy_rdlvl[102:97]	idel_dec_cntr	Number of IDELAY taps to be decremented, if needed by the calibration logic
dbg_phy_rdlvl[103]	tap_limit_cpt_r	Indicates the end of the phaser tap delay chain is reached
dbg_phy_rdlvl[115:104]	idelay_tap_delay	Total amount of valid window seen using idelay taps
dbg_phy_rdlvl[127:128]	phaser_tap_delay	Total amount of valid window seen using phaser taps
dbg_phy_rdlvl[255:128]	Reserved	

### Read Stage 2 Calibration Debug

Table 2-21 indicates the mapping between bits within the dbg\_rd\_stage2\_cal bus and debug signals in the PHY. All signals are found within the qdr\_rld\_phy\_read\_stage2\_cal module and are all valid in the clk domain.

**Table 2-21: Read Stage 2 Debug Signal Map**

<b>Bits</b>	<b>PHY Signal Name</b>	<b>Description</b>
dbg_stage2_cal[0]	en_mem_latency	Signal to enable latency measurement
dbg_stage2_cal[5:1]	latency_cntr[0]	Indicates the latency for the first byte lane in the interface
dbg_stage2_cal[6]	rd_cmd	Internal rd_cmd for latency calibration
dbg_stage2_cal[7]	latency_measured[0]	Indicates latency has been measured for byte lane 0
dbg_stage2_cal[8]	bl4_rd_cmd_int	Indicates calibrating for burst length of 4 data words
dbg_stage2_cal[9]	bl4_rd_cmd_int_r	Internal register stage for burst 4 read command
dbg_stage2_cal[10]	edge_adv_cal_start	Indicates start of edge_adv calibration, to see if the pi_edge_adv signal needs to be asserted
dbg_stage2_cal[11]	rd0_vld	Indicates valid ISERDES read data
dbg_stage2_cal[12]	fd0_vld	Indicates valid ISERDES read data
dbg_stage2_cal[13]	rd1_vld	Indicates valid ISERDES read data
dbg_stage2_cal[14]	fd1_vld	Indicates valid ISERDES read data

Table 2-21: Read Stage 2 Debug Signal Map (*Cont'd*)

Bits	PHY Signal Name	Description
dbg_stage2_cal[15]	phase_vld	Valid data is seen for the particular byte
dbg_stage2_cal[16]	rd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[17]	fd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[18]	rd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[19]	fd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[20]	phase_bslip_vld	Valid data is seen when bitslip applied to read data
dbg_stage2_cal[21]	clkdiv_phase_cal_done_4r	Indicates data validity complete, proceed to assert the pi_edge_adv signal if needed
dbg_stage2_cal[22]	pi_edge_adv	Phaser control signal to advance the Phaser clock, ICLKDIV by one fast clk cycle.
dbg_stage2_cal[25:23]	byte_cnt[2:0]	Indicates the byte that is being checked for data validity
dbg_stage2_cal[26]	inc_byte_cnt	Internal signal to increment to the next byte
dbg_stage2_cal[29:27]	pi_edge_adv_wait_cnt	Counter to wait between asserting the phaser control signal, pi_edge_adv signal in the various byte lanes.
dbg_stage2_cal[127:30]	reserved	Reserved

# RLDRAM II Memory Interface Solution

## Introduction

The RLDARAM II memory interface solution is a memory controller and physical layer for interfacing Xilinx® 7 series FPGAs user designs to RLDARAM II devices. An RLDARAM II device can transfer up to two, four, or eight words of data per request and are commonly used in applications such as look-up tables (LUTs), L3 cache, and graphics.

The RLDARAM II memory solutions core is composed of a user interface (UI), memory controller (MC), and physical layer (PHY). It takes simple user commands and converts them to the RLDARAM II protocol before sending them to the memory. Unique capabilities of Xilinx 7 series FPGAs 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 RLDARAM II memory interface core for Xilinx 7 series FPGAs.

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 272](#).

**Note:** RLDARAM II designs currently do not support memory-mapped AXI4 interfaces.

For detailed information and updates about the 7 series FPGAs RLDARAM II interface core, see the appropriate 7 series FPGAs data sheet [\[Ref 15\]](#).

## Getting Started with the CORE Generator Tool

This section is a step-by-step guide for using the CORE Generator™ tool to generate an RLDARAM II interface in a 7 series FPGA, run the design through implementation with the Xilinx tools, and simulate the example design using the synthesizable test bench provided.

## System Requirements

- ISE® Design Suite, v13.4

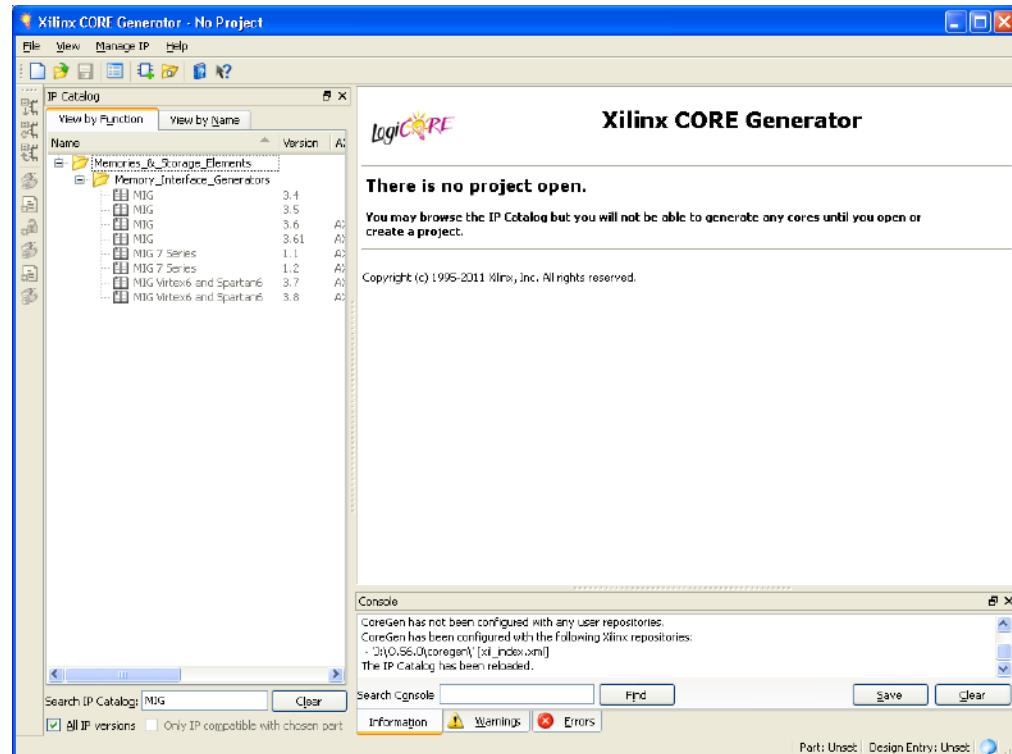
## Customizing and Generating the Core

### Generation through Graphical User Interface

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

These steps should be followed to generate an RLDRAM II design:

1. To launch the MIG tool from the CORE Generator tool, type **mig** in the search IP catalog box ([Figure 3-1](#)).



*Figure 3-1: Xilinx CORE Generator Tool*

2. Choose **File** → **New Project** to open the New Project dialog box. Create a new project named `7Series_MIG_Example_Design` ([Figure 3-2](#)).

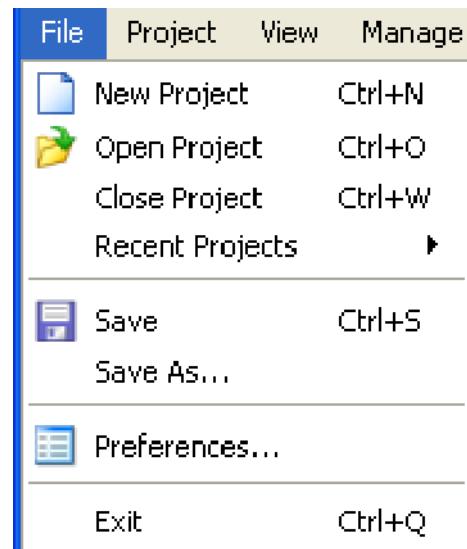


Figure 3-2: New CORE Generator Tool Project

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

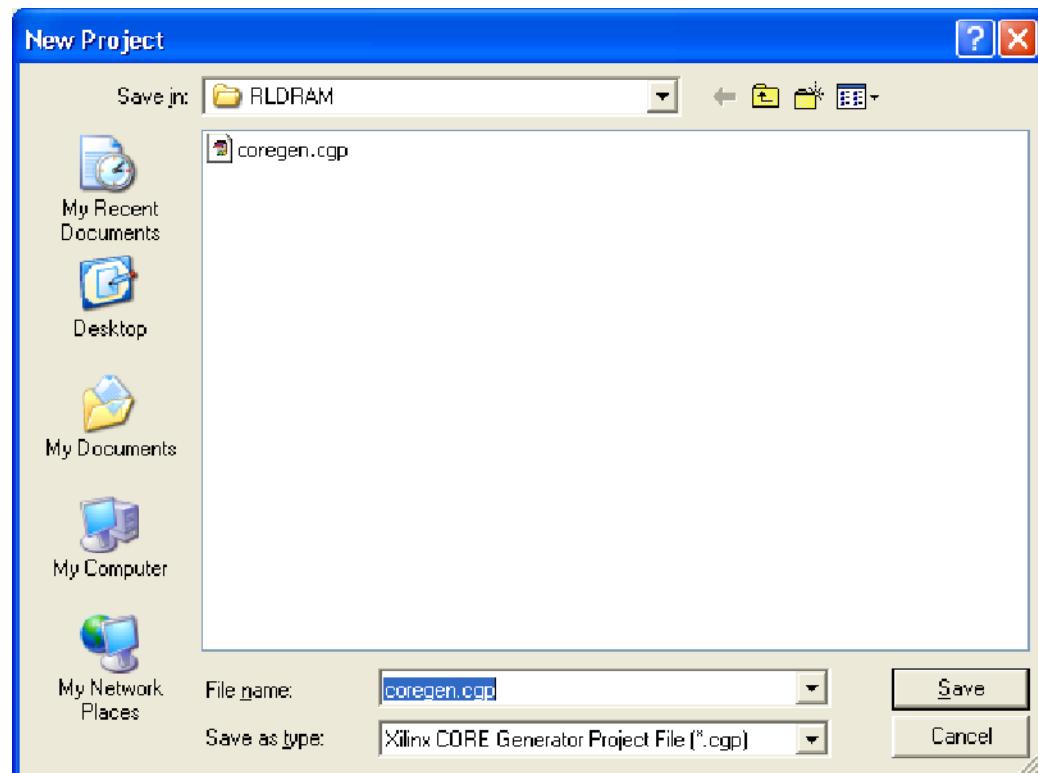


Figure 3-3: New Project Menu

4. Select these project options for the part ([Figure 3-4](#)):

- Select the target Kintex™-7 or Virtex®-7 device.

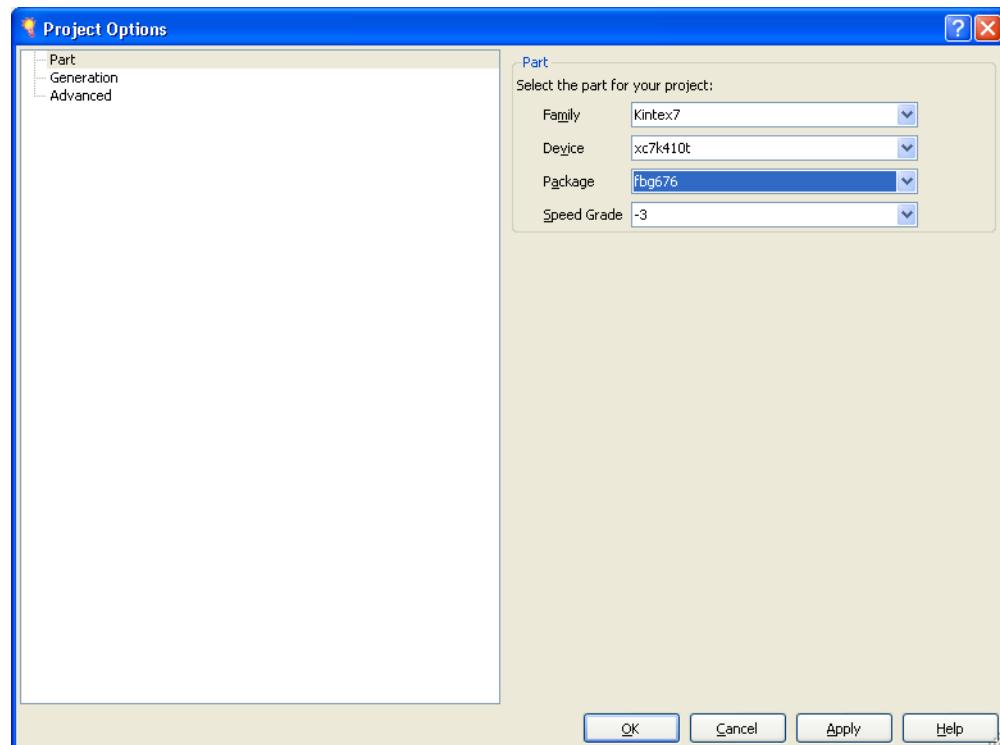


Figure 3-4: CORE Generator Tool Device Selection Page

5. Select **Verilog** as the Design Entry Option and **ISE** for the Vendor Flow Setting. Click **OK** to finish the Project Options setup ([Figure 3-5](#)).

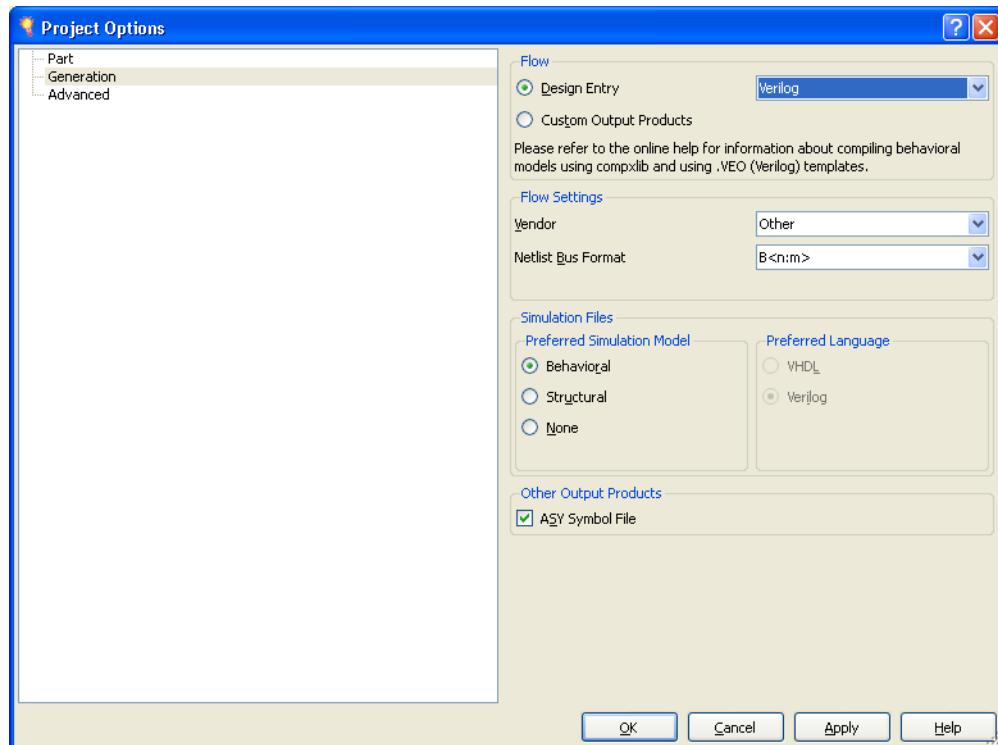


Figure 3-5: CORE Generator Tool Design Flow Setting Page

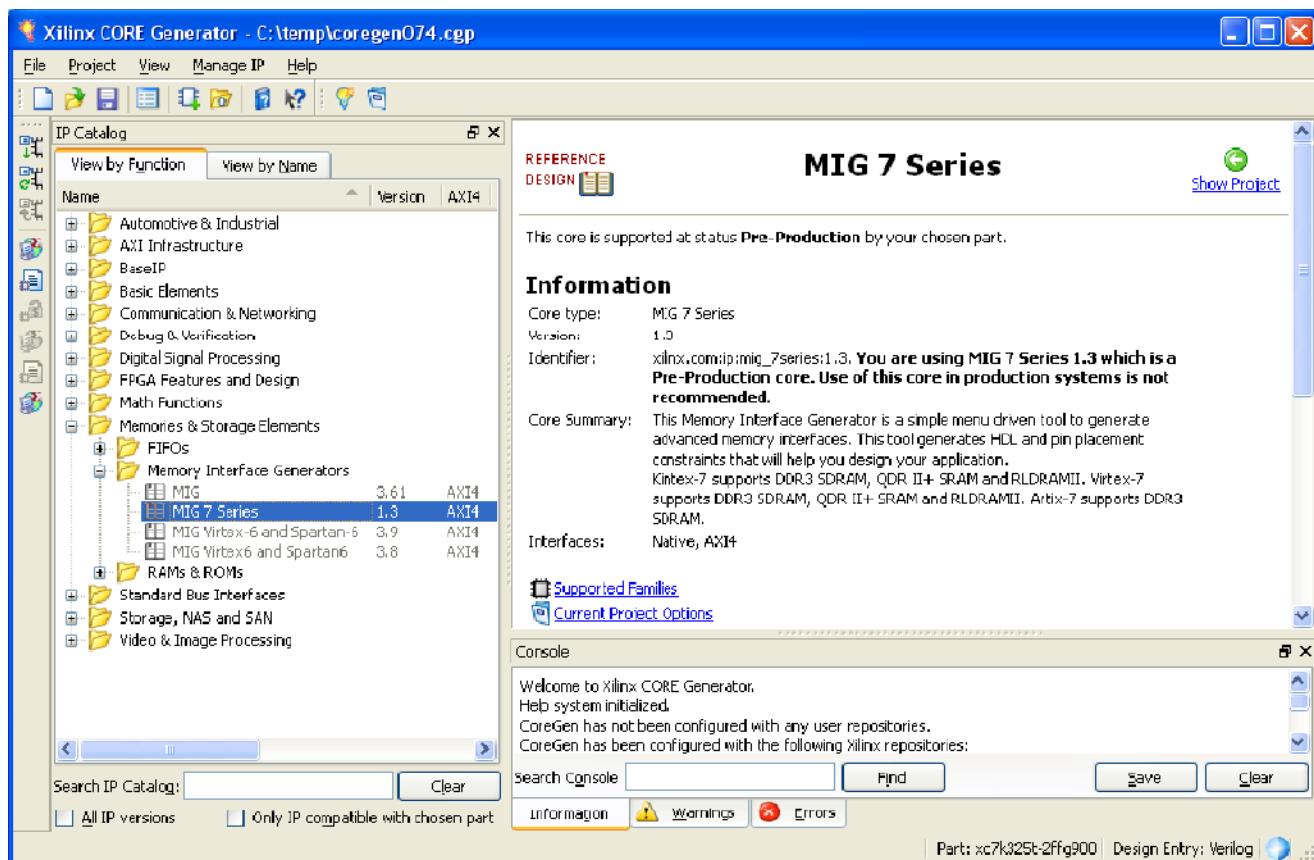
6. Select **MIG 7 Series 1.4** (Figure 3-6).

Figure 3-6: 7Series\_MIG\_Example\_Design Project Page

7. The options screen in the CORE Generator tool displays the details of the selected CORE Generator tool options that are selected before invoking the MIG tool ([Figure 3-7](#)).

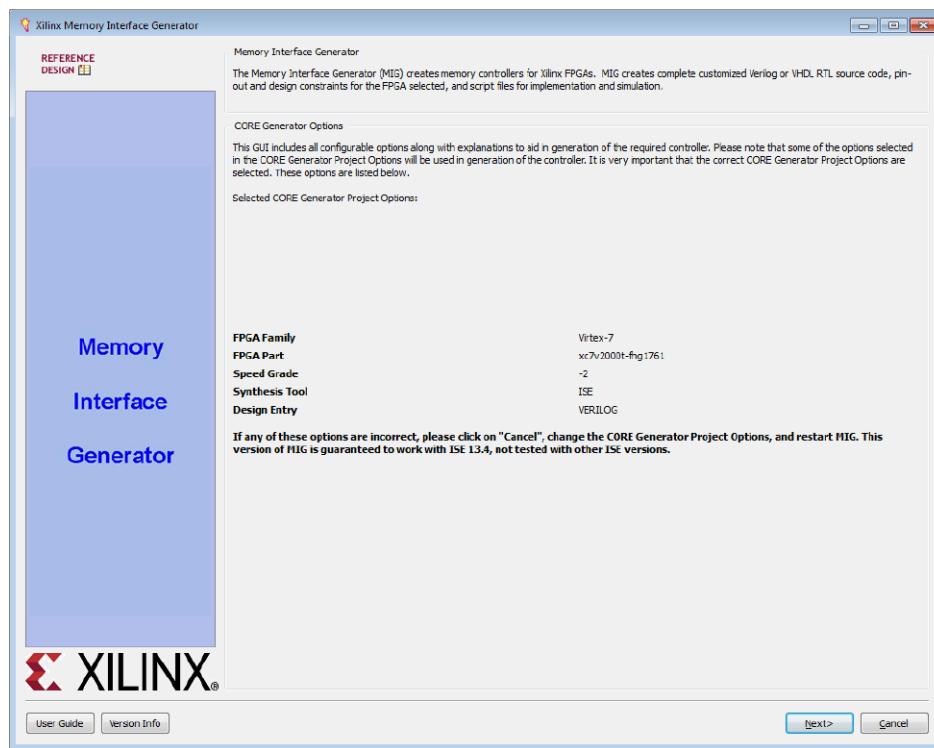
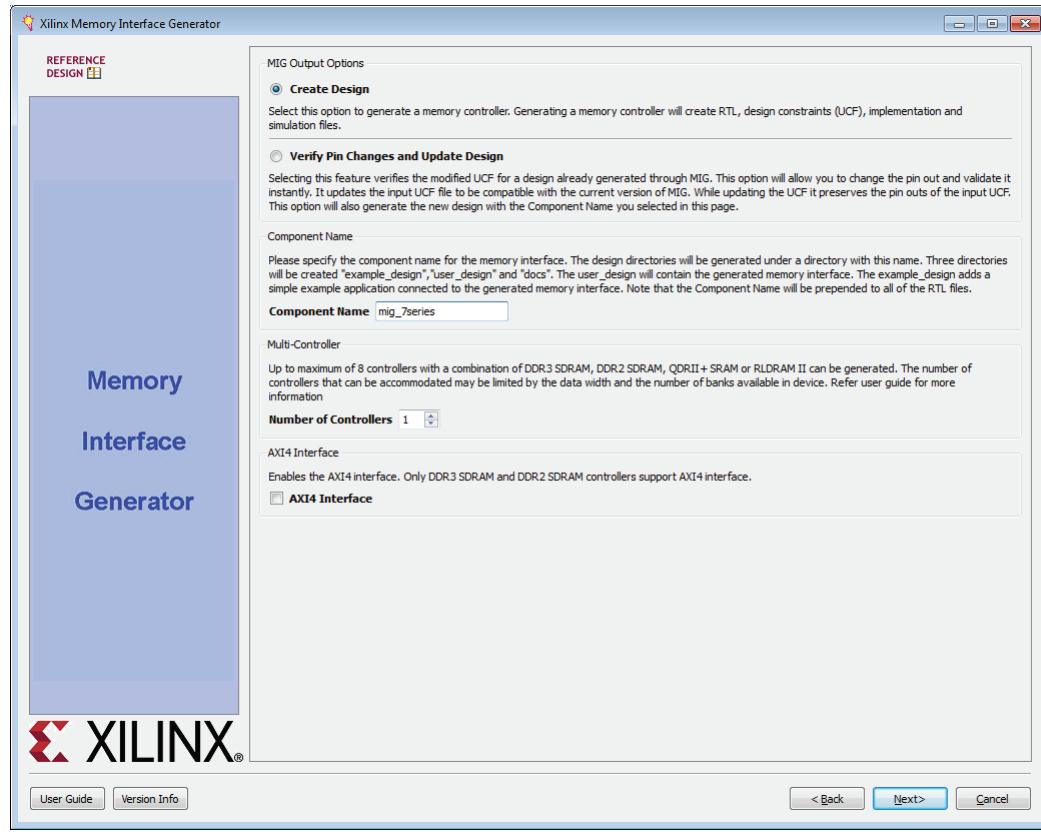


Figure 3-7: 7 Series FPGA Memory Interface Generator Front Page

8. Click **Next** to display the **Output Options** page.

## 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 3-8](#)).
2. Choose the number of controllers to be generated. This option determines the replication of further pages.



**Figure 3-8: 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, the component name is corrected to be the IP instance name from XPS.

3. Click **Next** to display the Pin Compatible FPGAs page.

## Pin Compatible FPGAs

The **Pin Compatible FPGAs** page 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 3-9).

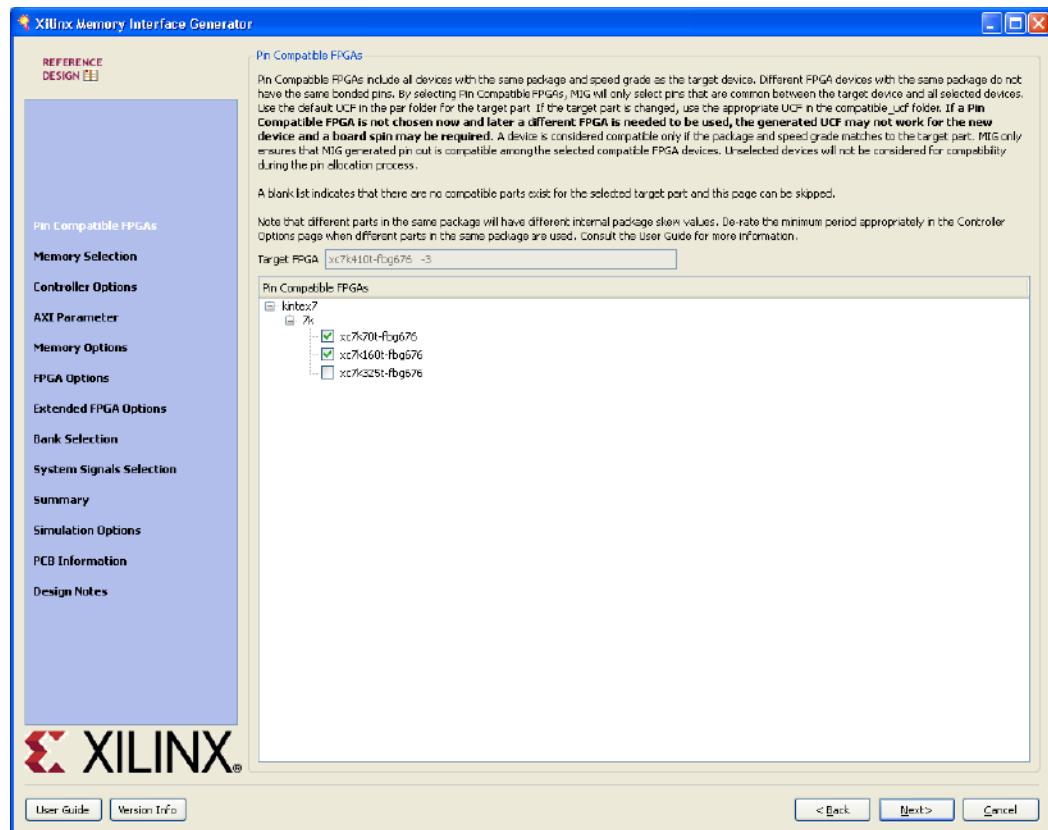


Figure 3-9: Pin-Compatible 7 Series FPGAs

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** page.

## Creating the 7 Series FPGAs 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** page.

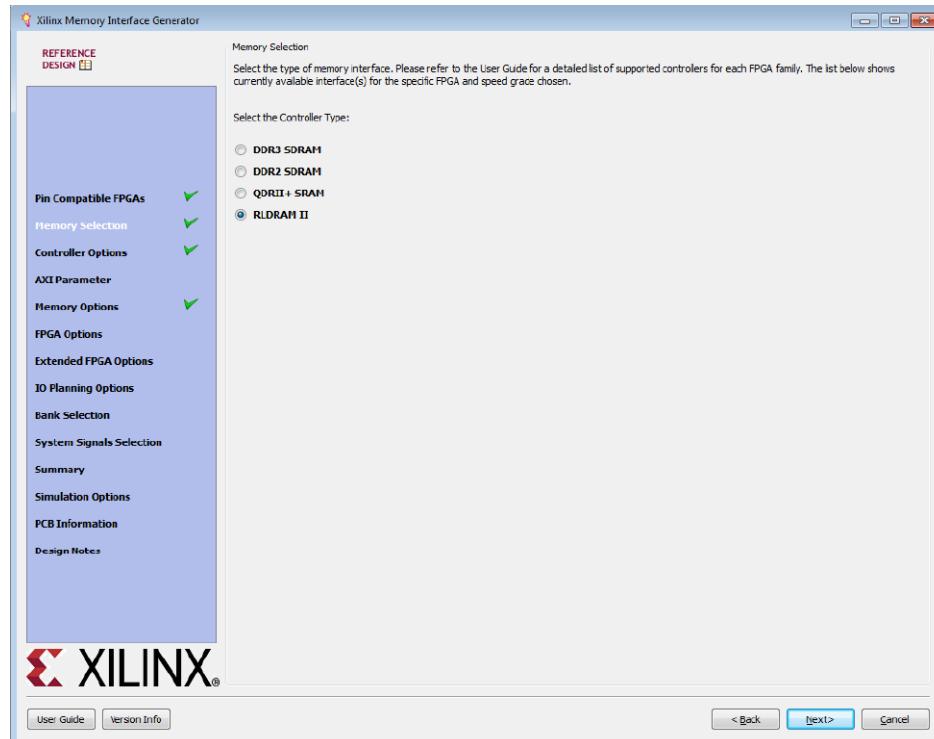


Figure 3-10: Memory Selection Page

RLDRAM II designs currently do not support memory-mapped AXI4 interfaces.

## Controller Options

This page shows the various controller options that can be selected.

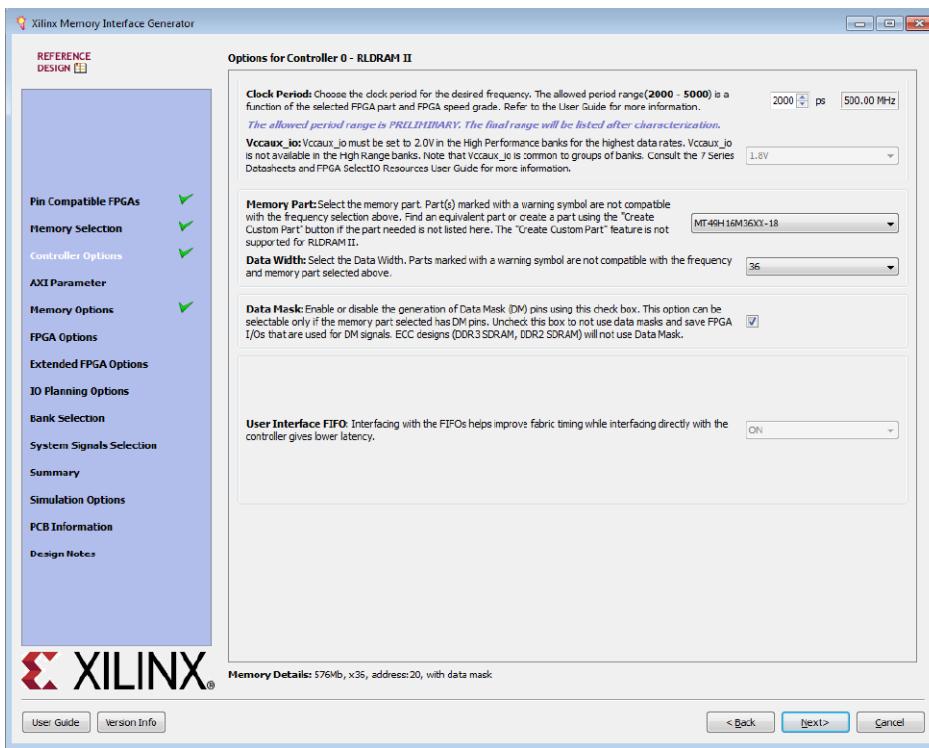


Figure 3-11: Controller Options Page

- **Frequency:** This feature indicates the operating frequency for all the controllers. The frequency block is limited by factors such as the selected FPGA and device speed grade.
- **Vccaux\_io:** Vccaux\_io is set based on the period/frequency setting. 2.0V is required at the highest frequency settings in the High Performance column. The MIG tool automatically selects 2.0V when required. Either 1.8 or 2.0V can be used at lower frequencies. Groups of banks share the Vccaux\_io supply. See the *7 Series FPGAs SelectIO Resources User Guide* [Ref 1] for more information.
- **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 (Create Custom Part). RLDRAM II 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.
- **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.

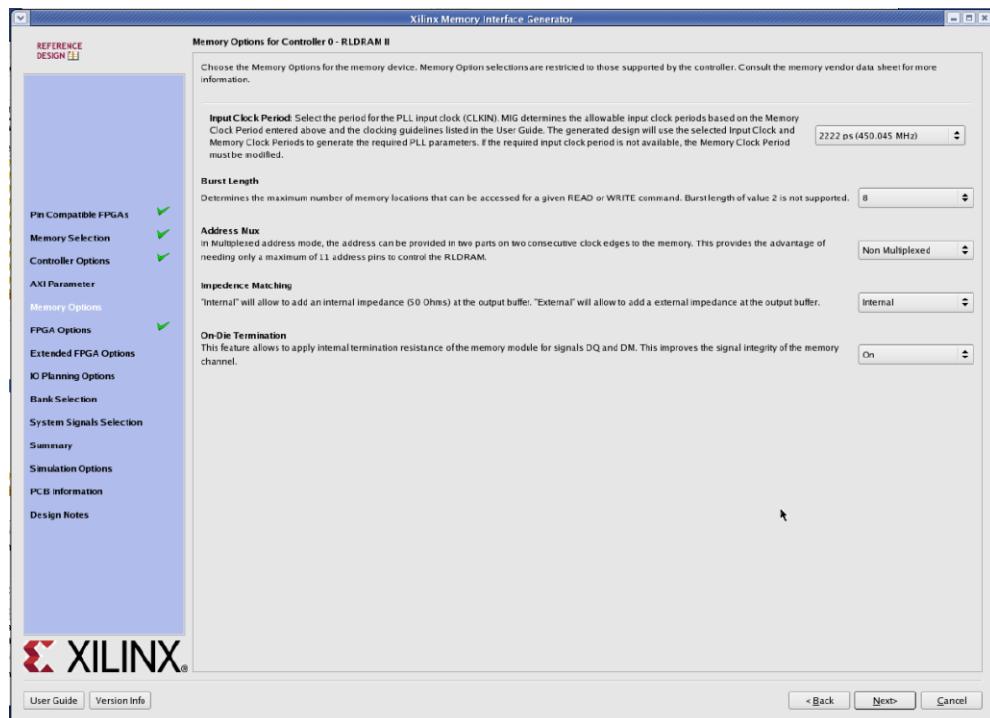
- **Memory Details:** The bottom of the **Controller Options** page. [Figure 3-12](#) displays the details for the selected memory configuration.

**Memory Details: 576Mb, x36, address:20, with data mask**

*Figure 3-12: Selected Memory Configuration Details*

## Memory Options

This feature allows the selection of various memory mode register values, as supported by the controller's specification ([Figure 3-13](#)).



*Figure 3-13: Memory Options Page*

The mode register value is loaded into the load mode register during initialization.

- **Input Clock Period:** The desired input clock period is selected from the list. These values are determined by the chosen memory clock period and the allowable limits of the PLL parameters. See [Clocking Architecture, page 255](#) for more information on the PLL parameter limits.
- **Configuration:** This option sets the configuration value associated with write and read latency values. Available values of 1, 2, and 3 are controlled based on the selected design frequency.
- **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.
- **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.

3. **Impedance Matching:** This option determines 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.
4. **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.

Click **Next** to display the **FPGA Options** page.

## FPGA Options

Figure 3-14 shows the **FPGA Options** page.

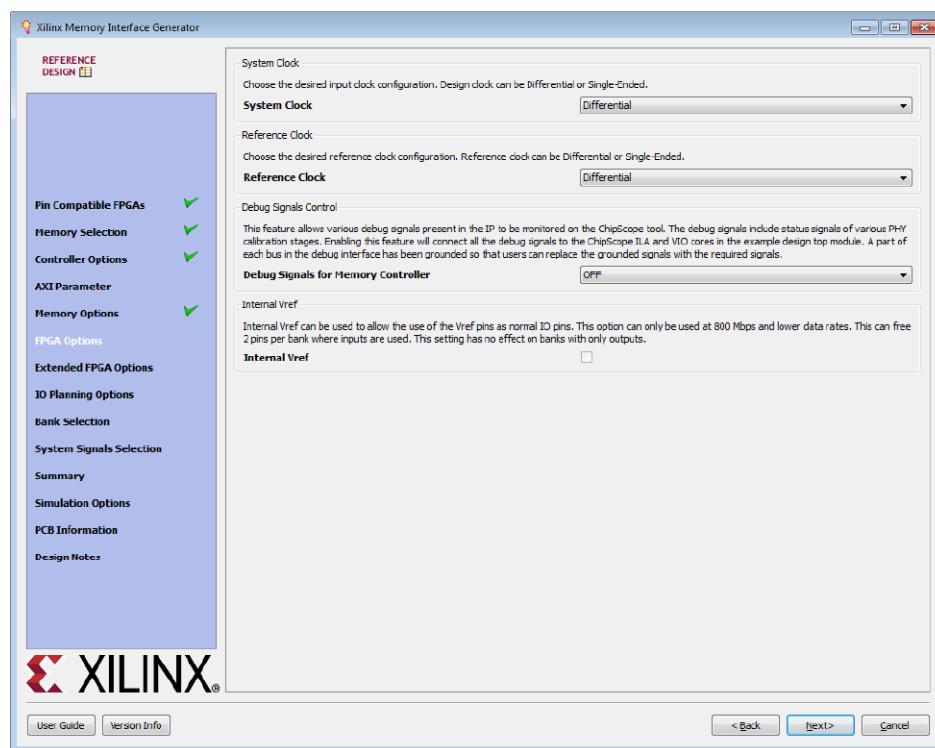


Figure 3-14: **FPGA Options Page**

- **System Clock.** This option selects the clock type (Single-Ended or Differential) for the sys\_clk signal pair.
- **Reference Clock.** This option selects the clock type (Single-Ended or Differential) for the ref\_clk signal pair.
- **Debug Signals Control.** 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. 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\_f1ow.bat in the design's par folder, the CORE Generator tool 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

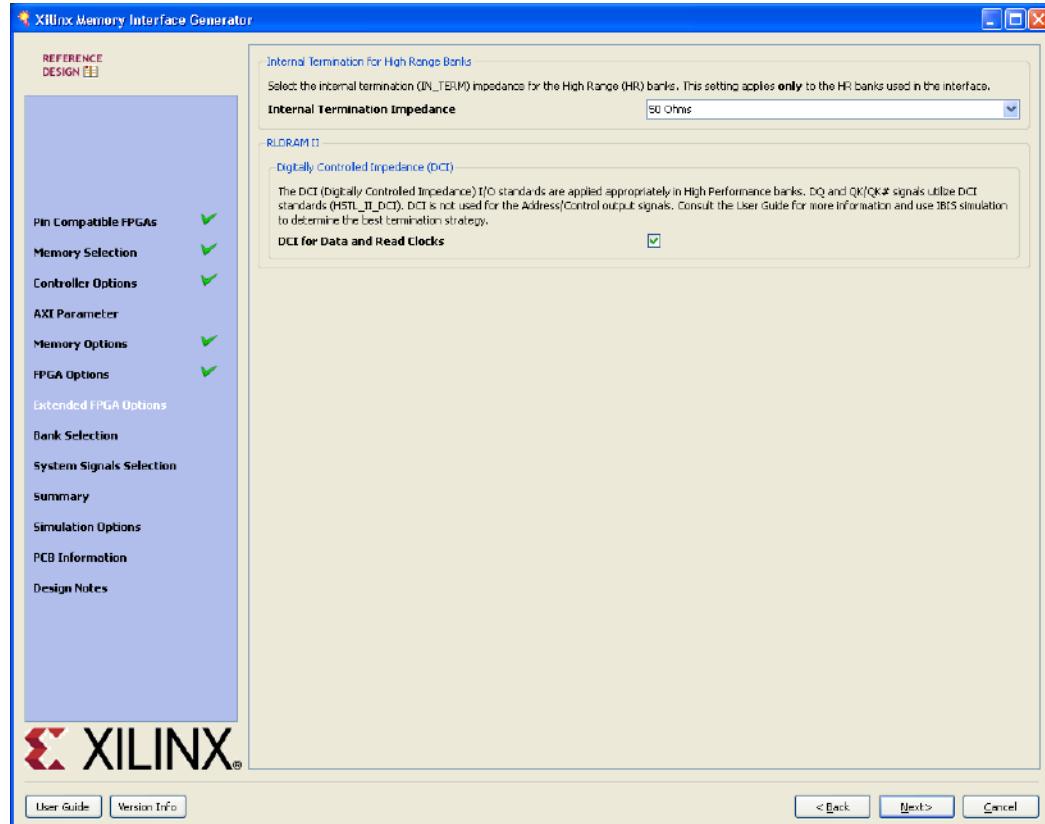
generated by the CORE Generator tool. The debug port is always disabled for functional simulations.

- **Internal Vref Selection.** Internal Vref can be used for data group bytes to allow the use of the  $V_{REF}$  pins for normal I/O usage. Internal Vref should only be used for data rates of 800 Mb/s or below.

Click **Next** to display the **Extended FPGA Options** page.

## Extended FPGA Options

[Figure 3-15](#) shows the Extended FPGA Options page.



[Figure 3-15: Extended FPGA Options Page](#)

- **Digitally Controlled Impedance (DCI):** When selected, this option internally terminates the signals from the RLDRAM II read path. DCI is available in the High Performance Banks.
- **Internal Termination for High Range Banks.** The internal termination option can be set to 40, 50, or 60 $\Omega$  or disabled. This termination is for the RLDRAM II read path. This selection is only for High Range banks.

## Bank Selection

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

- Address and control signals
- Data Read signals

- Data Write signals

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. To unselect the banks that are selected, click the **Deselect Banks** button. To restore the defaults, click the **Restore Defaults** button. Vccaux\_io groups are shown for HP banks in devices with these groups using dashed lines. Vccaux\_io is common to all banks in these groups. The memory interface must have the same Vccaux\_io for all banks used in the interface. MIG automatically sets the VCCAUX\_IO constraint appropriately for the data rate requested.

For devices implemented with SSI technology, the SLRs are indicated by a number in the header in each bank, for example, *SLR 1*. Interfaces cannot span across Super Logic Regions. Not all devices have Super Logic Regions.

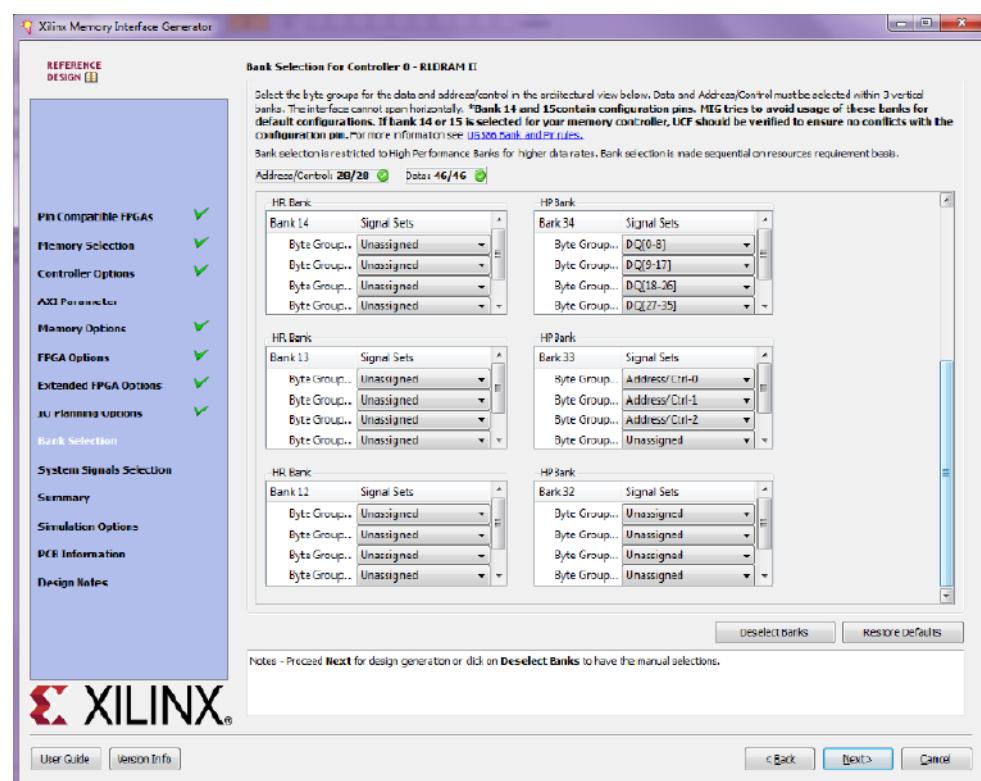


Figure 3-16: Bank Selection Page

## System Pins Selection

Figure 3-17 shows the System Pins Selection page.

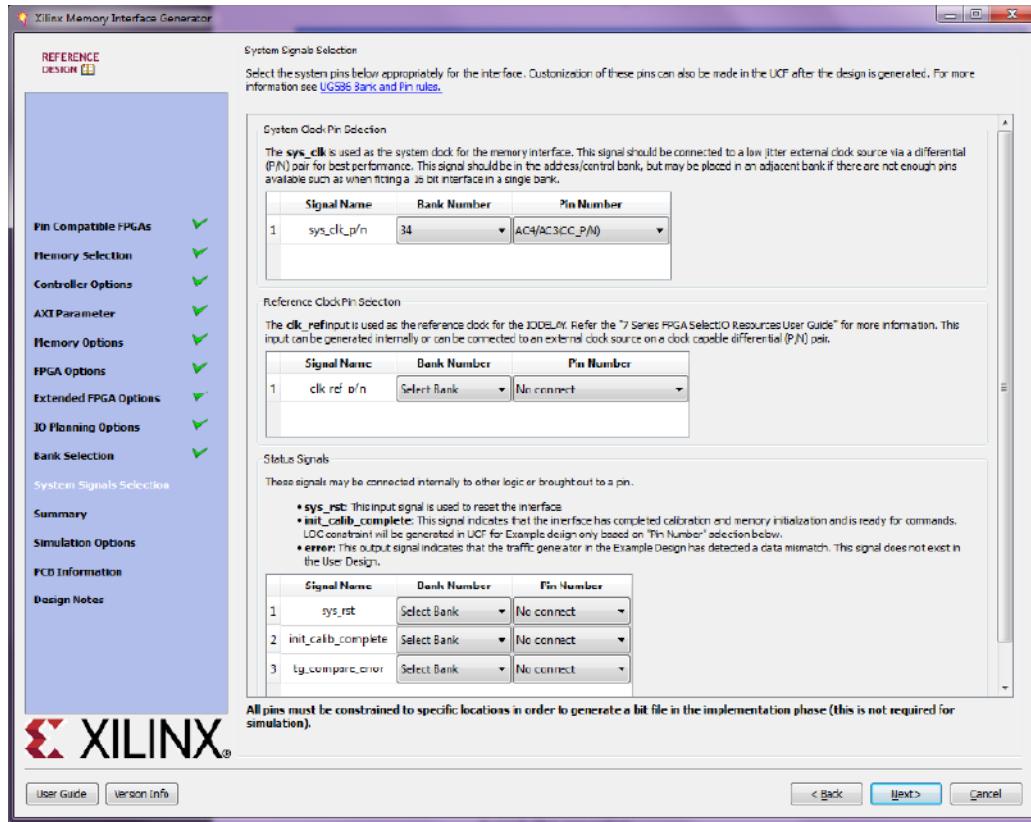


Figure 3-17: System Pins Selection Page

Select the pins for the system signals on this page. The MIG tool allows the selection of either external pins or internal connections, as desired.

- **sys\_clk:** This is the system clock input for the memory interface and is typically connected to a low-jitter external clock source. Either a single input or a differential pair can be selected based on the **System Clock** selection in the **FPGA Options** page (Figure 3-14). The sys\_clk input must be in the same column as the memory interface. If this pin is connected in the same banks as the memory interface, the MIG tool selects an I/O standard compatible with the interface, such as DIFF\_SSTL15 or SSTL15. If sys\_clk is not connected in a memory interface bank, the MIG tool selects an appropriate standard such as LVCMOS18 or LVDS. The UCF can be modified as desired after generation.
- **clk\_ref:** This is the reference frequency input for the IDELAY control. This is a 200 MHz input. The clk\_ref input can be generated internally or connected to an external source. A single input or a differential pair can be selected based on the **System Clock** selection in the **FPGA Options** page (Figure 3-14). The I/O standard is selected in a similar way as sys\_clk above.
- **sys\_rst:** This is the system reset input that can be generated internally or driven from a pin. The MIG tool selects an appropriate I/O standard for the input such as SSTL15 if the input is within the interface banks, and LVCMOS18 if it is not.

- **init\_calib\_complete**: This output indicates that the memory initialization and calibration is complete and that the interface is ready to use. The init\_calib\_complete signal is normally only used internally, but can be brought out to a pin if desired.
- **tg\_compare\_error**: This output indicates that the traffic generator in the example design has detected a data compare error. This signal is only generated in the example design and is not part of the user design. This signal is not typically brought out to a pin but can be, if desired.

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

## Summary

This page (Figure 3-18) provides the complete details about the memory core selection, interface parameters, CORE Generator tool options, and FPGA options of the active project.

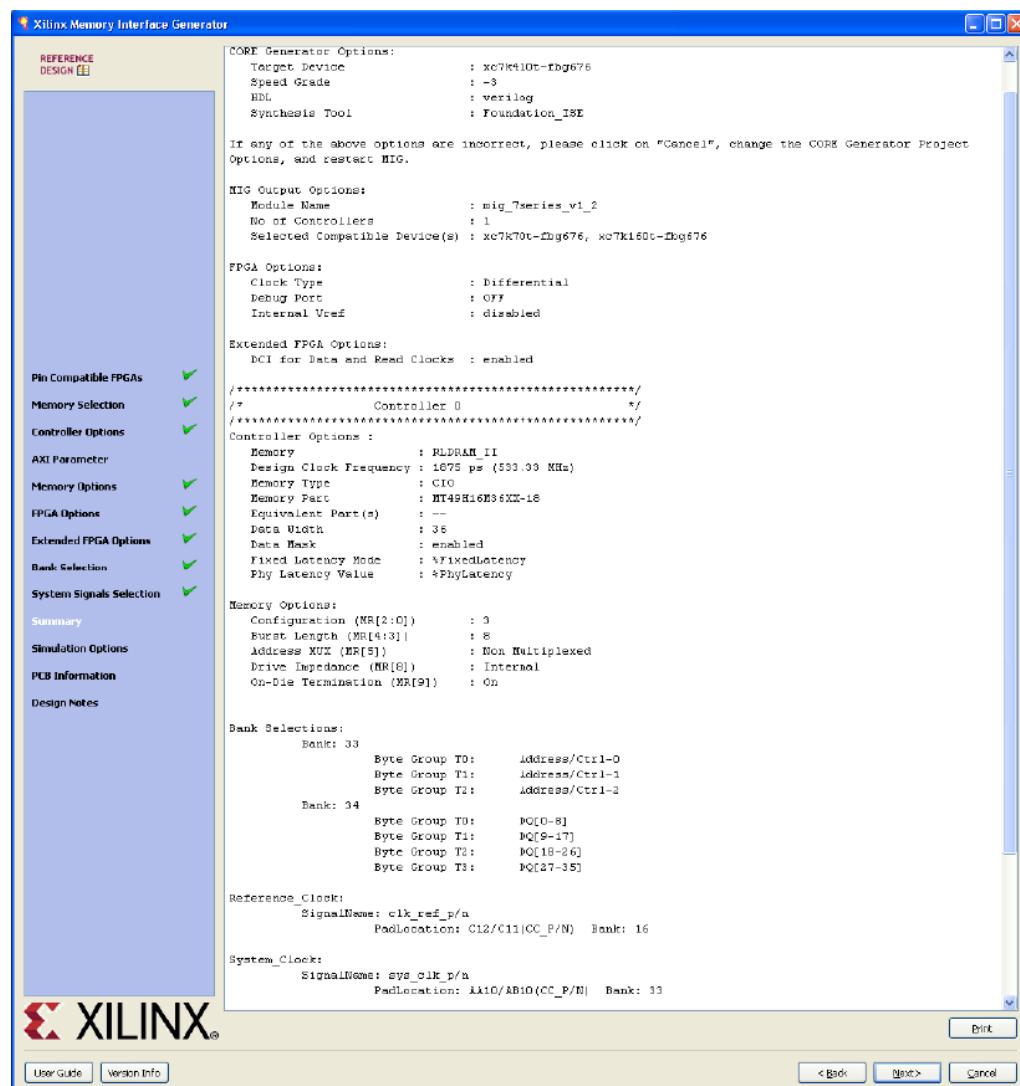


Figure 3-18: Summary Page

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 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.

## Finish

After the design is generated, a README page is displayed with additional useful information.

Click **Close** to complete the MIG tool flow.

## MIG Directory Structure and File Descriptions

This section explains the MIG tool directory structure and provides detailed output file descriptions.

### Output Directory Structure

The MIG tool places all output files and directories in a folder called <component\_name>, where <component\_name> was specified on the [MIG Output Options, page 228](#) of the MIG design creation flow.

Figure 3-19 shows the output directory structure for the memory controller design. There are three folders created within the <component\_name> directory:

- docs
- example\_design
- user\_design

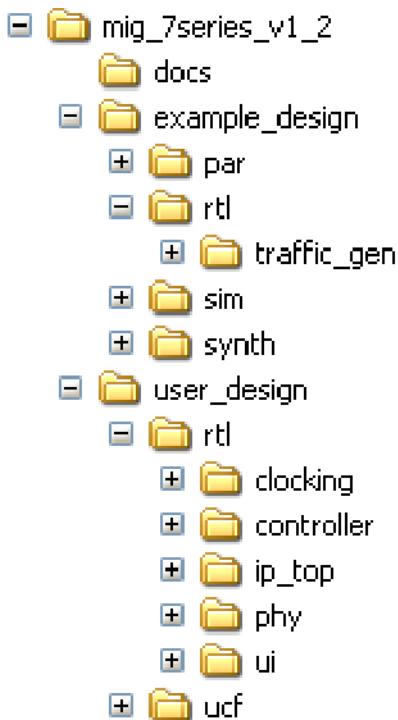


Figure 3-19: MIG Directory Structure

## Directory and File Contents

The 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 directory structure contains all necessary RTL, constraints, and script files for simulation and implementation of the complete MIG example design with a test bench. The optional ChipScope™ tool module is also included in this directory structure.

Table 3-1 lists the files in the example\_design/rtl directory.

Table 3-1: Files in example\_design/rtl Directory

Name	Description
example_top.v	This top-level module serves as an example for connecting the user design to the 7 series FPGAs memory interface core.

**Table 3-2** lists the files in the example\_design/rtl/traffic\_gen directory.

**Table 3-2: Modules in example\_design/rtl/traffic\_gen Directory**

Name	Description
memc_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.
memc_flow_vcontrol.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 sent to the memory controller; its FIFO output is used to generate expected 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/vhd	This module generates timing control for writes and ready signals to mcb_flow_control.v/vhd.
s7ven_data_gen.v/vhd	This module generates different data patterns.
a_fifo.v/vhd	This is a synchronous FIFO using LUT RAMs.
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.
traffic_gen_top.v/vhd	This module is the top level of the traffic generator and comprises the memc_traffic_gen and init_mem_pattern_ctr modules.

**Table 3-3** lists the files in the example\_design/sim directory.

**Table 3-3: Files in example\_design/sim Directory**

Name	Description
sim.do	This is the ModelSim simulator script file.
sim_tb_top.v	This file is the simulation top-level file.

**Table 3-4** lists the files in the example\_design/par directory.

**Table 3-4: Files in the example\_design/par Directory**

Name	Description
example_top.ucf	This file is the UCF for the core of the example design.
create_ise.bat	Double-click this file to create an ISE tool project. The generated ISE tool project contains the recommended build options for the design. To run the project in GUI mode, double-click the ISE tool project file to open up the ISE tool in GUI mode with all project settings.
ise_flow.bat	This script file runs the design through synthesis, build, map, and par. It sets all required options. Refer to this file for the recommended build options for the design.
rem_files.bat	This batch file moves all the implementation files generated during implementation.
set_ise_prop.tcl	List of properties to the ISE tool.
xst_options.txt	List of properties to the synthesis tool.
ila_cg.xco, icon_cg.xco, vio_cg.xco	XCO files for ChipScope modules to be generated when debug is enabled.

**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.

**Table 3-5** lists the files in the example\_design/synth directory.

**Table 3-5: Files in the example\_design/synth Directory**

Name	Description
example_top.prj	This file lists all the RTL files to be compiled by the XST tool.
example_top.lso	Custom library search order file for XST synthesis.

<component name>/user\_design/

**Table 3-6** lists the files in the user\_design/rtl/controller directory.

**Table 3-6: Files in user\_design/rtl/controller Directory**

Name	Description
rld_mc.v	This module implements the memory controller.

**Table 3-7** lists the files in the user\_design/rtl/ui directory.

**Table 3-7: Files in user\_design/rtl/ui Directory**

Name	Description
rld_ui_top.v	This is the top-level wrapper for the user interface.
rld_ui_wr.v	This module generates the FIFOs used to buffer write data for the user interface.
rld_ui_addr.v	This module generates the FIFOs used to buffer address and commands for the user interface.

**Table 3-8** lists the files in the user\_design/rtl/phy directory.

**Table 3-8: Files in user\_design/rtl/phy Directory**

Name	Description
rld_phy_top.v	This is the top-level module for the physical layer file.
rld_phy_write_top.v	This is the top-level wrapper for the write path.
qdr_rld_phy_read_top.v	This is the top-level of the read path.
qdr_rld_mc_phy.v	This module is a parameterizable wrapper instantiating up to three I/O banks each with four-lane PHY primitives.
rld_phy_write_init_sm.v	This module contains the logic for the initialization state machine.
rld_phy_write_control_io.v	This module contains the logic for the control signals going to the memory.
rld_phy_write_data_io.v	This module contains the logic for the data and byte writes going to the memory.
qdr_rld_prbs_gen.v	This PRBS module uses a many-to-one feedback mechanism for 2n sequence generation.
qdr_rld_phy_ck_addr_cmd_delay.v	This module contains the logic to provide the required delay on the address and control signals.
qdr_rld_phy_rdlvl.v	This module contains the logic for stage 1 calibration.
qdr_rld_phy_read_stage2_cal.v	This module contains the logic for stage 2 calibration.
qdr_rld_phy_read_data_align.v	This module realigns the incoming data.
qdr_rld_phy_read_vld_gen.v	This module contains the logic to generate the valid signal for the read data returned on the user interface.
rld_phy_byte_lane_map.v	This module handles the vector remapping between the mc_phy module ports and the user's memory ports.

**Table 3-8: Files in user\_design/rtl/phy Directory (Cont'd)**

Name	Description
qdr_rld_phy_4lanes.v	This module is the parameterizable four-lane PHY in an I/O bank.
qdr_rld_byte_lane.v	This module contains the primitive instantiations required within an output or input byte lane.
qdr_rld_byte_group_io.v	This module contains the parameterizable I/O logic instantiations and the I/O terminations for a single byte lane.

Table 3-9 lists the files in the user\_design/ucf directory.

**Table 3-9: user\_design/ucf Directory**

Name	Description
<component name>.ucf	This file is the UCF for the core of the user design.

## 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. See this file for the recommended build options for the design.

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

The MIG tool provides a synthesizable test bench to generate various traffic data patterns to the memory controller (MC). This test bench consists of a `rld_memc_ui_top` wrapper, a `traffic_generator` that generates traffic patterns through the user interface to a `rld_ui_top` core, and an infrastructure core that provides clock resources to the `rld_memc_ui_top` core. A block diagram of the example design test bench is shown in Figure 3-20.

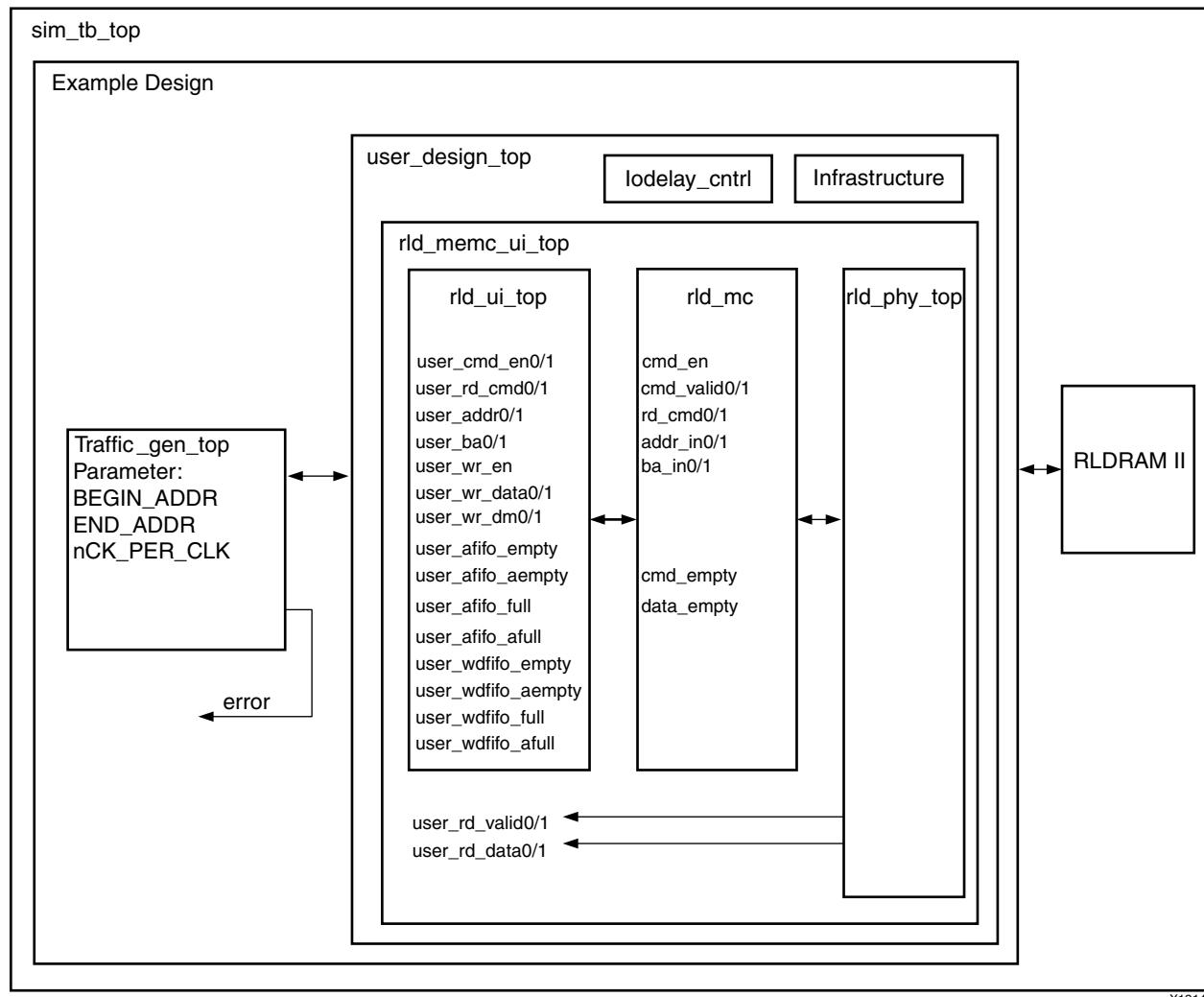
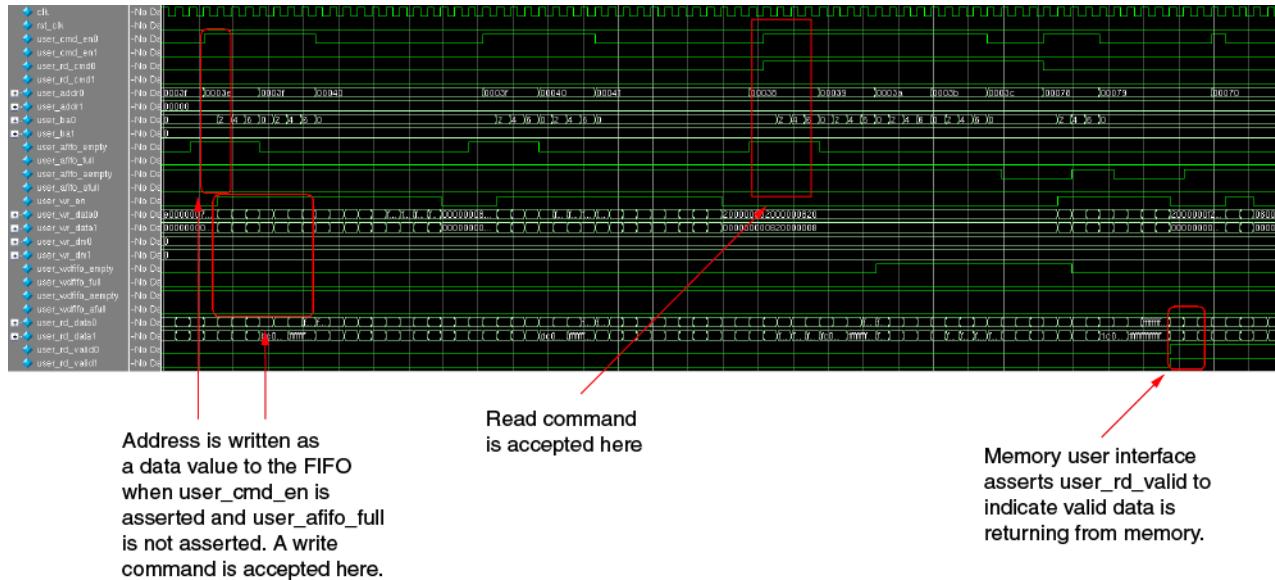


Figure 3-20: Synthesizable Example Design Block Diagram

**Figure 3-21** shows the simulation result of a simple read and write transaction between the tb\_top and memc\_intfc modules.



*Figure 3-21: User Interface Read and Write 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.

## 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 3-10](#) describes these parameters.

**Table 3-10: 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 "VIRTEX7".
MEMORY_TYPE	Indicate the memory controller type.	Current support is DDR2 SDRAM, DDR3 SDRAM, QDRII+ SRAM, and RLDRAM II.
nCK_PER_CLK	This is the memory controller clock to DRAM clock ratio.	This must be set to 2.
NUM_DQ_PINS	The is the total memory DQ bus width.	This parameter supports DQ widths from 8 to a maximum of 72 in increments of 9. The available maximum DQ width is frequency dependent on the selected memory device.
MEM_BURST_LEN	This is the memory data burst length.	This must be set to 8.
MEM_COL_WIDTH	This is the number of memory column address bits.	This must be set to 10.
DATA_WIDTH	This is the user interface data bus width.	For nCK_PER_CLK = 2, DATA_WIDTH = NUM_DQ_PINS * 4.
ADDR_WIDTH	This is the memory address bus width.	
MASK_SIZE	This parameter specifies the mask width in the user interface data bus.	This must be set to DATA_WIDTH/8.
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_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 3-10: 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.	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• CGEN_FIXED: The address, burst length, and instruction are taken directly from the fixed_addr_i, fixed_bl_i, and fixed_instr_i inputs.</li> <li>• CGEN_SEQUENTIAL: The address is incremented sequentially, and the increment is determined by the data port size.</li> <li>• 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.</li> <li>• CGEN_ALL (default): This option turns on all of the options above and allows addr_mode_i, instr_mode_i, and bl_mode_i to select the type of generation during run time.</li> </ul>

Table 3-10: Traffic Generator Parameters Set in the example\_top Module (Cont'd)

Parameter	Parameter Description	Parameter Value
DATA_PATTERN	This parameter sets the data pattern circuits to be generated through rtl logic. For larger devices, the DATA_PATTERN can be set to "DGEN_ALL", enabling all supported data pattern circuits to be generated. In hardware, the data pattern is selected and/or changed using vio_data_value_mode. The pattern can only be changed when DATA_PATTERN is set to DGEN_ALL.	<p>Valid settings for this parameter are:</p> <ul style="list-style-type: none"> <li>• ADDR (default): The address is used as a data pattern.</li> <li>• 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.</li> <li>• WALKING1: Walking 1s are on the DQ pins and the starting position of 1 depends on the address value.</li> <li>• WALKING0: Walking 0s are on the DQ pins and the starting position of 0 depends on the address value.</li> <li>• NEIGHBOR: The Hammer pattern is on all DQ pins except one. The address determines the exception pin location.</li> <li>• PRBS: A 32-stage LFSR generates random data and is seeded by the starting address.</li> <li>• DGEN_ALL: This option turns on all available options:           <ul style="list-style-type: none"> <li>0x1: FIXED - 32 bits of fixed_data.</li> <li>0x2: ADDRESS - 32 bits address as data.</li> <li>0x3: HAMMER</li> <li>0x4: SIMPLE8 - Simple 8 data pattern that repeats every 8 words.</li> <li>0x5: WALKING1s - Walking 1s are on the DQ pins.</li> <li>0x6: WALKING0s - Walking 0s are on the DQ pins.</li> <li>0x7: PRBS - A 32-stage LFSR generates random data. This mode only works with either a PRBS address or a SEQUENTIAL address pattern.</li> <li>0x9: SLOW HAMMER - This is the slow MHz hammer data pattern.</li> <li>0xF: PHY_CALIB pattern - 0xFF, 00, AA, 55, 55, AA, 99, 66. This mode only generates READ commands at address zero.</li> </ul> </li> </ul>
CMDS_GAP_DELAY	This parameter allows pause delay between each user burst command.	Valid values: 0 to 32.
SEL_VICTIM_LINE	Select a victim DQ line whose state is always at logic High.	<p>This parameter only applies to the Hammer pattern. Valid settings for this parameter are 0 to NUM_DQ_PINS.</p> <p>When value = NUM_DQ_PINS, all DQ pins have the same Hammer pattern.</p>

**Table 3-10: Traffic Generator Parameters Set in the example\_top Module (Cont'd)**

Parameter	Parameter Description	Parameter Value
EYE_TEST	Force the traffic generator to only generate writes to a single location, and no read transactions are generated.	Valid settings for this parameter are "TRUE" and "FALSE". When set to "TRUE", any settings in vio_instr_mode_value are overridden.

**Note:** The traffic generator might support more options than are available in the FPGA memory controller. The settings must match supported values in the memory controller.

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\_ctrl 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 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.
  - 4: 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.
  - 5: DGEN\_WALKING1. Walking 1s are on the DQ pins. The starting position of 1 depends on the address value.
  - 6: DGEN\_WALKING0. Walking 0s 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. The PRBS data pattern only works together with a PRBS address or a sequential address.

## Core Architecture

### Overview

Figure 3-22 shows a high-level block diagram of the 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.

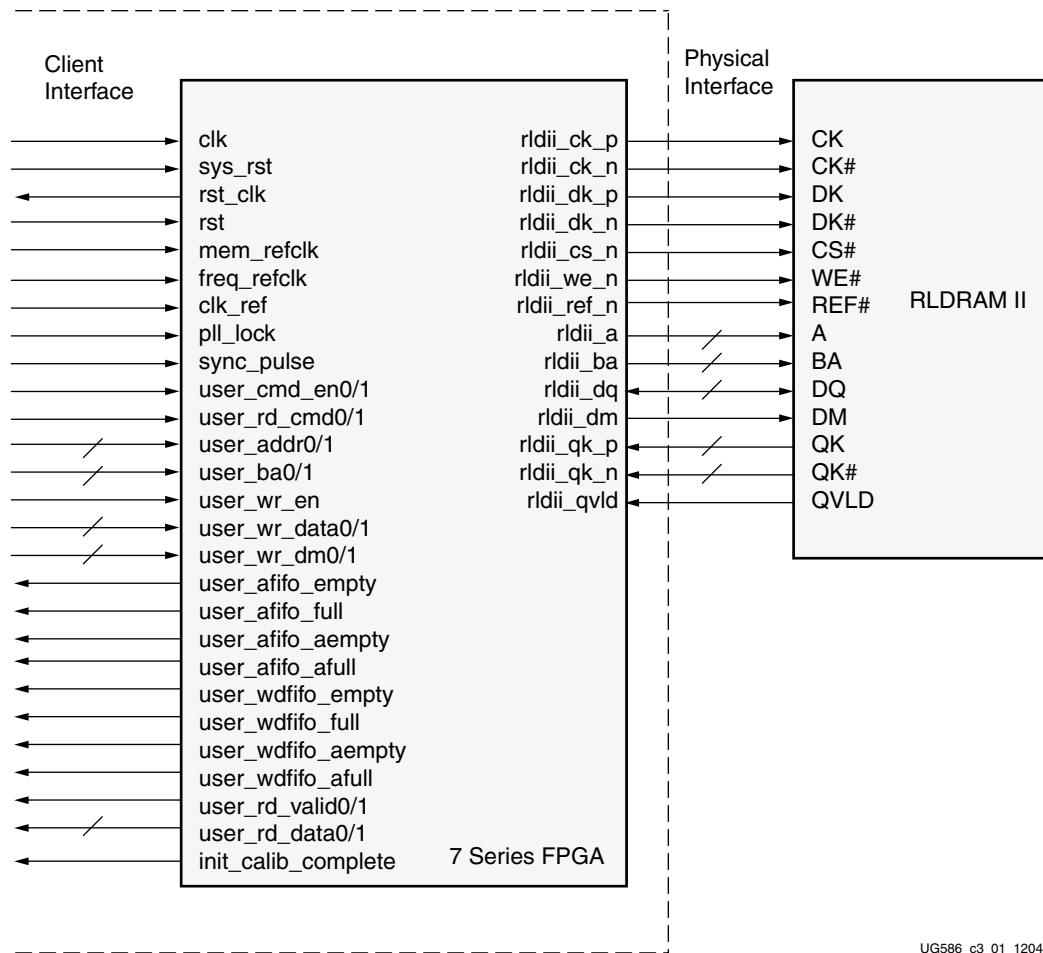
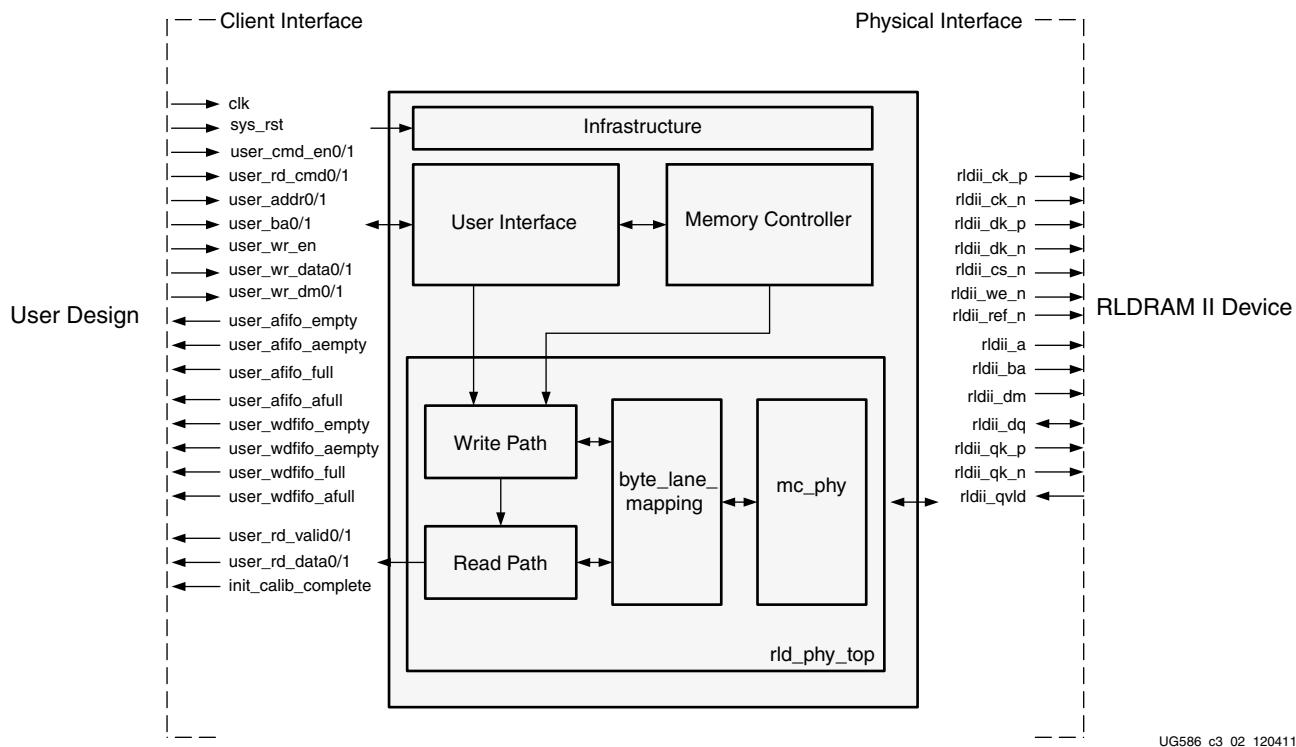


Figure 3-22: High-Level Block Diagram of RLDRAM II Interface Solution

The core is composed of these elements, as shown in [Figure 3-23](#):

- Client Interface
- Memory Controller
- Physical Interface
- Read Path
- Write Path



**Figure 3-23: 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](#) 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 device. Refer to [Memory Controller, page 259](#) 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 256](#) 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 264](#) for more details describing this process.

## Client Interface

The client interface connects the 7 series 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 3-11](#) and are listed assuming four-word or eight-word burst architectures.

**Table 3-11: Client Interface Request Signals**

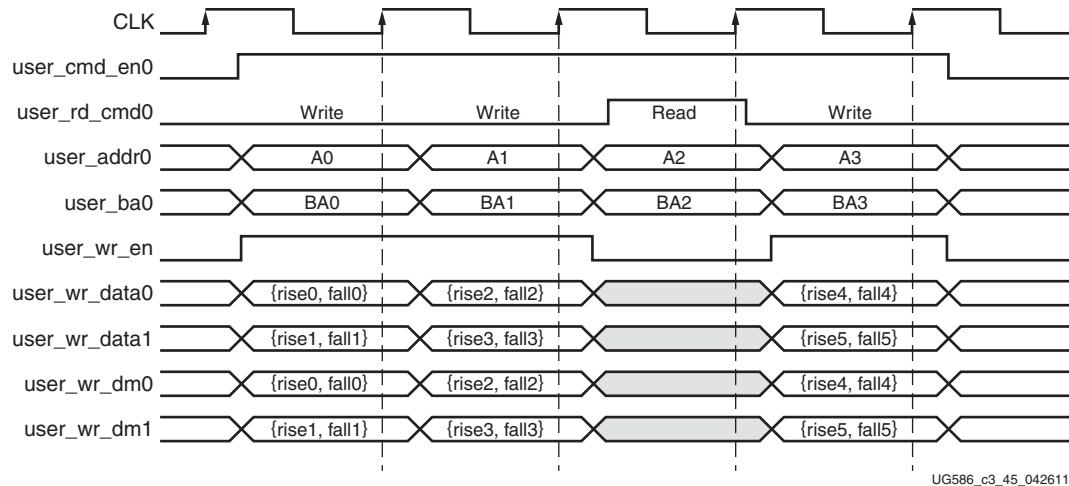
Signal	Direction	Description
user_cmd_en0	Input	Command Enable. This signal issues a read or write request and indicates that the corresponding command signals are valid.
user_rd_cmd0	Input	Read Command. This signal issues a read request. When user_cmd_en0 is asserted, this signal is active High for 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.
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 1. 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.

Table 3-11: Client Interface Request Signals (*Cont'd*)

Signal	Direction	Description
user_wdfifo_empty	Output	Write Data FIFO empty. If asserted, the write data buffer is empty.
user_afifo_full	Output	Address FIFO full. If asserted, the command buffer is full, and any writes to the FIFO are ignored until deasserted.
user_wdfifo_full	Output	Write Data FIFO full. If asserted, the write data buffer is full, and any writes to the FIFO are ignored until deasserted.
user_afifo_aempty	Output	Address FIFO almost empty. If asserted, the command buffer is almost empty.
user_afifo_afull	Output	Address FIFO almost full. If asserted, the command buffer is almost full.
user_wdfifo_aempty	Output	Write Data FIFO almost empty. If asserted, the write data buffer is almost empty.
user_wdfifo_afull	Output	Write Data FIFO almost full. If asserted, the Write Data buffer is almost full.
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.
init_calib_complete	Output	Calibration Done. This signal indicates back to the user design that read calibration is complete and requests can now take place.

## Interfacing with the Core through the Client Interface

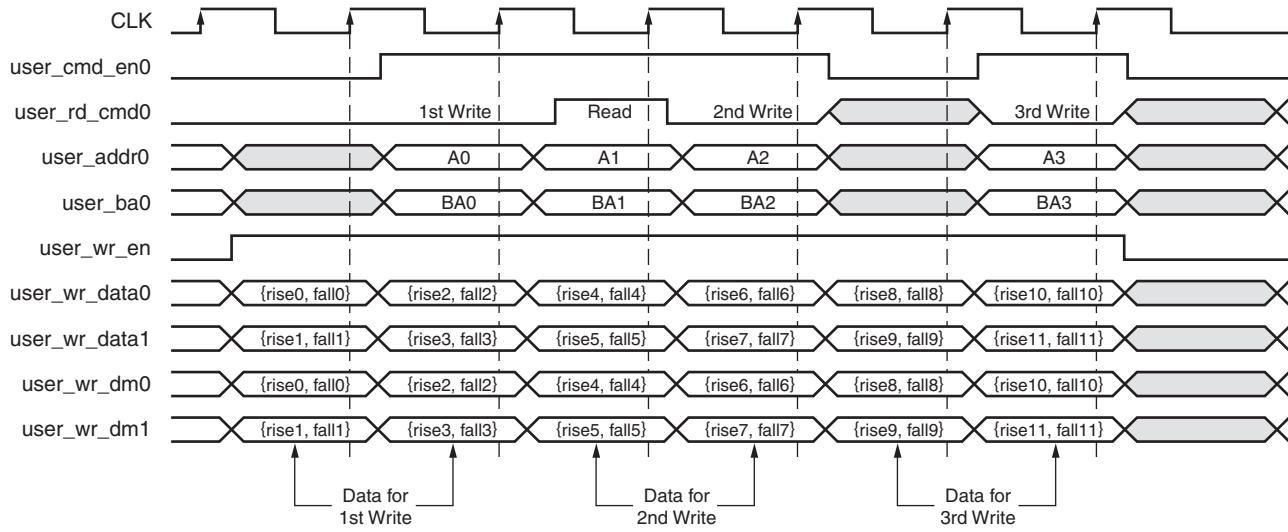
The client interface protocol is shown in [Figure 3-24](#) for the four-word burst architecture.



**Figure 3-24: Client Interface Protocol (Four-Word Burst Architecture)**

Before any requests can be accepted, the `ui_clk_sync_rst` signal must be deasserted Low. After the `ui_clk_sync_rst` 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.

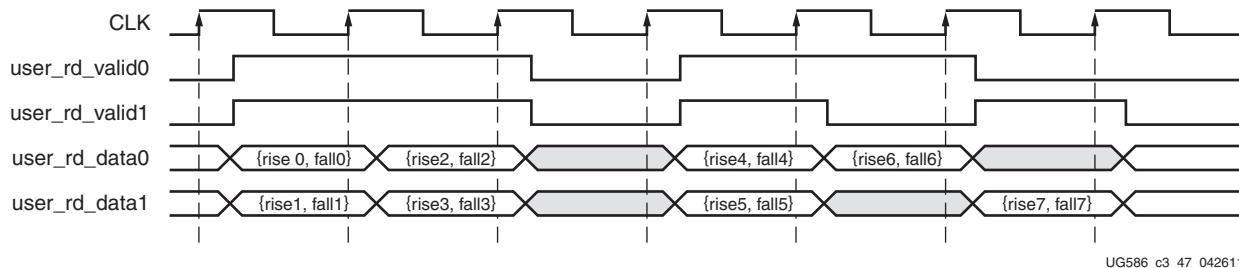
A command request is issued by asserting `user_cmd_en0` as a single cycle pulse. At this time, the `user_rd_cmd0`, `user_addr0`, and `user_ba0` signals must be valid. To issue a read request, `user_rd_cmd0` is asserted active High, while for a write request, `user_rd_cmd0` is kept Low. For a write request, 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 3-25](#). Any gaps in the command flow required can be filled with read commands, if desired.



UG586\_c3\_46\_042611

**Figure 3-25: 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 3-26](#). 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.



UG586\_c3\_47\_042611

**Figure 3-26: Client Interface Protocol Read Data**

## Clocking Architecture

The PHY design requires that a PLL module be used to generate various clocks. 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) logic
- Read path (input) and delay logic
- IDELAY reference clock (200 MHz)

One PLL is required for the PHY. The PLL generates the clocks for most of the internal logic, the input clocks to the phasers, and a synchronization pulse required to keep the PHASER blocks synchronized in a multi-I/O bank implementation.

The PHASER blocks require three clocks, a memory reference clock, a frequency reference clock and a phase reference clock from the PLL. The memory reference clock is required to be at the same frequency as that of the RLDRAM II interface clock. The frequency reference clock must be either 2x or 4x the memory clock frequency such that it meets the frequency range requirement of 400 MHz to 1066 MHz. The phase reference clock is used in the read banks, and is generated using the memory read clock (CQ/CQ#) routed internally and provided to the Phaser logic to assist with data capture.

The internal fabric clock generated by the PLL is clocked by a global clocking resource at half the frequency of the RDRAM II memory frequency.

A 200 MHz IDELAY reference clock must be supplied to the IDELAYCTRL module. The IDELAYCTRL module continuously calibrates the IDELAY elements in the I/O region to account for varying environmental conditions. The IP 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` module. This ensures that the clock is stable before being used.

[Table 3-12](#) lists the signals used in the infrastructure module that provides the necessary clocks and reset signals required in the design.

**Table 3-12: Infrastructure Clocking and Reset Signals**

Signal	Direction	Description
<code>mmcm_clk</code>	Input	System clock input
<code>sys_RST</code>	Input	Core reset from user application
<code>iodelay_ctrl_rdy</code>	Input	IDEDELAYCTRL lock status
<code>clk</code>	Output	Half frequency fabric clock
<code>mem_Refclk</code>	Output	PLL output clock at same frequency as the memory clock
<code>freq_Refclk</code>	Output	PLL output clock to provide the FREQREFCLK input to the Phaser. The freq_Refclk is generated such that its frequency in the range of 400 MHz - 1066 MHz
<code>sync_pulse</code>	Output	PLL output generated at 1/16 of mem_Refclk and is a synchronization signal sent to the PHY hard blocks that are used in a multi-bank implementation
<code>pll_locked</code>	Output	Locked output from PLLE2_ADV
<code>rstdiv0</code>	Output	Reset output synchronized to internal fabric half-frequency clock.
<code>rst_Phaser_Ref</code>	Output	Reset for the Phaser in the Physical Layer.

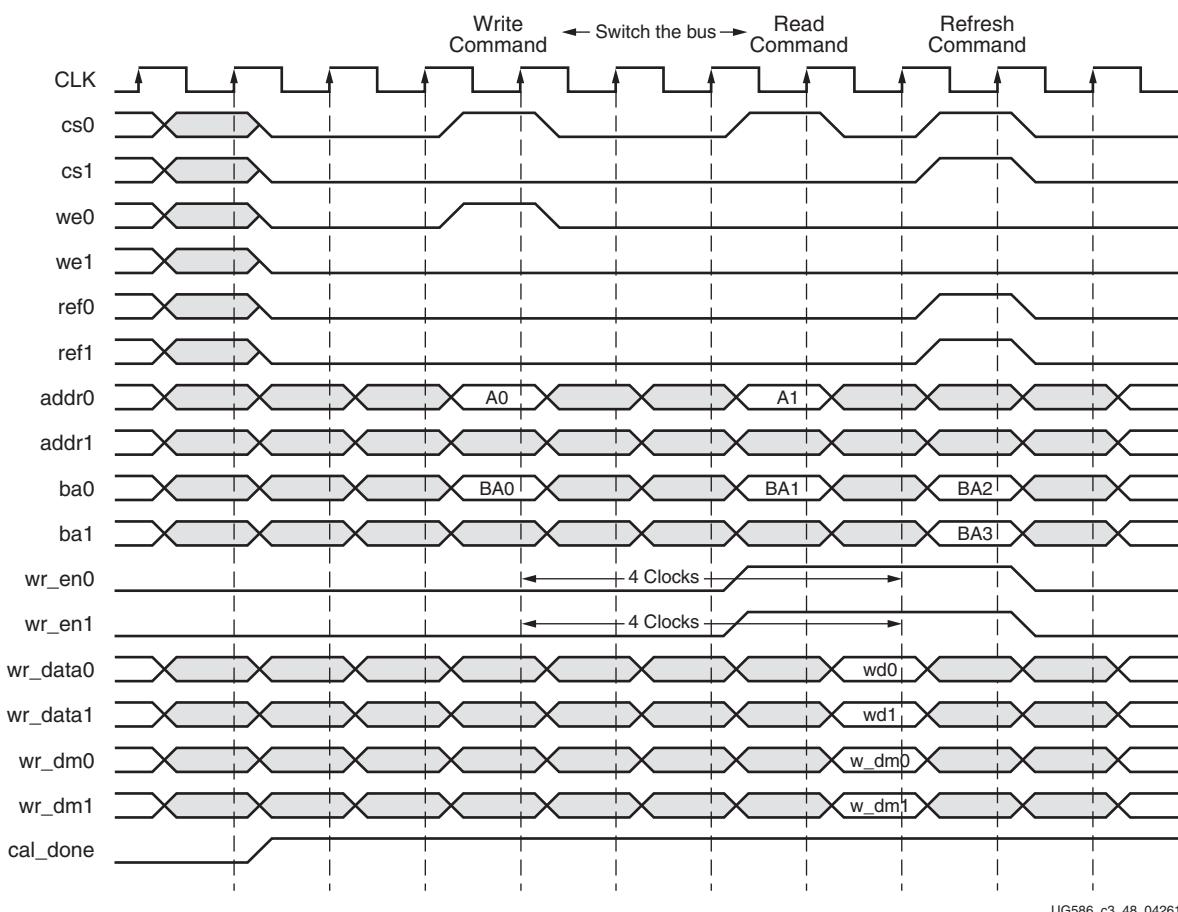
## Physical Interface

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 defined in [Table 3-13](#). These signals can be directly connected to the corresponding signals on the RLDRAM II device.

**Table 3-13: Physical Interface Signals**

<b>Signal</b>	<b>Direction</b>	<b>Description</b>
rldii_ck_p	Output	System Clock CK. This is the address/command clock to the memory device.
rldii_ck_n	Output	System Clock CK#. This is the inverted system clock to the memory device.
rldii_dk_p	Output	Write Clock DK. This is the write clock to the memory device.
rldii_dk_n	Output	Write Clock DK#. This is the inverted write clock to the memory device.
rldii_a	Output	Address. This is the address supplied for memory operations.
rldii_ba	Output	Bank Address. This is the bank address supplied for memory operations.
rldii_cs_n	Output	Chip Select CS#. This is the active-Low chip select control signal for the memory.
rldii_we_n	Output	Write Enable WE#. This is the active-Low write enable control signal for the memory.
rldii_ref_n	Output	Refresh REF#. This is the active-Low refresh control signal for the memory.
rldii_dm	Output	Data Mask DM. This is the active-High mask signal, driven by the FPGA to mask data that a user does not want written to the memory during a write command.
rldii_dq	Input/Output	Data DQ. This is a bidirectional data port, driven by the FPGA for writes and by the memory for reads.
rldii_qvld	Input	Read Data Valid QVLD. This signal indicates that the data on the rld_dq bus is valid.
rldii_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.
rldii_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.

Figure 3-27 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.



UG586\_c3\_48\_042611

**Figure 3-27: 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 3-26.

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 ensures 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 transitioning from a write to a read, and from a read to a write. This two clock cycle requirement depends on the PCB and might need to be increased for different board layouts.

## Memory Controller

The memory controller enforces the RLDRAM II access requirements and interfaces 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 immediately 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.

## PHY Architecture

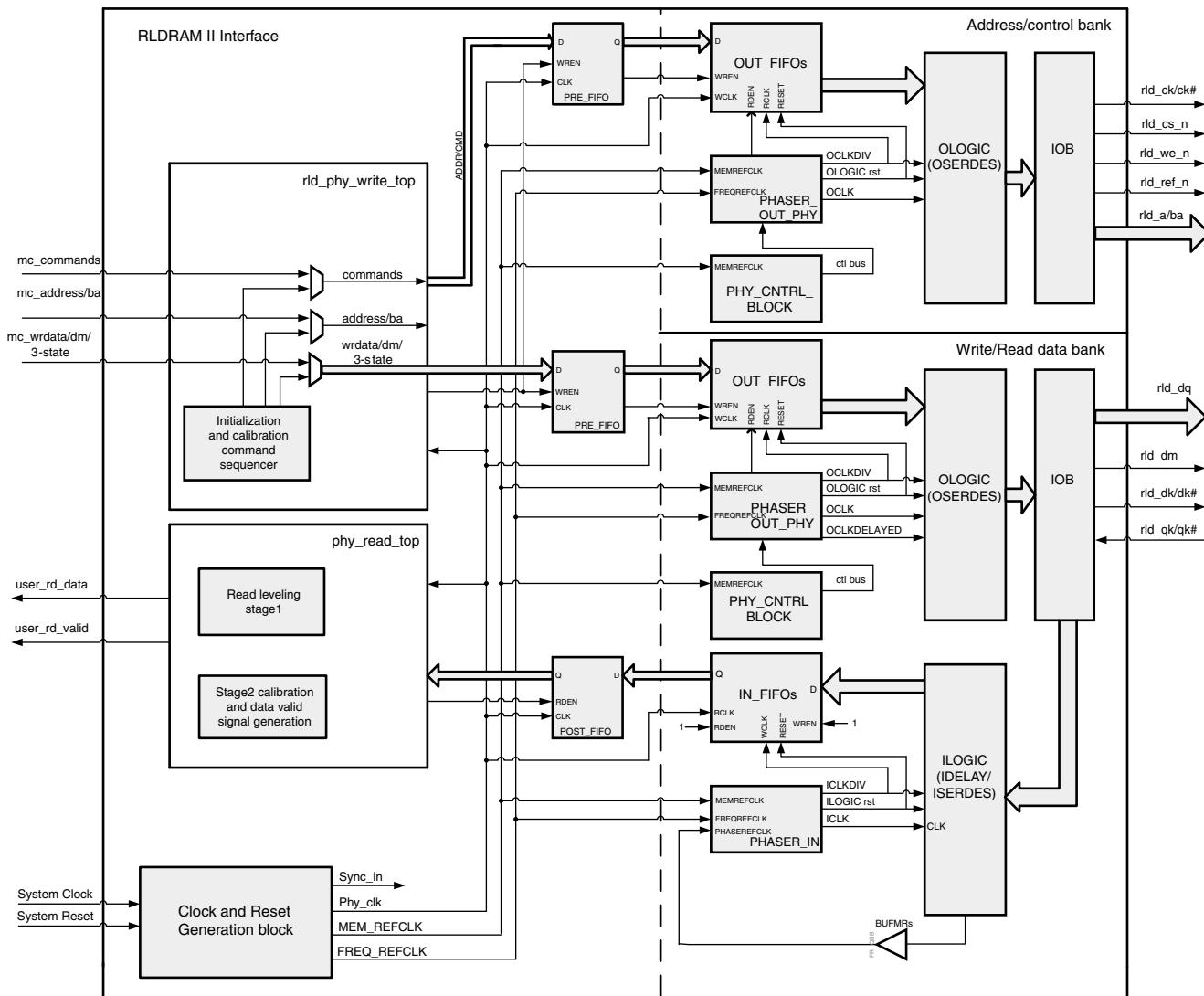
The PHY consists of dedicated blocks and soft calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high-performance physical layers.

Some of the dedicated blocks that are used in the RLDRAM II PHY and their features are described below:

- I/Os available within each FPGA bank are grouped into four byte groups, where each byte group consists of up to 12 I/Os.

- PHASER\_IN/PHASER\_OUT blocks are available in each byte group and are multistage programmable delay line loops that can provide precision phase adjustment of the clocks. Dedicated clock structures within an I/O bank, referred to as byte group clocks, generated by the PHASERS help minimize the number of loads driven by the byte group clock drivers.
- OUT\_FIFO and IN\_FIFO are shallow eight-deep FIFOs available in each byte group and serve to transfer data from the fabric domain to the I/O clock domain. OUT\_FIFOs are used to store output data and address/controls that need to be sent to the memory while IN\_FIFOs are used to store captured read data before transfer to the fabric.

[Pinout Requirements, page 272](#) explains the rules that need to be followed when placing the memory interface signals inside the byte groups.



UG586\_c3\_04\_051811

Figure 3-28: High-Level PHY Block Diagram of the RLDRAM II Interface Solution

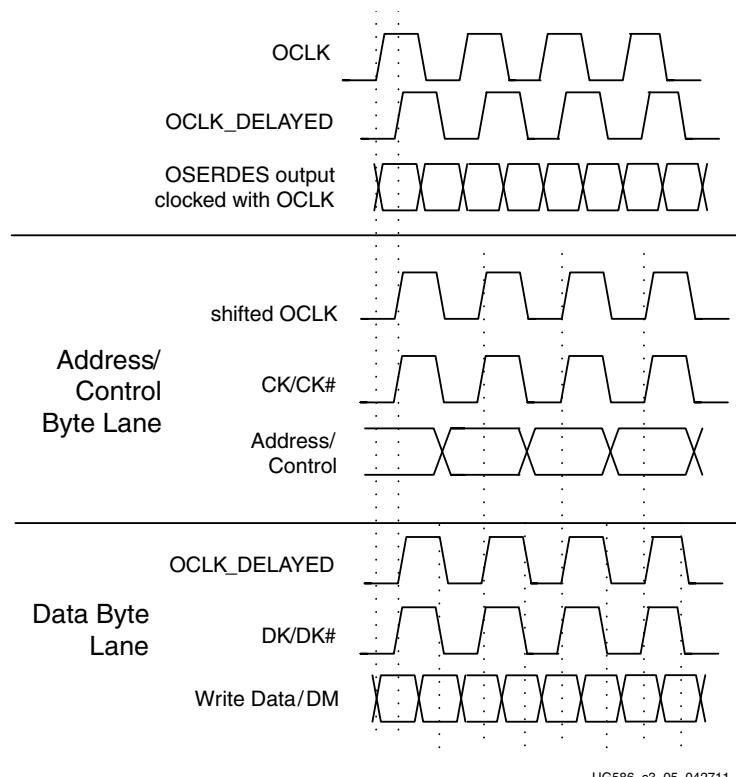
## Write Path

The write path to the RLDRAM II includes the address, data, and control signals necessary to execute any memory operation. The control strobes rldii\_cs\_n, rldii\_we\_n, and rldii\_ref\_n, and addresses rldii\_a and rldii\_ba to the memory all use SDR formatting. The write data values rldii\_dq and rldii\_dm also utilize DDR formatting to achieve the required four-word or eight-word burst within the given clock periods.

## Output Architecture

The output path of the RLDRAM II interface solution uses OUT\_FIFOs, PHASER\_OUT\_PHY, PHY\_CNTRL, and OSERDES primitives available in 7 series FPGAs. These blocks are used for clocking all outputs of the PHY to the memory device.

The PHASER\_OUT\_PHY block provides the clocks required to clock out the outputs to the memory. It provides synchronized clocks for each byte group, to the OUT\_FIFOs and to the OSERDES/ODDR. PHASER\_OUT\_PHY generates the byte clock (OCLK), the divided byte clock (OCLKDIV), and a delayed byte clock (OCLK\_DELAYED) for its associated byte group. The byte clock (OCLK) is the same frequency as the memory interface clock and the divided byte clock (OCLKDIV) is half the frequency of the memory interface clock. The byte clock (OCLK) is used to clock the Write data (DQ), Data Mask (DM), Address, controls, and system clock (CK/CK#) signals to the memory from the OSERDES/ODDR. The PHASER\_OUT\_PHY output, OCLK\_DELAYED, is a 90-degree phase-shifted output with respect to the byte clock (OCLK) and is used to generate the write clock (DK/DK#) to the memory. [Figure 3-29](#) shows the alignment of the various clocks and how they are used to generate the necessary signal alignment.



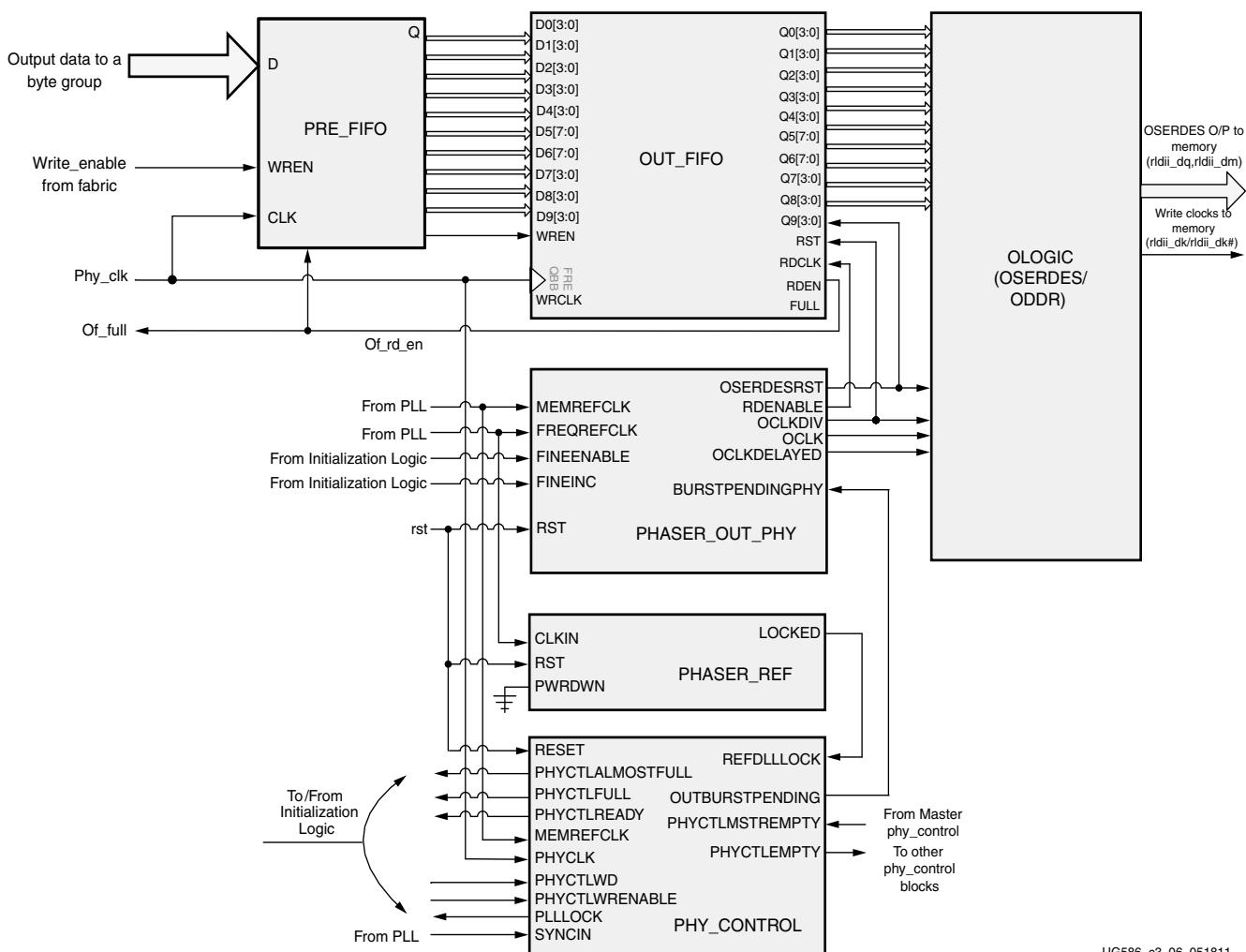
*Figure 3-29: Write Path Output Alignment*

UG586\_c3\_05\_042711

OCLK\_DELAYED generates a center-aligned clock for DDR write data but it does not produce an ideal alignment for SDR address/control signals. For this reason, OCLK is used to generate CK/CK#, and the address/control byte lanes are shifted so that the OCLK of these byte lanes are aligned with the OCLK\_DELAYED of write data banks.

The OUT\_FIFO serves as a temporary buffer to convert the write data from the fabric domain to the PHASER clock domain, which clocks out the output data from the I/O logic. The OUT\_FIFO runs in asynchronous mode, with the read and write clocks running at the same frequency yet an undetermined phase. A shallow, synchronous PRE\_FIFO drives the OUT\_FIFO with continuous data from the fabric in an event of a flag assertion from the OUT\_FIFO, which might potentially stall the flow of data through the OUT\_FIFO. The clocks required for operating the OUT\_FIFOs and OSERDES are provided by PHASER\_OUT\_PHY.

The clocking details of the write paths using PHASER\_OUT\_PHY are shown in [Figure 3-30](#). The PHY Control block is used to ensure proper startup of all PHASER\_OUT\_PHY blocks used in the interface.



UG586\_c3\_06\_051811

**Figure 3-30: Write Path Block Diagram of the RLDRAM II Interface Solution**

The OSERDES blocks available in every I/O simplifies generation of 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, and then passed through a parallel-to-serial conversion block. The OSERDES is used to clock all outputs from the PHY to the memory device. Upon exiting the OSERDES, all output signals must be presented center-aligned with respect to the generated clocks (CK/CK# for address/control signals, DK/DK# for data and data mask). For this reason, the PHASER\_OUT\_PHY block is also used in conjunction with the OSERDES to achieve center alignment.

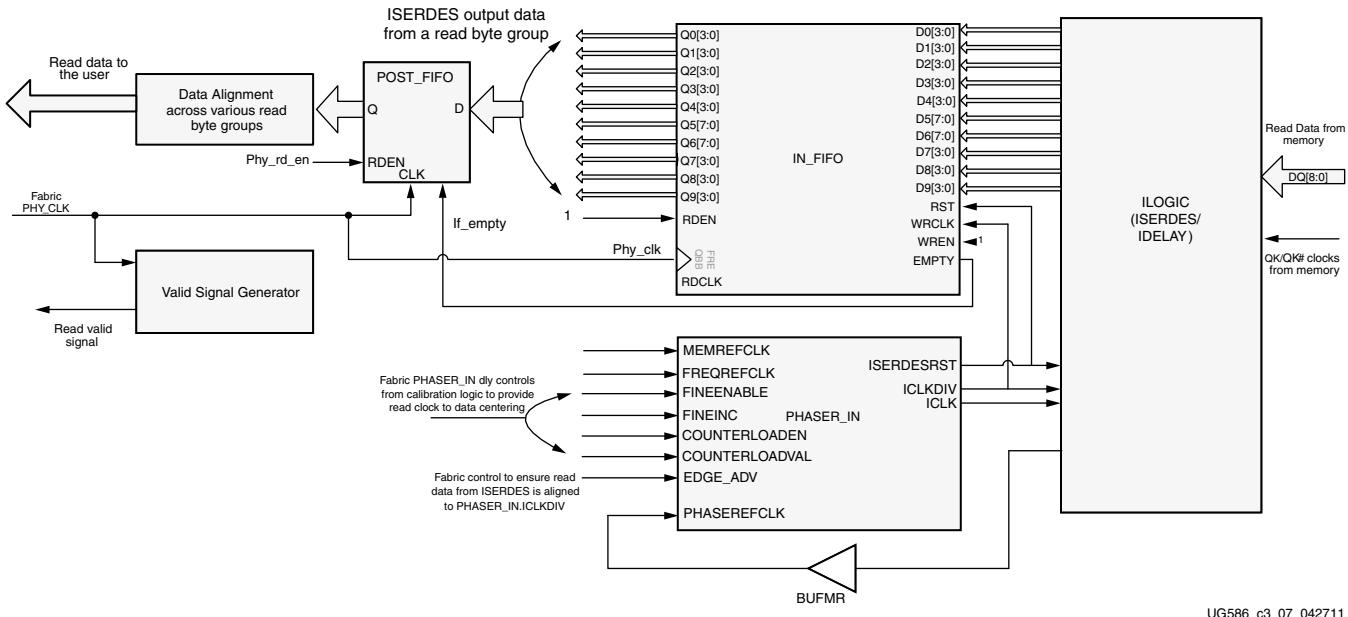
## Read Path

The read path includes data capture using the memory-provided read clocks and also ensures that the read clock is centered within the data window for good margin during data capture. Before any read can take place, calibration must occur. Calibration is the main function of the read path and needs to be performed before the user interface can start transactions to the memory.

### Data Capture

[Figure 3-31](#) shows a high-level block diagram of the path the read clock and the read data take from entering the FPGA until given to the user. The read clock bypasses the ILOGIC and is routed through PHASERs within each byte group through multiregion BUFMRs. The BUFMR output can drive the PHASERREFCLK inputs of the PHASERs in the immediate bank and also the PHASERs available in the bank above and below the current bank. The PHASER generated byte group clocks (ICLK and ICLKDIV) are then used to capture the read data (DQ) available within the byte group using the ISERDES block. The calibration logic makes use of the fine delay increments available through the PHASER to ensure the byte group clock, ICLK, is centered inside the read data window, ensuring maximum data capture margin.

IN\_FIFOs available in each byte group (shown in [Figure 3-31](#)) receive 4-bit data from each DQ bit captured in the ISERDES in a given byte group and write them into the storage array. The half-frequency PHASER\_IN generated byte group clock, ICLKDIV, that captures the data in the ISERDES is also used to write the captured read data to the IN\_FIFO. The write enables to the IN\_FIFO are always asserted to enable input data to be continuously written. A shallow, synchronous post\_fifo is used at the receiving side of the IN\_FIFO to enable captured data to be read out continuously from the fabric, in an event of a flag assertion in the IN\_FIFO which might potentially stall the flow of data from the IN\_FIFO. Calibration also ensures that the read data is aligned to the rising edge of the fabric half-frequency clock and that read data from all the byte groups have the same delay. More details about the actual calibration and alignment logic is explained in [Calibration](#).



UG586\_c3\_07\_042711

**Figure 3-31: Read Path Block Diagram of the RLDRAM II Interface Solution**

## Calibration

The calibration logic includes providing the required amount of delay on the read clock and read data to align the clock in the center of the data valid window. The centering of the clock is done using PHASERs, which provide very fine resolution delay taps on the clock. Each PHASER\_IN fine delay tap increments the clock by 1/64th of the reference clock period with a maximum of 32 taps possible.

Calibration begins after memory initialization. Prior to this point, all read path logic is held in reset. Calibration is performed in two stages:

1. Calibration of read clock with respect to DQ.
2. Data alignment and valid generation.

## Calibration of Read Clock and Data

PHASER\_IN clocks all ISERDES used to capture read data (DQ) associated with the corresponding byte group. ICLKDIV is also the write clock for the read data IN\_FIFOs. One PHASER\_IN block is associated with a group of 12 I/Os. Each I/O bank in the FPGA has four PHASER\_IN blocks, and hence four read data bytes can be placed in a bank.

## Implementation Details

This stage of read leveling is performed one byte at a time, where the read clock is center-aligned to the corresponding read data in that byte group. At the start of this stage, a write command is issued to a specified RLDRAM II address location with a specific data pattern. This write command is followed by back-to-back read commands to continuously read data back from the same address location that was written to.

The calibration logic reads data out of the IN\_FIFO and records it for comparison. The calibration logic checks for the sequence of the data pattern read, to determine the alignment of the clock with respect to the data. No assumption is made about the initial

relationship between the capture clock and the data window at tap 0 of the fine delay line. The algorithm tries to align the clock to the left edge of the data window, by delaying the read data through the IDELAY element.

Next, the clock is delayed using the PHASER taps and centered within the corresponding data window. The PHASER\_TAP resolution is based on the FREQ\_REF\_CLK period, and the per-tap resolution is equal to  $(\text{FREQ\_REFCLK\_PERIOD}/2)/64$  ps. For memory interface frequencies greater than or equal to 400 MHz, using the maximum of 64 PHASER taps can provide a delay of one data period or one-half the clock period. This enables the calibration logic to accurately center the clock within the data window.

For frequencies less than 400 MHz, because FREQ\_REF\_CLK has twice the frequency of MEM\_REF\_CLK, the maximum delay that can be derived from the PHASER is one-half the data period or one-fourth the clock period. Hence for frequencies less than 400 MHz, just using the PHASER delay taps might not be sufficient to accurately center the clock in the data window. For these frequency ranges, a combination of both data delay using IDELAY taps and PHASER taps is used. The calibration logic determines the best possible delays, based on the initial clock-data alignment. The algorithm first delays the read capture clock using the PHASER\_IN fine delay line until a data window edge is detected.

An averaging algorithm is used for data window detection, where data is read back over multiple cycles at the same tap value. The number of sampling cycles is set to 214. In addition to averaging, there is a counter that tracks whether the read capture clock is positioned in the unstable jitter region. A counter value of 3 means that the sampled data value is constant for three consecutive tap increments and the read capture clock is considered to be in a stable region. The counter value is reset to 0 whenever a value different from the previous value is detected.

The next step is to increment the fine phase shift delay line of the PHASER\_IN block one tap at a time until a data mismatch is detected. The data read out of IN\_FIFO after the required settling time is then compared with the recorded data at the previous tap value. This is repeated until a data mismatch is found, indicating detection of a valid data window edge. A valid window is the number of PHASER\_IN fine phase shift taps for which the stable counter value is a constant 3. This algorithm mitigates the risk of detecting a false valid edge in the unstable jitter regions.

## Data Alignment and Valid Generation

This phase of calibration:

- Ensures read data from all the read byte groups is aligned to the rising edge of the ISERDES CLKDIV capture clock
- Sets the latency for fixed-latency mode (supported for the PHY-only interface).
- Matches the latency for each memory when wider memories are derived from small memories.
- Sends the determined latency to the read valid generation logic.

After read data capture clock centering is achieved, the calibration logic writes out a known data pattern to the RLDRAM II device and issues continuous reads back from the memory. This is done to determine whether the read data comes back aligned to the positive edge or negative edge of the ICLKDIV output of the PHASER\_IN. Captured data from a byte group that is aligned to the negative edge, is made to align to the positive edge using the EDGE\_ADV input to the PHASER\_IN, which shifts the ICLKDIV output by one fast clock cycle.

The next stage is to generate the valid signal associated with the data on the client interface. During this stage of calibration, a burst-of-eight data pattern is written to memory and read back. This phase 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 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 init\_calib\_complete.

## Customizing the Core

The 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. These parameters are summarized in [Table 3-15](#).

**Table 3-15: RLDRAM II Memory Interface Solution Configurable Parameters**

Parameter	Value	Description
CLK_PERIOD		Memory clock period (ps).
ADDR_WIDTH	19-22	Memory address bus width.
RLD_ADDR_WIDTH	11, 19-22	Physical Memory address bus width when using Address Multiplexing mode.
BANK_WIDTH	3	Memory bank address bus width.
DATA_WIDTH		Memory data bus width and can be set through the MIG tool. A maximum DATA_WIDTH of 72 is supported.
QK_WIDTH	2 per x18/x36 device	Memory read clock bus width.
DK_WIDTH	2 per x36 device 1 per x18 device	Memory write clock bus width.
BURST_LEN	4, 8	Memory data burst length.
DM_PORT	ON, OFF	This parameter enables and disables the generation of the data mask ports.
NUM_DEVICES		Number of memory devices used.
MRS_CONFIG	1, 2, 3	This parameter sets the configuration setting in the RLDRAM II memory register.
MRS_ADDR_MUX	ON, OFF	This parameter sets the address multiplexing setting in the RLDRAM II memory register.
MRS_DLL_RESET	DLL_ON	This parameter sets the DLL setting in the RLDRAM II memory register.

Table 3-15: RLDRAM II Memory Interface Solution Configurable Parameters

Parameter	Value	Description
MRS_IMP_MATCH	INTERNAL, EXTERNAL	This parameter sets the impedance setting in the memory register.
MRS_ODT	ON, OFF	This parameter sets the ODT setting in the memory register.
MEM_TYPE	RLD2_CIO	This parameter specifies the memory type.
IODELAY_GRP		This is a unique name for the IODELAY_CTRL provided when multiple IP cores are used in the design.
REFCLK_FREQ	200.0	Reference clock frequency for IODELAYCTRLs.
BUFMR_DELAY		Simulation-only parameter used to model buffer delays.
RST_ACT_LOW	0,1	Active Low or active High reset.
IBUF_LPWR_MODE	ON, OFF	Enables or disables low power mode for the input buffers.
IODELAY_HP_MODE	ON, OFF	Enables or disables high-performance mode within the IODELAY primitive. When set to OFF, IODELAY operates in low power mode at the expense of performance.
SYSCLK_TYPE	DIFFERENTIAL, SINGLE_ENDED	This parameter indicates whether the system uses single-ended or differential system 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_i must be used.
REFCLK_TYPE	DIFFERENTIAL, SINGLE_ENDED	This parameter indicates whether the system uses single-ended or differential reference clocks. Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, ref_clk_p/ref_clk_n must be used. For single-ended clocks, ref_clk_i must be used.
CLKIN_PERIOD		Input clock period.
CLKFBOUT_MULT		PLL voltage-controlled oscillator (VCO) multiplier. This value is set by the MIG tool based on the frequency of operation.

Table 3-15: RLDRAM II Memory Interface Solution Configurable Parameters

Parameter	Value	Description
CLKOUT0_DIVIDE, CLKOUT1_DIVIDE, CLKOUT2_DIVIDE, CLKOUT3_DIVIDE		VCO output divisor for PLL outputs. This value is set by the MIG tool based on the frequency of operation.
DIVCLK_DIVIDE		MMCM VCO divisor. This value is set by the MIG tool based on the frequency of operation.
SIM_BYPASS_INIT_CAL	SKIP, FAST, NONE	This simulation-only parameter is used to speed up simulations, by skipping the initialization wait time and speeding up calibration.
SIMULATION	"TRUE", "FALSE"	Set to "TRUE" for simulation; set to "FALSE" for implementation.
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 through 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 simulation.
N_DATA_LANES	DATA_WIDTH/9	Calculated number of data byte lanes, used to set up signal widths for using the debug port.
DIFF_TERM_SYSCLK	"TRUE", "FALSE"	Differential Termination for System clock input pins
DIFF_TERM_REFCLK	"TRUE", "FALSE"	Differential Termination for IDELAY reference clock input pins
nCK_PER_CLK	2	Number of memory clocks per fabric clocks.
TCQ	100	Register delay for simulation.

Table 3-16 contains parameters set up by the MIG tool based on the pinout selected. When making pinout changes, it is recommended to rerun the MIG tool so the parameters are set up properly; otherwise see [Pinout Requirements, page 272](#). Mistakes to the pinout parameters can result in non-functional simulation, an unrouteable design, and/or trouble meeting timing. These parameters are used to set up the PHY and route all the necessary signals to and from it. The parameters are calculated based on Data and Address/Control byte groups selected. These parameters do not consider the System Signals selection (that is, system clock, reference clock, and status signals).

Table 3-16: RLDRAM II Memory Interface Solution Pinout Parameters

Parameter	Description	Example
BYTE_LANES_B0, BYTE_LANES_B1, BYTE_LANES_B2	Three fields, one per possible I/O bank. Defines the byte lanes being used in a given I/O bank. A "1" in a bit position indicates a byte lane is used, and a "0" indicates unused.	Ordering of bits from MSB to LSB is T0, T1, T2, and T3 byte groups. 4'b1101: Three byte lanes in use for a given bank, with one not in use.
DATA_CTL_B0, DATA_CTL_B1, DATA_CTL_B2	Three fields, one per possible I/O bank. Defines the byte lanes for a given I/O bank. A "1" in a bit position indicates a byte lane is used for data, and a "0" indicates it is used for address/control.	4'b1100: Two data byte lanes, and, if used with a BYTE_LANES_B0 parameter as in the example shown above, one address/control.
CPT_CLK_SEL_B0, CPT_CLK_SEL_B1, CPT_CLK_SEL_B2	Three fields, one per possible I/O bank. Defines which read capture clocks are used for each byte lane in given bank. MRCC read capture clocks are placed in byte lanes 1 and/or 2, where parameter is defined for each data byte lane to indicate which read clock to use for the capture clock. 8 bits per byte lane, defined such that: <ul style="list-style-type: none"> <li>[3:0] - 1, 2 to indicate which of two capture clock sources</li> <li>[7:4] - 0 (bank below), 1 (current bank), 2 (bank above) to indicate in which bank the clock is placed.</li> </ul>	32'h12_12_11_11: Four data byte lanes, all using the clocks in the same bank. 32'h21_22_11_11: Four data byte lanes, two lanes using the capture clock from the bank above (16'h21_22), two using the capture clock from the current bank (16'h11_11).
PHY_0_BITLANES, PHY_1_BITLANES, PHY_2_BITLANES	Three fields, one per possible I/O bank. 12-bit parameter per byte lane used to determine which I/O locations are used to generate the necessary PHY structures. This parameter is provided per bank. Except for the QVLD pin, all Data and Address/Control pins are considered for this parameter generation.	This parameter is denoted for all byte groups of a selected bank. All 12 bits are denoted for a byte lane. For example, this parameter is 48'hFFE_FFF_000_DF6 for one bank. 12'hBFC (12'b1011_1111_1100): bit lanes 0, 1, and 10 are not used, all others are used.

Table 3-16: RLDRAM II Memory Interface Solution Pinout Parameters (Cont'd)

Parameter	Description	Example
CK_MAP	<p>Bank and byte lane position information for the CK/CK#. 8-bit parameter provided per pair of signals.</p> <ul style="list-style-type: none"> <li>[3:0] - Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> <li>[7:4] - Bank position. Values of 0, 1, or 2 are supported</li> </ul>	<p>Upper-most Data or Address/Control byte group selected bank is referred to as Bank 0 in parameters notation. Numbering of banks is 0, 1, and 2 from top to bottom.</p> <p>Byte groups T0, T1, T2, and T3 are numbered in parameters as 3, 2, 1, and 0, respectively.</p> <p>96 'h00_00_00_00_00_00_00_00_00_00_00_10_13: This parameter is denoted for 12 clock pairs with 8 bits for each clock pin. In this case, two clock pairs are used. Ordering of parameters is from MSB to LSB (that is, CK[0]/ CK#[0] corresponds to the 8 LSBs of the parameter).</p> <p>8 'h13: CK/CK# placed in bank 1, byte lane 3.</p> <p>8 'h20: CK/CK# placed in bank 2, byte lane 0.</p>
DK_MAP	<p>Bank and byte lane position information for the DK/DK#. 8-bit parameter provided per pair of signals.</p> <ul style="list-style-type: none"> <li>[3:0] - Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> <li>[7:4] - Bank position. Values of 0, 1, or 2 are supported</li> </ul>	<p>See the <a href="#">CK_MAP</a> example for parameter values notation.</p> <p>8 'h13: DK/DK# placed in bank 1, byte lane 3.</p> <p>8 'h20: DK/DK# placed in bank 2, byte lane 0.</p>
QK_MAP	<p>Bank and byte lane position information for the QK/QK#. 8-bit parameter provided per pair of signals.</p> <ul style="list-style-type: none"> <li>[3:0] - Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> <li>[7:4] - Bank position. Values of 0, 1, or 2 are supported</li> </ul>	<p>See the <a href="#">CK_MAP</a> example for parameter values notation.</p> <p>8 'h11: QK/QK# placed in bank 1, byte lane 1.</p> <p>8 'h22: QK/QK# placed in bank 2, byte lane 2.</p>

**Table 3-16: RLDRAM II Memory Interface Solution Pinout Parameters (Cont'd)**

**Table 3-16: RLDRAM II Memory Interface Solution Pinout Parameters (Cont'd)**

Parameter	Description	Example
DQTS_MAP	Bank and byte lane position information for the 3-state control. See <a href="#">CS_MAP</a> description.	See <a href="#">CS_MAP</a> example
DM_MAP	Bank and byte lane position information for the data mask. See <a href="#">CS_MAP</a> description.	See <a href="#">CS_MAP</a> example
QVLD_MAP	Bank and byte lane position information for the QVLD. See <a href="#">CS_MAP</a> description.	See <a href="#">CS_MAP</a> example
DATA0_MAP, DATA1_MAP, DATA2_MAP, DATA3_MAP, DATA4_MAP, DATA5_MAP, DATA6_MAP, DATA7_MAP	Bank and byte lane position information for the data bus. See <a href="#">CS_MAP</a> description.	See <a href="#">CS_MAP</a> example

## Design Guidelines

### Design Rules

Memory types, memory parts, and data widths are restricted based on the selected FPGA, FPGA speed grade, and the design frequency. The final frequency ranges are subject to characterization results.

### 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. These internal delays can be found using the Pinout and Area Constraints Editor (PACE) tool. These rules indicate the maximum electrical delays between RLDRAM II signals:

- The maximum skew between any DQ/DM and DK/DK# should be  $\pm 15$  ps.
- The maximum skew between any DQ and its associated QK/QK# should be  $\pm 15$  ps.
- The maximum skew between any address and control signals and the corresponding CK/CK# should be  $\pm 50$  ps.
- The maximum skew between any DK/DK# and CK/CK# should be  $\pm 25$  ps.

### Pinout Requirements

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the RLDRAM II physical layer. Xilinx 7 series FPGAs have dedicated logic for each byte group. Four byte groups are available in each 50-pin bank. Each 50-pin bank consists of four byte groups that contain 1 DQS clock-capable I/O

pair and 10 associated I/Os. Two pairs of multiregion clock-capable I/O (MRCC) pins are available in a bank, and are used for placing the read clocks (QK/QK#).

In a typical RLDRAM II data bank configuration, 9 of these 10 I/Os are used for the data (DQ) and one can be used for the data mask (DM). The write clocks (DK/DK#) use one of the DQSCCIO pairs inside the data bank. QK/QK# clocks must be placed on MRCC pins in a given data bank or in the bank above or below the data. QVLD is not used in the design but should be placed on a free pin in a data or address/control bank for future use. Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, RLDRAM II interfaces must be arranged in the banks vertically and not horizontally. In addition, the maximum height is three banks.

After a core is generated through the MIG tool, the most optimal pinout 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 CK/CK# clocks must be placed in an address/control byte lane. The CK/CK# clocks also need to be placed on a DQSCCIO pin pair. CK must be placed on the P location, and CK# must be placed on the N location.
- The DK/DK# clocks must be placed in a data byte lane. The DK/DK# clocks also need to be placed on a DQSCCIO pin pair. DK must be placed on the P location, and DK# must be placed on the N location.
- Data (DQ) is placed such that all signals corresponding to one byte (9 bits) are placed inside a byte group. DQ must not be placed on the DQSCCIO N location in a byte lane, because this location is used for the 3-state control.
- It is recommended to keep all the data generated from a single memory component within a bank.
- Read clocks (QK and QK#) need to be placed on the MRCC pins that are available in each bank, respectively. Data must be in the same bank as the associated QK/QK#, or in the bank above or below.
- Address/control signals can be placed in byte groups that are not used for data and all should be placed in the same bank.
- For a given byte lane, the DQSCC\_N location is used to generate the 3-state control signal. The 3-state can share the location with QVLD, DK#, or DM only data.
- The system clock input must be in the same column as the memory interface. The system clock input is strongly recommended to be in the address/control bank. If this is not possible, the system clock input must be in the bank above or below the address/control bank.

## Manual Pinout Changes

For manually manipulating the parameters described in [Table 3-16](#), the following examples show how to allocate parameters for a given byte lane. [Table 3-17](#) shows a typical data byte lane, indicating the bank, byte lane, and bit position for each signal.

**Table 3-17: Example Byte Lane #1**

Bank	Byte Lane	Bit	DDR	Byte Group	I/O Type	I/O Number	Special Designation	BITLANES		
0	0	9	VREF	A_11	P	12	VREF	1 1 1 1 1 1 1 1 1 1 1 1	0	0
		8	DQ8	A_10	N	11			1	1
		7	DQ7	A_09	P	10			0	0
		6	DQ6	A_08	N	9			0	1
		B	DK0_P	A_07	P	8	DQSCC-P		1	0
		A	DK0_N	A_06	N	7	DQSCC-N		0	1
		5	DQ5	A_05	P	6			1	1
		4	DQ4	A_04	N	5			1	F
		3	DQ3	A_03	P	4			1	
		2	DQ2	A_02	N	3			1	
		1	DQ1	A_01	P	2			1	1111
		0	DQ0	A_00	N	1			1	F
			VRN	N/A	SE	0				

The byte lane parameters for [Table 3-17](#) are shown in [Table 3-18](#).

**Table 3-18: Parameters for Example Data Byte Lane #1**

Parameter	Value
DK_MAP	8'h00
DQTS_MAP	12'h00A
PHY_0_BITLANES	12'h1FF
DATA0_MAP	108'h008_007_006_005_004_003_002_001_000

[Table 3-19](#) shows another data byte lane, with QVLD placed in the 3-state location, which is valid, as the 3-state location only uses the OSERDES location of the I/O.

**Table 3-19: Example Byte Lane #2**

Bank	Byte Lane	Bit	DDR	Byte Group	I/O Type	I/O Number	Special Designation	BITLANES		
0	1	9	QK0_P	B_11	P	24	CCIO-P	1 1 1 1 0 1 1 1 1 1 1 1	0	0 0 1 1 0 1111 F 1111 F 8FF
		8	QK0_N	B_10	N	23	CCIO-N		0	
		7	DQ17	B_09	P	22	CCIO-P		0	
		6	DQ16	B_08	N	21	CCIO-N		0	
		B	DQ15	B_07	P	20	DQSCC-P		1	
		A	QVLD	B_06	N	19	DQSCC-N		1	
		5	DQ14	B_05	P	18			1	
		4	DQ13	B_04	N	17			1	
		3	DQ12	B_03	P	16			1	
		2	DQ11	B_02	N	15			1	
		1	DQ10	B_01	P	14			1	
		0	DQ9	B_00	N	13			1	

The byte lane parameters for [Table 3-19](#) are shown in [Table 3-20](#).

**Table 3-20: Parameters for Example Data Byte Lane #2**

Parameter	Value
QVLD_MAP	12'h01A
DQTS_MAP	12'h01A
PHY_0_BITLANES	12'h8FF
DATA1_MAP	108'h017_016_01B_015_014_013_012_011_010
QK_MAP	8'h01

[Table 3-21](#) shows the same byte lane as [Table 3-19](#), but instead of QVLD, the Data Mask (DM) is placed in this byte lane. While the DM can share the OSERDES location with the 3-state control, they cannot share the same location in the OUT\_FIFO in the PHY. Thus some signals from the OUT\_FIFO have to shift as shown in [Table 3-21](#). In this case, the direction of the shift is determined on the byte lane location, with byte lanes 0, 1 shifted up, and 2, 3 shifted down. In this case, the PHY merges the 3-state control with the DM to share the same OSERDES location.

Table 3-21: Example Byte Lane #3, Shared 3-State with DM in Byte Lane #1

Bank	Byte Lane	Bit	MAP	DDR	Byte Group	I/O Type	I/O Number	Special Designation	BITLANES	
UCF										
0	1	9		QK0_P	B_11	P	24	CCIO-P	0	
		8	DQ17	QK0_N	B_10	N	23	CCIO-N	1	
		7	DQ16	DQ17	B_09	P	22	CCIO-P	1	
		6	DQ15	DQ16	B_08	N	21	CCIO-N	1	
		B	3-state	DQ15	B_07	P	20	DQSCC-P	0	0101
		A	DM	DM	B_06	N	19	DQSCC-N	1	5
		5	DQ14	DQ14	B_05	P	18		1	1111
		4	DQ13	DQ13	B_04	N	17		1	F
		3	DQ12	DQ12	B_03	P	16		1	
		2	DQ11	DQ11	B_02	N	15		1	
		1	DQ10	DQ10	B_01	P	14		1	1111
		0	DQ9	DQ9	B_00	N	13		1	F

The byte lane parameters for Table 3-21 are shown in Table 3-22.

Table 3-22: Parameters for Example Data Byte Lane #3

Parameter	Value
DM_MAP	12'h01A
DQTS_MAP	12'h01B
PHY_0_BITLANES	12'h5FF
DATA1_MAP	108'h018_017_016_015_014_013_012_011_010
QK_MAP	8'h01

[Table 3-23](#) shows another byte lane with the QVLD sharing the location of the 3-state control (or leaving the location unused).

**Table 3-23: Example Byte Lane #4, Shared 3-State with QVLD**

Bank	Byte Lane	Bit	MAP	DDR	Byte Group	I/O Type	I/O Number	Special Designation	BITLANES	
UCF										
0	2	9	DQ26	DQ26	C_11	P	12		1	
		8	DQ25	DQ25	C_10	N	11		1	
		7	DQ24	DQ24	C_09	P	10		1	
		6	DQ23	DQ23	C_08	N	9		1	
		B	DQ22	DQ22	C_07	P	8	DQSCC-P	1	
		A	<b>QVLD</b>	<b>QVLD</b>	C_06	N	7	DQSCC-N	0	
		5	DQ21	DQ21	C_05	P	6		1	1111
		4	DQ20	DQ20	C_04	N	5		1	F
		3	DQ19	DQ19	C_03	P	4	CCIO-P	1	
		2	DQ18	DQ18	C_02	N	3	CCIO-N	1	
		1		<b>QK1_P</b>	C_01	P	2	CCIO-P	0	1100
		0		<b>QK1_N</b>	C_00	N	1	CCIO-N	0	C
										<b>BFC</b>

The byte lane parameters for [Table 3-23](#) are shown in [Table 3-24](#).

**Table 3-24: Parameters for Example Data Byte Lane #4**

Parameter	Value
DQTS_MAP	12'h02A
PHY_0_BITLANES	12'hBFC
DATA1_MAP	108'h029_028_027_026_02B_025_024_023_022
QK_MAP	8'h02

[Table 3-25](#) shows the same byte lane as [Table 3-23](#), but instead of QVLD the DM is placed in this byte lane. In this situation the signals are shifted down in the OUT\_FIFO.

**Table 3-25: Example Byte Lane #5, Shared 3-State with DM in Byte Lane #2**

Bank	Byte Lane	Bit	MAP	DDR	Byte Group	I/O Type	I/O Number	Special Designation	BITLANES	
UCF										
0	2	9	DQ26	DQ26	C_11	P	12		1	
		8	DQ25	DQ25	C_10	N	11		1	
		7	DQ24	DQ24	C_09	P	10		1	
		6	DQ23	DQ23	C_08	N	9		1	
		B	DQ22	DQ22	C_07	P	8	DQSCC-P	1	1111
		A	DM	DM	C_06	N	7	DQSCC-N	1	F
		5	3-state	DQ21	C_05	P	6		0	1101
		4	DQ21	DQ20	C_04	N	5		1	D
		3	DQ20	DQ19	C_03	P	4	CCIO-P	1	
		2	DQ19	DQ18	C_02	N	3	CCIO-N	1	
		1	DQ18	QK1_P	C_01	P	2	CCIO-P	1	1110
		0		QK1_N	C_00	N	1	CCIO-N	0	E

The byte lane parameters for [Table 3-25](#) are shown in [Table 3-26](#).

**Table 3-26: Parameters for Example Data Byte Lane #5**

Parameter	Value
DM_MAP	12'h02A
DQTS_MAP	12'h025
PHY_0_BITLANES	12'hFDE
DATA1_MAP	108'h029_028_027_026_02B_024_023_022_021
QK_MAP	8'h02

## I/O Standards

The MIG tool generates the appropriate UCF for the core with SelectIO™ standards based on the type of input or output to the 7 series FPGAs. These standards should not be changed. [Table 3-27](#) contains a list of the ports with the I/O standard used.

**Table 3-27: I/O Standards**

Signal	Direction	I/O Standard
rldii_ck_p, rldii_ck_n	Output	DIFF_HSTL_I
rldii_dk_p, rldii_dk_n	Output	DIFF_HSTL_I
rldii_cs_n	Output	HSTL_I

**Table 3-27: I/O Standards (Cont'd)**

<b>Signal</b>	<b>Direction</b>	<b>I/O Standard</b>
rldii_we_n	Output	HSTL_I
rldii_ref_n	Output	HSTL_I
rldii_a	Output	HSTL_I
rldii_ba	Output	HSTL_I
rldii_dm	Output	HSTL_I
rldii_dq	Input/Output	HSTL_II_T_DC1, HSTL_II
rldii_qk_p, rldii_qk_n	Input	DIFF_HSTL_II_DC1, DIFF_HSTL_II

DCI (HR banks) or IN\_TERM (HR banks) is required at the FPGA to meet the specified performance. Designs generated by the MIG tool use the DCI standards for Data (DQ) and Read Clock (QK\_P and QK\_N) in the High-Performance banks. In the High-Range banks, the MIG tool uses the HSTL\_II and DIFF\_HSTL\_II standards with the internal termination (IN\_TERM) attribute chosen in the GUI.

## Debugging RLDRAM II 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 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 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 3-32 shows the overall flow for debugging problems associated with these two general types of issues.

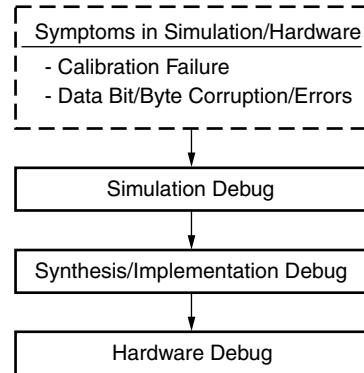


Figure 3-32: RLDRAM II MIG Tool Debug Flowchart

## 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

RLDRAM II 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.

### 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.

### 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 the application and the MIG tool signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool [Ref 8].

## Simulation Debug

Figure 3-33 shows the debug flow for simulation.

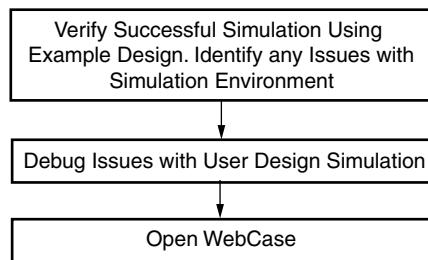


Figure 3-33: **Simulation Debug Flowchart**

### 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. 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 9] and the *Synthesis and Simulation Design Guide* [Ref 10]. For simulator tool support, refer to the *7 Series FPGAs Memory Interface Solutions Data Sheet* [Ref 11].

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 init\_calib\_complete signal. When this signal is asserted, the Traffic Generator takes control and begins executing writes and reads according to its parameterization.

Table 3-28 shows the signals and parameters of interest, respectively, during simulation.

Table 3-28: **Signals of Interest During Simulation**

Signal Name Usage	Signal Name Usage
tg_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; it is held until the design is reset.
dbg_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 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.

Table 3-28: Signals of Interest During Simulation (Cont'd)

Signal Name Usage	Signal Name Usage
user_ba	This is the bank address location for the current command.
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 this signal is 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 this signal is 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.

### Memory Initialization

For simulation, the MIG tool sets up the design parameters such that long wait times usually required for memory initialization are skipped. These parameters can result in memory model warnings. For the design to properly initialize and calibrate the full memory array in hardware, the top-level MIG tool design file (`example_top.v`) 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.

### Calibration

Calibration completes read leveling and read enable calibration. This is completed over three stages. This sequence successfully completes when the `init_calib_complete` signal is asserted. For more details, refer to [Physical Interface, page 256](#).

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 3-34](#).

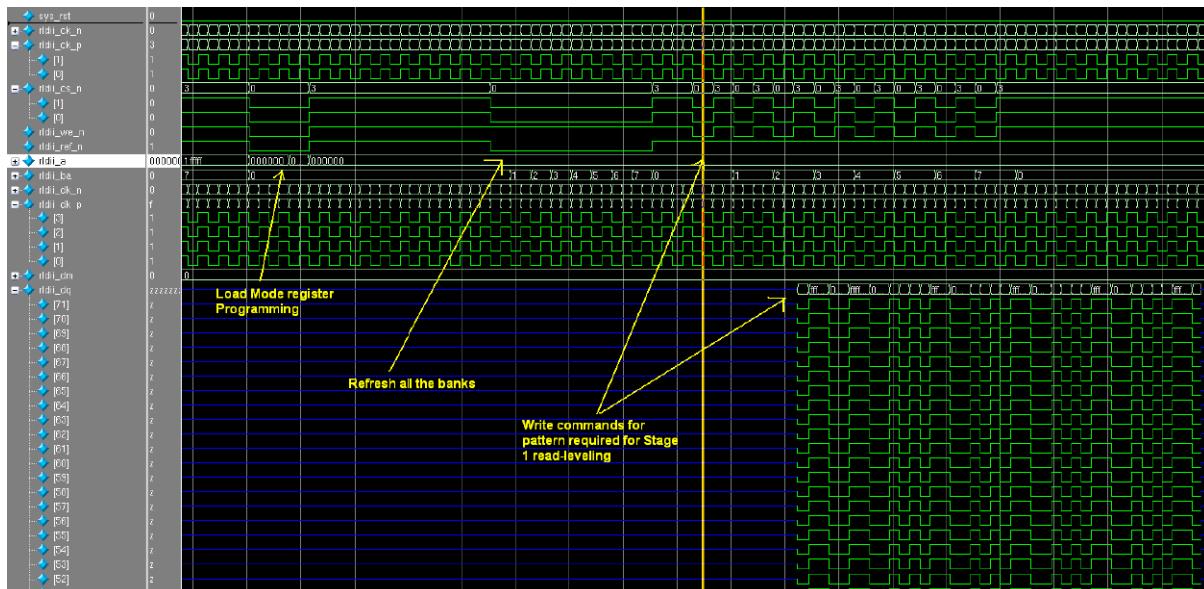


Figure 3-34: Writes for First Stage Read Calibration

This pattern is then continuously read back while the calibration is completed, as shown in Figure 3-35.

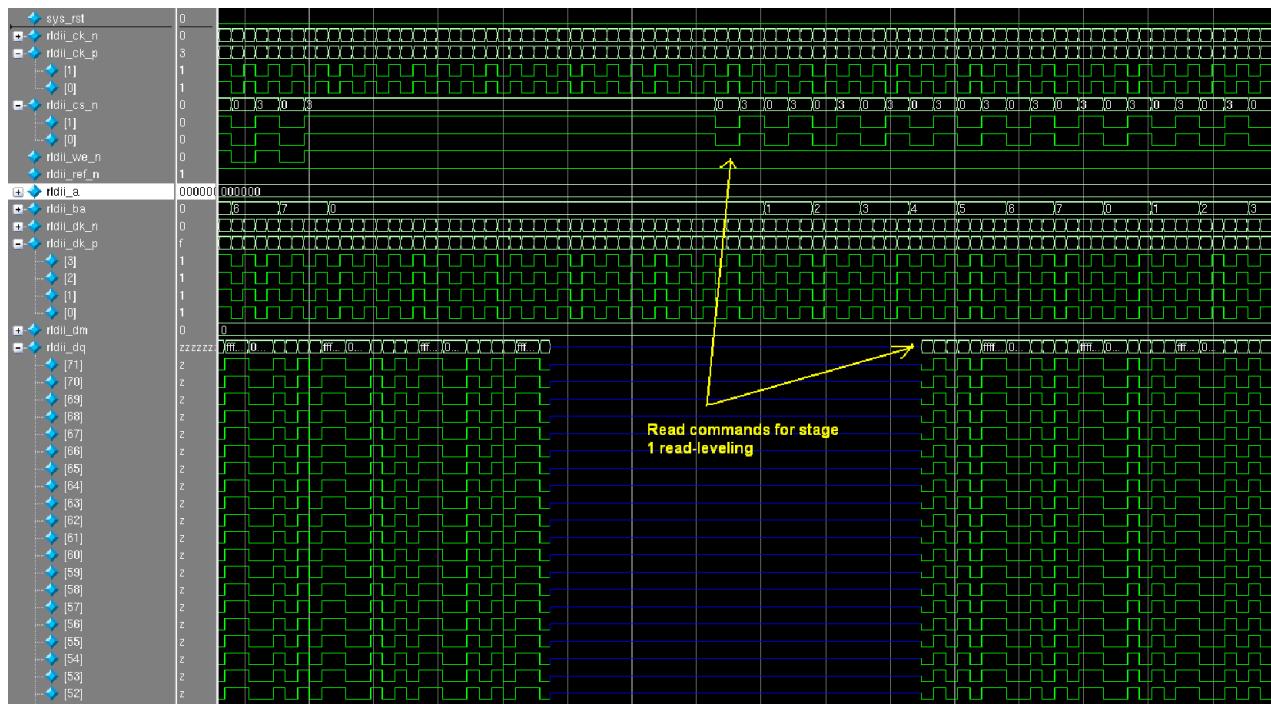


Figure 3-35: Reads for First Stage Read Calibration

The second stage performs an alignment to ensure data is returned in the correct order. The data pattern of AA .. 55 .. 00 .. FF is first written to the memory and continuously read back and adjusted internally if required. The third stage performs a read enable

calibration. The data pattern used during this stage is the same pattern used during the second stage of calibration. The data pattern is first written to the memory, and then read back for the read enable calibration, as shown in Figure 3-36.

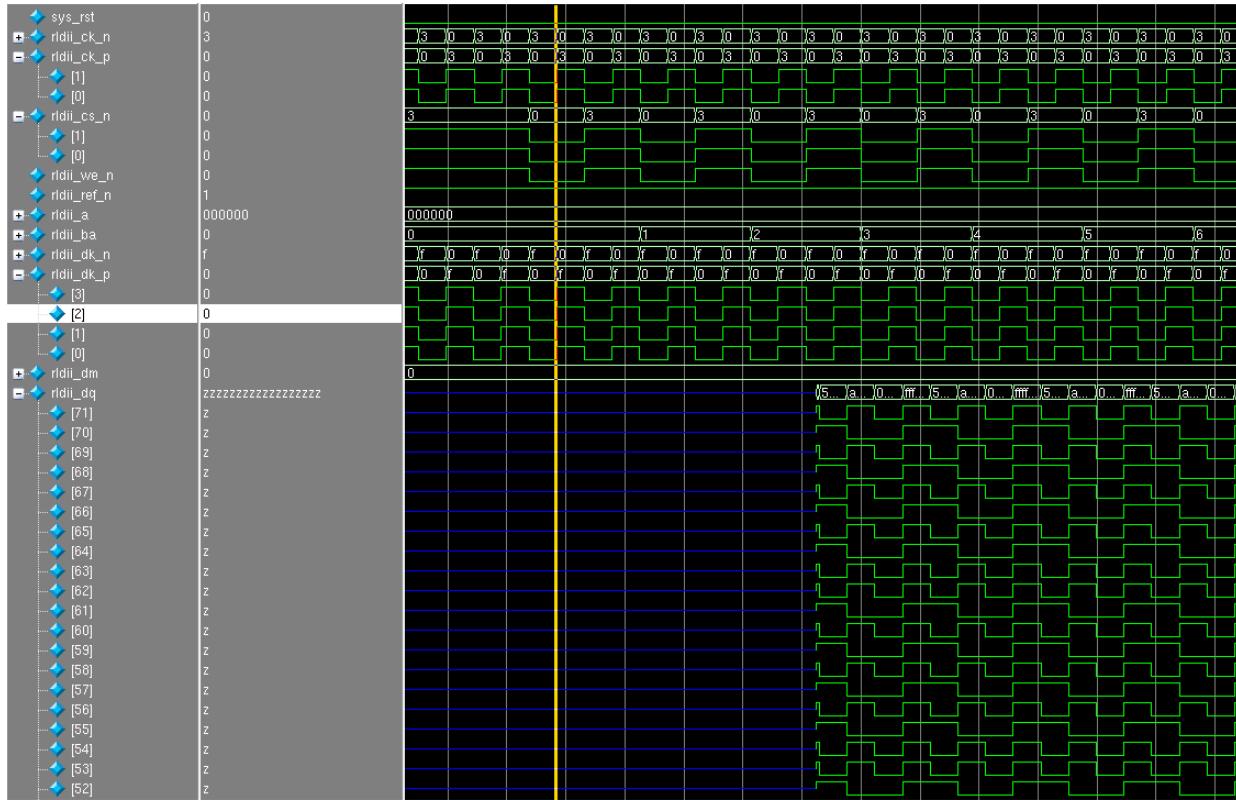


Figure 3-36: Write and Read for Second Stage Read Calibration

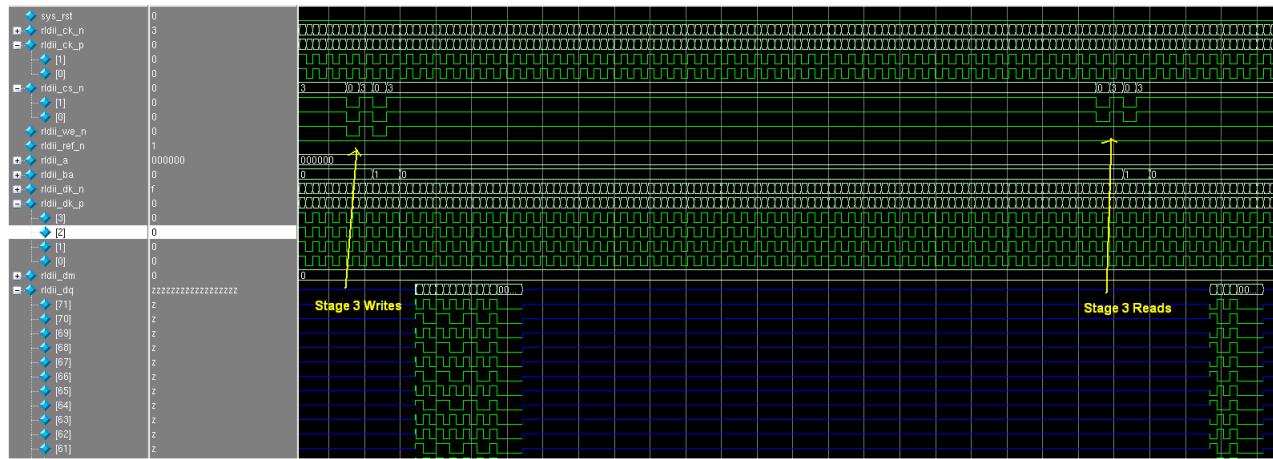


Figure 3-37: Write and Read for Third Stage Read Calibration

An additional write/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 third stage calibration completes, init\_calib\_complete is asserted, signifying successful completion of the calibration process.

### Test Bench

After init\_calib\_complete is asserted, 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 3-38](#) shows a successful implementation of the test bench with no assertions on error.



*Figure 3-38: Test Bench Operation After Completion of Calibration*

### Proper Write and Read Commands

When sending write and read commands, the user must properly assert and deassert the corresponding UI inputs. Refer to [Client Interface, page 252](#) and [Interfacing with the Core through the Client Interface, page 254](#) 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, 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 3-39](#). Highlight the user interface signals noted in [Table 3-28, page 281](#), right-click, and select **Add > To Wave > Selected Signals**.

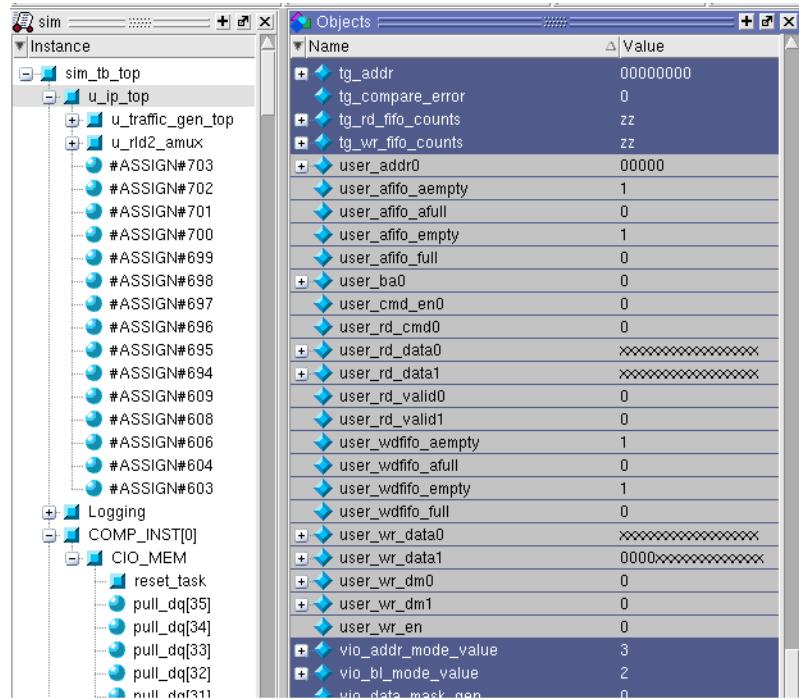


Figure 3-39: ModelSim Instance Window

Figure 3-40 and Figure 3-41 show example waveforms of a write and read on both the user interface and the RLDRAM II interface.

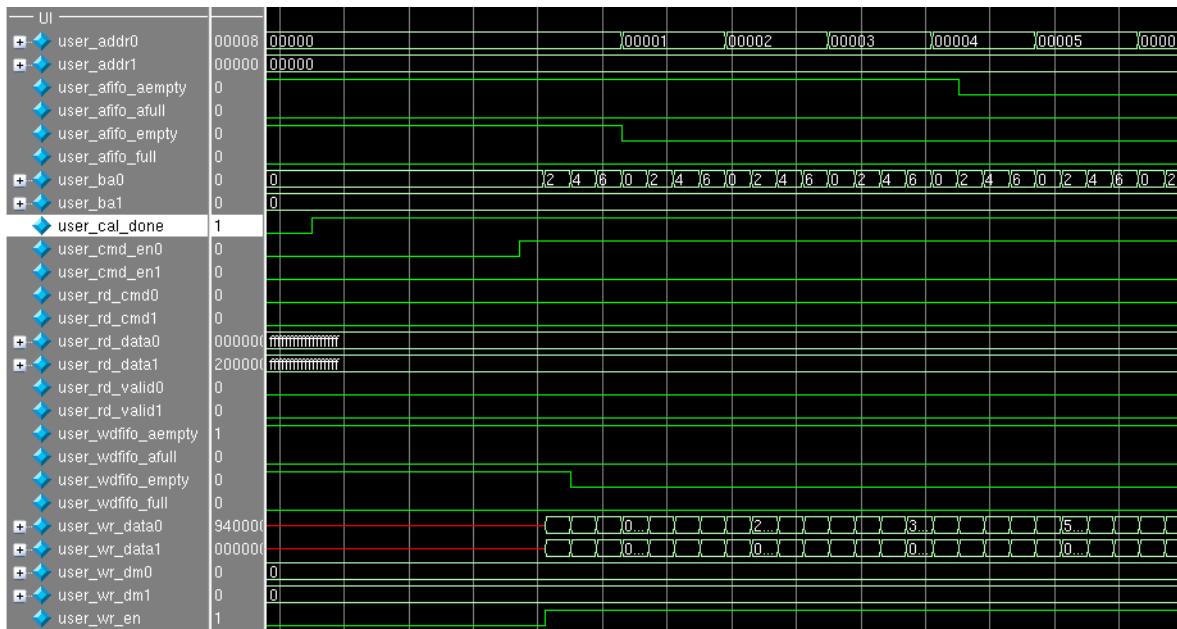


Figure 3-40: User Interface Write and Read

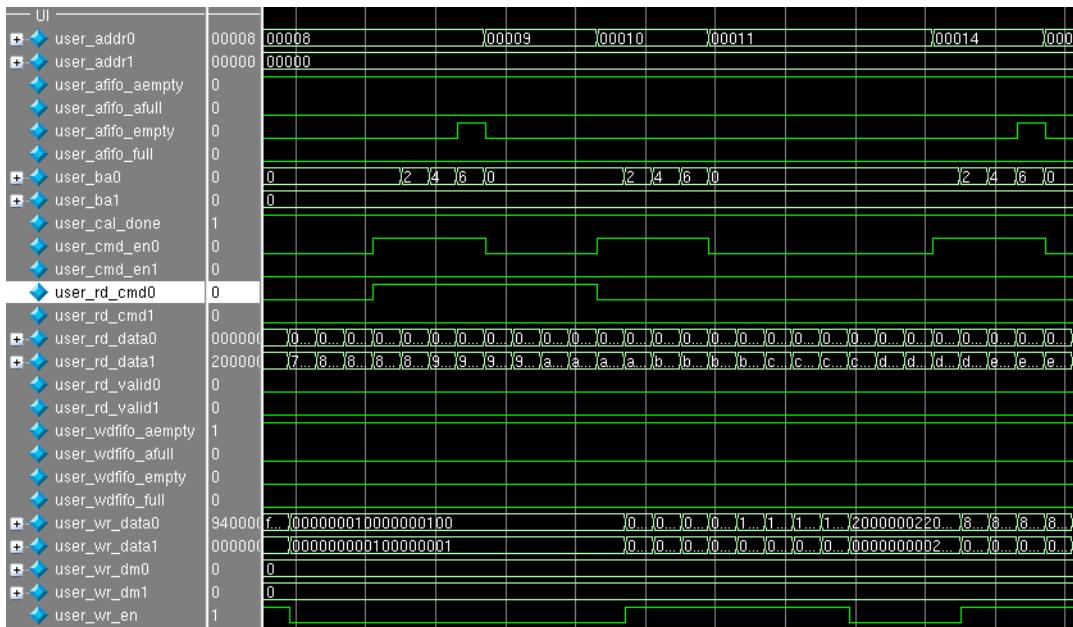


Figure 3-41: RLDRAM II Interface Write and Read

## Synthesis and Implementation Debug

Figure 3-42 shows the debug flow for synthesis and implementation.

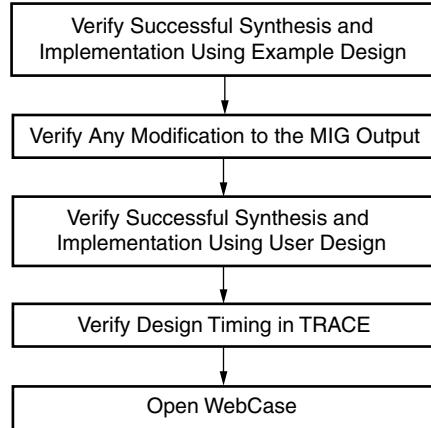


Figure 3-42: Synthesis and Implementation Debug Flowchart

### 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`/`ise_flow.sh`, 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 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

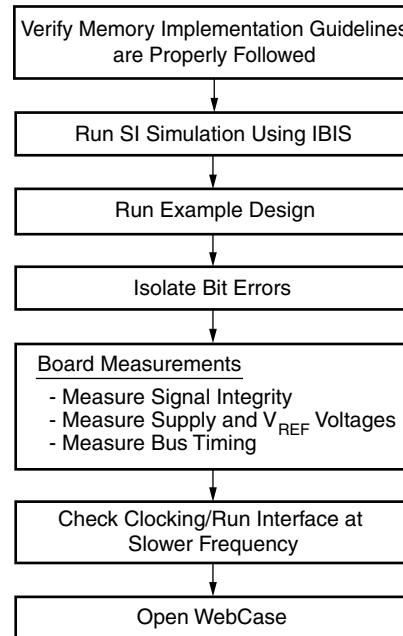
The MIG tool RLDRAM II designs have been verified to meet timing using the example design across a wide range of configurations. However, timing violations might occur, such as when integrating the MIG tool design with the user's specific application logic.

Any timing violations that are encountered must be isolated. The timing report output by TRACE (.twx/.tvr) 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 12] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 13] provides valuable information on all available Xilinx constraints.

## Hardware Debug

Figure 3-43 shows the debug flow for hardware.



*Figure 3-43: Hardware Debug Flowchart*

## 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 14] 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 MIG designs with 7 series FPGAs.

### Run the Example Design

The example design provided with the MIG tool 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 8].

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 init\_calib\_complete and no assertions of tg\_compare\_error. Assertion of init\_calib\_complete signifies successful completion of calibration while no assertion of tg\_compare\_error signifies that the data is written to and read from the memory compare with no data errors.

The dbg\_cmp\_err signal can be used to indicate if a single error was encountered or if multiple errors are encountered. With each error encountered, dbg\_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 QK 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, if available, to vary the phase of DQ relative to the DK clocks.
  - Verify the timing relationship between CK and DK clocks.

- Vary only read timing:
    - Check the IDELAY/PHASER\_IN values after calibration. Look for variations between IDELAY/PHASER\_IN values. IDELAY values should be very similar for DQs in the same byte group.
    - Vary the IDELAY/PHASER\_IN 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 3-29](#) 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 3-29: Physical Layer Simple Status Bus Description Defined in the rld\_phy\_top Module**

Debug Port Signal	Name	Description	If Problems Arise
dbg_phy_status[0]	rst_wr_clk	Fabric reset based on PLL lock and system input reset	If this signal stays asserted, check your clock source and system reset input
dbg_phy_status[1]	po_delay_done	I/O FIFO initialization to ensure the I/O FIFOs are in an almost full condition and the phaser out delay to provide the 90° phase shift to address/control signals are done	Check if the PHY control ready signal is asserted
dbg_phy_status[2]	init_done	RLDRAM II initialization sequence is complete	N/A <sup>(1)</sup>
dbg_phy_status[3]	rdlvl_stg1_start	Stage 1 read calibration start signal	N/A
dbg_phy_status[4]	rdlvl_stg1_done	Stage 1 calibration is complete	N/A
dbg_phy_status[5]	edge_adv_cal_done	Edge Advance calibration is complete	Make sure the expected data is being returned from the memory. Check results of Stage 1 read calibration.
dbg_phy_status[6]	init_cal_done	Latency calibration completed	N/A
dbg_phy_status[7]	init_calib_complete	Calibration complete	N/A

**Notes:**

1. N/A indicates that as long as previous stages have completed, this stage is also completed.

The read calibration results are provided as part of the Debug port as various output signals. These signals can be used to capture and evaluate the read calibration results.

Read calibration uses the IODELAY to align the capture clock in the data valid window for captured data. The algorithm shifts the IDELAY/PHASER\_IN values and looks for edges of the data valid window on a per-byte 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 begins with *dbg\_*. The DEBUG\_PORT parameter is always set to OFF in the sim\_tb\_top module of the sim folder, which disables the debug option for functional simulations. These signals and their associated data are described in [Table 3-30](#).

*Table 3-30: DEBUG\_PORT Signal Descriptions*

Signal	Direction	Description
dbg_phy_cmd_n[5:0]	Output	This active-Low signal is the internal command that is sent to the memory used for debug with the ChipScope analyzer
dbg_phy_addr[RLD_ADDR_WIDTH*2-1:0]	Output	Address being accessed for the commands given with dbg_phy_cmd_n.
dbg_phy_ba[BANK_WIDTH*2-1:0]	Output	Control bank address bus used for debug with the ChipScope analyzer
dbg_phy_wr_data[DATA_WIDTH× 4 – 1:0]	Output	Data being written that is used for debug with the ChipScope analyzer.
dbg_phy_init_wr_only	Input	When this input is High, the state machine in the rld_phy_write_init_sm module keeps issuing write commands for Stage 1 data to the RLDRAM II. This can be used to verify write timing, for measuring the DK to DQ timing relationship at the memory using an oscilloscope. This signal must be Low for normal operation.
dbg_phy_init_rd_only	Input	When this input is High, the state machine in the rld_phy_write_init_sm module keeps issuing read commands for Stage 1 read calibration. This is useful to probe the signals on the PCB and look for any SI issues or verify the previous write command occurred properly. This signal must be Low for normal operation.

Table 3-30: DEBUG\_PORT Signal Descriptions (Cont'd)

Signal	Direction	Description
dbg_byte_sel[CQ_BITS-1:0]	Input	This input selects the corresponding byte lane whose phaser/IDELAY tap controls need to be controlled by the user
dbg_bit_sel[Q_BITS-1:0]	Input	This input selects the corresponding bit lane whose IDELAY tap controls need to be controlled by the user. It also controls which read data signals are presented to the user for debug (dbg_rd_data_rd0, dbg_rd_data_rd1, dbg_rd_data_fd0, dbg_rd_data_fd1).
dbg_idel_up_all	Input	This input increments all IDELAY tap values for the entire bus.
dbg_idel_down_all	Input	This input decrements all IDELAY tap values for the entire bus.
dbg_idel_up	Input	This input increments all IDELAY tap values for a single bit, selected by dbg_bit_sel.
dbg_idel_down	Input	This input decrements all IDELAY tap values for a single bit, selected by dbg_bit_sel.
dbg_pi_f_inc	Input	This signal increments the phaser_in generated ISERDES clk that is used to capture rising data
dbg_pi_f_dec	Input	This signal decrements the phaser_in generated ISERDES clk that is used to capture rising data
dbg_po_f_inc	Input	This signal increments the phaser_out generated OSERDES clk that is used to capture falling data
dbg_po_f_dec	Input	This signal increments the phaser_out generated OSERDES clk that is used to capture falling data
dbg_pi_tap_cnt[5:0]	Output	This output indicates the current phaser_in tap count position
dbg_po_tap_cnt[5:0]	Output	This output indicates the current phaser_out tap count position

Table 3-30: DEBUG\_PORT Signal Descriptions (Cont'd)

Signal	Direction	Description
dbg_cq_num[CQ_BITS-1:0]	Output	This signal indicates the current byte lane selected (either during calibration or through the debug port)
dbg_valid_lat[4:0]	Output	Latency in cycles of the delayed read command
dbg_idel_tap_cnt_sel[TAP_BITS-1:0]	Output	Current IDELAY tap setting for bits selected using dbg_bit_sel
dbg_inc_latency	Output	This output indicates that the latency of the corresponding byte lane was increased to ensure proper alignment of the read data to the user interface.
dbg_error_max_latency	Output	This signal indicates that the latency could not be measured before the counter overflowed. Each device has one error bit.
dbg_error_adj_latency	Output	This signal indicates that the target PHY_LATENCY could not be achieved
ddbg_rd_data_rd0[8:0]	Output	This bus shows the captured output of the first rising data for a single byte lane, selected using dbg_bit_sel
dbg_rd_data_rd1[8:0]	Output	This bus shows the captured output of the second rising data for a single byte lane, selected using dbg_bit_sel
dbg_rd_data_fd0[8:0]	Output	This bus shows the captured output of the first falling data for a single byte lane, selected using dbg_bit_sel
dbg_rd_data_fd1[8:0]	Output	This bus shows the captured output of the second falling data for a single byte lane, selected using dbg_bit_sel
dbg_rd_valid0	Output	Read data valid signal that aligns with the dbg_rd_data_rd0, dbg_rd_data_fd0 signals.
dbg_rd_valid1	Output	Read data valid signal that aligns with the dbg_rd_data_rd1, dbg_rd_data_fd1 signals

## Write Init Debug Signals

[Table 3-31](#) indicates the mapping between the write init debug signals on the dbg\_wr\_init bus and debug signals in the PHY. All signals are found within the rld\_phy\_write\_init\_sm module and are all valid in the clk domain.

**Table 3-31: Write Init Debug Signal Map**

Bits	PHY Signal Name	Description
dbg_phy_init_sm[3:0]	phy_init_cs	Current state of the initialization state machine
dbg_phy_init_sm[6:4]	start_cal	Flags to determine stages of calibration
dbg_phy_init_sm[7]	init_complete	Memory initialization is complete
dbg_phy_init_sm[8]	refr_req	Refresh request
dbg_phy_init_sm[9]	refr_done	Refresh complete
dbg_phy_init_sm[10]	stage2_done	Stage 2 calibration is complete
dbg_phy_init_sm[22:11]	refr_cnt	Refresh counter
dbg_phy_init_sm[26:23]	phy_init_ps	Previous state of the initialization state machine
dbg_phy_init_sm[31:27]	Reserved	N/A

## Read Stage 1 Calibration Debug Signals

[Table 3-32](#) indicates the mapping between bits within the dbg\_rd\_stage1\_cal bus and debug signals in the PHY. All signals are found within the qdr\_rld\_phy\_rdlvl module and are all valid in the clk domain.

**Table 3-32: Read Stage 1 Debug Signal Map**

Bits	PHY Signal Name	Description
dbg_phy_rdlvl[0]	rdlvl_stg1_start	Input from initialization state machine indicating start of stage 1 calibration
dbg_phy_rdlvl[1]	rdlvl_start	Indicates when calibration logic begins
dbg_phy_rdlvl[2]	found_edge_r	Indicates a transition of data window was detected
dbg_phy_rdlvl[3]	pat0_data_match_r	Expected data pattern seen at ISERDES outputs
dbg_phy_rdlvl[4]	pat1_data_match_r	Expected data pattern is seen at ISERDES outputs; however, this is shifted due to the first read data aligning to the negative edge of the ICLKDIV
dbg_phy_rdlvl[5]	data_valid	Valid calibration data is seen

Table 3-32: Read Stage 1 Debug Signal Map (Cont'd)

Bits	PHY Signal Name	Description
dbg_phy_rdlvl[6]	cal1_wait_r	Wait state for observing the data after the IDELAY/phaser taps have been varied
dbg_phy_rdlvl[7]	Reserved	
dbg_phy_rdlvl[8]	detect_edge_done_r	Edge detection completed
dbg_phy_rdlvl[13:9]	cal1_state_r	Calibration state machine states
dbg_phy_rdlvl[20:14]	cnt_idel_dec_cpt_r	Number of phaser taps to be decremented for centering the clock in the data window
dbg_phy_rdlvl[21]	found_first_edge_r	First edge transition detected
dbg_phy_rdlvl[22]	found_second_edge_r	Second edge transition detected
dbg_phy_rdlvl[23]	Reserved	
dbg_phy_rdlvl[24]	store_sr_r	Signal to store current read data prior to incrementing tap delays
dbg_phy_rdlvl[32:25]	sr_fall1_r, sr_rise1_r sr_fall0_r, sr_rise0_r	Read data stored in shift registers for comparison
dbg_phy_rdlvl[40:33]	old_sr_fall1_r, old_sr_rise1_r old_sr_fall0_r, old_sr_rise0_r	Read data stored in registers prior to tap increments
dbg_phy_rdlvl[41]	sr_valid_r	Determines when it is safe to load the ISERDES data for comparison
dbg_phy_rdlvl[42]	found_stable_eye_r	Indicates a stable eye is seen
dbg_phy_rdlvl[48:43]	tap_cnt_cpt_r	Phaser tap counter
dbg_phy_rdlvl[54:49]	first_edge_taps_r	Number of taps to detect first edge
dbg_phy_rdlvl[60:55]	second_edge_taps_r	Number of taps to detect second edge
dbg_phy_rdlvl[64:61]	cal1_cnt_cpt_r	Indicates the white byte lane is being calibrated
dbg_phy_rdlvl[65]	cal1_dlyce_cpt_r	Phaser control to enable phaser delays
dbg_phy_rdlvl[66]	cal1_dlyinc_cpt_r	Phaser control to increment phaser tap delay
dbg_phy_rdlvl[67]	found_edge_r	Indicates a transition of the data window is detected
dbg_phy_rdlvl[68]	found_stable_eye_last_r	Indicates a stable eye is seen

**Table 3-32: Read Stage 1 Debug Signal Map (Cont'd)**

<b>Bits</b>	<b>PHY Signal Name</b>	<b>Description</b>
dbg_phy_rdlvl[74:69]	idelay_taps	Number of IDELAY taps that detect a valid data window while delaying incoming read data
dbg_phy_rdlvl[80:75]	start_win_taps	Number of IDELAY taps to be delayed to detect start of valid window
dbg_phy_rdlvl[81]	idel_tap_limit_cpt_r	Indicates the end of the IDELAY tap chain is reached
dbg_phy_rdlvl[82]	qdlly_inc_done_r	Indicates valid window detection by delaying IDELAY taps is done
dbg_phy_rdlvl[83]	start_win_detect	Indicates start of window detection using IDELAY taps
dbg_phy_rdlvl[84]	detect_edge_done_r	Edge detection is completed
dbg_phy_rdlvl[90:85]	idel_tap_cnt_cpt_r	Counter that keeps track of the number of IDELAY taps used
dbg_phy_rdlvl[96:91]	idelay_inc_taps_r	Number of IDELAY taps to be incremented, if needed by the calibration logic
dbg_phy_rdlvl[102:97]	idel_dec_cntr	Number of IDELAY taps to be decremented, if needed by the calibration logic
dbg_phy_rdlvl[103]	tap_limit_cpt_r	Indicates the end of the phaser tap delay chain is reached
dbg_phy_rdlvl[115:104]	idelay_tap_delay	Total amount of valid window seen using idelay taps
dbg_phy_rdlvl[127:128]	phaser_tap_delay	Total amount of valid window seen using phaser taps
dbg_phy_rdlvl[255:128]	Reserved	

### Read Stage 2 Calibration Debug

**Table 3-33** indicates the mapping between bits within the dbg\_rd\_stage2\_cal bus and debug signals in the PHY. All signals are found within the qdr\_rld\_phy\_read\_stage2\_cal module and are all valid in the clk domain.

**Table 3-33: Read Stage 2 Debug Signal Map**

<b>Bits</b>	<b>PHY Signal Name</b>	<b>Description</b>
dbg_stage2_cal[0]	en_mem_latency	Signal to enable latency measurement
dbg_stage2_cal[5:1]	latency_cntr[0]	Indicates the latency for the first byte lane in the interface
dbg_stage2_cal[6]	rd_cmd	Internal rd_cmd for latency calibration

Table 3-33: Read Stage 2 Debug Signal Map (Cont'd)

Bits	PHY Signal Name	Description
dbg_stage2_cal[7]	latency_measured[0]	Indicates latency has been measured for byte lane 0
dbg_stage2_cal[8]	bl4_rd_cmd_int	Indicates calibrating for burst length of 4 data words
dbg_stage2_cal[9]	bl4_rd_cmd_int_r	Internal register stage for burst 4 read command
dbg_stage2_cal[10]	edge_adv_cal_start	Indicates start of edge_adv calibration, to see if the pi_edge_adv signal needs to be asserted
dbg_stage2_cal[11]	rd0_vld	Indicates valid ISERDES read data
dbg_stage2_cal[12]	fd0_vld	Indicates valid ISERDES read data
dbg_stage2_cal[13]	rd1_vld	Indicates valid ISERDES read data
dbg_stage2_cal[14]	fd1_vld	Indicates valid ISERDES read data
dbg_stage2_cal[15]	phase_vld	Valid data is seen for the particular byte
dbg_stage2_cal[16]	rd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[17]	fd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[18]	rd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[19]	fd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip
dbg_stage2_cal[20]	phase_bslip_vld	Valid data is seen when bitslip applied to read data
dbg_stage2_cal[21]	clkdiv_phase_cal_done_4r	Indicates data validity complete, proceed to assert the pi_edge_adv signal if needed
dbg_stage2_cal[22]	pi_edge_adv	Phaser control signal to advance the Phaser clock, ICLKDIV by one fast clk cycle.
dbg_stage2_cal[25:23]	byte_cnt[2:0]	Indicates the byte that is being checked for data validity
dbg_stage2_cal[26]	inc_byte_cnt	Internal signal to increment to the next byte
dbg_stage2_cal[29:27]	pi_edge_adv_wait_cnt	Counter to wait between asserting the phaser control signal, pi_edge_adv signal in the various byte lanes.
dbg_stage2_cal[127:30]	Reserved	Reserved

## Margin Check

Debug signals are provided to move either clocks or data to verify functionality and to confirm sufficient margin is available for reliable operation. These signals can also be used to check for signal integrity issues affecting a subset of signals or to deal with trace length mismatches on the board. To verify read window margin, enable the debug port when generating a design in the MIG tool and use the provided example design. The steps to follow are:

1. Open ChipScope Analyzer and program the FPGA under test.
2. Set up the ChipScope Analyzer project with signals of interest (if not already done).
3. Verify that calibration completes (`init_calib_complete` should be asserted) and no errors currently exist in the example design (both `tg_compare_error` and `dbg_cmp_err` should be Low).
4. Select a given byte lane using `dbg_byte_sel`.
5. Select a given bit lane using `dbg_bit_sel`.
6. Observe the tap values on `PHASER_IN` for the selected byte lane using `dbg_pi_tap_cnt`. Observe the IDELAY tap values on the given DQ bit selected using `dbg_idel_tap_cnt_sel`.
7. Increment the tap values on `PHASER_IN` until an error occurs (`tg_compare_error` should be asserted) using `dbg_pi_f_inc`. Record how many phaser taps it took to get an error from the starting location. This value is the tap counts to reach one side of the window for the entire byte lane.
8. Decrement the tap values on `PHASER_IN` using `dbg_pi_f_dec` back to the starting value.
9. Clear the error recorded previously by asserting `dbg_clear_error`.
10. Depending on the tap count of `PHASER_IN`, you can either decrement the taps of `PHASER_IN` using `dbg_pi_f_dec` to find the other edge of the window or, if there are not enough taps to find the other edge, increment the IDELAY taps for the given bit `dbg_idel_down` until another error occurs.
11. Record those results, return the taps (either `PHASER_IN` or IDELAY) to the starting location, and clear the error again (`dbg_clear_error`).

This simple technique uses the error signal that is common for the entire interface, so any marginality in another bit or byte not being tested might affect the results. For better results, a per-bit error signal should be used. `PHASER_IN` and IDELAY taps need to be converted into a common unit of time to properly analyze the results.

## Automated Margin Check

Manually moving taps to verify functionality is useful to check problem bits or bytes, but it can be difficult to step through an entire interface looking for issues. For this reason, the RLDRAM II Memory Interface Debug port contains automated window checking that can be used to step through the entire interface. A simple state machine is used to take control of the debug port signals and report results of the margin found per-bit. Currently, the automated window check only uses `PHASER_IN` to check window sizes, so depending on the tap values after calibration, the left edge of the read data window might not be found properly.

Table 3-34 lists the signals associated with this automated window checking functionality.

Table 3-34: Debug Port Signals

Signal	Description
dbg_win_start	Single pulse that starts the chk_win state machine. Use the ChipScope Analyzer VIO to control this.
dbg_win_dump	Single pulse that starts the automated reporting mechanism. Use the ChipScope Analyzer VIO to control this.
dbg_win_bit_select[6:0]	Manual bit selection for reporting of results. The results are provided on dbg_left_ram_out and dbg_right_ram_out for the bit indicated.
dbg_win_active	Flag to indicate chk_win is active and measuring read window margins. While active, the state machine has control over the debug port signals.
dbg_win_dump_active	Flag to indicate chk_win is automatically reporting results sequentially for all bits.
dbg_win_clr_error	Clear error control signal controlled by chk_win.
dbg_win_current_bit[6:0]	Feedback to indicate which bit is currently being monitored during automatic window checking.
dbg_win_current_byte[3:0]	Feedback to indicate which byte is currently being monitored (and used to select the byte lane controls with dbg_byte_sel).
dbg_current_bit_ram_out[6:0]	Feedback to indicate which bit is currently being reported back during win_dump.
dbg_win_left_ram_out [WIN_SIZE-1:0]	PHASER_IN tap count to reach the left edge of the read window for a given bit.
dbg_win_right_ram_out [WIN_SIZE-1:0]	PHASER_IN tap count to reach the right edge of the read window for a given bit.
dbg_win_inc	chk_win control signal to increment PHASER_IN.
dbg_win_dec	chk_win control signal to decrement PHASER_IN.

An extra simulation-only state machine is provided to start the chk\_win and win\_dump functionalities to verify the connectivity and functionality of the debug port and to allow a better understanding of how it works. To enable this, the simulation environment has to set the DEBUG\_PORT parameter to “ON” and a local parameter called SIMULATE\_CHK\_WIN under example\_top.v has to be set to “TRUE.”

# Multicontroller Design

---

## Introduction

This chapter describes the specifications (including the supported features and unsupported features) and pinout rules for multicontroller designs.

The supported and unsupported features are:

- Supports up to 8 controllers
  - Multi-interface support includes the combination of all memory interfaces as DDR3 SDRAM (Native only), QDRII+ SRAM, and RLDRAM II up to total of 8 controllers. Multi-interface support with the DDR3 SDRAM AXI interface combined with other memory interfaces is not supported.
  - Multicontroller for DDR3 SDRAM (AXI only) interface is supported up to 8 independent controllers. Multicontroller support combining DDR3 SDRAM Native and AXI interface designs is not supported.
  - Banks selected for one of the controllers are not allowed for other controllers; that is, across the same memory interfaces and different memory interfaces.
  - Memory options (frequency, data width, and so on) and all other options remain the same as for single controller options.
  - Sharing of banks across two different controllers is not allowed.
  - Rules for all memory interfaces (DDR3 SDRAM, QDRII+SRAM, and RLDRAM II) remain the same as for single controller designs.

## Getting Started with the CORE Generator Tool

Invoking the MIG tool from the CORE Generator™ tool is the same as with single controller designs. See the appropriate memory interface chapter in this document for more information. The MIG GUI pages that are different for multicontroller designs are explained in this chapter.

## Multiple Controllers

Select the number of controllers from the MIG Output Options page (Figure 4-1). The number of controllers that can be accommodated varies based on the number of banks available in the device and depends on the memory interface configuration chosen (that is, the selected data width and number of banks).

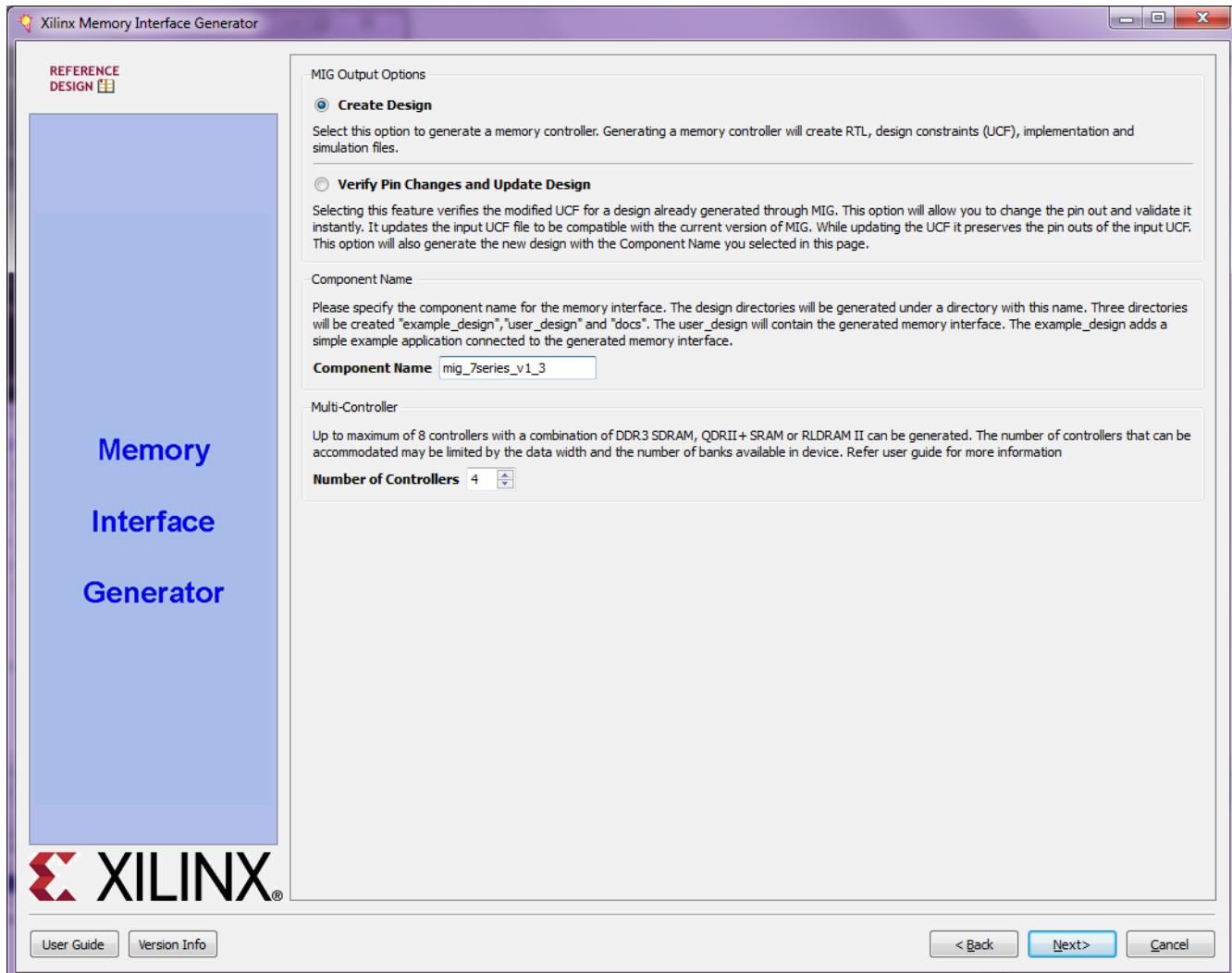


Figure 4-1: MIG Output Options Page

## Memory Selection

Memory interface selection is different for a multicontroller design compared with a single controller design. Select the number of controllers for each memory interface on the Memory Selection page ([Figure 4-2](#)).

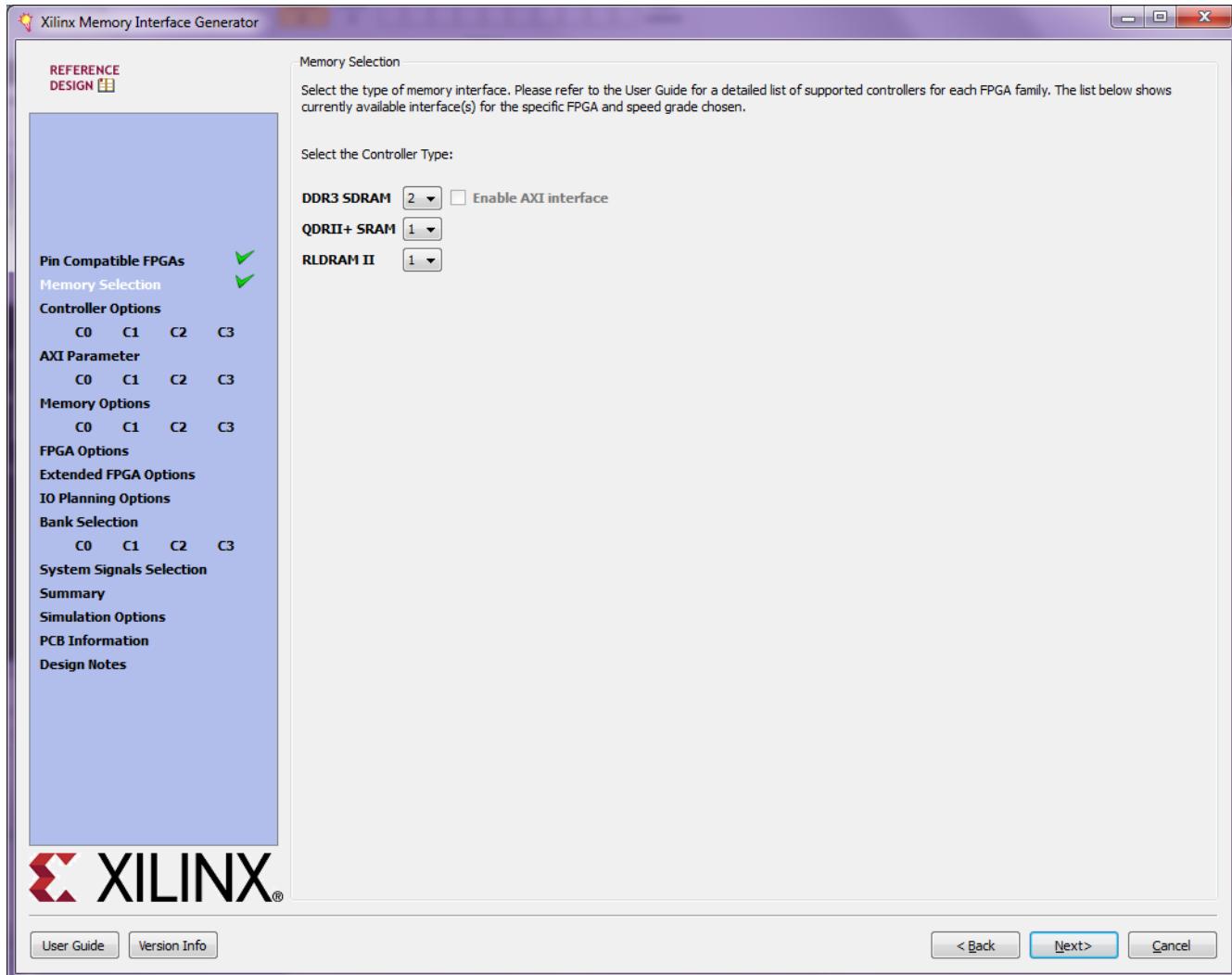


Figure 4-2: Memory Selection Page

## FPGA Options

The Debug option can be selected for one controller only. Debug logic is generated for the selected controller (Figure 4-3).

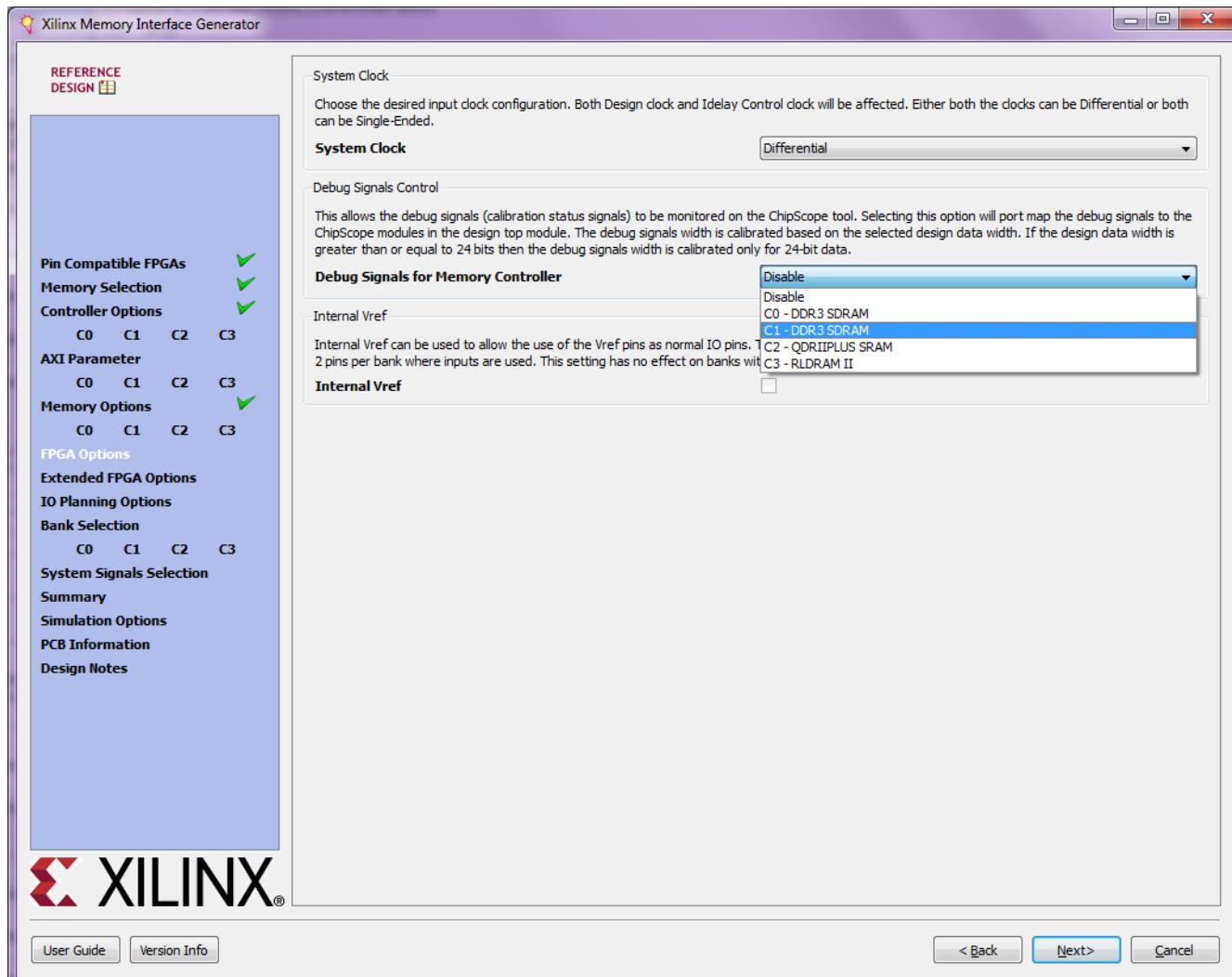


Figure 4-3: FPGA Options Page

## Extended FPGA Options Page

Figure 4-4 shows the Extended FPGA Options page for a multicontroller design with all three memory interfaces chosen.

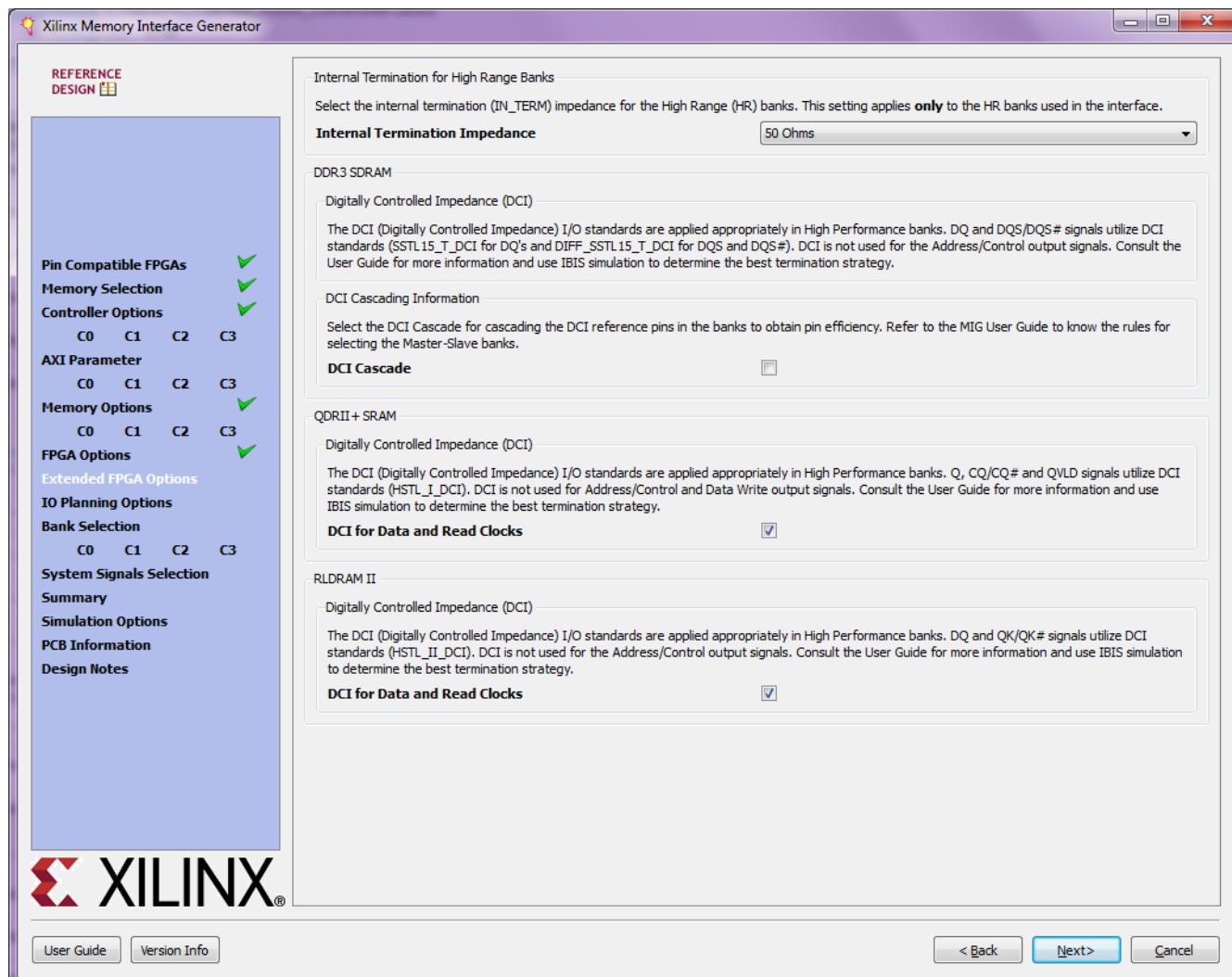
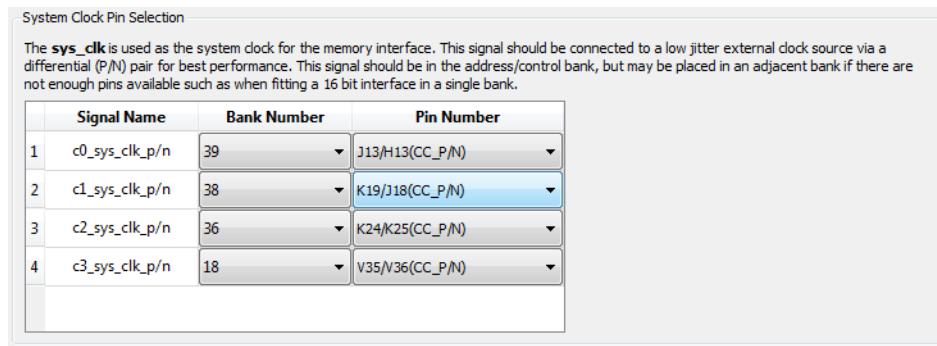


Figure 4-4: Extended FPGA Options Page

## System Clock Pins Selection

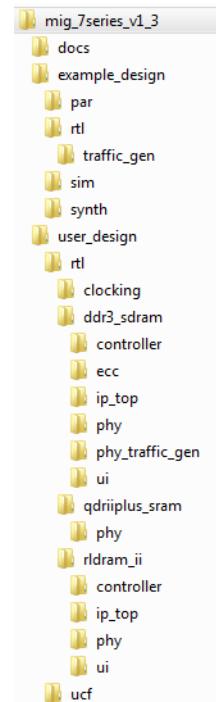
Select the system clock pins on the System Pins Selection page. System clock pins can be selected for each controller; this varies based on the number of controllers ([Figure 4-5](#)).



*Figure 4-5: System Clock Pins Selection Page*

## MIG Directory Structure

The MIG output directory structure is slightly different for the user design RTL folder compared with the single controller design. The user design RTL folder contains the subfolders for each memory interface, and related RTL files are generated in the corresponding memory interface folders. All chosen memory interfaces for multicontroller designs are shown in [Figure 4-6](#).



*Figure 4-6: MIG Directory Structure*

# 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/company/terms.htm>.

## 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 the [Xilinx MIG Solution Center](#).

## References

These references provide supplemental information useful for this document:

1. [UG471, 7 Series FPGAs SelectIO Resources User Guide](#)
2. [UG475, 7 Series FPGAs Packaging and Pinout Specification](#)
3. ARM® AMBA® Specifications  
<http://www.arm.com/products/system-ip/amba/amba-open-specifications.php>
4. JESD79-3E, DDR3 SDRAM Standard, JEDEC Solid State Technology Association  
<http://www.jedec.org/sites/default/files/docs/JESD79-3E.pdf>
5. [UG683, EDK Concepts, Tools, and Techniques](#)
6. [UG111, Embedded System Tools Reference Manual](#)
7. [TN-47-01, DDR2-533 Memory Design Guide For Two-DIMM Unbuffered Systems](#). Micron Technology, Inc.
8. ChipScope Pro Logic Analyzer tool  
<http://www.xilinx.com/tools/cspro.htm>
9. [UG628, Command Line Tools User Guide, COMPXLIB](#)
10. [UG626, Synthesis and Simulation Design Guide](#)
11. [DS176, 7 Series FPGAs Memory Interface Solutions Data Sheet](#)
12. PlanAhead™ Design Analysis tool  
[www.xilinx.com/tools/planahead.htm](http://www.xilinx.com/tools/planahead.htm)

13. [UG612, Xilinx Timing Constraints User Guide](#)
14. [UG199, Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide](#)
15. 7 series FPGAs Data Sheets  
[www.xilinx.com/support/documentation/7\\_series.htm](http://www.xilinx.com/support/documentation/7_series.htm)
16. [UG029, ChipScope Pro Software and Cores User Guide](#)
17. "Improving DDR SDRAM Efficiency with a Reordering Controller", XCELL Journal Issue 69, [www.xilinx.com/publications/archives/xcell/Xcell69.pdf](http://www.xilinx.com/publications/archives/xcell/Xcell69.pdf)
18. [UG472, 7 Series FPGAs Clocking Resources User Guide](#)