

# Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions

## v4.2

### ***User Guide***

UG586 December 5, 2018



12/05/2018	4.2	<ul style="list-style-type: none"> <li>Updated to core version 4.2.</li> <li>Updated Vivado MIS GUIs.</li> </ul>
04/04/2018	4.1	Vivado Design Suite release for MIS v4.1.
03/02/2018	4.1	<ul style="list-style-type: none"> <li>Reverted doc version to v4.1 to match Vivado Design Suite release for MIS core v4.1.</li> <li>Added routing constraints note in General Memory Routing Guidelines appendix.</li> </ul>
10/04/2017	4.2	<b>DDR3 and DDR2</b> <ul style="list-style-type: none"> <li>Updated CLOCK_DEDICATED_ROUTE description in Reference Clock section.</li> </ul>
06/07/2017	4.2	Vivado Design Suite release for MIS v4.2.
04/05/2017	4.2	<b>DDR3 and DDR2</b> <ul style="list-style-type: none"> <li>Updated Fig. 1-93 and Fig. 1-94 captions.</li> </ul>
11/30/2016	4.1	<ul style="list-style-type: none"> <li>Renamed QuestaSim to Questa Advanced Simulator.</li> </ul> <p><b>QDR II+</b></p> <ul style="list-style-type: none"> <li>Updated qdr_k_n/p directions in Physical Interface Signals table.</li> <li>Updated in qdr_k_n/p directions I/O Standards table.</li> </ul> <p><b>RLDRAM II/RLDRAM 3</b></p> <ul style="list-style-type: none"> <li>Updated rld_dk_p/n directions in Physical Interface Signals table.</li> <li>Updated rld_dk_p/n directions in RLDIMM II I/O Standards and RLDIMM 3 Standards tables.</li> </ul>
10/05/2016	4.1	<ul style="list-style-type: none"> <li>Updated to core version 4.1.</li> <li>Updated file name path to _ex/imports in all sections.</li> </ul> <p><b>DDR3 and DDR2</b></p> <ul style="list-style-type: none"> <li>Updated Controller Options Page figure.</li> <li>Added Number of Bank Machines bullet in the Controller Options section.</li> </ul>
06/08/2016	4.0	<p><b>DDR3 and DDR2</b></p> <ul style="list-style-type: none"> <li>Updated Memory Part description in Controller Option section.</li> <li>Added app_ecc_single_err[7:0] in Table 1-17: User Interface table.</li> <li>Added app_ecc_single_err[7:0] and note in Table 1-56: User Interface for ECC Operation.</li> <li>Updated description in dbg_pi_phase_locked_phy4lanes and dbg_pi_dqs_found_lanes_phy4lanes in Table 1-74: DDR2/DDR3 Debug Signals.</li> </ul>
04/06/2016	3.0	<ul style="list-style-type: none"> <li>Updated to core version 3.0.</li> <li>Updated Termination for all sections.</li> <li>Updated 1.0 µF capacitor in General Memory Routing Guideline chapter.</li> </ul> <p><b>DDR3 and DDR2</b></p> <ul style="list-style-type: none"> <li>Added note in FPGA Options section.</li> <li>Added note in Interfacing to the Core section.</li> <li>Updated sys_RST descriptions in DDR3 and DD2 Configuration sections.</li> <li>Added note in Debug Signals section.</li> <li>Updated reset description in General Checks section.</li> </ul>

11/18/2015	2.4	<ul style="list-style-type: none"> <li>Added asynchronous to sys_rst in all sections.</li> <li>Added note to RELAXED mode in DDR3/DDR2 and LDDR2 sections.</li> <li>Updated code in all Configuration sections</li> <li>Added Important jitter note in Pinout Requirements in all sections.</li> </ul> <p><b>DDR3 and DDR2</b></p> <ul style="list-style-type: none"> <li>Added Synplify Pro Black Box Testing section.</li> </ul> <p><b>QDR II+</b></p> <ul style="list-style-type: none"> <li>Updated DEBUG_PORT Signal Descriptions, Write Init Debug Signal Map, and Read Stage 1 Debug Signal Map tables.</li> <li>Updated Calibration of Read Clock and Data description.</li> <li>Updated Write Calibration description.</li> </ul> <p><b>RLDRAM II/ RLDRAM 3</b></p> <ul style="list-style-type: none"> <li>Updated Read Stage 1 Debug Signal Map table.</li> <li>Updated Calibration of Read Clock and Data description.</li> </ul>
09/30/2015	2.4	<ul style="list-style-type: none"> <li>Added CLOCK_DEDICATED_ROUTE Constraints in all sections.</li> </ul> <p><b>DDR3 and DDR2</b></p> <ul style="list-style-type: none"> <li>Updated Trace Lengths section.</li> </ul> <p><b>QDR II+</b></p> <ul style="list-style-type: none"> <li>Added Termination section.</li> </ul> <p><b>RLDRAM II/ RLDRAM 3</b></p> <ul style="list-style-type: none"> <li>Added Termination section.</li> <li>Updated Margin Check section.</li> <li>Updated Automatic Margin Check section.</li> <li>Updated Table 3-33: Debug Port Signals.</li> </ul> <p><b>LPDDR2</b></p> <ul style="list-style-type: none"> <li>Updated Trace Lengths section.</li> </ul> <p><b>Appendix</b></p> <ul style="list-style-type: none"> <li>Added General Memory Routing Guidelines.</li> </ul>
06/24/2015	2.3	<ul style="list-style-type: none"> <li>Added Simulation Flow Using VCS and IES to all sections.</li> <li>Added Clocking sections to QDR II+, RLDRAM II/RLDRAM 3, and LPDDR2 chapters.</li> </ul> <p><b>RLDRAM II/ RLDRAM 3</b></p> <ul style="list-style-type: none"> <li>Added address/control signal and SSI descriptions in Pinout Requirements section.</li> <li>Updated Input Clock Guidelines section.</li> </ul>

04/01/2015	2.3	<ul style="list-style-type: none"> <li>• Updated description in all Configuration sections.</li> <li>• Updated SIM_BYPASS_INIT_CAL.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>• Added description in Setting DDR3 Memory Parameter Option section.</li> <li>• Added Note to Answer Record: 54025 in Controller Options section.</li> <li>• Added description to app_rd_data_end in Table 1-17: User Interface.</li> <li>• Updated Table 1-19: AXI4 Slave Interface Parameters.</li> <li>• Updated description in AXI4 Slave Interface Signals section.</li> <li>• Updated Time Division Multiplexing (TDM), Round-Robin, and Read Priority (RD_PRI_REG) sections.</li> <li>• Updated GES description in Calibration Times section.</li> <li>• Updated Fig. 1-50: Clocking Architecture.</li> <li>• Updated Table 1-87: Memory Controller to Calibration Logic Interface Signals.</li> <li>• Updated AXI Addressing section.</li> <li>• Updated Write Path section.</li> <li>• Updated Fig. 1-84: Command Processing.</li> <li>• Updated Physical Layer Interface (Non-Memory Controller Design) section.</li> <li>• Updated CK signal description in Trace Length section.</li> <li>• Updated Fig. 1-93: Calibration Stages.</li> <li>• Updated description in Determine the Failing Calibration Stage section.</li> <li>• Updated Table 1-100: DDR2/DDR3 Debug Signals.</li> <li>• Updated Table 1-102: Debug Signals of Interest for Write Leveling Calibration.</li> <li>• Updated Table 1-103: Debug Signals of Interest for MPR Read Leveling Calibration.</li> <li>• Updated calibration overview in Debugging OCLKDELAYED Calibration Failures section.</li> <li>• Updated Debug bullets in Debugging OCLKDELAYED Calibration Failures section.</li> <li>• Updated Table 1-104: Debug Signals of Interest for OCLKDELAYED Calibration to Table 1-106: Debug Signals of Interest for Read Leveling Stage 1 Calibration.</li> <li>• Updated Table 1-108: Calibration Time in Hardware.</li> <li>• Updated Checking and Varying Read Timing to Manual Window Check sections.</li> <li>• Updated Calibration Times section.</li> </ul> <p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>• Updated Fig. 2-43: High-Level PHY Block Diagram for a 36-Bit QDR II+ Interface.</li> <li>• Updated Margin Check and Automated Margin Check sections.</li> </ul>
Continued		<p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>• Updated description in Interfacing with the Core through the Client Interface section.</li> </ul> <p><b>Chapter 4</b></p> <ul style="list-style-type: none"> <li>• Corrected app_wdf_data[APP_DATA_WIDTH - 1:0] and app_wdf_mask[APP_MASK_WIDTH - 1:0] sections.</li> <li>• Updated Fig. 4-43: Clocking Architecture.</li> <li>• Updated Read Path section.</li> </ul>

11/19/2014	2.3	<p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>• Updated description in Round-Robin section.</li> <li>• Updated RTT_WR in Table 1-92: 7 Series FPGA Memory Solution Configuration Parameters.</li> <li>• Updated description in Debugging OCLKDELAYED Calibration Failures section.</li> <li>• Updated Table 1-106: Debug Signals of Interest for OCLKDELAYED Calibration.</li> <li>• Updated GES time in Calibration Times section.</li> <li>• Updated bits in left_loss_pb and right_gain_pb in Table 1-109: Debug Signals of Interest for PRBS Read Leveling Calibration.</li> </ul>
10/01/2014	2.2	<ul style="list-style-type: none"> <li>• Global update to example design link in Files in example_design/sim Directory tables, updated links in Simulation Flow Using IES and VCS Script Files section, updated Simulation Flow Using Vivado Simulator section, and updated Simulation Flow Using QuestaSim section.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>• Updated Reference Clock description in FPGA Option section.</li> <li>• Updated C_S_AXI_DATA_WIDTH description in Table 1-19: AXI4 Slave Interface Parameters.</li> <li>• Updated Fig. 1-50: Clocking Architecture.</li> <li>• Updated OCLKDELAYED Calibration section.</li> <li>• Updated Write Path section.</li> <li>• Added REF_CLK_MMCM_IODELAY_CTRL in Table 1-92: 7 Series FPGA Memory Solution Configuration Parameters.</li> <li>• Added note for nBANK_MACHS in Table 1-93: Embedded 7 Series FPGAs Memory Solution Configuration Parameters.</li> <li>• Added row and updated Table 1-94: DDR2/DDR3 SDRAM Memory Interface Solution Pinout Parameters</li> <li>• Updated CK/CK# bullet in Trace Length section.</li> <li>• Updated Table 1-102: DDR2/DDR3 Debug Signals.</li> <li>• Updated debug signals in Table 1-112: Debug Signals Used for Checking and Varying Read/Write Timing.</li> </ul>
Continued		<p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>• Added Bank Sharing Among Controllers section in Design Guideline section.</li> </ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>• Added Bank Sharing Among Controllers section in Design Guideline section.</li> </ul> <p><b>Chapter 4</b></p> <ul style="list-style-type: none"> <li>• Updated Figs. 4-57 to 4-59 and Figs. 4-62 to 4-63.</li> <li>• Updated 2:1 description in Write Path section.</li> <li>• Updated rules in Termination section.</li> </ul>

06/04/2014	2.1	<p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>Added reference to data sheet in Features section.</li> <li>Added Important note about Data Mask in Controller Options section.</li> <li>Added note in Precharge Policy section.</li> <li>Added PRBS_SADDR_MASK)POS to Table 1-11: Traffic Generator Parameters Set in the example_top Module.</li> <li>Updated IDELAYCTRL frequency in IDELAYCTRL section.</li> <li>Updated IDELAY Reference Clock section.</li> <li>Updated PRBS Read Leveling section.</li> <li>Updated CL description for DDR3 in Table 1-93: Embedded 7 Series FPGAs Memory Solution Configuration Parameters.</li> <li>Updated package length descriptions in Trace Length section.</li> <li>Added simulation description in Note in Debugging DDR3/DDR2 Designs.</li> <li>Updated description in Debugging PRBS Read Leveling Failures section.</li> <li>Updated Table 109: Debug Signals of Interest for PRBS Read Leveling Calibration.</li> </ul> <p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>Added reference to data sheet in Introduction section.</li> <li>Updated package length descriptions in Trace Length Requirements section.</li> <li>Added CPT_CLK_SEL_* row in Table 2-11: QDR II+ SRAM Memory Interface Solution Pinout Parameters.</li> <li>Added simulation description in Note in Debugging QDR II+ Designs.</li> </ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>Added reference to data sheet in Features section.</li> <li>Added note in Memory Controller section.</li> <li>Added PRBS_SADDR_MASK)POS to Table 3-8: Traffic Generator Parameters Set in the example_top Module.</li> <li>Updated rules and package length descriptions in Trace Length Requirements section.</li> <li>Added simulation description in Note in Debugging RLDRAM II and 3 Designs.</li> </ul>
Continued		<p><b>Chapter 4</b></p> <ul style="list-style-type: none"> <li>Added note in Precharge Policy section.</li> <li>Added PRBS_SADDR_MASK)POS to Table 4-11: Traffic Generator Parameters Set in the example_top Module.</li> <li>Updated package length descriptions in Trace Length Requirements section.</li> <li>Added simulation description in Note in Read Path section.</li> </ul>

**Chapter 1**

- Updated book to DQS.
- Updated Table 1-4: Files in example\_design/sim Directory.
- Updated file descri %

12/18/2013	2.0	<ul style="list-style-type: none"> <li>• Vivado Design Suite release only for MIS v2.0.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>• Added Out of Context content.</li> <li>• Updated Table 1-4: Modules in example_design/sim Directory.</li> <li>• Updated &lt;component name&gt;/user_design section.</li> <li>• Updated Fig. 1-39: Synthesizable Example Design Block Diagram.</li> <li>• Added simulator flows.</li> <li>• Added Bits[39:32] to Table 1-15: Debug Status for the Write Transaction.</li> <li>• Added Bits[39:32] to Table 1-16: Debug Status for the Read Transaction.</li> <li>• Added OOC description in Customizing the Core section.</li> <li>• Added ILA trigger settings in Vivado Lab Tools section.</li> <li>• Added note on read latency in Debug section.</li> <li>• Updated Chipscope triggers to R in Debug section.</li> </ul> <p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>• Added Out of Context content.</li> <li>• Updated Table 2-3: Modules in example_design/sim Directory.</li> <li>• Updated &lt;component name&gt;/user_design section.</li> <li>• Added OOC description in Customizing the Core section.</li> <li>• Added simulator flows.</li> <li>• Added ILA trigger settings in Vivado Lab Tools section.</li> </ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>• Added Out of Context content.</li> <li>• Updated Table 3-3: Modules in example_design/sim Directory.</li> <li>• Updated &lt;component name&gt;/user_design section.</li> <li>• Updated Fig. 3-35: Synthesizable Example Design Block Diagram.</li> <li>• Added OOC description in Customizing the Core section.</li> <li>• Added simulator flows.</li> <li>• Added ILA trigger settings in Vivado Lab Tools section.</li> <li>• Updated Fig. 3-48 Write Path Block Diagram of the RLDRAM II Interface Solution.</li> <li>• Added note on read latency in Debug section.</li> </ul> <p><b>Chapter 4</b></p> <ul style="list-style-type: none"> <li>• Added Out of Context content.</li> <li>• Updated Table 4-4: Modules in example_design/sim Directory.</li> <li>• Updated &lt;component name&gt;/user_design section.</li> <li>• Updated Fig. 4-37: Synthesizable Example Design Block Diagram.</li> <li>• Added OOC description in Customizing the Core section.</li> <li>• Added simulator flows.</li> <li>• Added note on read latency in Debug section.</li> </ul> <p><b>Chapter 5</b></p> <ul style="list-style-type: none"> <li>• Added Out of Context content.</li> </ul>

		<ul style="list-style-type: none"> <li>• Vivado Design Suite release only for MIS v2.0.</li> <li>• Removed ISE content throughout book and updated screenshots to v2.0.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>• Updated Memory Part bullet description.</li> <li>• Updated Table 1-4 sim.do description and simulation directory.</li> <li>• Updated Fig. 1-44 7 Series FPGAs MIS.</li> <li>• Added aresetn in Table 1-20 AXI4 Slave Interface Signals.</li> <li>• Added Caution note in Single Error and Double Error Reporting section.</li> <li>• Updated Table 1-77 Memory Interface Commands.</li> <li>• Updated and added stage 3 tap in OCLKDELAYED Calibration section.</li> <li>• Added #4 table note to Table 1-91 7 Series FPGA Memory Solution Configuration Parameters.</li> <li>• Updated description in app_wdf_mask[APP_MASK_WIDTH - 1:0] section.</li> <li>• Added Memory Address Mapping description in User Interface section.</li> <li>• Updated Table 1-106 Debug Signals of Interest for OCLKDELAYED Calibration</li> </ul> <p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>• Updated Table 2-3 sim.do description and simulation directory.</li> <li>• Updated DIFF_HSTL_I in I/O Standards table.</li> <li>• Updated reference clock descriptions in Clocking Architecture section.</li> <li>• Added #1 table note to Table 2-11 7 Series FPGAs QDR II+ SRAM Memory Interface Solution Configurable Parameters. And updated SIM_BYPASS_INIT_CAL.</li> </ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>• Updated Table 3-3 sim.do description and simulation directory.</li> <li>• Updated reference clock descriptions in Clocking Architecture section.</li> <li>• Added #1 table note to Table 3-13 RLDRAM II Memory Interface Solution Configurable Parameters. And updated SIM_BYPASS_INIT_CAL.</li> </ul> <p><b>Chapter 4</b></p> <ul style="list-style-type: none"> <li>• Updated Table 4-4 sim.do description and simulation directory.</li> <li>• Updated Fig. 4-37 7 Series FPGAs MIS.</li> <li>• Updated Table 4-14 User Interface.</li> <li>• Added #4 note to Table 4-25 7 Series FPGA Memory Solution Configuration Parameters.</li> <li>• Updated description in app_wdf_mask[APP_MASK_WIDTH - 1:0] section.</li> <li>• Added Memory Address Mapping description in User Interface section.</li> </ul>
10/02/2013	2.0	

06/19/2013	2.0	<ul style="list-style-type: none"> <li>• Vivado Design Suite release only for MIS v2.0. Revision number advanced to 2.0 to align with core version number.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>• Updated ChipScope to Vivado logic analyzer, VIO, and ILA.</li> <li>• Updated ui_clk and ui_clk_sync_rst descriptions in Table 1-17 User Interface.</li> <li>• Updated ui_clk and ui_clk_sync_rst descriptions.</li> <li>• Added Ordering Modes in Reordering section and added modes in Table 1-91.</li> <li>• Updated ECC enable in AXI4 Slave Interface Block section.</li> <li>• Updated Read Priority (RD_PRI) section.</li> <li>• Updated Table 1-19 AXI4 Slave Interface Parameters, C_S_AXI_ADDR_WIDTH value and descriptions.</li> <li>• Added Write Priority description.</li> <li>• Updated PHASER_IN DQSFOUND Calibration section.</li> <li>• Removed Downsizing Option.</li> <li>• Added DM in DQ descriptions.</li> <li>• Added Dynamic Calibration and Periodic Read Behavior section.</li> <li>• Added Vivado Lab Tools section.</li> <li>• Added AR 54025 for Vivado.</li> <li>• Updated Debugging PHASER_IN DQSFOUND Calibration Failures (dbg_pi_dqsfound_err = 1) section.</li> </ul> <p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>• Updated ChipScope to Vivado logic analyzer, VIO, and ILA.</li> <li>• Added Fixed Latency Mode description in Controller Options section.</li> <li>• Removed qdr_qvld in Table 2-12 Physical Interface Signals.</li> <li>• Updated Figure 2-26 Four-Word Burst Length Memory Device Protocol.</li> <li>• Updated Output Architecture section in Write Path.</li> <li>• Added Write Calibration section.</li> <li>• Removed QVLD.</li> <li>• Updated Table 2-20 Write Init Debug Signal Map.</li> <li>• Updated Tables 2-21 and 2-22 Read Stage 1 and Stage 2 Debug Signal Map tables.</li> </ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>• Updated ChipScope to Vivado logic analyzer, VIO, and ILA.</li> <li>• Removed rld_qvld in Table 3-13 Physical Interface Signals.</li> <li>• Removed QVLD and QVLD_MAP in Table 3-16 RLDRAM II Memory Interface Solution Pinout Parameters.</li> <li>• Removed QVLD.</li> <li>• Updated descriptions in Manual Pinout Changes section.</li> <li>• Added new calibration description in Calibration section.</li> <li>• Updated Table 3-26 Physical Layer Simple Status Bus Description Defined in the rld_phy_top Module.</li> </ul>

		<ul style="list-style-type: none"> <li>Updated Table 3-27 DEBUG_PORT Signal with dbg_rd_stage1_rtr_error[N_DATA_LANES - 1:0] and dbg_rd_stage1_error[N_DATA_LANES - 1:0].</li> <li>Updated Tables 3-31 and 3-32 Read Stage 1 and Stage 2 Debug Signal Map tables.</li> <li>Added Fig. 3-36 Calibration Flow Diagram and Fig. 3-37 Read Level Stage 1.</li> <li>Added description to Data Alignment and Valid Generation section.</li> <li>Updated description and added Figs. 3-38 to 3-43 in Write Calibration section.</li> <li>Added Write Calibration Debug Map section.</li> </ul> <p><b>Chapter 4</b></p> <ul style="list-style-type: none"> <li>Updated ChipScope to Vivado logic analyzer, VIO, and ILA.</li> <li>Updated ui_clk and ui_clk_sync_rst descriptions in Table 4-14 User Interface.</li> <li>Updated ui_clk and ui_clk_sync_rst descriptions.</li> <li>Added Ordering Modes in Reordering section and added modes in Table 4-25.</li> <li>Added DM in DQ descriptions.</li> <li>Added Termination description in LPDDR2 Pinout Examples section.</li> </ul> <p><b>Chapter 6</b></p> <ul style="list-style-type: none"> <li>Added Upgrading the ISE/CORE Generator MIS Core in Vivado section.</li> </ul>
03/20/2013	1.9	<ul style="list-style-type: none"> <li>ISE 14.5 and Vivado Design Suite 2013.1 releases for MIS v1.9 and v1.9a.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>Added Memory Part frequency in Controller Options section.</li> <li>Added No Buffer option description in FPGA Options section.</li> <li>Added pinout description in Verify Pin Changes and Update Design section.</li> <li>Updated Fig. 1-15 Setting Memory Mode Options.</li> <li>Updated Fig. 1-16 FPGA Options.</li> <li>Updated Fig. 1-30 7 Series FPGAs Memory Interface Solution to User's FPGA Logic</li> <li>Added ECC description in AXI4 Slave Interface Block section.</li> <li>Updated Table 1-91 7 Series FPGA Memory Solution Configuration Parameters.</li> <li>Updated Table 1-92 Embedded 7 Series FPGAs Memory Solution Configuration Parameters.</li> <li>Updated Table 1-93 DDR2/DDR3 SDRAM Memory Interface Solution Pinout Parameters.</li> <li>Added description in Verifying the Simulation Using the Example Design section.</li> <li>Reworked Design Guidelines DDR3 SDRAM section.</li> <li>Added new debug section.</li> </ul>

Continued		<p><b>Chapter 2</b></p> <ul style="list-style-type: none"><li>Added No Buffer option description in FPGA Options section.</li><li>Added pinout description in Verify Pin Changes and Update Design section.</li><li>Updated Fig. 2-15 FPGA Options.</li><li>Updated REFCLK_FREQ and RST_ACT_LOW in Table 2-13 7 Series FPGAs QDR II+ SRAM Memory Interface Solution Configurable Parameters</li><li>Updated Table 2-14 QDR II+ SRAM Memory Interface Solution Pinout Parameters.</li><li>Added description in Verifying the Simulation Using the Example Design section.</li></ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"><li>Added No Buffer option description in FPGA Options section.</li><li>Updated Fig. 3-14 FPGA Options.</li><li>Added Verify Pin Changes and Update Design section.</li><li>Updated nCK_PER_CLK in Table 3-10 Traffic Generator Parameters Set in the example_top Module</li><li>Updated Table 3-15 RLDRAM II Memory Interface Solution Configurable Parameters.</li><li>Updated Table 3-16 RLDRAM II Memory Interface Solution Pinout Parameters.</li><li>Added description in Verifying the Simulation Using the Example Design section.</li></ul> <p><b>Chapter 4</b></p> <ul style="list-style-type: none"><li>Added new LPDDR2 SDRAM section.</li></ul> <p><b>Chapter 6</b></p> <ul style="list-style-type: none"><li>Updated to new GUIs.</li></ul>

12/18/2012	1.8	<ul style="list-style-type: none"> <li>ISE 14.4 and Vivado 2012.4 Design Suite releases for MIS v1.8.</li> </ul> <p><b>Chapter 1</b></p> <ul style="list-style-type: none"> <li>Updated Table 1-2 to 1-9 with new table note and.v name.</li> <li>Updated Fig. 1-16 FPGA Options GUI.</li> <li>Added XADC Instantiation bullet.</li> <li>Added description to sim.do in Table 1-4.</li> <li>Updated Table 1-11 DATA_PATTERN to OxA.</li> <li>Updated Table 1-13 vio_data_mode_value[3:0] to OxA.</li> <li>Updated description in Setting Up for Simulation.</li> <li>Added description to EDK Clocking.</li> <li>Updated ui_clk and ui_clk_sync_rst in Table 1-17.</li> <li>Added description in Internal (FPGA) Logic Clock.</li> <li>Added TEMP_MON_CONTROL to Table 1-91.</li> <li>Added DATA_IO_IDLE_PWRDWN and CA_MIRROR to Table 1-92.</li> <li>Added HP bank description in Bank and Pin Selection Guides for DDR3 Designs.</li> <li>Added DDR3 SDRAM interface description to Configuration.</li> <li>Added HP bank description in Bank and Pin Selection Guides for DDR2 Designs.</li> <li>Added DDR2 SDRAM interface description to Configuration.</li> </ul> <p><b>Chapter 2</b></p> <ul style="list-style-type: none"> <li>Updated Table 2-2 and 2-7 to 2-8 with new table note and.v name.</li> <li>Added description to sim.do in Table 2-3.</li> <li>Updated descriptions and added Fig 2-26 to Clocking Architecture.</li> <li>Updated description in Write Path Output Architecture.</li> <li>Updated descriptions in Trace Length Requirements.</li> <li>Added QDRII description in Configuration.</li> <li>Added description to Verifying the Simulation Using the Example Design.</li> <li>Added Margin Check and Automated Margin Check sections.</li> </ul> <p><b>Chapter 3</b></p> <ul style="list-style-type: none"> <li>Updated Table 3-2 and 3-6 to 3-8 with new table note and.v name.</li> <li>Added description to sim.do in Table 3-3.</li> <li>Updated Table 3-10 DATA_PATTERN to OxA.</li> <li>Updated descriptions and added Fig 3-30 to Clocking Architecture.</li> <li>Updated descriptions in Trace Length Requirements.</li> <li>Added descriptions in RLDRAM II.</li> <li>Added RLDRAM II description in Configuration.</li> <li>Added description to Verifying the Simulation Using the Example Design.</li> <li>Added Debug section.</li> </ul>

10/16/2012	1.7	<ul style="list-style-type: none"> <li>MIS 1.7 release. Updated ISE Design Suite version to 14.3.</li> <li>Chapter 1: Added AXI4-Lite Slave Control/Status Register Interface Block section. Updated figures (1-32 and 1-37) and added PRBS and Temperature Monitor sections. Added CLKIN_PERIOD to USE_DM_PORT parameters in Table 1-37. Updated Table 1-38 PHYO_BITLANES description.</li> <li>Chapter 2: Added CLKIN_PERIOD to DIVCLK_DIVIDE parameters in Table 2-13.</li> <li>Chapter 3: Added RLDRAM 3 content throughout. Updated/added figures (3-10, 3-13, 3-23 to 3-32, 3-36 to 3-37, 3-40 to 3-41, 3-45 to 3-47, and 3-50). Added mem_ck_lock_complete parameter in Table 3-11. Added CLKOUTO_PHASE parameter in Table 3-15. Updated descriptions in Table 3-16 and added Table 3-28. Updated Table 3-29 user_cmd signal. Updated Table 3-31 and 3-34 descriptions. Added Debugging Write Calibration section.</li> <li>Chapter 4: Added System Clock Sharing section</li> <li>Chapter 5: Updated figures (5-15, 5-17 to 5-20), updated steps in Getting Started with Vivado – MIS IP Generation</li> </ul>
07/25/2012	1.6	<ul style="list-style-type: none"> <li>MIS 1.6 release. Updated ISE Design Suite version to 14.2. Updated GUI screen captures throughout document.</li> <li>Chapter 1: Added No Buffer, Use System Clock, and Sample Data Depth in FPGA Options, page 36. Changed the parameters NCK_PER_CLK, tZQI, SYSCLK_TYPE, REFCLK_TYPE, and APP_DATA_WIDTH. Added bulleted item about multiple CK outputs to Bank and Pin Selection Guides for DDR3 Designs, page 186. Updated Trace Lengths, page 191 and Termination, page 200.</li> <li>Chapter 2: Added No Buffer, Use System Clock, and Sample Data Depth in FPGA Options, page 282. Changed the parameters SYSCLK_TYPE and REFCLK_TYPE.</li> <li>Chapter 3: Added No Buffer, Use System Clock, and Sample Data Depth in FPGA Options, page 282. Changed the parameters SYSCLK_TYPE and REFCLK_TYPE.</li> <li>Chapter 6: Added new chapter on migrating to Vivado Design Suite.</li> </ul>
06/13/2012	1.5	Revised the recommended total electrical delay on CK/CK# relative to DQS/DQS# on page 191.

04/24/2012	1.4	<ul style="list-style-type: none"> <li>MIS 1.5 release. Updated ISE Design Suite version to 14.1. Updated GUI screen captures throughout document. Replaced IODELAYCTRL with IDELAYCTRL throughout.</li> <li>Chapter 1: Added I/O Power Reduction option to FPGA Options. Revised I/O standards for sys_rst option in Bank Selection. Added Creating ISE Project Navigator Flow for MIS Example Design, Power-Saving Features, Multi-Purpose Register Read Leveling, OCLKDELAYED Calibration, Upsizing, and External Vref sections. Changed bits [16:15] to from Rank Count to Reserved in the PHY Control word. Revised maximum setting of NUM_DQ_PINS in Table 1-11. Revised Figure 1-55 flowchart. Removed RankSel[1:0] from Figure 1-56 and Figure 1-58. Added mc_odt and mc_cke to Table 1-87. Replaced AXI Addressing. Updated REFCLK_FREQ, RANK_WIDTH, and WRLVL in Table 1-92. Added DATA_IO_PRIM_TYPE to Table 1-93. Added bullet about DQS pins to Bank and Pin Selection Guides for DDR3 Designs. Changed DIFF_SSTL_15 to DIFF_SSTL18_II and SSTL15 to SSTL18_II.</li> <li>Chapter 2: Changed DIFF_SSTL_15 to DIFF_HSTL_I and SSTL15 to HSTL_I. Revised I/O standards for sys_rst option in System Pins Selection. Revised the PHY_BITLANE parameters in Table 2-11. Added System Clock, PLL Location, and Constraints and Configuration sections.</li> <li>Chapter 3: Changed DIFF_SSTL_15 to DIFF_HSTL_I and SSTL15 to HSTL_I. to Revised I/O standards for sys_rst option in System Pins Selection. Added the Write Calibration, System Clock, PLL Location, and Constraints, and Configuration sections. Revised the PHY_BITLANE parameters in Table 3-15. In Table 3-28, added dbg_wrcal_sel_stg[1:0], dbg_wrcal[63:0], dbg_wrcal_done[2:0], dbg_wrcal_po_first_edge[5:0], dbg_wrcal_po_second_edge[5:0], and dbg_wrcal_po_final[5:0].</li> </ul>
01/18/2012	1.3	<ul style="list-style-type: none"> <li>MIS 1.4 release. Updated ISE Design Suite version to 13.4. Updated GUI screen captures throughout document.</li> <li>Chapter 1: Added support for DDR2 SDRAM. Added option 3 to MIS Output Options. Added EDK Clocking. Added Replaced Figure 1-41 and Figure 1-69.</li> <li>Chapter 2: Removed Input Clock Period option from Controller Options. Added Memory Options. Added Reference Clock option to FPGA Options. Updated Debug Signals.</li> <li>Chapter 3: Removed Input Clock Period option from Controller Options. Added Input Clock Period option to Memory Options. Added Reference Clock option to FPGA Options. Added Debugging RLDRAM II and RLDRAM 3 Designs.</li> </ul>

10/19/2011	1.2	<ul style="list-style-type: none"> <li>• MIS 1.3 release. Updated ISE Design Suite version to 13.3.</li> <li>• Chapter 1: Added step 2 to MIS Output Options, page 26. Added note about optional use of the memory controller to Controller Options, page 30. Added arbitration scheme to AXI Parameter Options, page 33. Added description of DCI Cascade under Figure 1-23. Updated text about devices with SSI technology and SLRs on page 41 and page 187. Changed error to tg_compare_error on page 42. Replaced Table 1-8. Added qdr_wr_cmd_o, vio_fixed_instr_value, vio_fixed_b1_value, vio_pause_traffic, and vio_data_mask_gen signals to Table 1-13. Added signals to the User Interface in Figure 1-49 and Figure 1-51. Added app_sr_req, app_sr_active, app_ref_req, app_ref_ack, app_zq_req, and app_zq_ack signals to Table 1-17. Added app_wdf_rdy, app_ref_req, app_ref_ack, app_zq_req, app_zq_ack, Read Priority with Starve Limit (RD_PRI_REG_STARVE_LIMIT), Native Interface Maintenance Command Signals, User Refresh, and User ZQ sections. Added C_RD_WR_ARB_ALGORITHM to Table 1-19. Updated fields in Table 1-84, changed Hi Index (Rank) to Rank Count, and added CAS slot field. Updated AXI Addressing and Physical Layer Interface (Non-Memory Controller Design). Added Figure 1-75 through Figure 1-77 in Write Path. In Table 1-92, removed DISABLED option from RTT_NOM for DDR3_SDRAM, changed RTT_NOM to RTT_WR in RTT_WR, updated SIM_BYPASS_INIT_CAL, and updated table note 2. In Table 1-93, updated tZQI and added USER_REFRESH. Added Table 1-94. In Configuration, updated constraints example and removed paragraph about SCL and SDA.</li> <li>• Chapter 2: Added step 2 to MIS Output Options, page 275. Added Input Clock Period description in Controller Options, page 279. Added Debug Signals Control and Internal Vref Selection options to FPGA Options, page 282. Added I/O Planning Options, page 285. In System Pins Selection, page 288, changed cal_done signal to init_calib_complete and error signal to tg_compare_error. Replaced Table 2-2. Changed file names in Table 2-5. Updated signal names in Figure 2-38, Figure 2-39, and Figure 2-40. Updated signal names in Table 2-7. Added CPT_CLK_CQ_ONLY and updated value for SIM_BYPASS_INIT_CAL in Table 2-10. Added Table 2-11. Updated pinout rules in Pinout Requirements, page 337. Added paragraph about DCI and IN_TERM after Table 2-12. Added Debugging QDR II+ SRAM Designs, page 340.</li> <li>• Chapter 3: Added step 2 to MIS Output Options, page 375. Added Input Clock Period description in Controller Options. Added Debug Signals Control and Internal Vref Selection options to FPGA Options, page 382. In System Pins Selection, changed cal_done signal to init_calib_complete and error signal to tg_compare_error. Changed file names in Table 3-6. Removed Table 3-12, which contained Reserved signals not used. Added rst_phaser_ref to Table 3-11. Removed PHY-Only Interface section. In Table 3-14, added RLD_ADDR_WIDTH, MEM_TYPE, CLKIN_PERIOD, and SIMULATION, and renamed CLKFBOUT_MULT, CLKOUT0_DIVIDE, CLKOUT1_DIVIDE, CLKOUT2_DIVIDE, and CLKOUT3_DIVIDE. Updated Table 3-15. Added paragraph about DCI and IN_TERM after Table 3-24.</li> <li>• Added Chapter 5, Multicontroller Design.</li> </ul>

- MIS 1.2 release. Updated ISE Design Suite version to 13.2. Updated GUI screen captures throughout document.
- Chapter 1: Added Verify Pin Changes and Update Design, Simulating the Example Design (for Designs with the AXI4 Interface), Error Correcting Code, and DDR3 Pinout Examples sections. Added paragraph about SLRs to Pin Compatible FPGAs, page 27. Added Input Clock Period and PHY to Controller bullets in Controller Options, page 30. To Setting DDR3 Memory Parameter Option, page 35, indicated that DDR3 SDRAM supports burst lengths of 8. Added Internal Termination for High Range Banks option under Figure 1-23. Added bulleted item about Pin/Bank selection mode on page E<sup>a</sup>

06/22/2011 1.1



# Table of Contents



# DDR3 and DDR2 SDRAM Memory Interface Solution

---

The Xilinx® 7 series FPGAs Memory Interface Solutions (MIS) 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 interface core for 7 series FPGAs. The user guide describes the core architecture and provides details on customizing and interfacing to the core.



**IMPORTANT:** *Memory Interface Solutions v4.2 only supports the Vivado® Design Suite. The ISE® Design Suite is not supported in this version.*

---

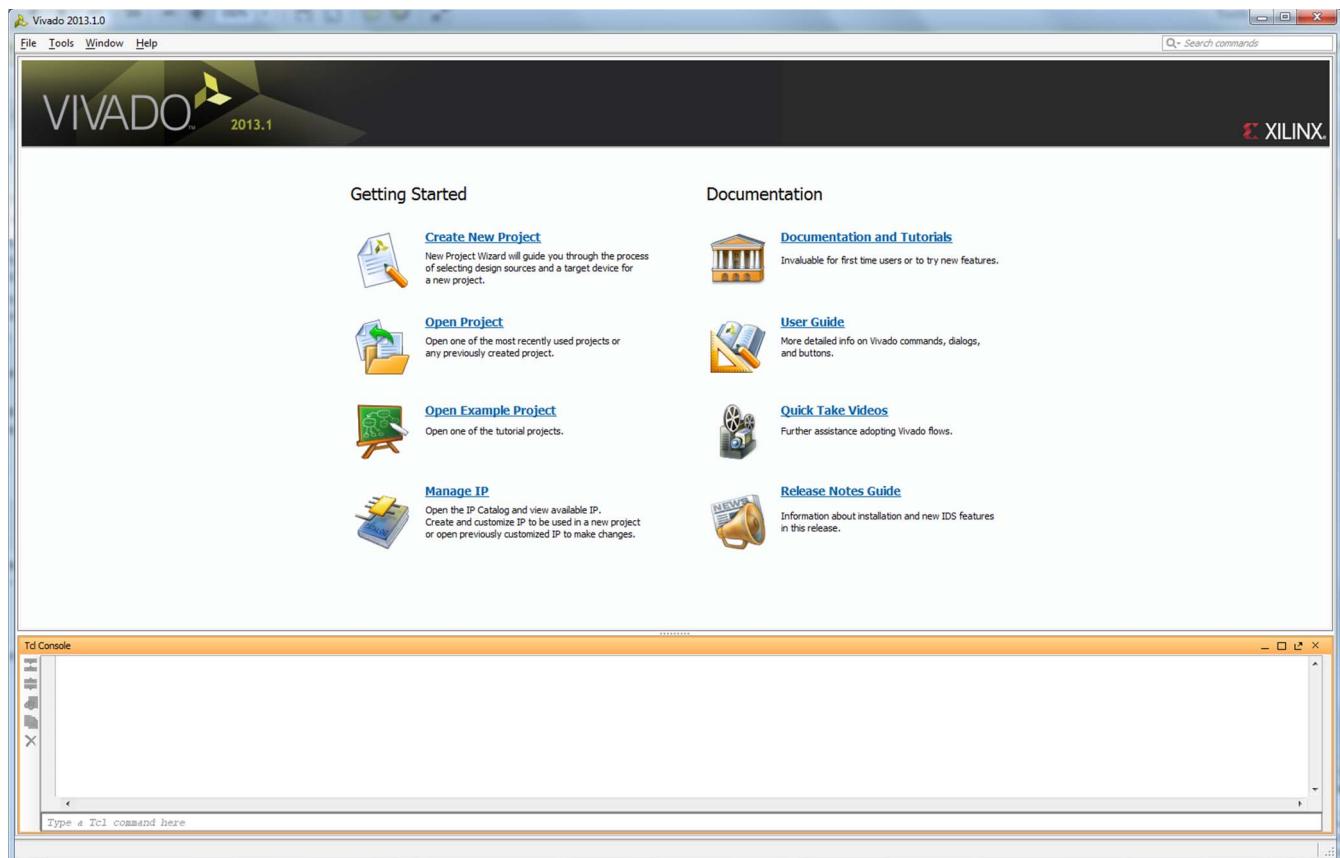
Enhancements to the Xilinx 7 series FPGA memory interface solutions from earlier memory interface solution device families include:

- 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 90](#)).
- Pinout rules changed due to the hardware blocks (see [Design Guidelines, page 192](#)).
- Controller and user interface operate at 1/4 the memory clock frequency.

For a full list of supported features, see the *Zynq-7000 SoC and 7 Series FPGAs Memory Interface Solutions Data Sheet* (DS176) [\[Ref 1\]](#).

This section provides the steps to generate the Memory Interface Generator (MIG) IP core using the Vivado Design Suite and run implementation.

1. Start the Vivado Design Suite (see [Figure 1-1](#)).



*Figure 1-1:*

2. To create a new project, click the **Create New Project** option shown in [Figure 1-1](#) to open the page as shown in [Figure 1-2](#).

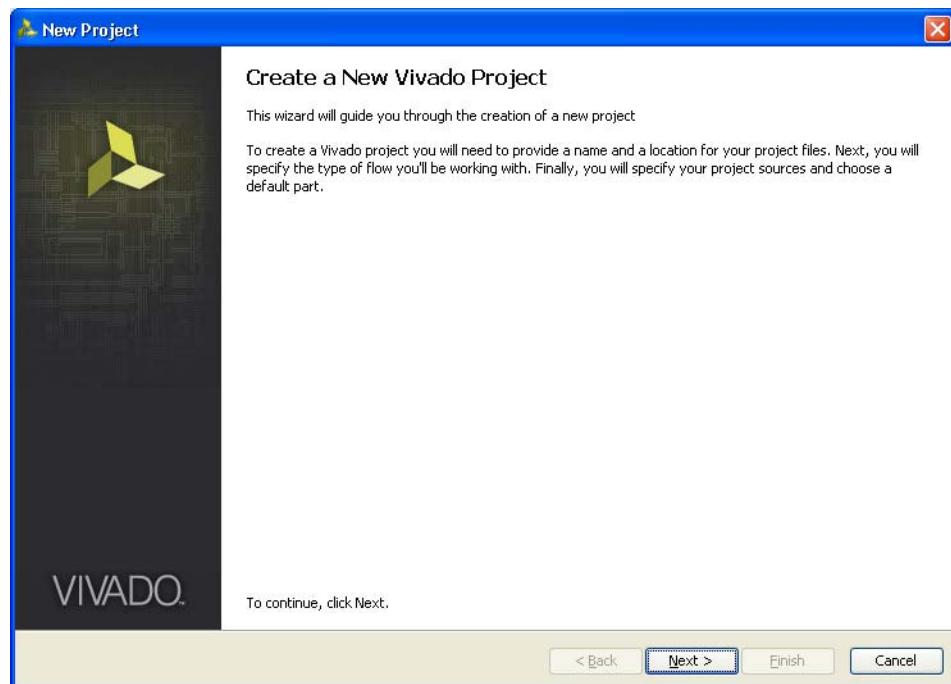


Figure 1-2:

3. Click **Next** to proceed to the **Project Name** page (Figure 1-3). Enter the **Project Name** and **Project Location**. Based on the details provided, the project is saved in the directory.

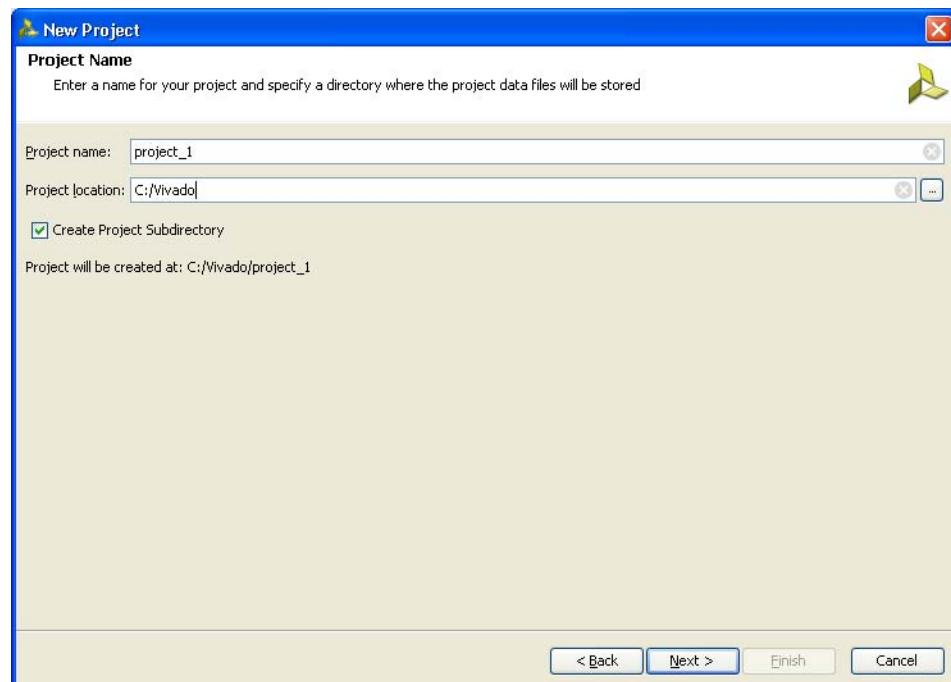


Figure 1-3:

- Click **Next** to proceed to the **Project Type** page (Figure 1-4). Select the **Project Type** as **RTL Project** because MIG deliverables are RTL files.

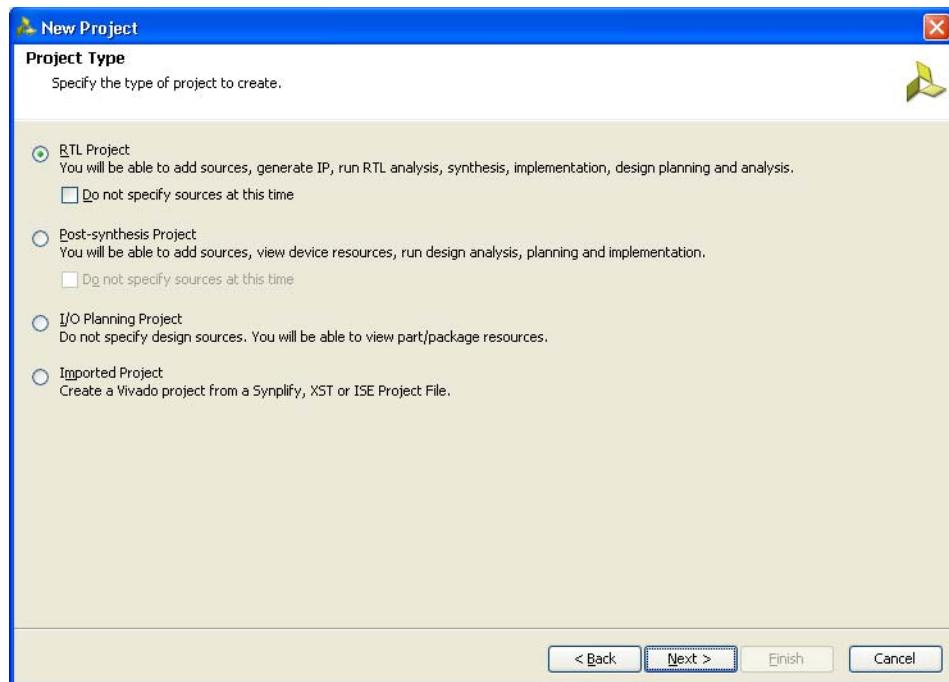


Figure 1-4:

- Click **Next** to proceed to the **Add Sources** page (Figure 1-5). RTL files can be added to the project in this page. If the project was not created earlier, proceed to the next page.

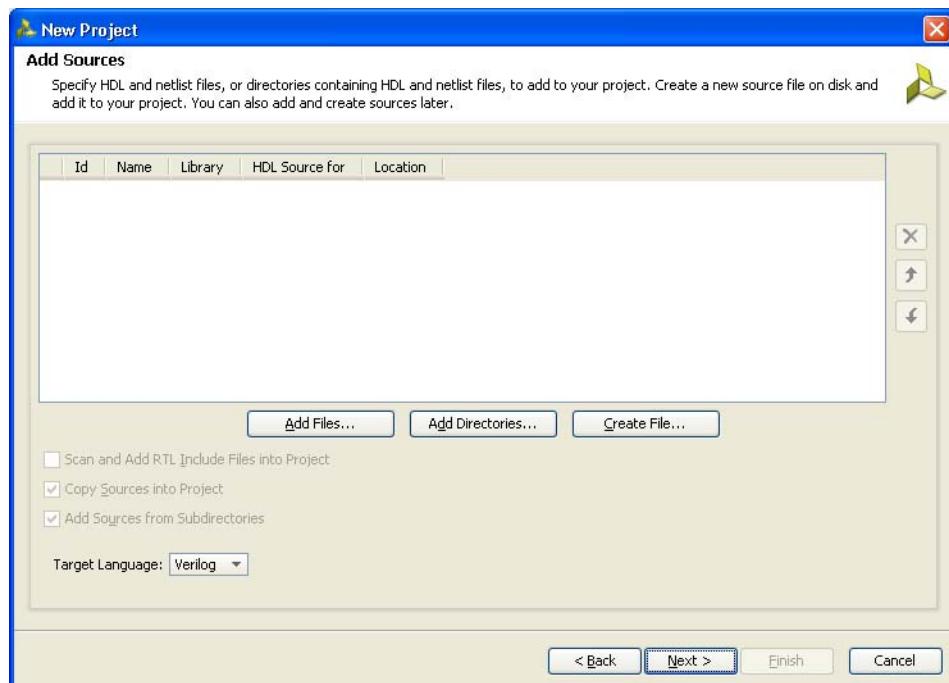


Figure 1-5:

6. Click **Next** to open the **Add Existing IP (Optional)** page (Figure 1-6). If the IP is already created, the XCI file generated by the IP can be added to the project and the previous created IP files are automatically added to the project. If the IP was not created earlier, proceed to the next page.

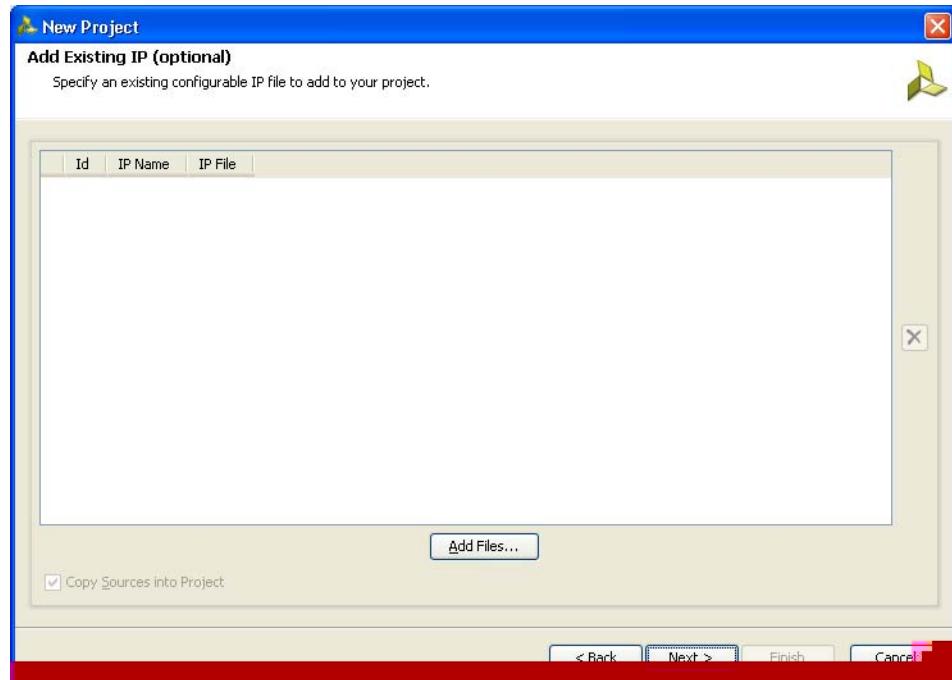


Figure 1-6:

7. Click **Next** to open the **Add Constraints (Optional)** page (Figure 1-7). If the constraints file exists in the repository, it can be added to the project. Proceed to the next page if the constraints file does not exist.

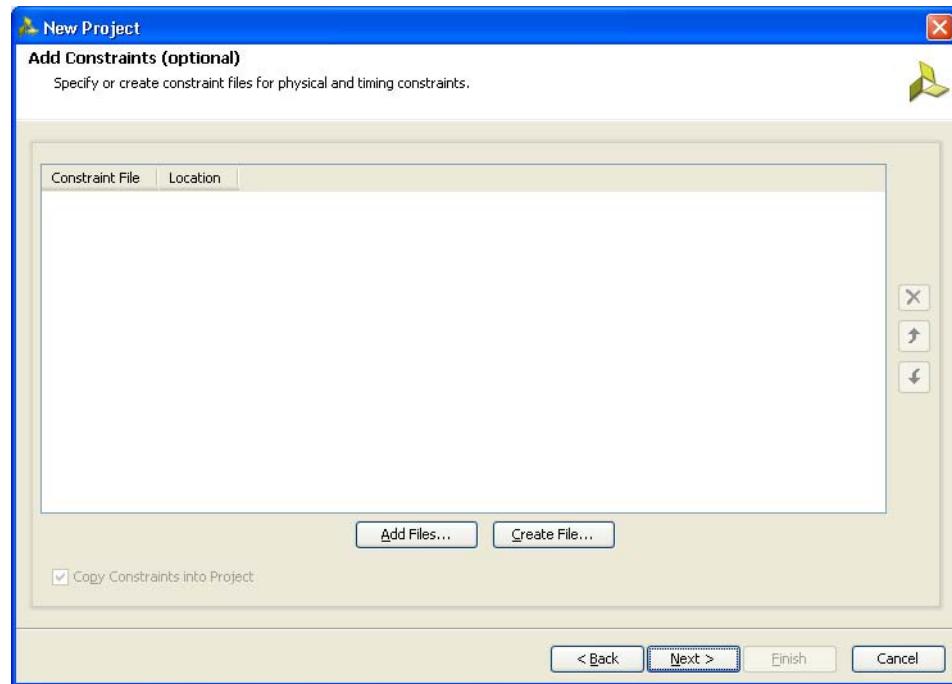


Figure 1-7:

- Click **Next** to proceed to the **Default Part** page (Figure 1-8) where the device that needs to be targeted can be selected. The **Default Part** page appears as shown in Figure 1-8.

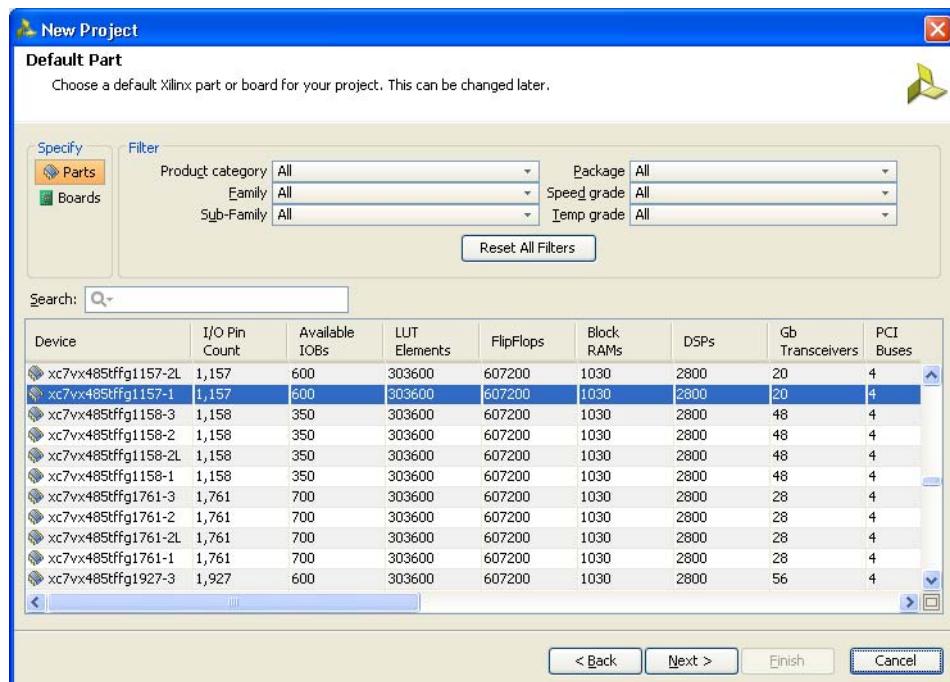
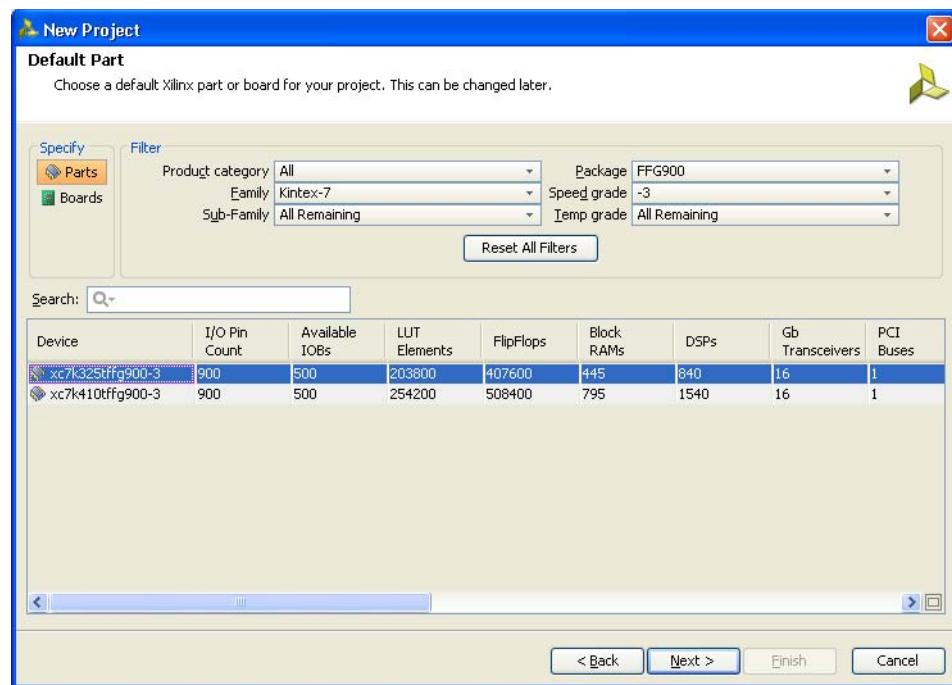


Figure 1-8:

Select the target **Family**, **Package**, and **Speed Grade**. The valid devices are displayed in the same page, and the device can be selected based on the targeted device ([Figure 1-9](#)).



*Figure 1-9:*

Apart from selecting the parts by using **Parts** option, parts can be selected by choosing the **Boards** option, which brings up the evaluation boards supported by Xilinx ([Figure 1-10](#)). With this option, design can be targeted for the various evaluation boards. If the XCI file of an existing IP was selected in an earlier step, the same part should be selected here.

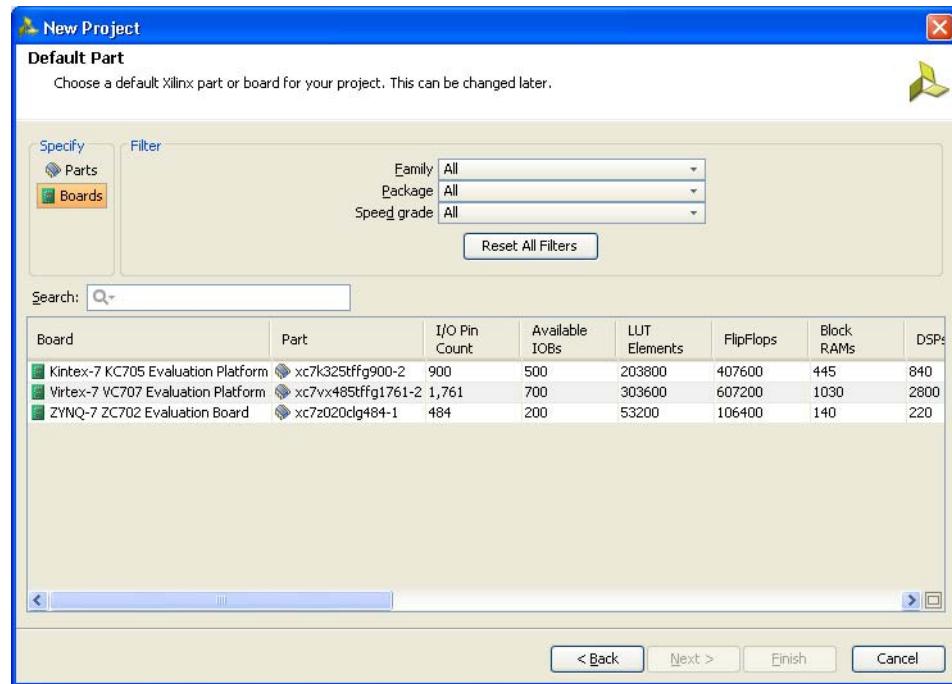


Figure 1-10:

- Click **Next** to open the **New Project Summary** page (Figure 1-11). This includes the summary of selected project details.

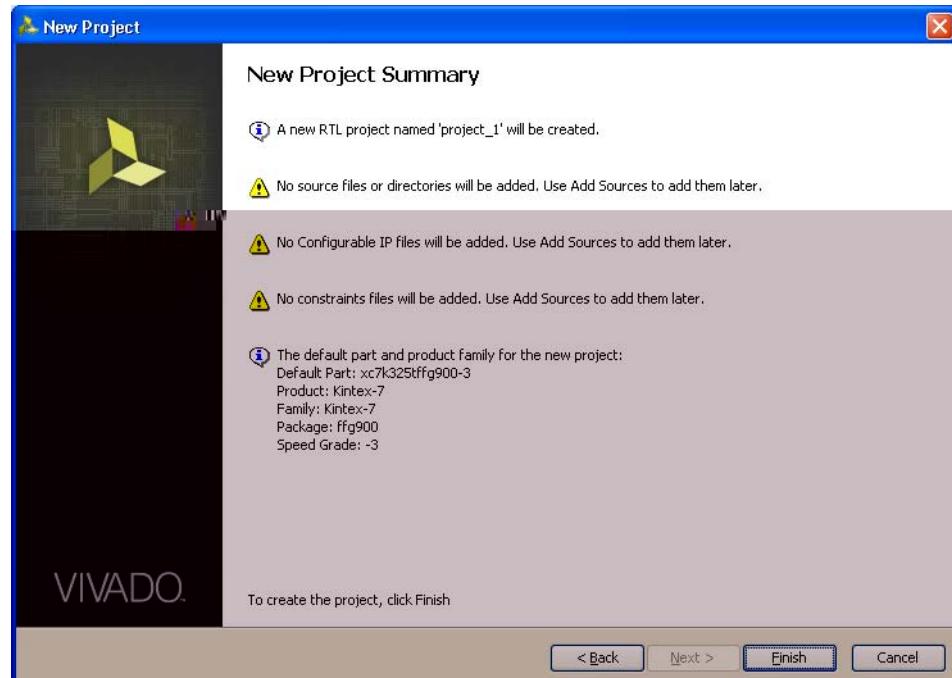
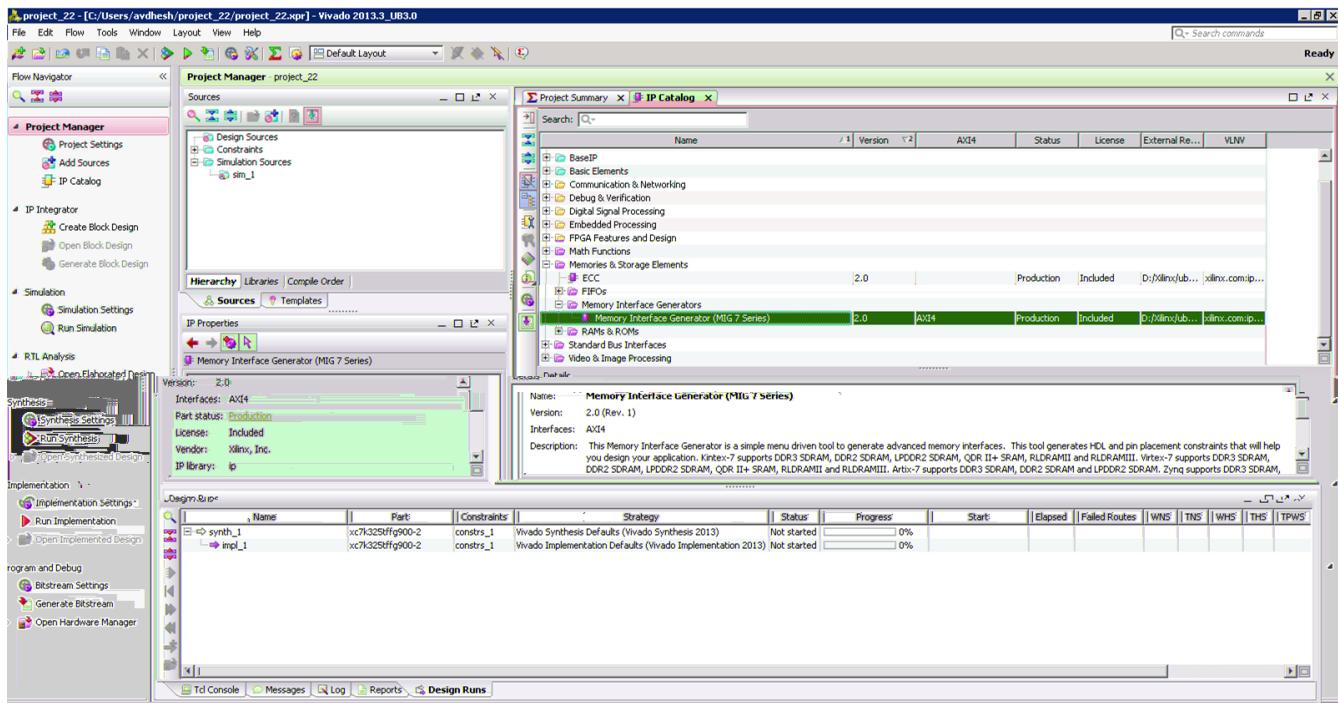


Figure 1-11:

- Click **Finish** to complete the project creation.

11. Click **IP Catalog** on the **Project Manager** window to open the IP catalog window. The Vivado IP catalog window appears on the right side panel (see [Figure 1-12](#), highlighted in a red circle).
12. The MIG tool exists in the **Memories & Storage Elements > Memory Interface Generators** section of the IP catalog window ([Figure 1-12](#)) or you can search from the Search tool bar for the string "MIG."



*Figure 1-12:*

13. Select **MIG 7 Series** to open the MIG tool (Figure 1-13).

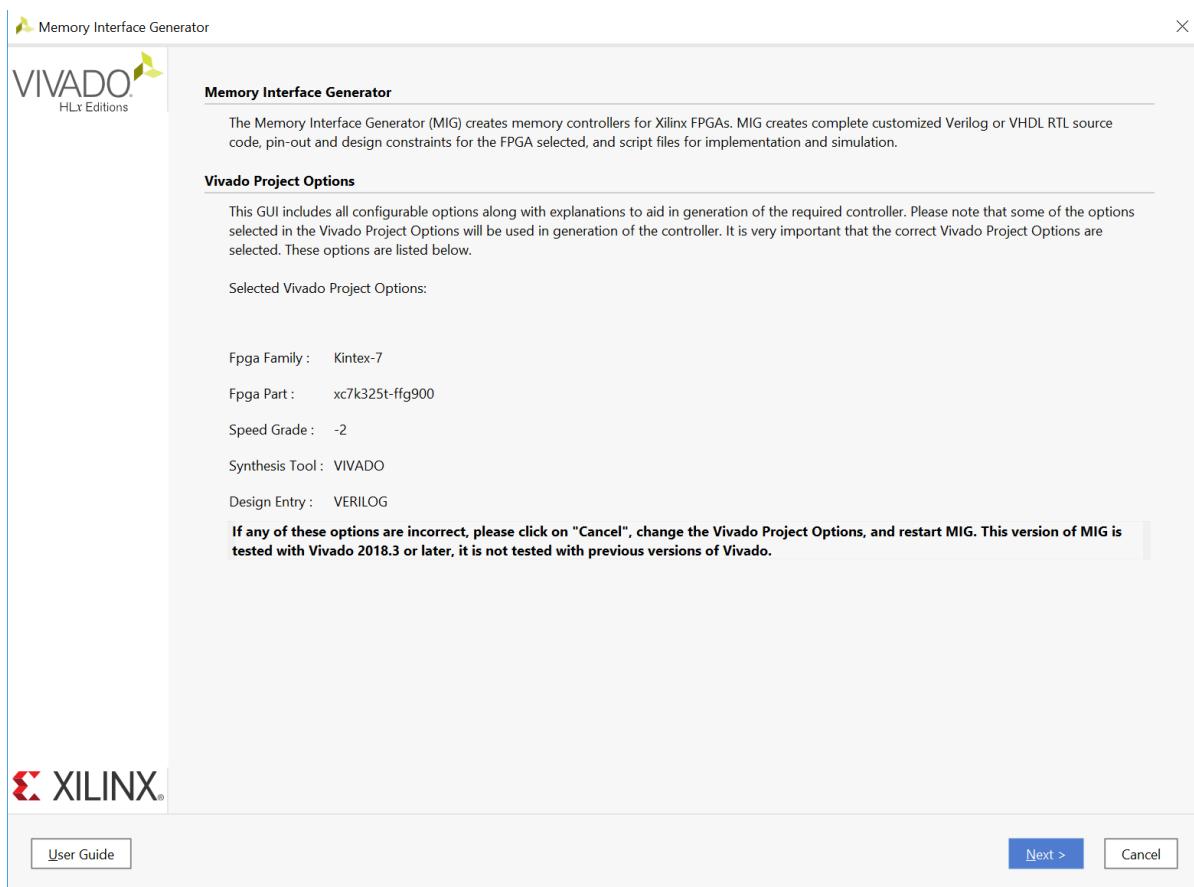


Figure 1-13:

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

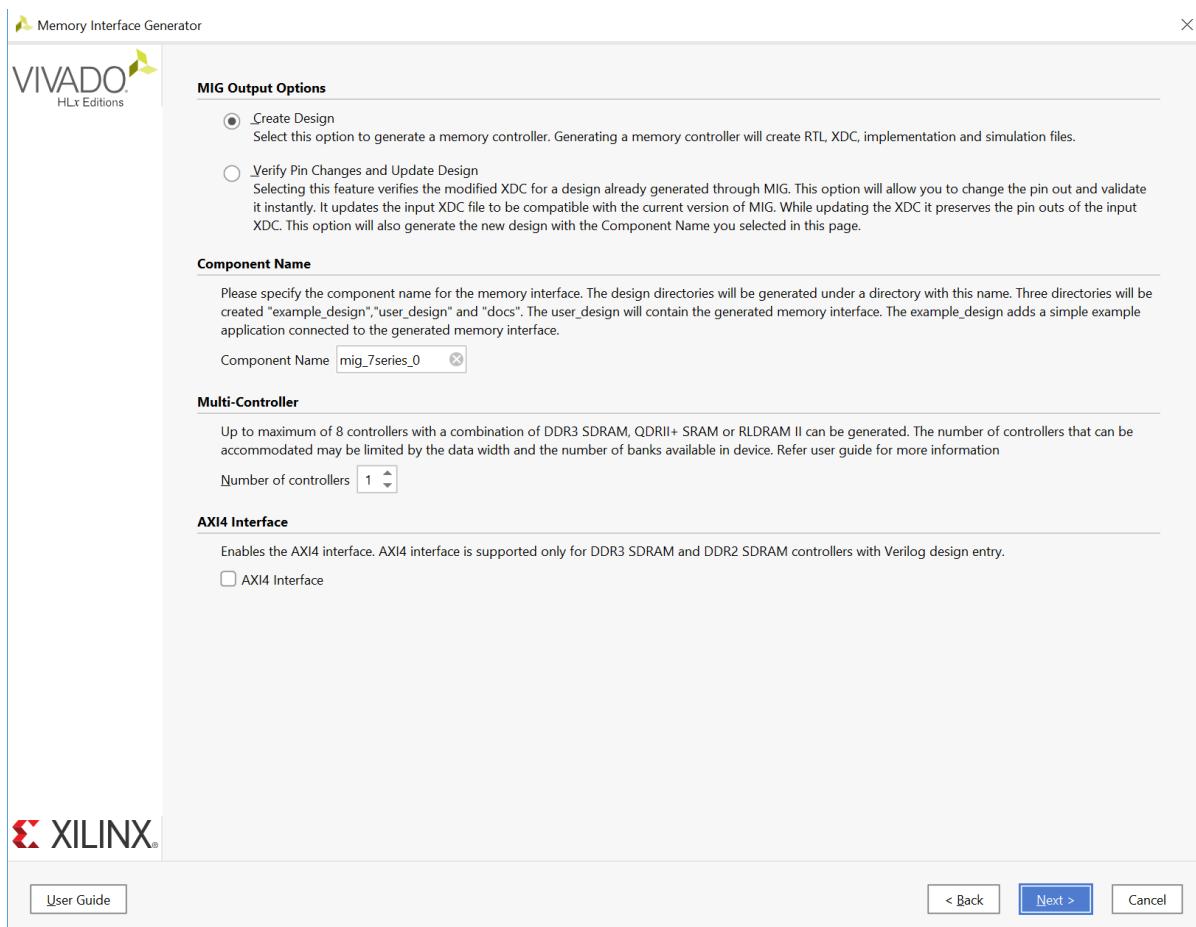


**CAUTION!** *The Windows operating system has a 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs.*

## MIG Output Options

1. Select the **Create Design** to create a new Memory Controller design. Enter a component name in the **Component Name** field (Figure 1-14).
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 Vivado Design Suite before invoking the MIG tool. If the AXI4 interface is not selected, the user interface (UI) is the primary interface.



*Figure 1-14:*

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



**IMPORTANT:** 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 Xilinx Platform Studio (XPS), the component name is corrected to be the IP instance name from XPS.

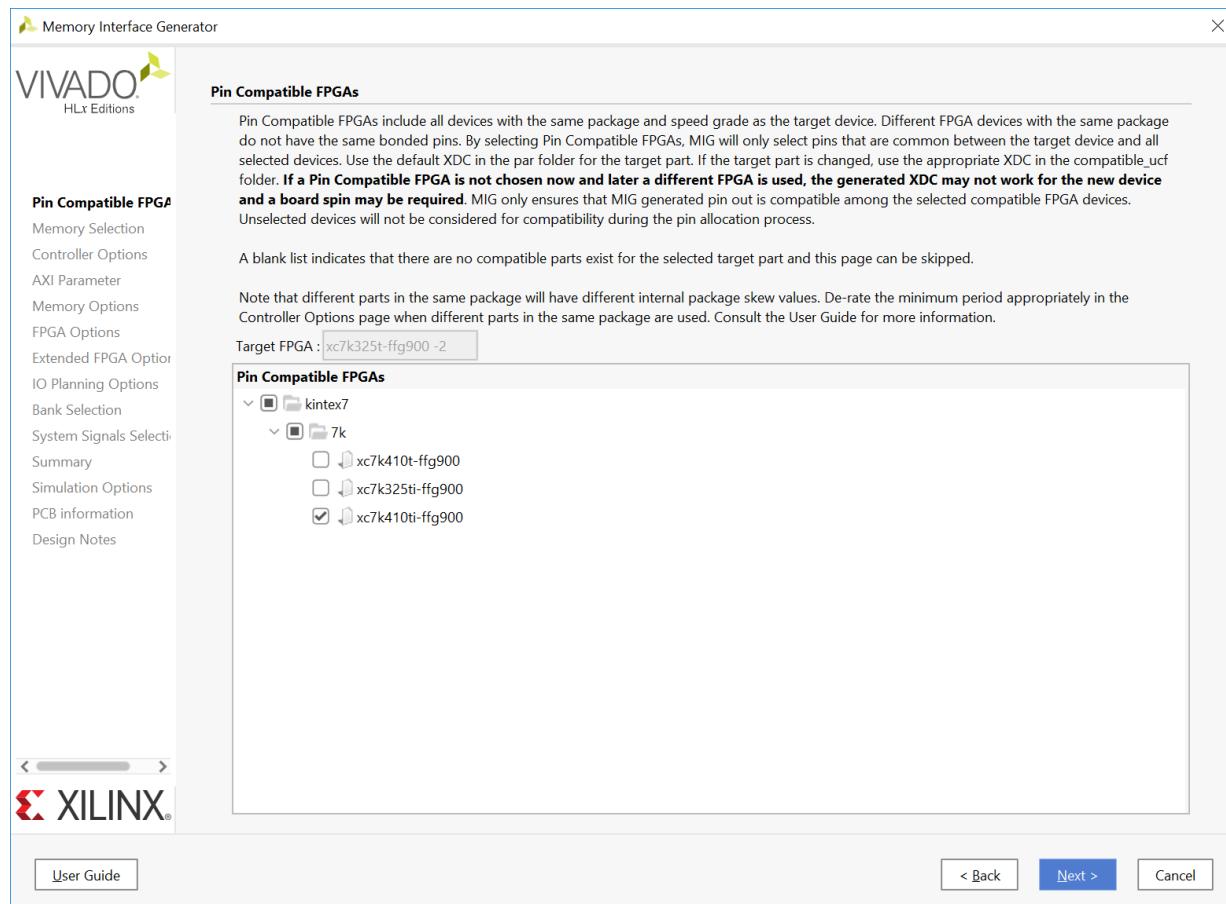
4. Click **Next** to display the **Pin Compatible FPGAs** page.

### ***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 1-15](#)).

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 1-15:*

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*

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 1-16).

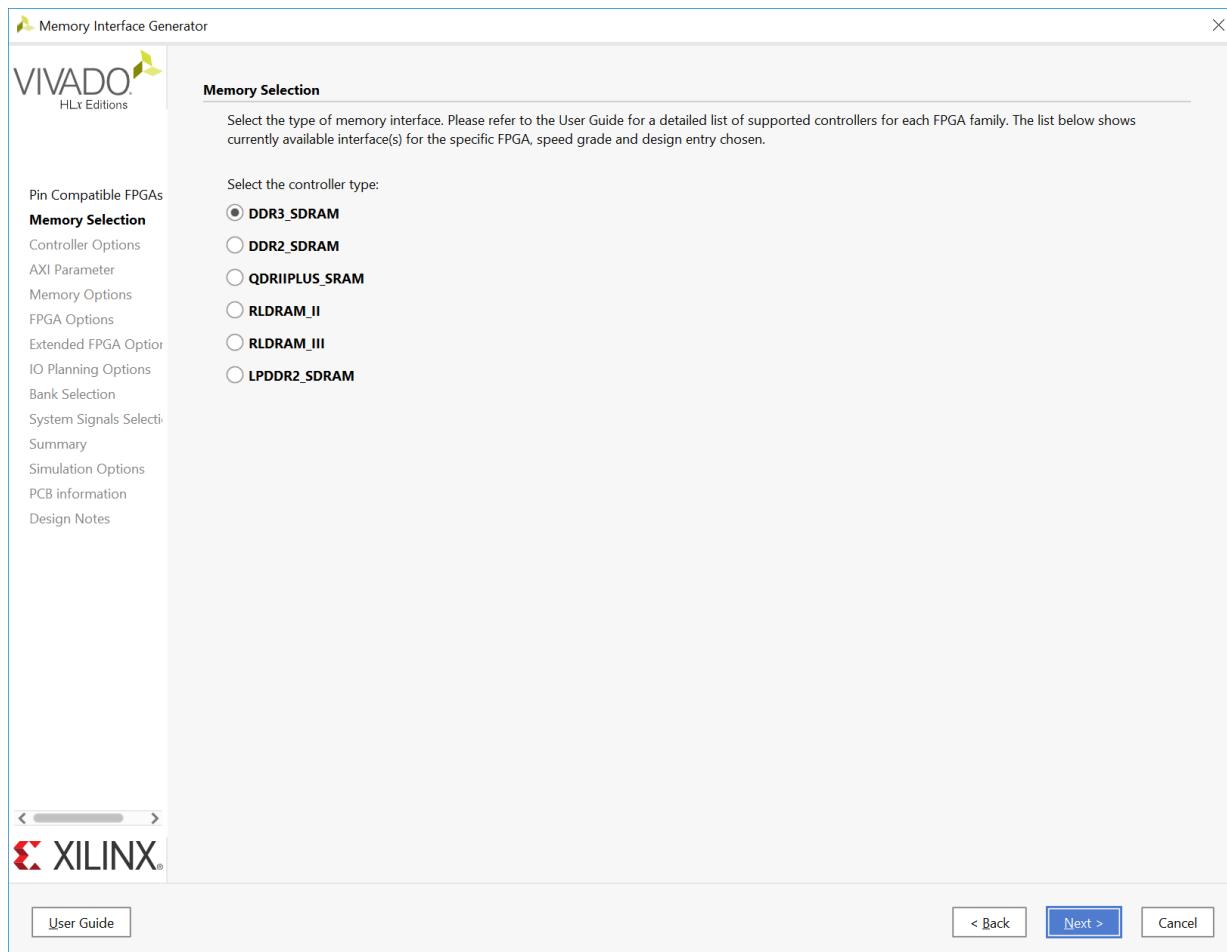


Figure 1-16:

This page shows the various controller options that can be selected (Figure 1-17).

**TIP:** *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 175](#) 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.*

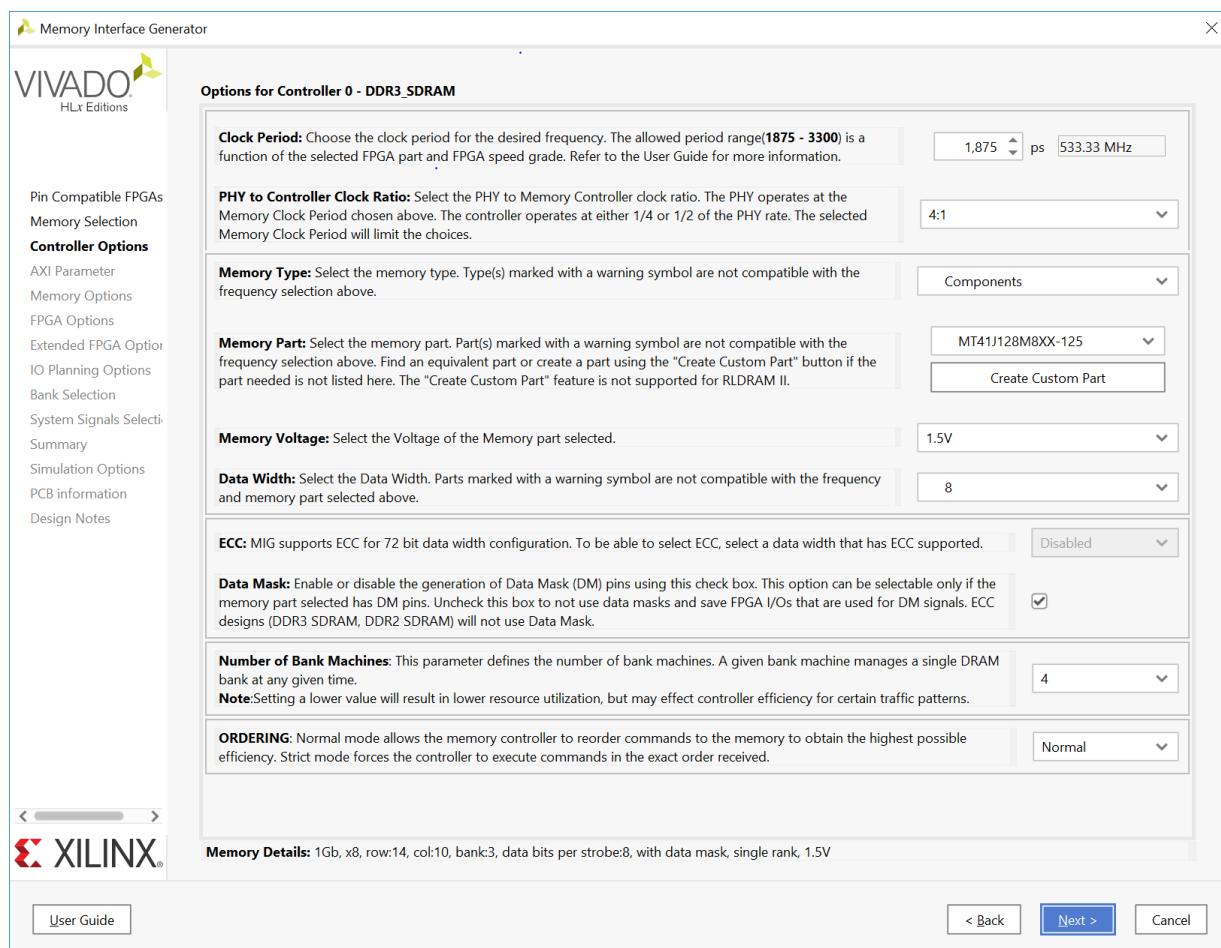


Figure 1-17:

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:

- **Clock Period** – This feature indicates the operating frequency for all of the controllers. The frequency block is limited by factors such as the selected FPGA and device speed grade.

- **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 FPGA logic timing limitations. The user interface data bus width of the 2:1 ratio is four times the width of the physical memory interface width, while the bus width of the 4:1 ratio is eight 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** – 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. For more information, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2] and the *7 Series FPGAs Packaging and Pinout Specification* (UG475) [Ref 3].
- **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.

**Note:** For a complete list of memory parts available, see Answer Record: [54025](#).

- **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 eight bits is also a valid value.
- **Data Mask** – This option allocates data mask pins when selected. This should be deselected to deallocate data mask pins and increase pin efficiency. Also, this is disabled for memory parts that do not support data mask.



**IMPORTANT:** *The Data Mask (DM) option is always selected for AXI designs and is grayed out (you cannot select it). For AXI interfaces, Read Modify Write (RMW) is supported and for RMW to mask certain bytes of Data Mask bits should be present. Therefore, the DM is always enabled for AXI interface designs. This is the case for all data widths except 72-bit.*

*For 72-bit interfaces, Error Correcting Code (ECC) is enabled and DM is deselected and grayed out. If DM is enabled for 72-bit designs, computing ECC is not compatible, therefore DM is disabled for 72-bit designs.*

- **Number of Bank Machines** – The list shows the number of bank machines that are supported for the selected design configuration.
- **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 1-17](#)) displays the details for the selected memory configuration ([Figure 1-18](#)).

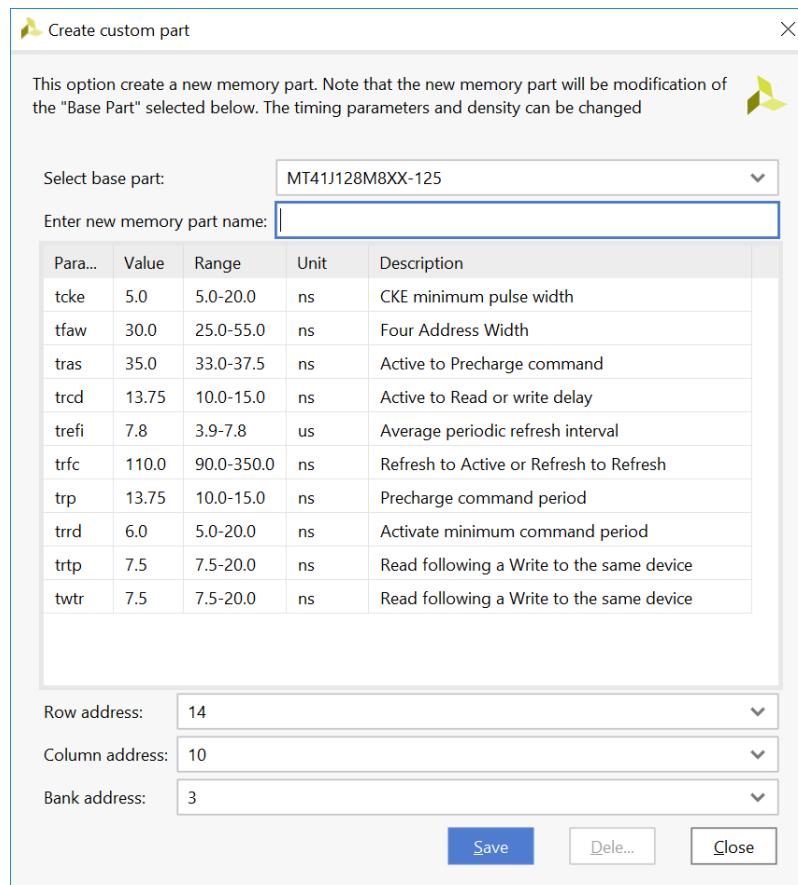


Figure 1-19:

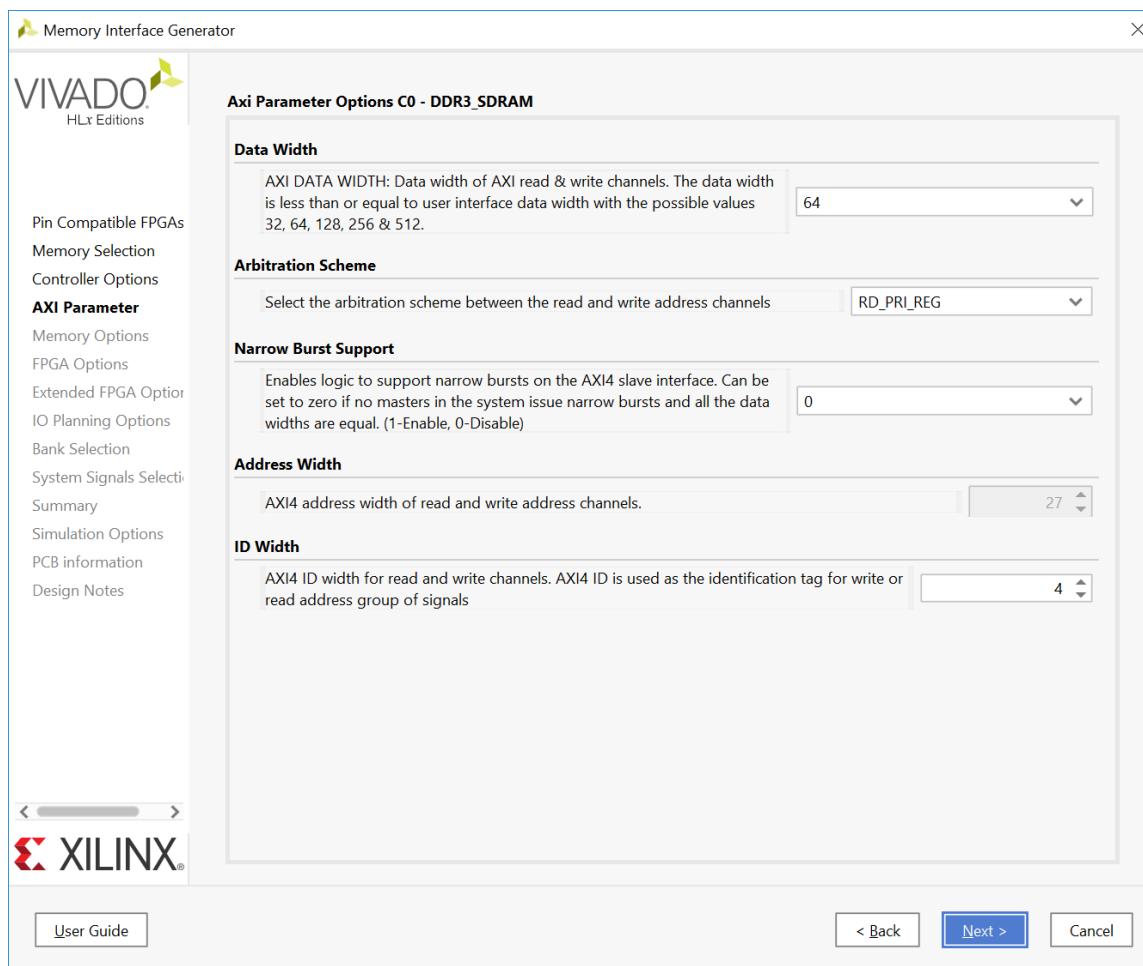
The **Create Custom Part** page includes all the specifications of the memory component selected in the **Select Base Part** pull-down menu.

3. Select the suitable base part from the **Select Base Part** list.
4. Enter the appropriate memory part name in the text box.
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 **Save**. 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).

This feature allows the selection of AXI parameters for the controller ([Figure 1-20](#)). These are standard AXI parameters or parameters specific to the AXI4 interface. Details are available in the Arm® AMBA® specifications [\[Ref 4\]](#).

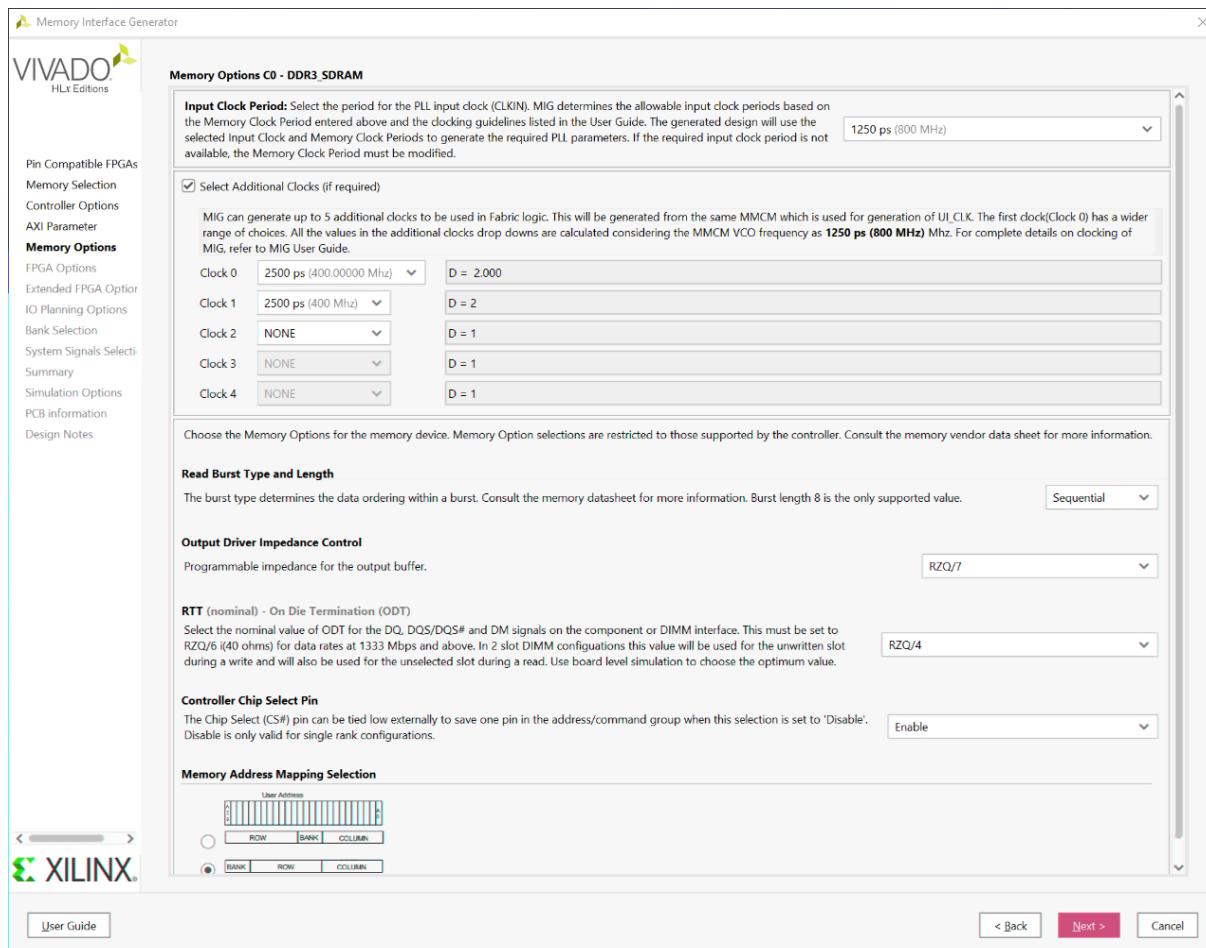
These parameters specific to the AXI4 interface logic can be configured:

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



*Figure 1-20:*

This feature allows the selection of various memory mode register values, as supported by the controller specification ([Figure 1-21](#)).



*Figure 1-21:*

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.

The **Output Driver Impedance Control** sets the output driver impedance on the DRAM. The selections listed are determined by specific DRAM chosen. RZQ is  $24\Omega$ . For example, if RZQ/6 is chosen, the output drive impedance is  $4\Omega$ . For more information, consult the memory vendor data sheet.

The DDR2 SDRAM interface has a separate option to select the number of memory clocks called **Memory Clock Selection**. Each component has a **Number of Memory Clocks** setting, and the maximum number of clocks allowed is four.

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 192](#) for more information on the PLL parameter limits.

**Select Additional Clocks** option appears for AXI interface designs only. Selection is allowed for up to five additional clocks which are generated from the same MMCM that generates **UI\_CLK**.



**IMPORTANT:** *The **Select Additional Clocks** option appears in Vivado IP integrator flow only.*

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

Figure 1-22 shows the **FPGA Options** page.

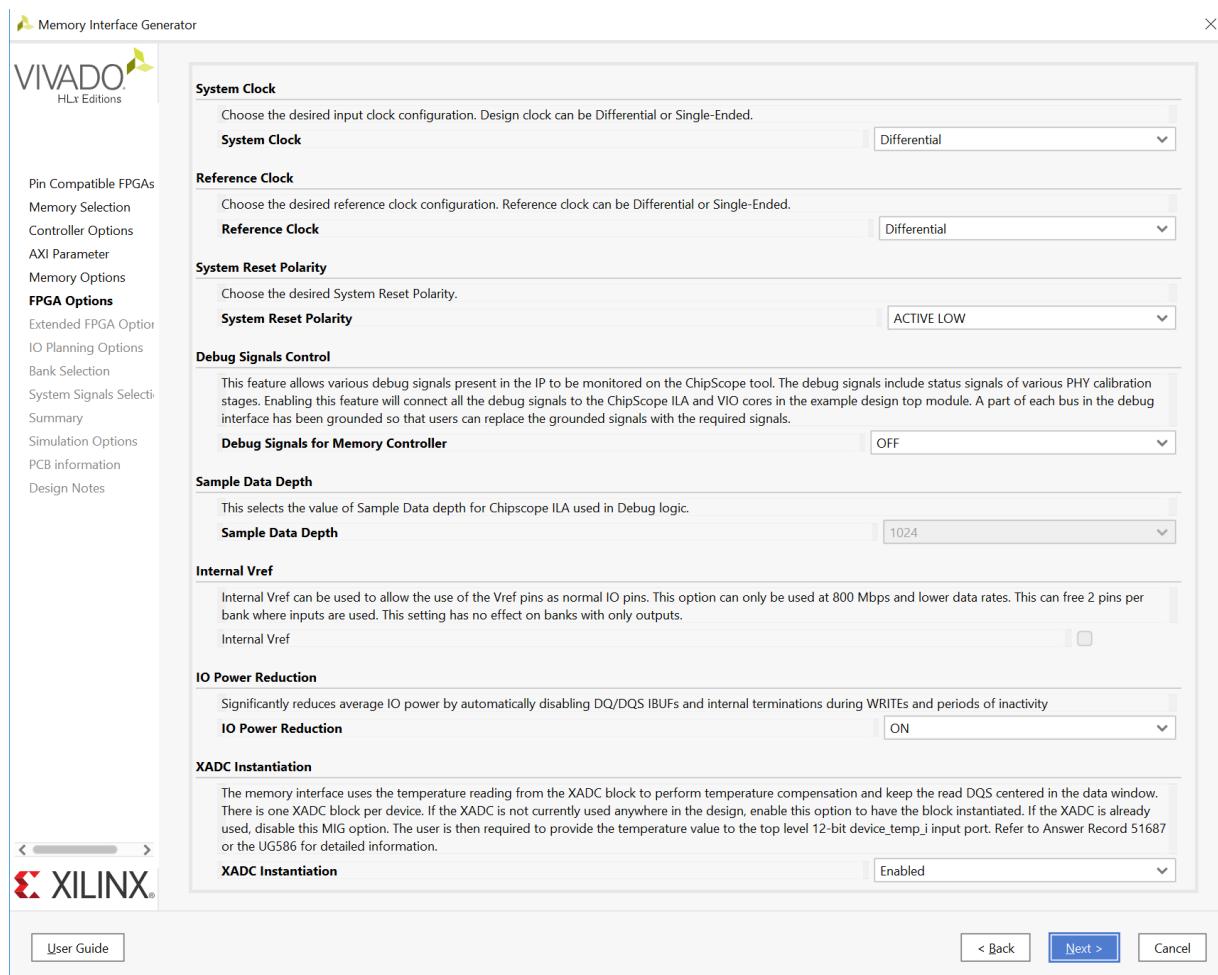


Figure 1-22:

- **System Clock** – This option selects the clock type (Single-Ended, Differential, or No Buffer) for the `sys_clk` signal pair. When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the system clock.

If the designs generated from MIG for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the `sys_clk_i` signal. So for **No Buffer** scenarios, `sys_clk_i` signal needs to be connected to an internal clock.

The **No Buffer** option must only be selected for designs that already have a system input clock assigned that meets all rules specified in the [Clocking, page 210](#).

- **Reference Clock** – This option selects the clock type (Single-Ended, Differential, No Buffer, or Use System Clock) for the `clk_ref` signal pair. The **Use System Clock** option appears when the input frequency is between 199 and 201 MHz (that is, the Input Clock Period is between 5,025 ps (199 MHz) and 4,975 ps (201 MHz). The reference clock frequency is based on the data rate and note that an MMCM is added to create the appropriate `ref_clk` frequency above 1,333 Mb/s. When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the reference clock.

If the designs generated from MIG for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the `ref_clk_i` signal. So for **No Buffer** scenarios, `ref_clk_i` signal needs to be connected to an internal clock.

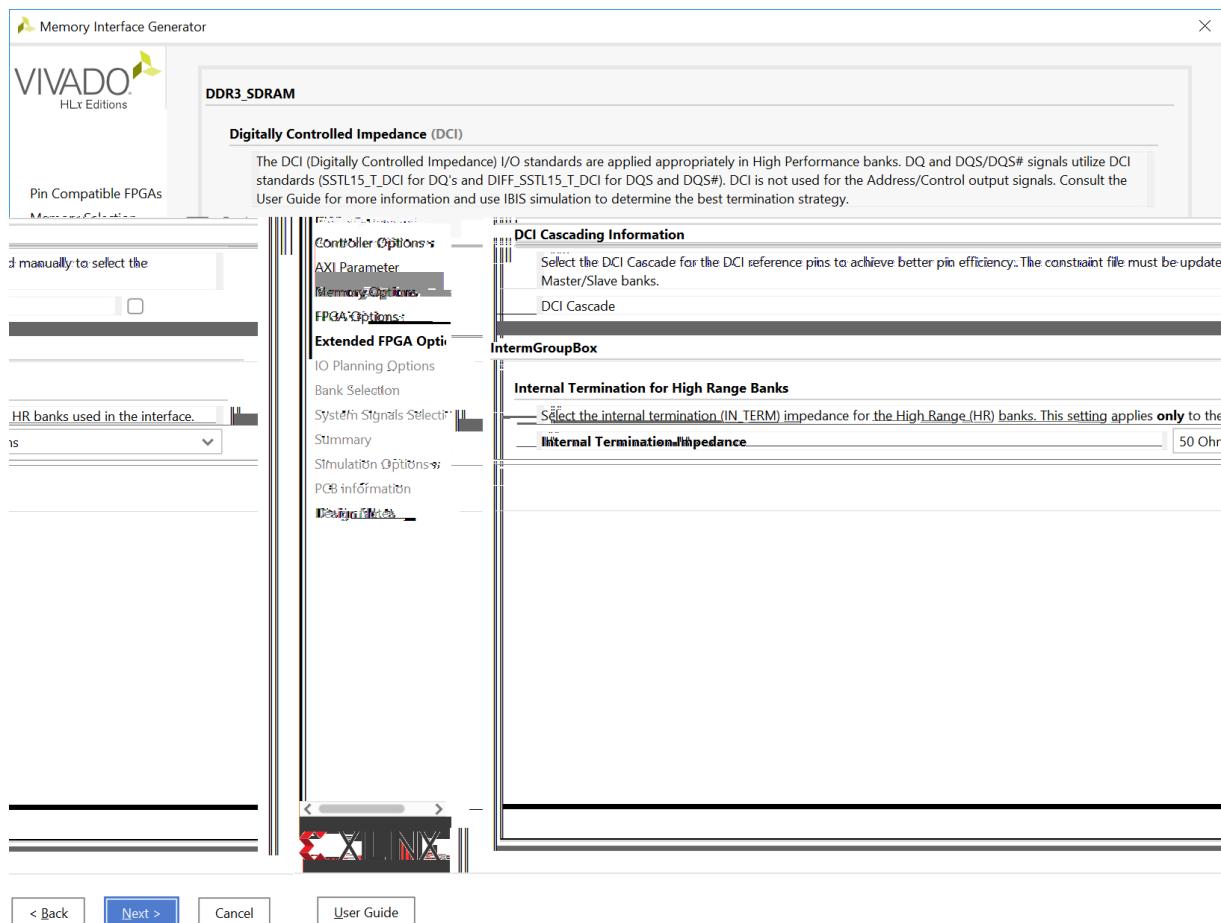
- **System Reset Polarity** – The polarity for system reset (`sys_rst`) can be selected. If the option is selected as active-Low, the parameter `RST_ACT_LOW` is set to 1 and if set to active-High the parameter `RST_ACT_LOW` is set to 0.
- **Debug Signals Control** – Selecting this option enables calibration status and user port signals to be port mapped to the ILA and VIO in the `example_top` module. This helps in monitoring traffic on the user interface port with the Vivado Design Suite debug feature. Deselecting the **Debug Signals Control** option leaves the debug signals unconnected in the `example_top` module and no ILA/VIO modules are generated by the IP catalog. Additionally, the debug port is always disabled for functional simulations.

**Note:** This option is not available in the Vivado IP integrator flow.

- **Sample Data Depth** – This option selects the Sample Data depth for the ILA module used in the Vivado debug logic. This option can be selected when the **Debug Signals for Memory Controller** option is ON.
- **Internal V<sub>REF</sub> Selection** – Internal V<sub>REF</sub> can be used for data group bytes to allow the use of the V<sub>REF</sub> pins for normal I/O usage. Internal V<sub>REF</sub> should only be used for data rates of 800 Mb/s or below.

- **I/O Power Reduction** – This option reduces the average I/O power by disabling DQ and DQS IBUFs automatically whenever the controller is in the idle state.
- **XADC Instantiation** – When enabled, this option directs MIG core to instantiate the XADC and a temperature polling circuit for the Temperature Monitor feature (see [Temperature Monitor](#)). This option can be disabled if the XADC is already used elsewhere in the design. In this case, the device temperature must be periodically sampled and driven onto the `device_temp_i` bus in the memory interface top-level user design module. If the `device_temp_i` signal is left unconnected, then the XADC is instantiated. Otherwise the XADC is not instantiated.

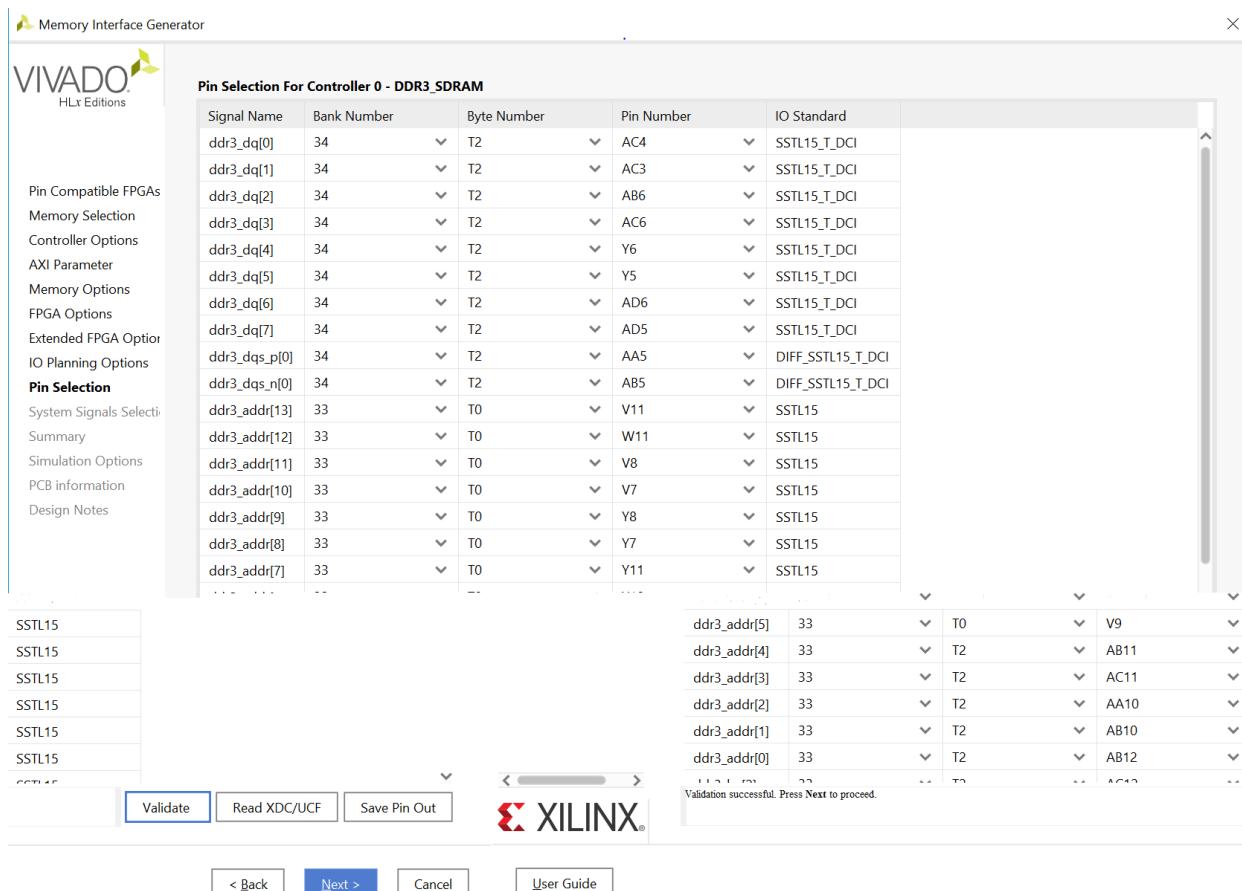
Click **Next** to display the **DCI Description** page ([Figure 1-23](#)).



*Figure 1-23:*

- **Digitally Controlled Impedance (DCI)** – The DCI option allows the use of the FPGA 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 you to specify an existing pinout and generate the RTL for this pinout, or pick banks for a new design. [Figure 1-24](#) shows the options for using an existing pinout. You 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. You cannot proceed until the MIG DRC has been validated by clicking **Validate**.



*Figure 1-24:*

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

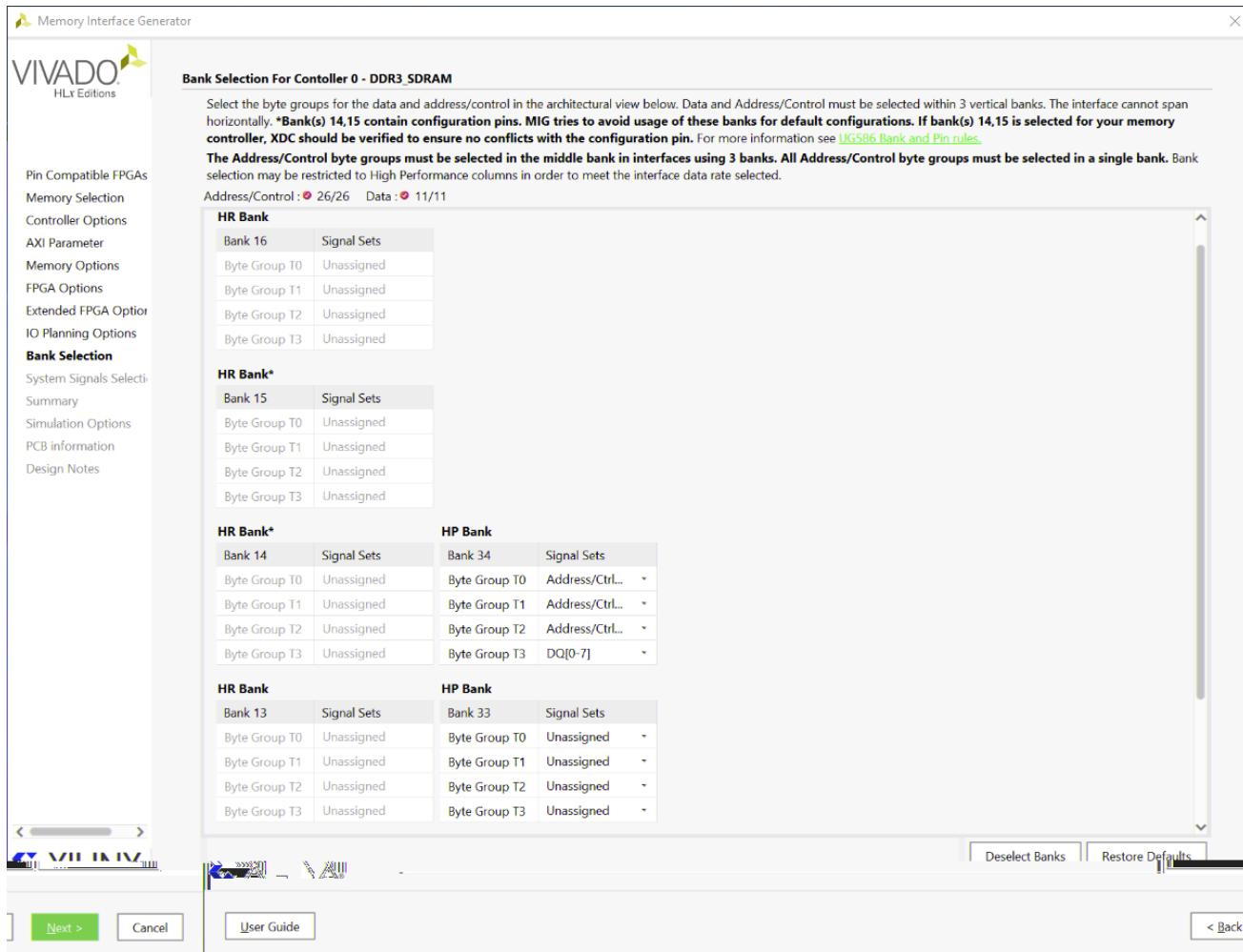


Figure 1-25:

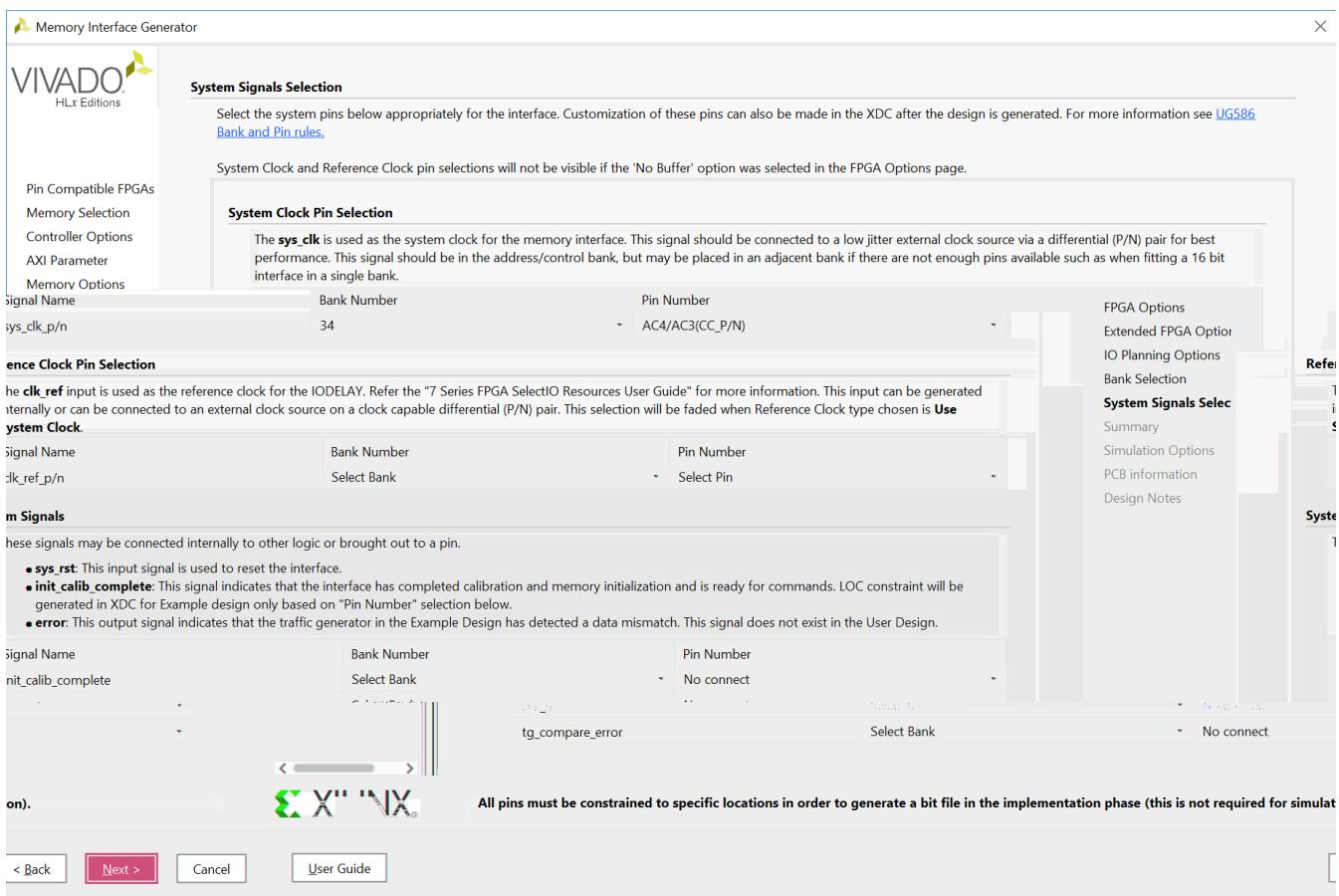
For customized settings, click **Deselect Banks** and select the appropriate bank and memory signals. Click **Next** to move to the next page if the default setting is used.

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

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. The MIG core 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 1-26](#)). The MIG tool allows the selection of either external pins or internal connections, as desired.



*Figure 1-26:*

- **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 1-22](#)). 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 XDC can be modified as desired after generation.
- **clk\_ref** – This is the reference frequency input for the IDELAY control. 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 1-22](#)). The I/O standard is selected in a similar way as **sys\_clk**.

- **sys\_rst** – This is the asynchronous 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 LVCMOS18 and LVCMOS25 for HP and HR banks, respectively. The default polarity of **sys\_rst** pin is active-Low. The polarity of **sys\_rst** pin varies based on the **System Reset Polarity** option chosen in **FPGA Options** page ([Figure 1-22](#)).
- **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.

This page provides the complete details about the 7 series FPGA memory core selection, interface parameters, IP catalog options, and FPGA options of the active project ([Figure 1-27](#)).

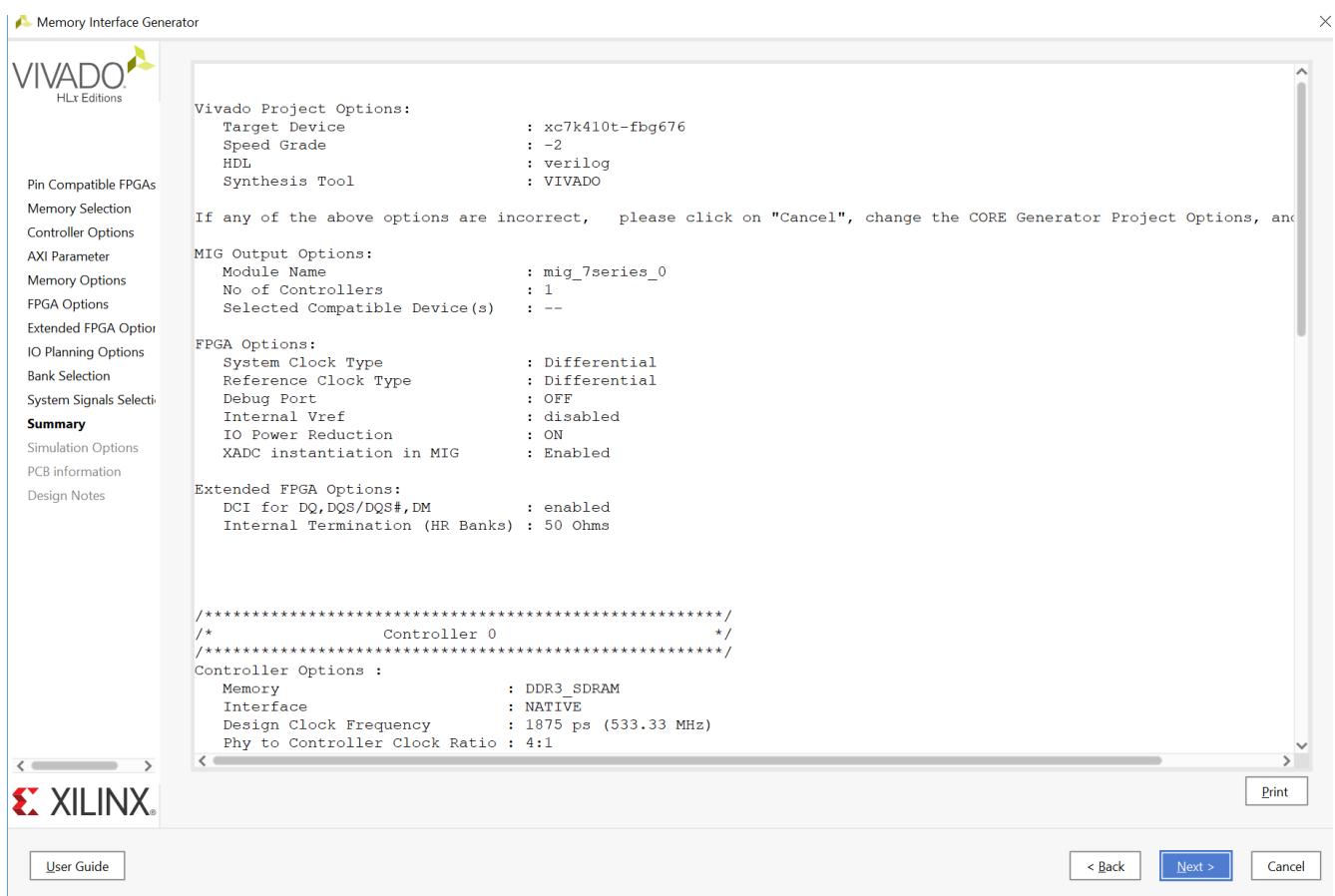


Figure 1-27:

The MIG tool can output a chosen vendor's memory model for simulation purposes for memories such as DDR2 or DDR3 SDRAMs. To access the models in the output **sim** folder, click the license agreement (Figure 1-28). 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.

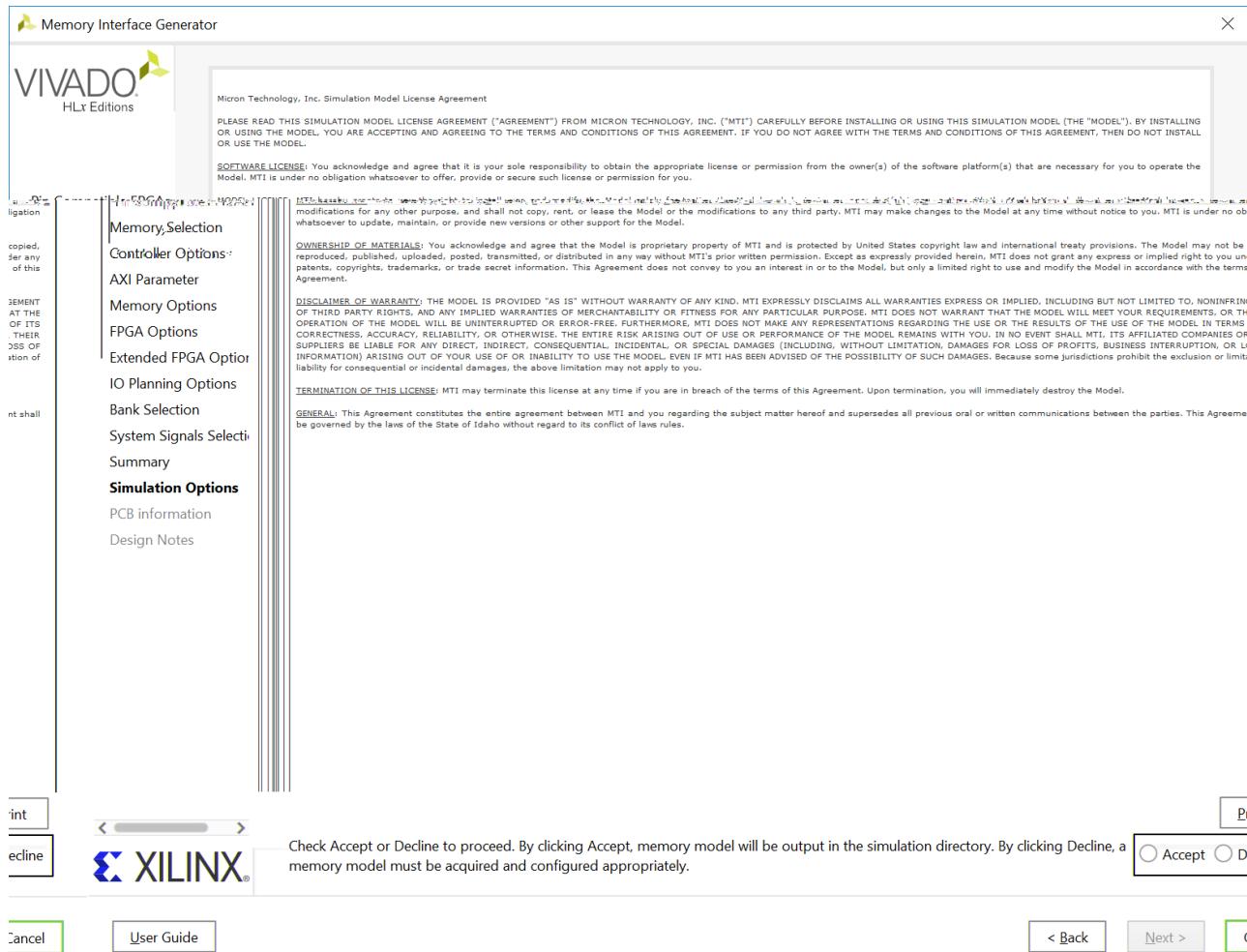


Figure 1-28:

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

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.

Click **Generate** 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.

1. After clicking **Generate**, the **Generate Output Products** window appears. This window has the **Out-of-Context Settings** as shown in [Figure 1-29](#).

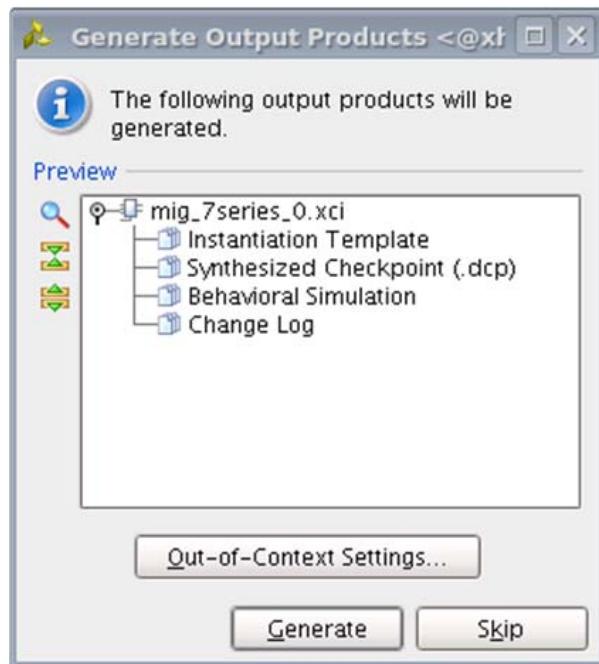
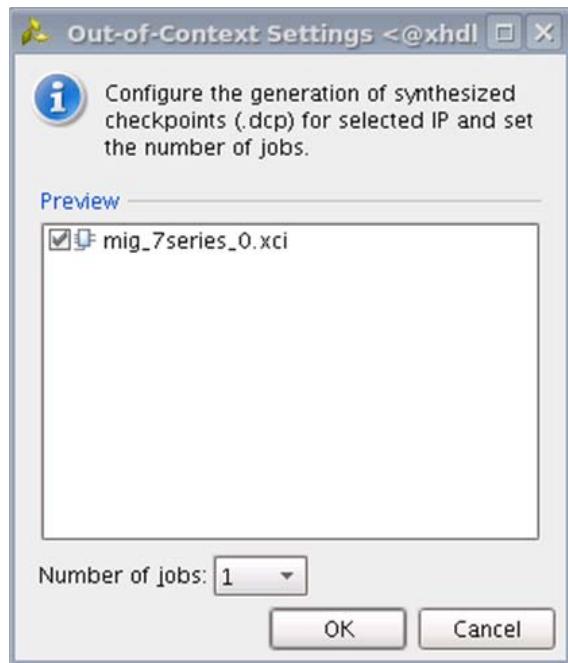


Figure 1-29:

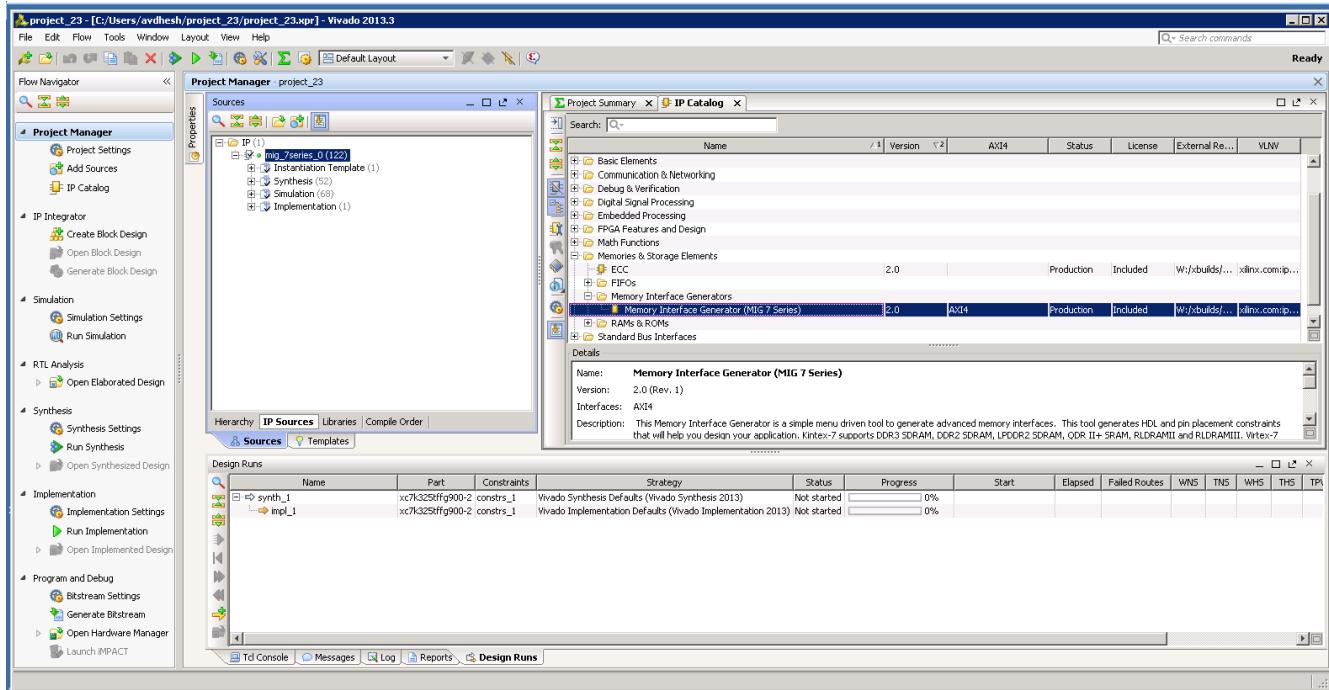
2. Click **Out-of-Context Settings** to configure generation of synthesized checkpoints. To enable the **Out-of-Context** flow, enable the check box. To disable the **Out-of-Context** flow, disable the check box. The default option is "enable" as shown in [Figure 1-30](#).



*Figure 1-30:*

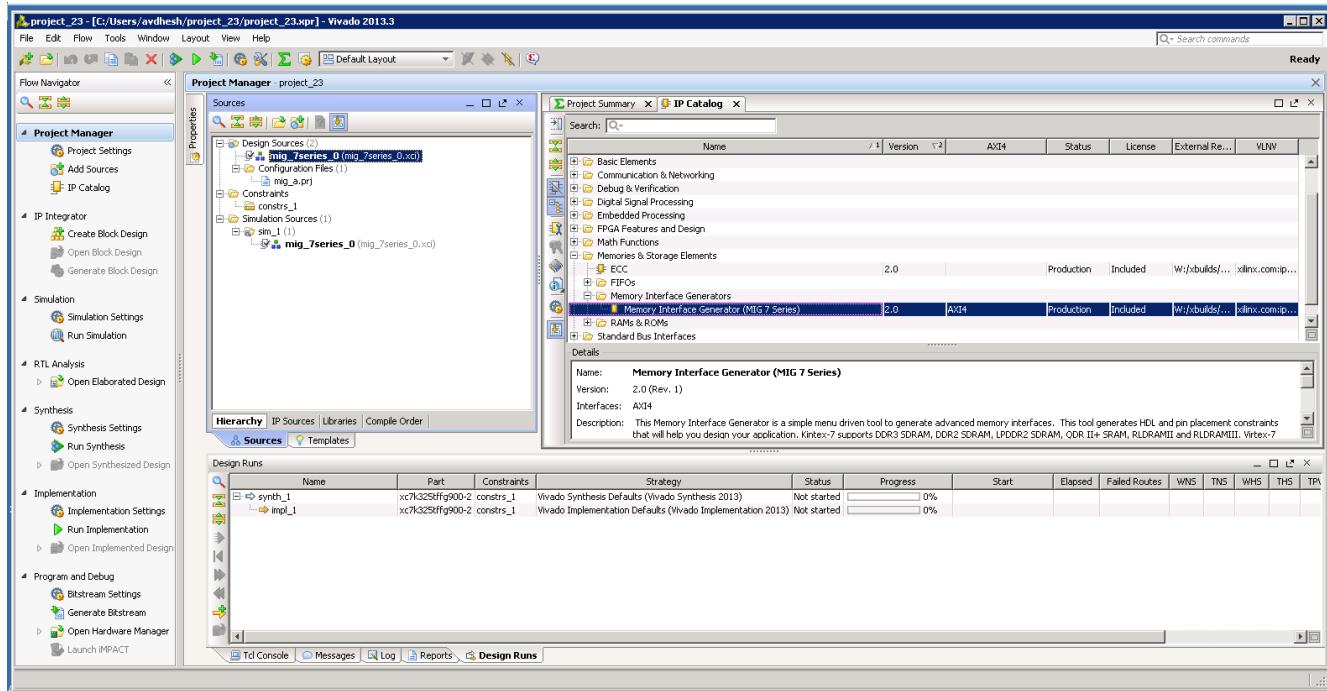
3. MIG core designs comply with "Hierarchical Design" flow in Vivado. For more information, see the *Vivado Design Suite User Guide: Hierarchical Design* (UG905) [\[Ref 5\]](#) and the *Vivado Design Suite Tutorial: Hierarchical Design* (UG946) [\[Ref 6\]](#).

4. After generating the MIG core design, the project window appears as shown in Figure 1-31.



*Figure 1-31:*

5. After project creation, the XCI file is added to the Project Hierarchy. The same view also displays the module hierarchies of the user design. The list of HDL and XDC files is available in the **IP Sources** view in the **Sources** window. Double-clicking on any module or file opens the file in the Vivado Editor. These files are read only.



*Figure 1-32:*

Design generation from the MIG tool can be generated using the **Create Design** flow or the **Verify Pin Changes** and **Update Design** flows. There is no difference between the flow when generating the design from the MIG tool. Irrespective of the flow by which designs are generated from the MIG tool, the XCI file is added to the Vivado tool project. The implementation flow is the same for all scenarios because the flow depends on the XCI file added to the project.

6. All MIG generated user design RTL and XDC files are automatically added to the project. If files are modified and you wish to regenerate them, right-click the XCI file and select **Generate Output Products** (Figure 1-33).

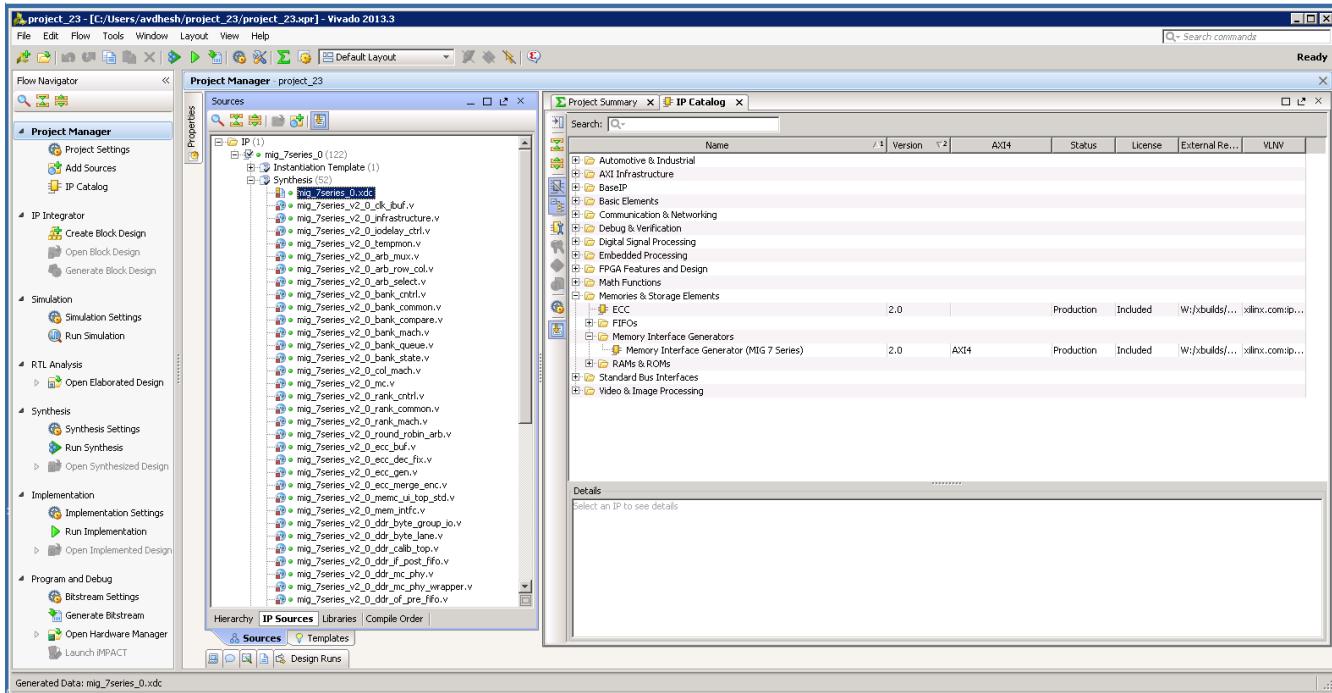


Figure 1-33:

7. Clicking the Generate Output Products option brings up the Manage Outputs window (Figure 1-34).

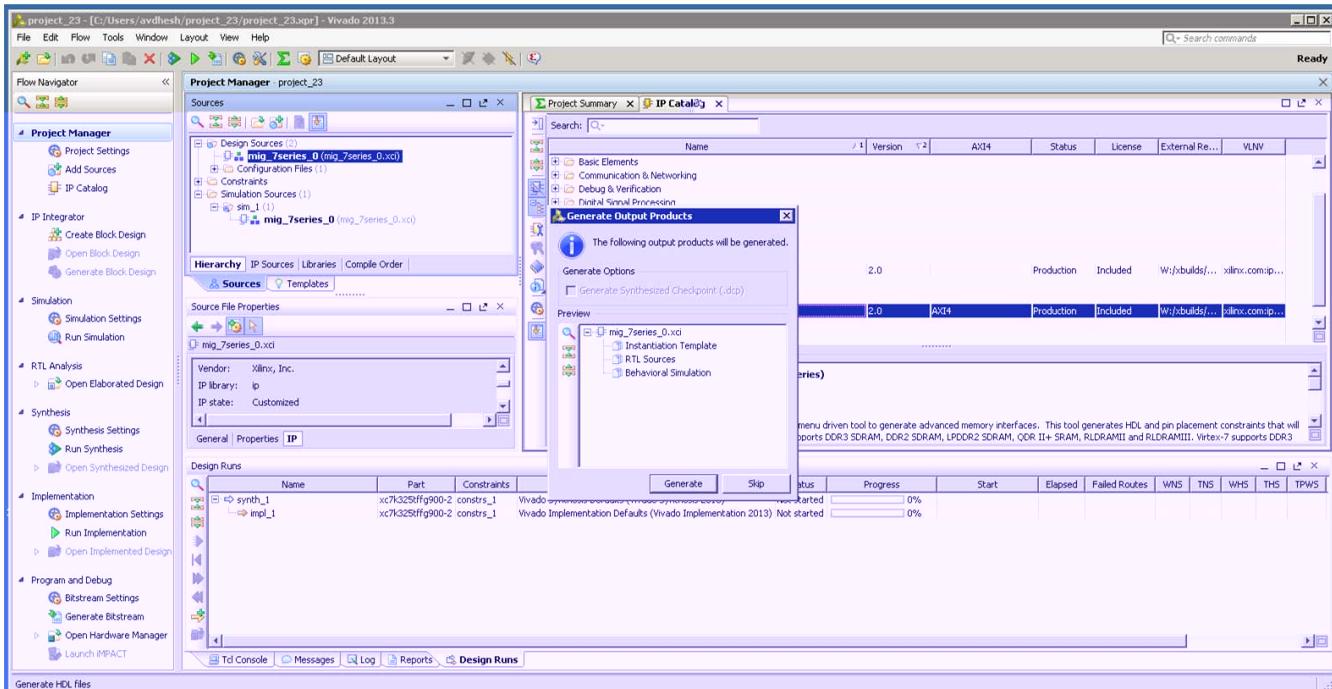
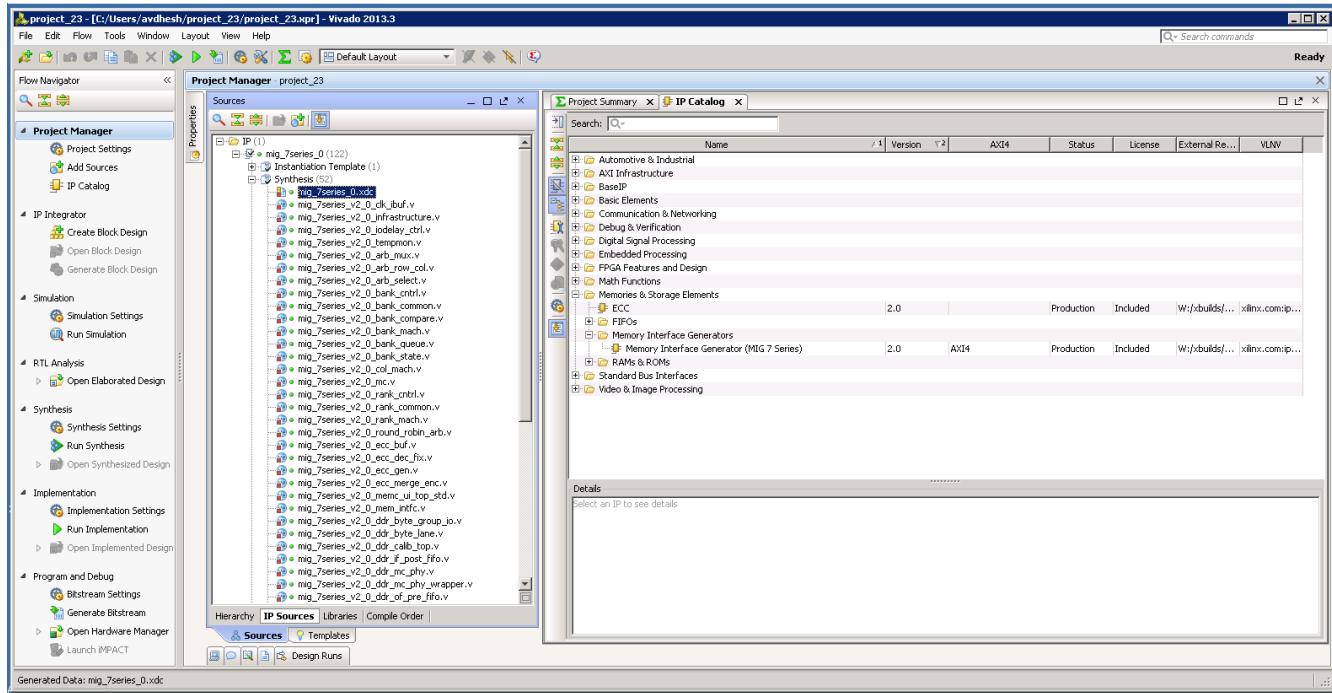


Figure 1-34:

8. All user-design RTL files and constraints files (XDC files) can be viewed in the **Sources > Libraries** tab ([Figure 1-35](#)).



*Figure 1-35:*

9. The Vivado Design Suite supports **Open IP Example Design** flow. To create the example design using this flow right-click the IP in the **Source Window**, as shown in [Figure 1-36](#) and select.

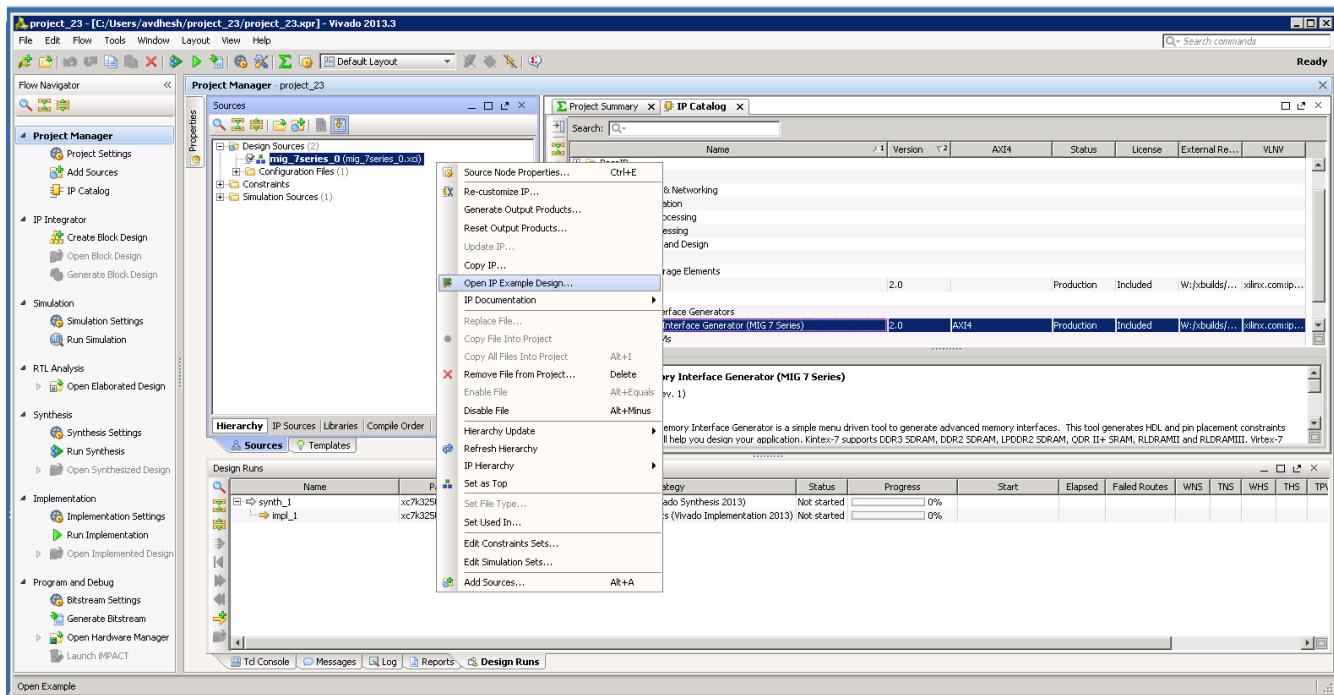


Figure 1-36:

10. This option creates a new Vivado project. Selecting the menu brings up a dialog box, which guides you to the directory for a new design project. Select a directory (or use the defaults) and click **OK**.

This launches a new Vivado project with all example design files and a copy of the IP. This project has **example\_top** as the Implementation top directory, and **sim\_tb\_top** as the Simulation top directory, as shown in Figure 1-37.

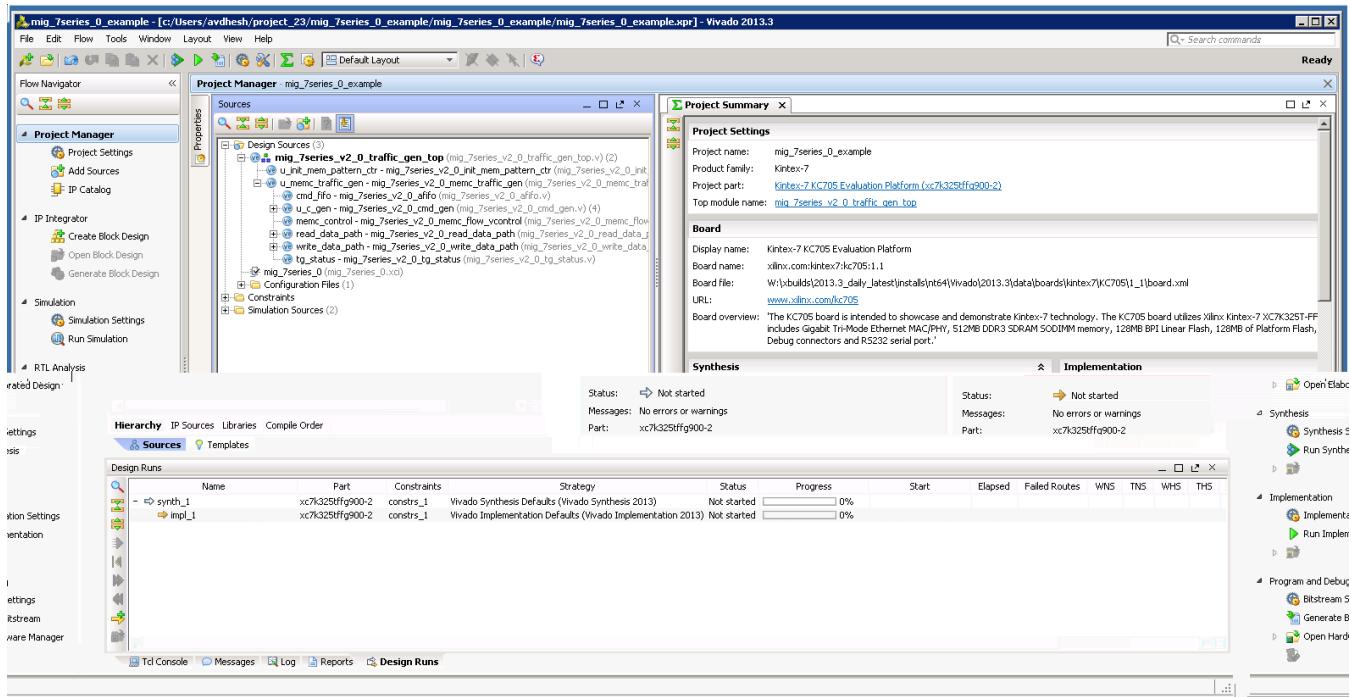
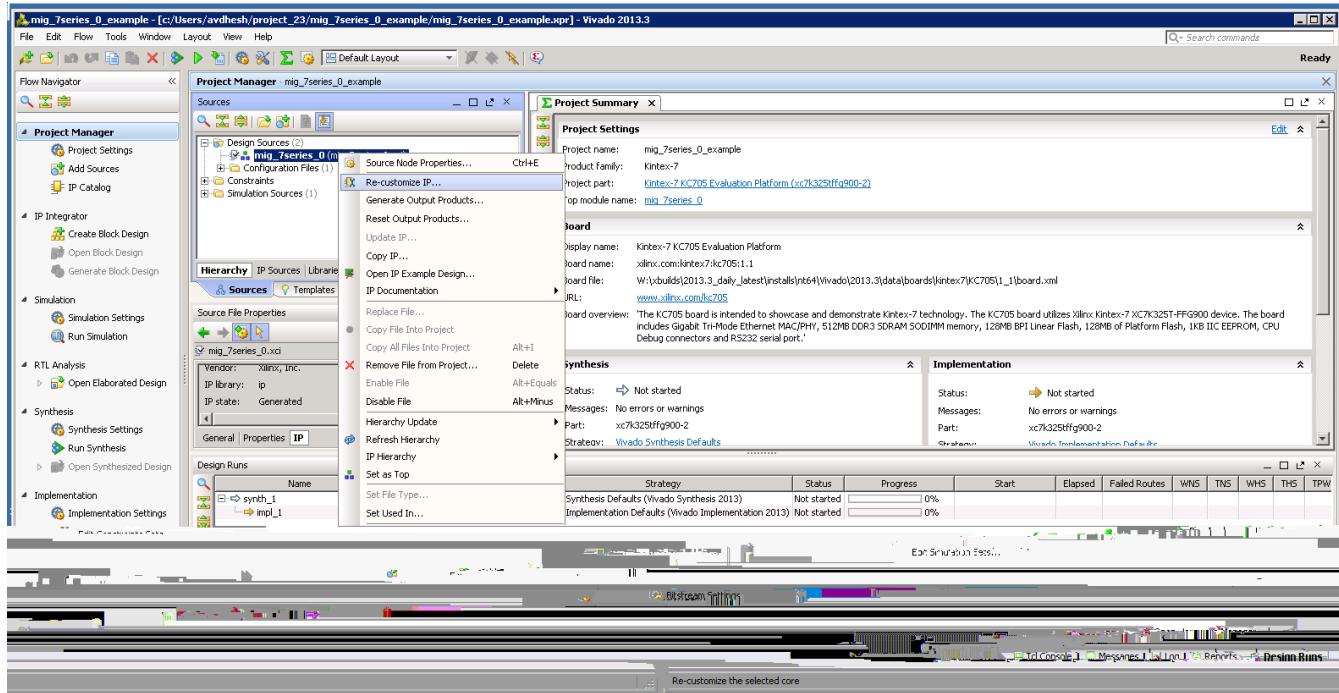


Figure 1-37:

11. Click Generate Bitstream under Project Manager > Program and Debug to generate the BIT file for the generated design.

The `<project directory>/<project directory>.runs/ impl_1` directory includes all report files generated for the project after running the implementation. It is also possible to run the simulation in this project.

12. Recustomization of the MIG IP core can be done by using the **Recustomize IP** option. It is not recommended to recustomize the IP in the `example_design` project. The correct solution is to close the `example_design` project, go back to original project and customize there. Right-click the XCI file and click **Recustomize IP** (Figure 1-38) to open the MIG GUI and regenerate the design with the preferred options.

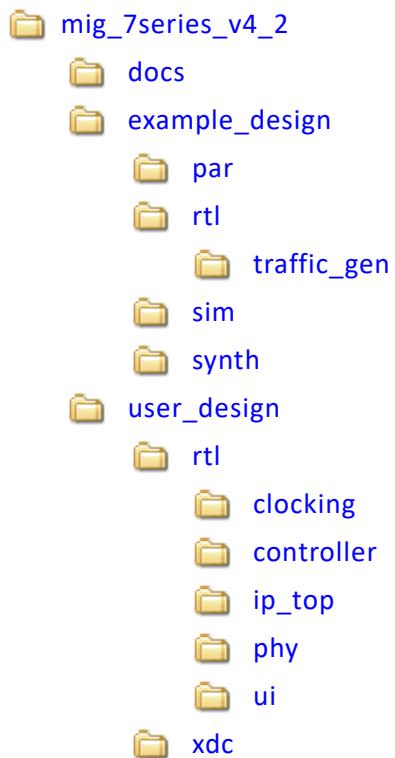


*Figure 1-38:*

### *Directory Structure and File Descriptions*

The output directory structure of the selected Memory Controller (MC) design from the MIG tool is shown here. In the `<component_name>` directory, three folders are created:

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



The 7 series FPGAs core directories and their associated files are listed in this section for Vivado implementations.

**<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:*

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:*

memc_traffic_gen.v	This is the top-level of the traffic generator.
cmd_gen.v	This is the command generator. This module provides independent control of generating the types of commands, addresses, and burst lengths.
cmd_prbs_gen.v	This is a pseudo-random binary sequence (PRBS) generator for generating PRBS commands, addresses, and burst lengths.
memc_flow_vcontrol.v	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	This is the top-level for the read datapath.
read_posted_fifo.v	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	This module generates timing control for reads and ready signals to memc_flow_vcontrol.v.
write_data_path.v	This is the top-level for the write datapath.
wr_data_g.v	This module generates timing control for writes and ready signals to memc_flow_vcontrol.v.
s7ven_data_gen.v	This module generates different data patterns.
a_fifo.v	This is a synchronous FIFO using LUT RAMs.
data_prbs_gen.v	This is a 32-bit linear feedback shift register (LFSR) for generating PRBS data patterns.
init_mem_pattern_ctr.v	This module generates flow control logic for the traffic generator.
traffic_gen_top.v	This module is the top-level of the traffic generator and comprises the memc_traffic_gen and init_mem_pattern_ctr modules.

**Notes:**

1. All file names are prefixed with the MIG core version number. For example, the MIG 4.2 release module name of cmd\_gen in generated output is now mig\_7series\_v4\_2\_cmd\_gen.

**`<component name>/example_design/par`**

[Table 1-3](#) lists the modules in the `example_design/par` directory.

*Table 1-3:*

example_top.xdc	This is the XDC for the core and the example design.

**<component\_name>/example\_design/sim**

Table 1-4 lists the modules in the **example\_design/sim** directory.

*Table 1-4:*

ddr2_model.v ddr3_model.v	These are the DDR2 and DDR3 SDRAM models.
ddr2_model_parameters.vh ddr3_model_parameters.vh	These files contain the DDR2 and DDR3 SDRAM model parameter setting.
ies_run.sh <sup>(1)</sup>	Linux Executable file for simulating the design using IES simulator.
vcs_run.sh <sup>(1)</sup>	Linux Executable file for simulating the design using VCS simulator.
readme.txt <sup>(1)</sup>	Contains the details and prerequisites for simulating the designs using Mentor Graphics Questa Advanced Simulator, IES, and VCS simulators.
sim_tb_top.v	This is the simulation top file.

**Notes:**

1. The **ies\_run.sh** and **vcs\_run.sh** files are generated in the folder **mig\_7series\_0\_ex/imports** when the example design is created using **Open IP Example Design** for the design generated with **Component Name** entered in Vivado IDE as **mig\_7series\_0**.

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

The **user\_design** folder contains the following:

- **rtl** and **xdc** folders
- Top-level wrapper module **<component\_name>.v/vhd**
- Top-level modules **<component\_name>\_mig.v/vhd** and **<component\_name>\_mig\_sim.v/vhd**

The top-level wrapper file **<component\_name>.v/vhd** has an instantiation of top-level file **<component\_name>\_mig.v/vhd**.

Top-level files **<component\_name>\_mig.v/vhd** and **<component\_name>\_mig\_sim.v/vhd** have the same module name as **<component\_name>\_mig**. These two files are same in all respects except that the file **<component\_name>\_mig\_sim.v/vhd** has parameter values set for simulation where calibration is in fast mode **viz., SIM\_BYPASS\_INIT\_CAL = "FAST"** etc.



**IMPORTANT:** The top-level file **<component\_name>\_mig.v/vhd** is used for design synthesis and implementation, whereas the top-level file **<component\_name>\_mig\_sim.v/vhd** is used in simulations.

The top-level wrapper file serves as an example for connecting the **user\_design** to the MIG core.

**user\_design/rtl/clocking**

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

*Table 1-5:*

clk_ibuf.v	This module instantiates the input clock buffer.
iodelay_ctrl.v	This module instantiates IDELAYCNTRL primitives needed for IDELAY use.
infrastructure.v	This module helps in clock generation and distribution, and reset synchronization.

**Notes:**

1. All file names are prefixed with the MIG core version number. For example, for the MIG 4.2 release module name of clk\_ibuf in generated output is now mig\_7series\_v4\_2\_clk\_ibuf.

**user\_design/rtl/controller**

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

*Table 1-6:*

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

**Table 1-6:**
*(Cont'd)*

round_robin_arb.v	This is a simple round-robin arbiter.

**Notes:**

1. All file names are prefixed with the MIG core version number. For example, for the MIG 4.2 release module name of arb\_mux in generated output is now mig\_7series\_v4\_2\_arb\_mux.

**user\_design/rtl/ip\_top**

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

**Table 1-7:**

mem_intfc.v	This is the top-level memory interface block that instantiates the controller and the PHY.
memc_ui_top.v	This is the top-level Memory Controller module.

**Notes:**

1. All file names are prefixed with the MIG core version number. For example, for the MIG 4.2 release module name of mem\_intfc in generated output is now mig\_7series\_v4\_2\_mem\_intfc.

**user\_design/rtl/phy**

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

**Table 1-8:**

ddr_byte_group_io	This module contains the parameterizable I/O logic instantiations and the I/O terminations for a single byte lane.
ddr_byte_lane	This module contains the primitive instantiations required within an output or input byte lane.
ddr_calib_top	This is the top-level module for the memory physical layer interface.
ddr_if_post_fifo	This module extends the depth of a PHASER IN_FIFO up to four entries.
ddr_mc_phy	This module is a parameterizable wrapper instantiating up to three I/O banks, each with 4-lane PHY primitives.
ddr_mc_phy_wrapper	This wrapper file encompasses the MC_PHY module instantiation and handles the vector remapping between the MC_PHY ports and your DDR2 or DDR3 ports.
ddr_of_pre_fifo	This module extends the depth of a PHASER OUT_FIFO up to four entries.
ddr_phy_4lanes	This module is the parameterizable 4-lane PHY in an I/O bank.
ddr_phy_ck_addr_cmd_delay	This module contains the logic to provide the required delay on the address and control signals.
ddr_phy_dqs_delay	This module contains the DQS to DQ phase offset logic.

Table 1-8:

(Cont'd)

ddr_phy_dqs_found_cal	This module contains the Read leveling calibration logic (PHASER_JN DQSFOUND calibration logic).
ddr_phy_init	This module contains the memory initialization and overall master state control during initialization and calibration.
ddr_phy_rdlvl	This module contains the Read leveling Stage1 calibration logic (Window detection with PRBS pattern).
ddr_phy_top	This is the top-level module for the physical layer.
ddr_phy_wrcal	This module contains the write calibration logic.
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.

**Notes:**

1. All file names are prefixed with the MIG core version number. For example, for the MIG 4.2 release module name of ddr\_byte\_group\_io in generated output is now mig\_7series\_v4\_2\_ddr\_byte\_group\_io.

**`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:

ui_cmd.v	This is the user interface command port.
ui_rd_data.v	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	This is the user interface write buffer.
ui_top.v	This is the top-level of the Memory Controller user interface.

**Notes:**

1. All file names are prefixed with the MIG core version number. For example, for the MIG 4.2 release module name of ui\_cmd in generated output is now mig\_7series\_v4\_2\_ui\_cmd.

**`<component_name>/user_design/xdc`**

[Table 1-10](#) lists the modules in the `user_design/xdc` directory.

Table 1-10:

<component_name>.xdc	This is the XDC for the core and the user design.

This feature verifies the input XDC for bank selections, byte selections, and pin allocation. It also generates errors and warnings in a separate dialog box when you click **Validate** on the page. This feature is useful to verify the XDC for any pinout changes made after the design is generated from the MIG tool. You must load the MIG tool generated **.prj** file, the original **.prj** file without any modifications, and the XDC that needs to be verified. In the Vivado Design Suite, 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 XDC is not sufficient; it is mandatory to proceed with design generation to get the XDC with updated clock and phaser related constraints and RTL top-level module for various updated Map parameters.

The Update Design feature is required in the following scenarios:

- A pinout is generated using an older version of the MIG tool and the design is to be revised to the current version of MIG. In MIG, the pinout allocation algorithms have been changed for certain MIG designs.
- A pinout is generated independent of the MIG tool or is modified after the design is generated. When a design is generated from the MIG tool, the XDC and HDL code are generated with the correct constraints.

Here are the rules verified from the input XDC:

- If a pin is allocated to more than one signal, the tool reports an error. Further verification is not done if the XDC 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.
  - $V_{REF}$  I/Os should be used as GPIOs when an internal  $V_{REF}$  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 DQS I/O pair.

- An FPGA byte lane should not contain pins related to two different strobe sets.
- $V_{REF}$  I/O can be used only when the internal  $V_{REF}$  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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  is used.
  - Status signals:
    - The `sys_rst` signal should be allocated in the bank where the  $V_{REF}$  I/O is unallocated or the internal  $V_{REF}$  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.

## Overview

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

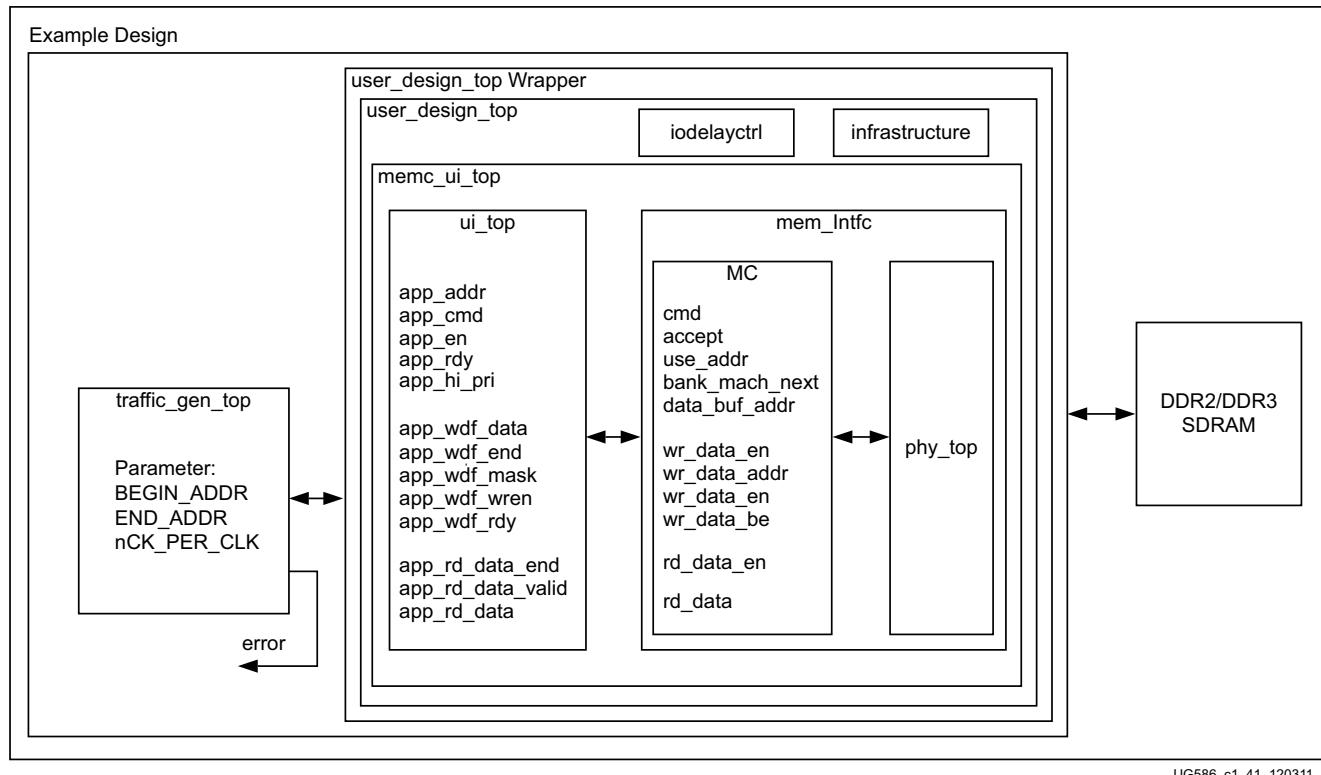
## Implementing the Example Design

For more information on using an IP example design, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

## 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 1-39](#).

ddr2\_sim\_tb\_top or ddr3\_sim\_tb\_top



UG586\_c1\_41\_120311

Figure 1-39:

[Figure 1-40](#) shows the simulation result of a simple read and write transaction between the **tb\_top** and **memc\_intfc** modules.

## ***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.

You can define the address range through the BEGIN\_ADDRESS and END\_ADDRESS parameters. The Init Memory Pattern Control block directs the traffic generator to s

T

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

The 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:*

FAMILY	Indicates the family type.	"VIRTEX7"
MEMORY_TYPE	Indicate the Memory Controller type.	"DDR2", "DDR3"
nCK_PER_CLK	This is the Memory Controller clock to DRAM clock ratio.	4, 2 (depends on the PHY to Controller Clock ratio chosen in the GUI)
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 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.	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.

Table 1-11:

(Cont'd)

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.
PRBS_SADDR_MASK_POS	Sets the 32-bit OR MASK position.	This parameter is used with the PRBS address generator to shift random addresses up into the port address space. The START_ADDRESS value is ORed with the PRBS address for bit positions that have a 1 in this mask
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 increased 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:

(Cont'd)

		Valid settings for this parameter are: <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: Ox1: FIXED – 32 bits of fixed_data. Ox2: ADDRESS – 32 bits address as data. Ox3: HAMMER Ox4: SIMPLE8 – Simple eight data pattern that repeats every eight words. Ox5: WALKING1s – Walking 1s are on the DQ pins. Ox6: WALKING0s – Walking 0s are on the DQ pins. Ox7: PRBS – A 32-stage LFSR generates random data. Ox9: SLOW HAMMER – This is the slow MHz hammer data pattern. OxA: PHY_CALIB pattern – OxFF, O0, AA, 55, 55, AA, 99, 66. This mode only generates READ commands at address zero.</li> </ul>
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.	Valid values: 0 to 32.
CMDS_GAP_DELAY	This parameter allows pause delay between each user burst command.	This parameter only applies to the Hammer pattern. Valid settings for this parameter are 0 to NUM_DQ_PINS. When value = NUM_DQ_PINS, all DQ pins have the same Hammer pattern.
SEL_VICTIM_LINE	Select a victim DQ line whose state is always at logic High.	

Table 1-11:

(Cont'd)

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.
----------	---	--

**Notes:**

1. 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 you to modify the command pattern in real-time using the Vivado debug logic core 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 **1**s are on the DQ pins. The starting position of **1** depends on the address value.

6: **DGEN\_WALKING0**. Walking **0**s are on the DQ pins. The starting position of **0** depends on the address value.

7: **DGEN\_PRBS**. A 32-stage LFSR generates random data and is seeded by the starting address. 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:

0x1000	0xFFFF	0x00001000	0xFFFF0000
0x2000	0xFFFF	0x00002000	0xFFFF0000
0x3000	0xFFFF	0x00003000	0xFFFF0000
0x4000	0xFFFF	0x00004000	0xFFFF0000
0x5000	0xFFFF	0x00005000	0xFFFF0000
0x2000	0x1FFF	0x00002000	0xFFFFE000
0x2000	0x2FFF	0x00002000	0xFFFFD000
0x2000	0x3FFF	0x00002000	0xFFFFC000
0x2000	0x4FFF	0x00002000	0xFFFF8000
0x2000	0x5FFF	0x00002000	0xFFFF8000
0x2000	0x6FFF	0x00002000	0xFFFF8000
0x2000	0x7FFF	0x00002000	0xFFFF8000
0x2000	0x8FFF	0x00002000	0xFFFF0000
0x2000	0x9FFF	0x00002000	0xFFFF0000

*Table 1-12:*
*(Cont'd)*

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:*

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 QDR II+ user interface.

Table 1-13:

(Cont'd)

vio_modify_enable	Input	Allow vio_xxxx_mode_value to alter traffic pattern.
vio_data_mode_value[3:0]	Input	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• Ox0: Reserved.</li> <li>• Ox1: FIXED – 32 bits of fixed_data as defined through fixed_data_i inputs.</li> <li>• Ox2: ADDRESS – 32 bits address as data. Data is generated based on the logical address space. If a design has a 256-bit user data bus, each write beat in the user bus would have a 256/8 address increment in byte boundary. If the starting address is 1,300, the data is 1,300, followed by 1,320 in the next cycle. To simplify the logic, the user data pattern is a repeat of the increment of the address value Bits[31:0].</li> <li>• Ox3: 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>• Ox4: SIMPLE8 – Simple 8 data pattern that repeats every 8 words. The patterns can be defined by the "simple_datax" inputs.</li> <li>• Ox5: 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>• Ox6: 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>• Ox7: 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>• Ox9: SLOW HAMMER – This is the slow MHz hammer data pattern.</li> <li>• OxA: PHY_CALIB pattern – OxFF, OO, 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>• Ox1: FIXED address mode. The address comes from the fixed_addr_i input bus. With FIXED address mode, the data_mode is limited to the fixed_data_input. No PRBS data pattern is generated.</li> <li>• Ox2: 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>• Ox3: SEQUENTIAL address mode. The address is generated from the internal address counter. The increment is determined by the user interface port width.</li> </ul>

Table 1-13:

(Cont'd)

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>
vio_fixed_instr_value	Input	<p>Valid settings are:</p> <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	<p>This signal latches these values when the error signal is asserted:</p> <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.

Table 1-13:

(Cont'd)

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_b1_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.

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. The **bl\_mode\_i** is set to **01** to get the burst length from the **fixed\_b1\_i** input.
5. The **fixed\_b1\_i** input is set to either 16 or 32.
6. The **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. The **addr\_mode\_i** is set to **11** for the sequential address mode to fill up the memory space.
9. The **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).

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 73](#).

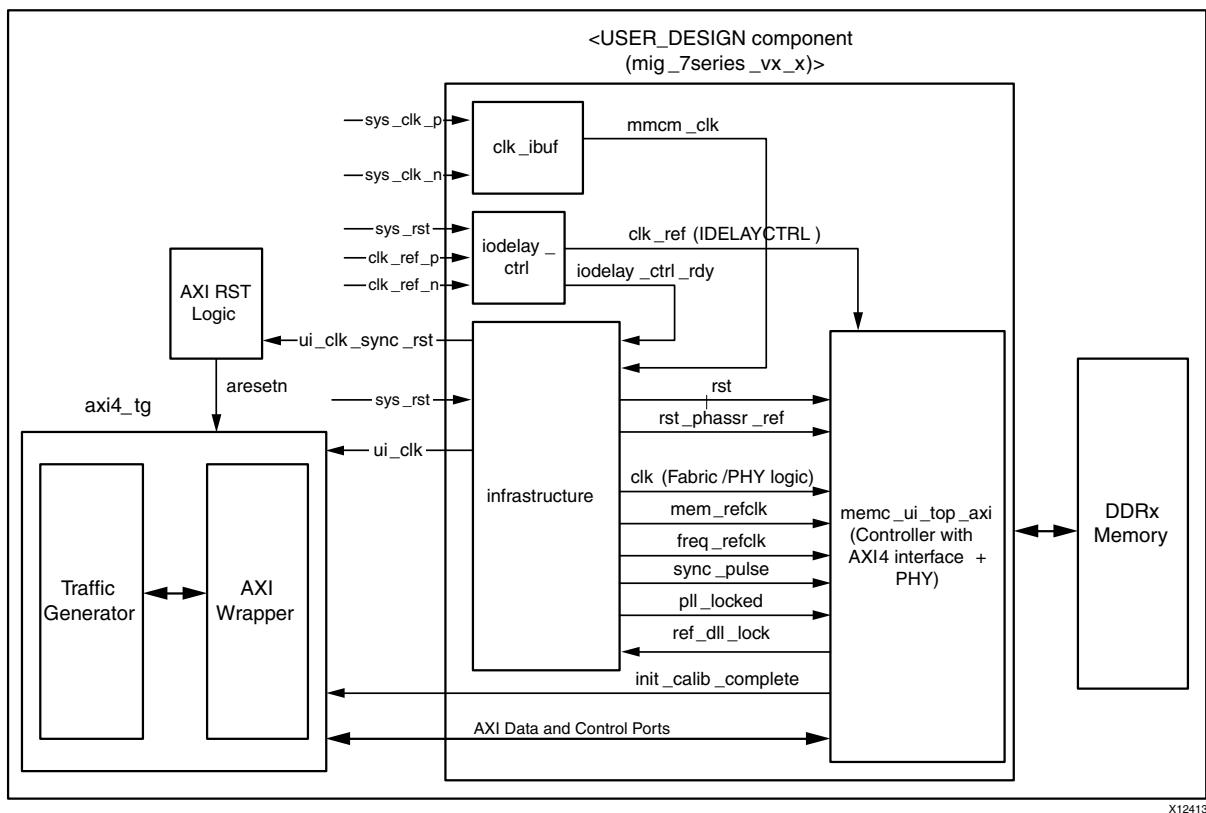
With some modifications, the example design can be changed to allow `addr_mode_i`, `instr_mode_i`, and `bl_mode_i` to be changed dynamically when `run_traffic_i` is deasserted. However, after changing the setting, the memory initialization steps need to be repeated to ensure that the proper pattern is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

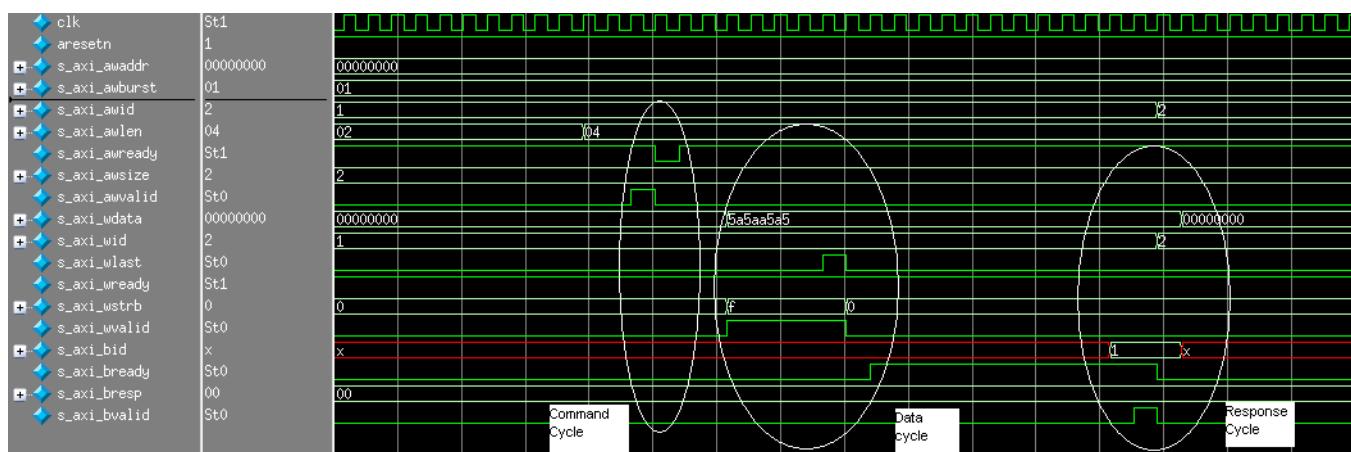
### *Simulating the Example Design (for Designs with the AXI4 Interface)*

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 1-41](#). The infrastructure block inside the user design provides clock resources to both the controller and the traffic generator. [Figure 1-41](#) shows a block diagram of the example design test bench. The details of the clocks in [Figure 1-41](#) are provided in [Clocking Architecture, page 120](#).



*Figure 1-41:*

[Figure 1-42](#) 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 1-42:*

Figure 1-43 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.

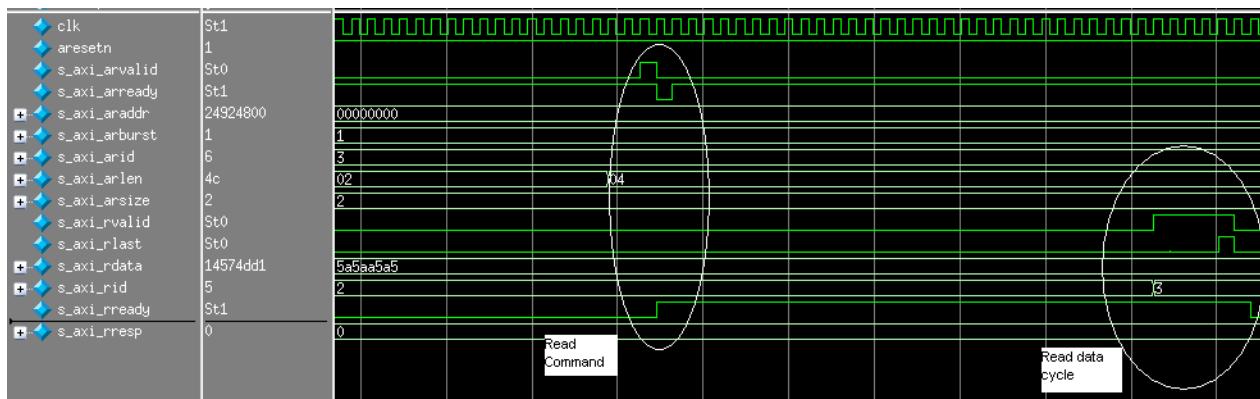


Figure 1-43:

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:

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.
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.

Table 1-14:

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 66](#). The status that is generated for a write transaction can be found in [Figure 1-44](#).

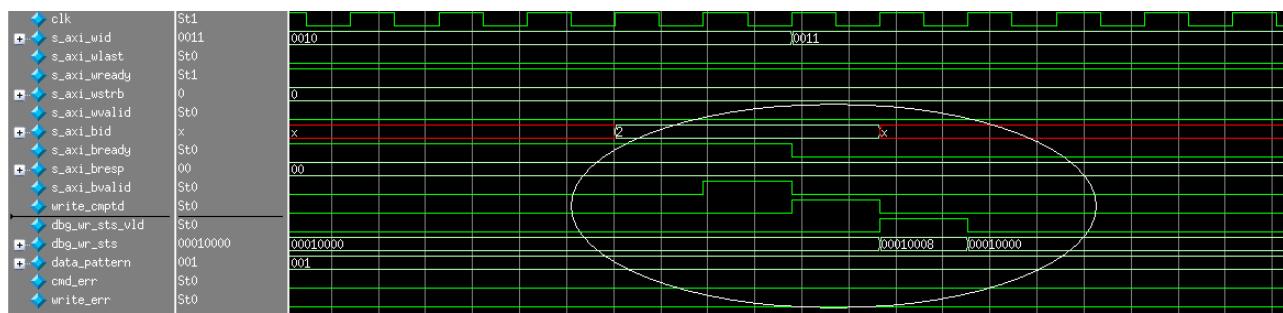


Figure 1-44:

Table 1-15:

39:32	Number of beats/write transfers completed for last burst
31:21	Reserved
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>
17	Write error occurred. The write transaction could not be completed.
16	Command error occurred during a write transaction.
15:9	Reserved
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>

Table 1-15:

(Cont'd)

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

The status generated for a read transaction is shown in Figure 1-45.

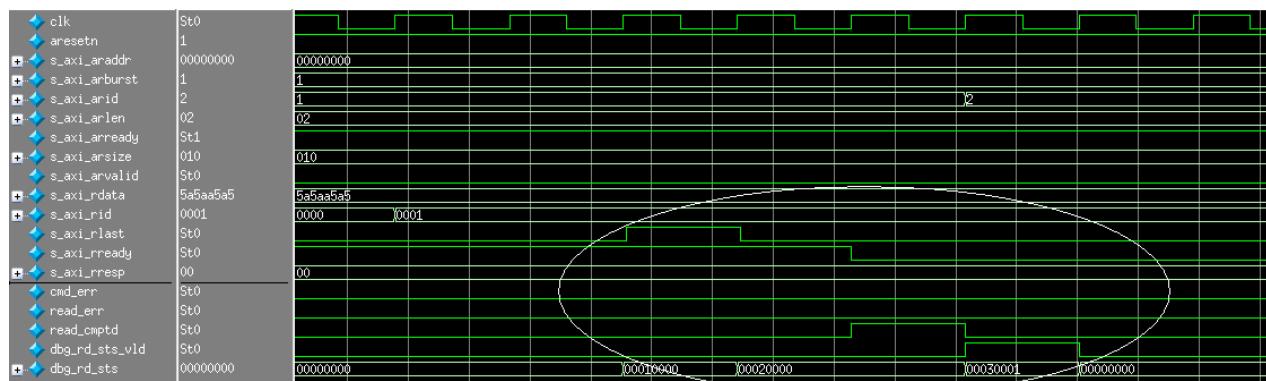


Figure 1-45:

Table 1-16:

39:32	Number of beats/read transfers completed for last burst
31:30	Reserved
29:27	Data pattern used for the current check: <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>
26:19	Pointer value for which the mismatch occurred
18	Data mismatch occurred between the written data and read data
17	Read error occurred, read transaction could not be completed
16	Command error occurred during read transaction
15:4	Reserved
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>
1	Incorrect response ID presented by the AXI slave
0	Read error response on AXI

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 66](#). The AXI4 write and read transactions are started only after the `init_calib_complete` signal is asserted.

## ***Setting Up for Simulation***



---

**IMPORTANT:** *The Xilinx UNISIM library must be mapped into the simulator.*

---

The test bench provided with the example design supports these pre-implementation simulations:

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

The Questa Advanced Simulator, Vivado Simulator, IES, and VCS simulation tools are used for verification of the MIG IP core at each software release. Script files to run simulations with IES and VCS simulators are generated in MIG generated output. Simulations using Questa Advanced Simulator and Vivado simulators can be done through the Vivado Tcl Console commands or in the Vivado IDE.



---

**IMPORTANT:** *Other simulation tools can be used for MIG IP core simulation but are not specifically verified by Xilinx.*

---

To run the simulation, go to this directory:

`<project_dir>/<Component_Name>_ex/imports`

For a project created with the name set as `project_1` and the Component Name entered in Vivado IDE as `mig_7series_0`, go to the directory as follows:

`project_1/mig_7series_ex/imports`

IES and VCS simulation scripts are meant to be executed only in Linux operating systems.

The `ies_run.sh` and `vcs_run.sh` files are the executable files for running simulations using IES and VCS simulators respectively. Library files should be added to the `ies_run.sh` and `vcs_run.sh` files respectively. See the `readme.txt` file for details regarding simulations using IES and VCS.

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings** ([Figure 1-46](#)).

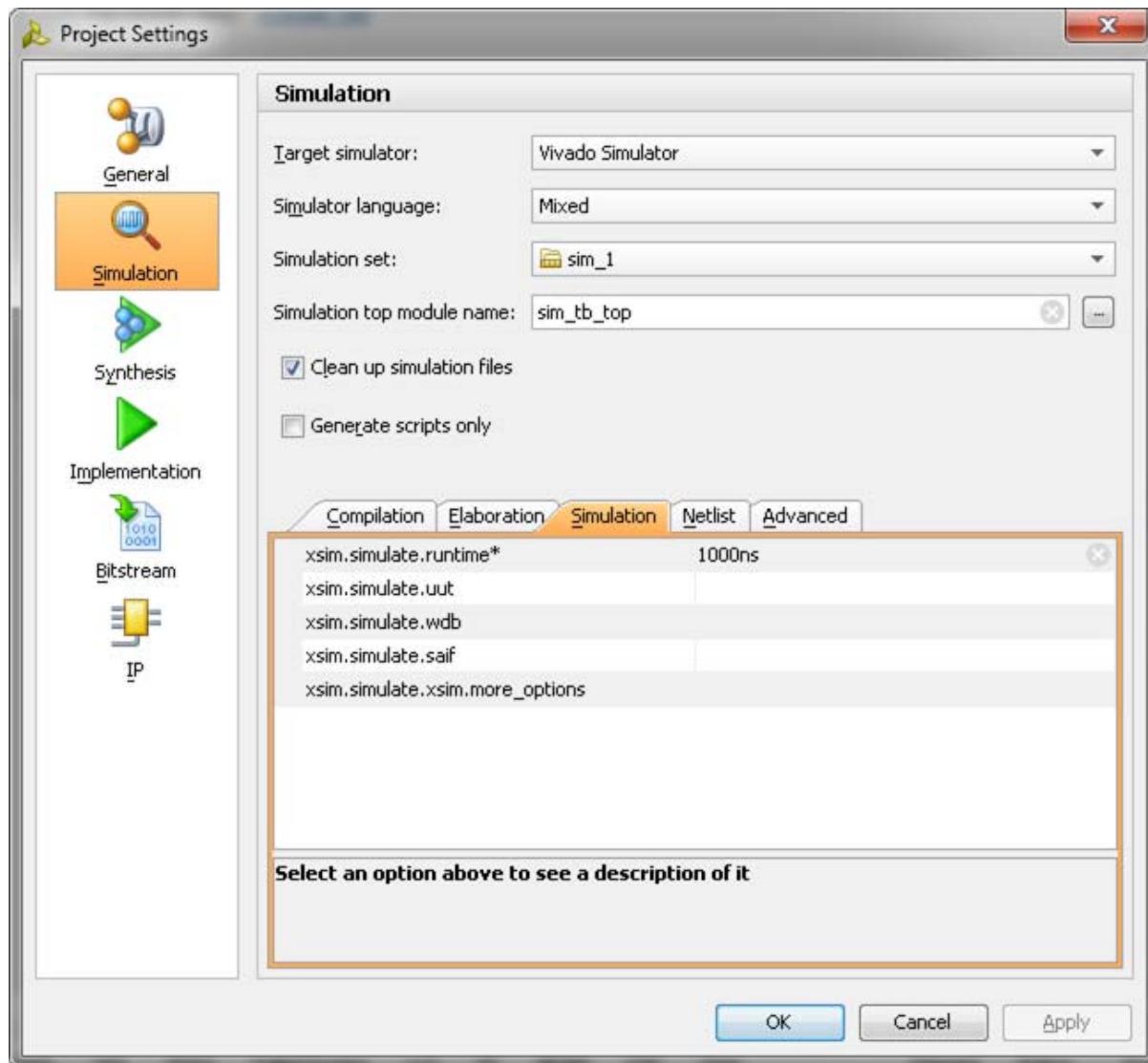
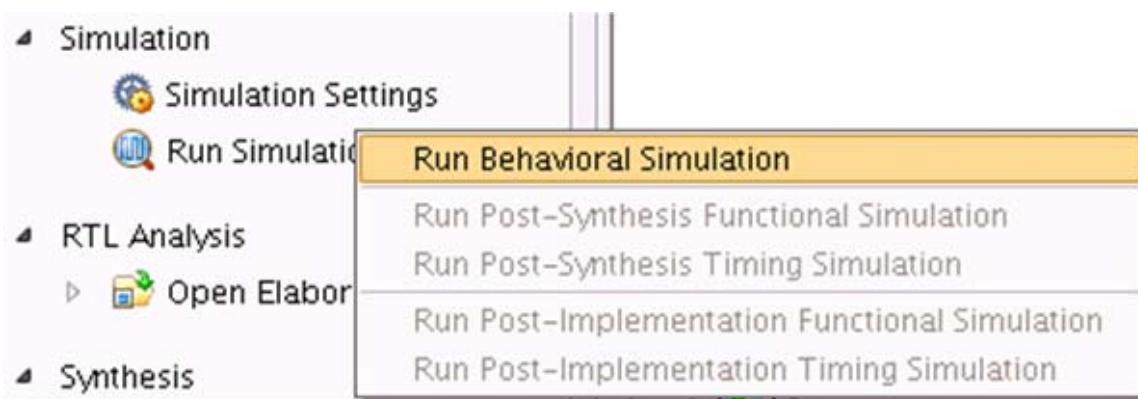


Figure 1-46:

2. Under the **Simulation** tab as shown in [Figure 1-46](#), set the **xsim.simulate.runtime** as 1 ms (there are simulation RTL directives which stop the simulation after a certain period of time, which is less than 1 ms). Apply the settings and select **OK**.

3. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 1-47](#).



*Figure 1-47:*

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Questa Advanced Simulator/ModelSim.
  - a. Browse to the **Compiled libraries location** and set the path on **Compiled libraries location** option.
  - b. Under the **Simulation** tab, set the **modelsim.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms), set **modelsim.simulate.vsim.more\_options** to **-novopt** as shown in [Figure 1-46](#).
3. Apply the settings and select **OK**.

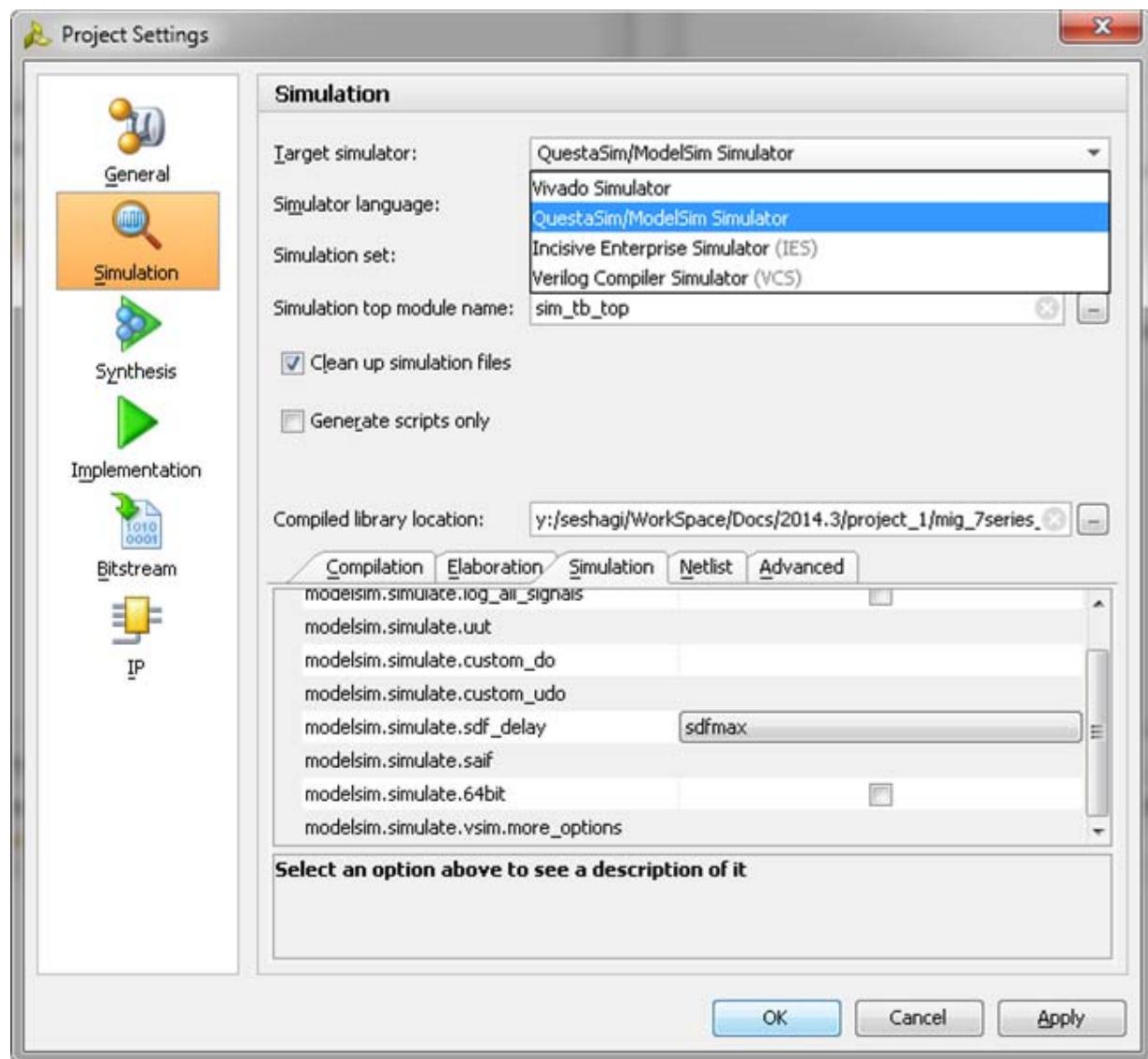


Figure 1-48:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 1-47](#).
  5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG 900)* [Ref 8].
- 
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
  2. Select **Target simulator** as Verilog Compiler Simulator (VCS).

- a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **vcs.compile.vlogan.more\_options** to **-sverilog**.
  - c. Under the **Simulation** tab, set the **vcs.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after a certain period of time which is less than 1 ms) as shown in [Figure 1-49](#).
3. Apply the settings and select **OK**.

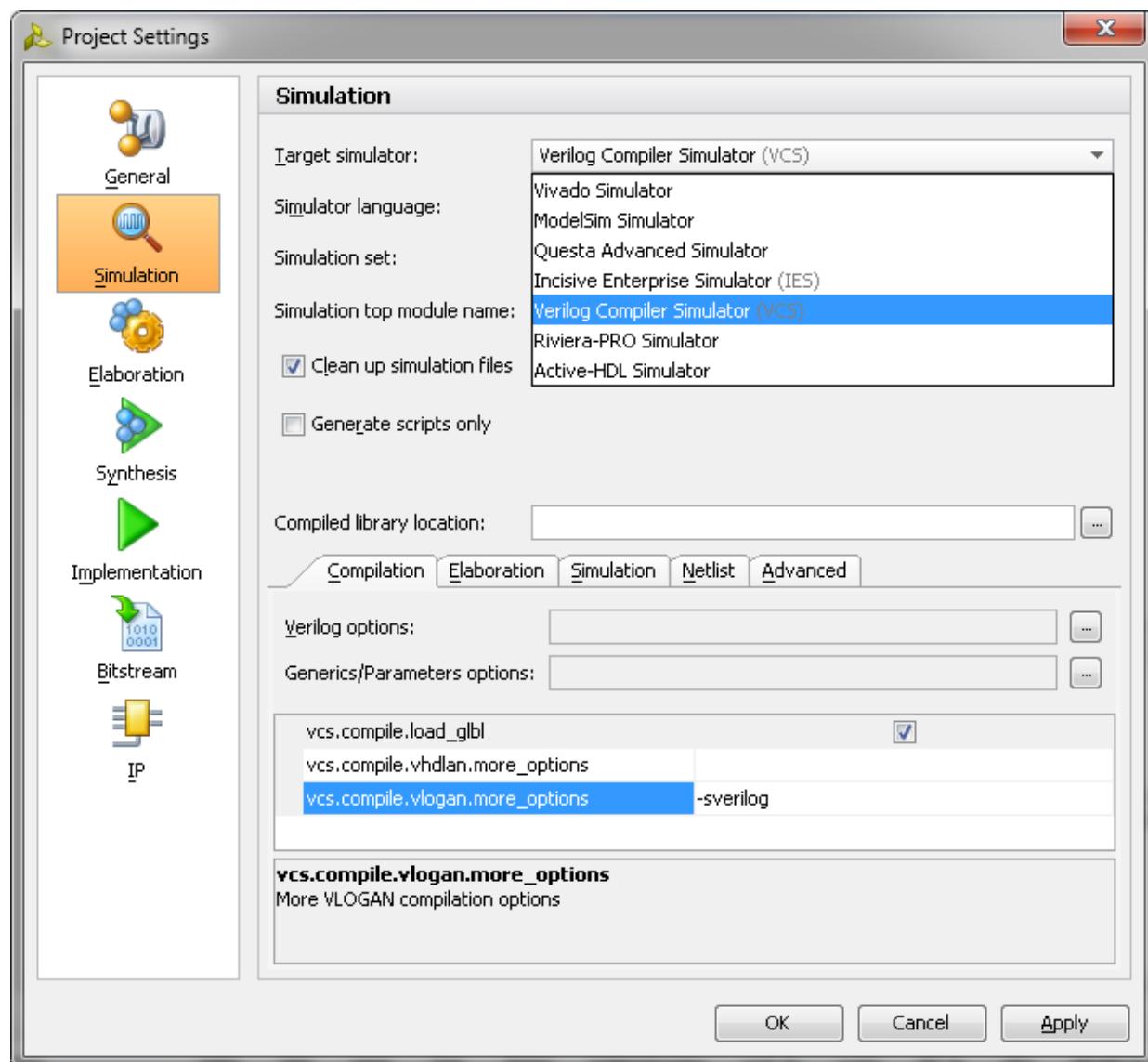


Figure 1-49:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 1-47](#).

5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [Ref 8].
  
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
  - a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **ies.compile.ncvlog.more\_options** to **-sv**.
  - c. Under the **Elaboration** tab, set the **ies.elaborate.ncelab.more\_options** to **-namemap\_mixgen**.
  - d. Under the **Simulation** tab, set the **ies.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time which is less than 1 ms) as shown in [Figure 1-50](#).

3. Apply the settings and select **OK**.

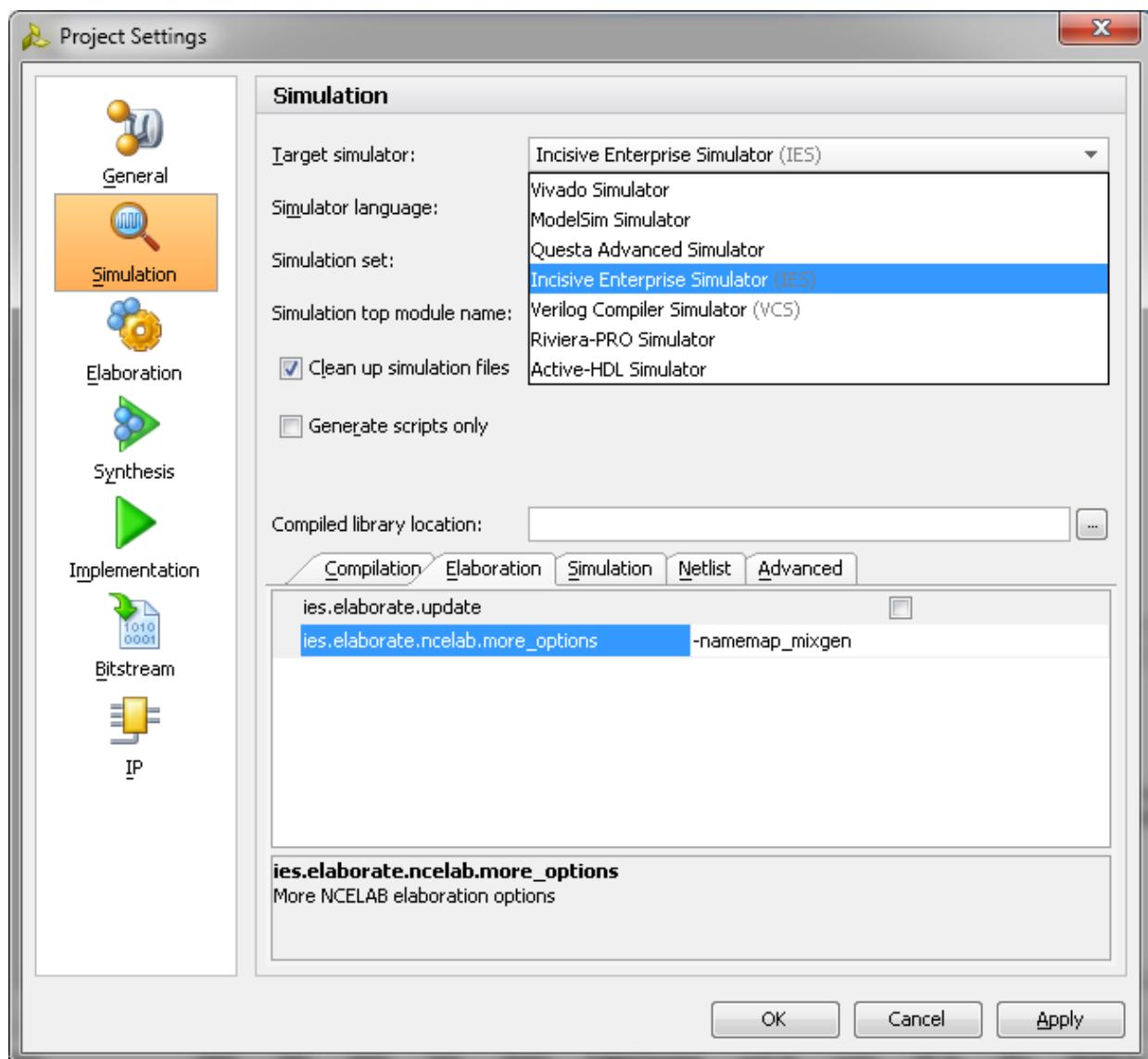


Figure 1-50:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 1-47](#).
5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [\[Ref 8\]](#).

Using the Synopsys® Synplify Pro® flow for **example\_design**, follow these steps to run black box synthesis with Synplify Pro and implementation with Vivado.

1. Generate the 7 series architecture DDR3 SDRAM IP core with OOC flow to generate the **.dcp** file for implementation. The **Target Language** for the project can be selected as **Verilog** or **VHDL**.
2. Create the example design for the DDR3 SDRAM IP core using the information provided in the example design section and close the Vivado project.
3. Invoke the Synplify Pro software which supports 7 series FPGA and select the same 7 series FPGA part selected at the time of generating the IP core.
4. Add the following files into Synplify Pro project based on the **Target Language** selected at the time of invoking Vivado:
  - a. For Verilog:

```
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/ip/<Component_Name>/*stub.v
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/imports/rtl/example_top.v
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/imports/rtl/traffic_gen/*.v
```
  - b. For VHDL:

```
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/ip/<Component_Name>/*stub.vhd
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/imports/rtl/example_top.vhd
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/imports/rtl/traffic_gen/*.v
```

5. Specify top-level module/entity name of the design. In this case it is **example\_top**. Run Synplify Pro synthesis to generate the **.edf** file. Then, close the Synplify Pro project.
6. Open a new Vivado project with Project Type as **Post-synthesis Project** and select the **Target Language**, same as selected at the time of generating the IP core.
7. Add the Synplify Pro generated **.edf** file to the Vivado project as **Design Source**.
8. Add the DDR3 IP **.dcp** file present inside the example project in step 2 to this Vivado project as **Design Source**. For example:

```
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/sources_1/ip/<Component_Name>/<Component_Name>.dcp
```

9. Add the **.xdc** file generated in step 2 to the Vivado project as a **constraint** file. For example:

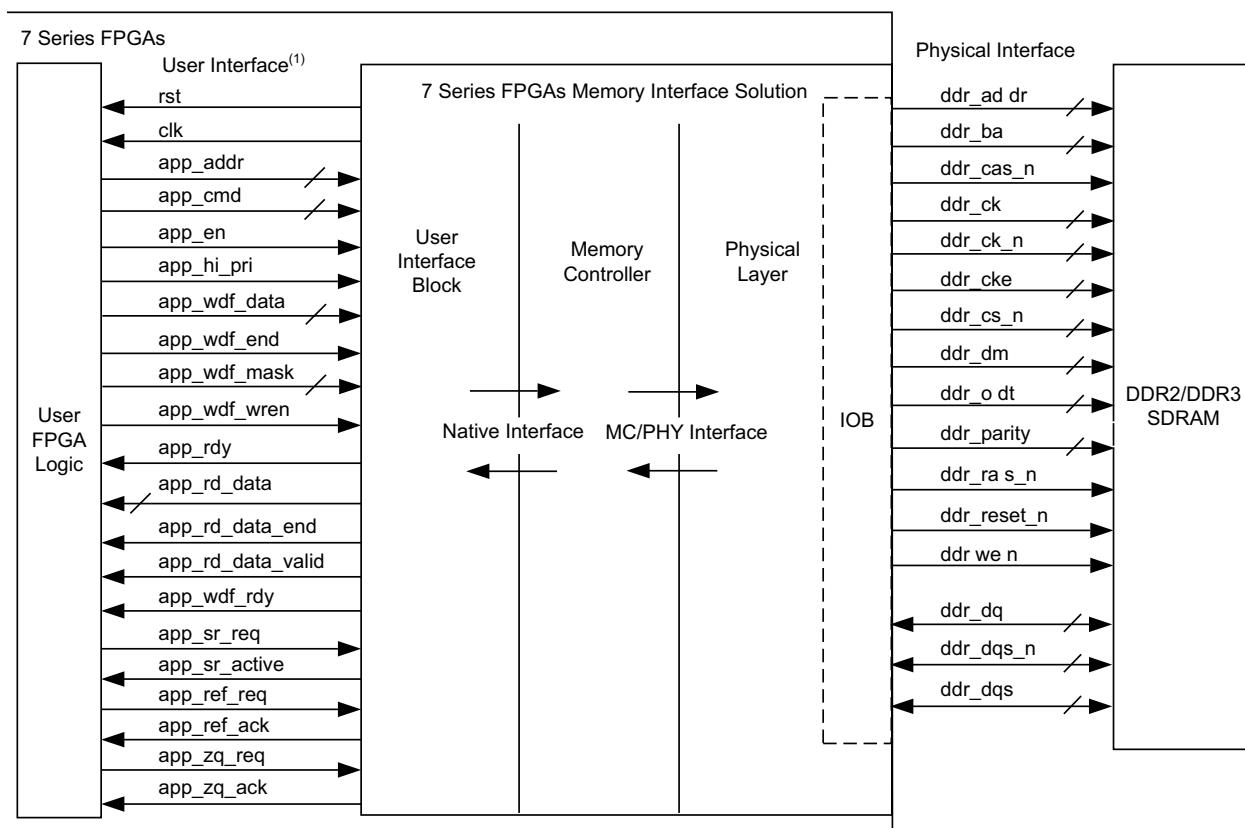
```
<project_dir>/<Component_Name>_example/
<Component_Name>_example.srcs/constrs_1/imports/par/example_top.xdc
```

10. Run implementation flow with the Vivado tool. For details about implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

**Note:** Similar steps can be followed for the user design using appropriate .dcp and .xdc files.

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

The 7 series FPGAs memory interface solutions core is shown in [Figure 1-51](#).



1. System clock (sys\_clk\_p and sys\_clk\_n/sys\_clk\_i), Reference clock (clk\_ref\_p and clk\_ref\_n/clk\_ref\_i), and system reset (sys\_rst\_n) port connections are not shown in block diagram.

*Figure 1-51:*

## *User FPGA Logic*

The user FPGA logic block shown in [Figure 1-51](#) is any FPGA design that requires to be connected to an external DDR2 or DDR3 SDRAM. The user FPGA logic connects to the Memory Controller through the user interface. An example user FPGA logic 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 FPGA logic 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 backend 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 backend 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 IDELAYCTRL and then uses the IODELAY\_GROUP attribute (see the `iodelay_ctrl.v` module). Based on this attribute, the Vivado Design Suite properly replicates IDELAYCTRLs as needed within the design.

The IDELAYCTRL reference frequency is set by the MIG tool to either 200 MHz, 300 MHz, or 400 MHz depending on memory interface frequency and speed grade of the FPGA. Based on the IODELAY\_GROUP attribute that is set, the Vivado Design Suite replicates the IDELAYCTRLs for each region where the IDELAY blocks exist.

When a user creates a multicontroller design on their own, each MIG output has the component instantiated with the primitive. This violates the rules for IDELAYCTRLs and the usage of the IODELAY\_GRP attribute. IDELAYCTRLs need to have only one instantiation of the component with the attribute set properly, and allow the tools to replicate as needed.

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:*

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.
D S U G B 00 @ %p @ & A 0 € p		D S U G B 00 @ %p @ & A 0 € p
app_rdy	Output	This output indicates that the UI is ready to accept commands. This signal is asserted when app_en is asserted, app_cmd and app_addr must be retried until app_rdy is asserted.
app_hi_pri	Input	This active-High input elevates the priority of the current request.
app_rd_data [APP_DATA_WIDTH - 1:0]	Output	This provides the output data from read commands.
app_rd_data_end	Output	This active-High output indicates that the current clock cycle is the last cycle of output data on app_rd_data[]. This is valid only when app_rd_data_valid is active-High.
app_rd_data_valid	Output	This active-High output indicates that app_rd_data[] is valid.
app_sz	Input	This input is res

B G D W D  
S B : , ' 7 + ^ 2 @

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.
ui_clk	Output	This UI clock must be a half or quarter of the DRAM clock.
init_calib_complete	Output	PHY asserts init_calib_complete when calibration is finished.

a nMem

### ***app\_addr[ADDR\_WDTH - 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 you.

### ***app\_cmd[20]***

This input specifies the command for the request currently being submitted to the UI. The available commands are shown in [Table 1-18](#).

### *app\_en*

This input strobes in a request. You 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\_WDTH - 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\_WDTH - 1:0]***

This bus indicates which bytes of **app\_wdf\_data[]** are written to the external memory and which bytes remain in their current state. The bytes are masked by setting a value of 1 to the corresponding bits in **app\_wdf\_mask**. For example, if the application data width is 256, the mask width takes a value of 32. The least significant byte [7:0] of **app\_wdf\_data** is masked using Bit[0] of **app\_wdf\_mask** and the most significant byte [255:248] of **app\_wdf\_data** is masked using Bit[31] of **app\_wdf\_mask**. Hence if you have to mask the last DWORD, that is, bytes 0, 1, 2, and 3 of **app\_wdf\_data**, the **app\_wdf\_mask** should be set to 32'h0000\_000F.

### *app\_wdf\_wren*

This input indicates that the data on the **app\_wdf\_data[]** bus is valid.

### *app\_rdy*

This output indicates to you 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\_WDTH - 1:0]***

This output contains the data read from the external memory.

### ***app\_rd\_data\_end***

This output indicates that the data on the **app\_rd\_data[]** bus in the current cycle is the last data for the current request.

### ***app\_rd\_data\_valid***

This output indicates that the data on the **app\_rd\_data[]** bus is valid.

### ***app\_wdf\_rdy***

This output indicates that the write data FIFO is ready to receive data. Write data is accepted when both **app\_wdf\_rdy** and **app\_wdf\_wren** are asserted.

### ***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.

### ***ui\_dk\_sync\_rst***

This is the reset from the UI which is in synchronous with **ui\_clk**.

### ***ui\_dk***

This is the output clock from the UI. It must be a half or quarter the frequency of the clock going out to the external SDRAM, which depends on 2:1 or 4:1 mode selected in GUI.

### ***init\_calib\_complete***

The 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.

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 RTL is consistent between both tools. For details on the AXI4 signaling protocol, see the Arm AMBA specifications [Ref 4].

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.

If ECC is enabled, all write commands with any of the mask bits enabled are issued as read-modify-write operation.

If ECC is enabled, all write commands with none of the mask bits enabled are issued as write operation.

***AXI4 Slave Interface Parameters***

Table 1-19 lists the AXI4 slave interface parameters.

Table 1-19:

C_S_AXI_ADDR_WIDTH	32	32	This is the width of address read and address write signals. This value must be set to 32.
C_S_AXI_DATA_WIDTH	32	32, 64, 128, 256	This is the width of data signals; a width of APP_DATA_WIDTH is recommended for better performance. Using a smaller width invokes an Upsizer, which would spend clocks in packing the data.
C_S_AXI_ID_WIDTH	4	1-16	This is the width of ID signals for every channel.
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, WRITE_PRIORITY_REG, WRITE_PRIORITY	This parameter indicates the Arbitration algorithm scheme. See <a href="#">Arbitration in AXI Shim, page 101</a> for more information.
C_S_AXI_BASEADDR	-	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 two. Additionally, the base/high address pair must be aligned to a multiple of the accessible size. The minimum accessible size is 4,096 bytes.

**Table 1-19:**
*(Cont'd)*

C_S_AXI_HIGHADDR	-	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 two. Additionally, the base/high address pair must be aligned to a multiple of the accessible size. The minimum accessible size is 4,096 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. All of the AXI interface signals are synchronous to **ui\_clk**.

**Table 1-20:**

aresetn	1	Input	Low	Input reset to the AXI Shim and it should be in synchronous with FPGA logic clock.
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 strb
i_ssPtoqnR d	a n			
W, dQ M y				
s_axi_wb	Hput			
		Ho	Hau	

Table 1-20:

(Cont'd)

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

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. For example, the read requests are served in alternative clock cycles, even when there are no write requests. The slots are fixed and they are served in their respective slots only.

### ***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 last served request granted from the AXI master. For example, if the last performed operation is write, then it gives precedence for read operation to be served over write operation. Similarly, if the last performed operation is read, then it gives precedence for write operation to be served over read operation. If both read and write channels requests at the same time when there are no pending requests, this scheme serves write channel ahead of read.

### ***Read Priority (RD\_PRI\_REG)***

Read and write address channels are served with equal priority in this mode. The requests from the write address channel are processed when one of the following occurs:

- No pending requests from read address channel.
- Read starve limit of 256 is reached. It is only checked at the end of the burst.
- Read wait limit of 16 is reached.
- Write QOS is higher which is non-zero. It is only checked at the end of the burst.

The requests from the read address channel are processed in a similar method.

### ***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.

## ***Write Priority (WRITE\_PRIORITY, WRITE\_PRIORITY\_REG)***

Write address channel is always given priority in this mode. The requests from the read address channel are processed when there are no pending requests from the write address channel. Arbitration outputs are registered in WRITE\_PRIORITY\_REG mode.

The AXI4-Lite Slave Control register block provides a processor accessible interface to the ECC memory option. The interface is available when ECC is enabled and the primary slave interface is AXI4. The block provides interrupts, interrupt enable, ECC status, ECC enable/disable, ECC correctable errors counter, first failing correctable/uncorrectable data, ECC and address. Fault injection registers for software testing is provided when the ECC\_TEST\_FI\_XOR (C\_ECC\_TEST) parameter is "ON." The AXI4-Lite interface is fixed at 32 data bits and signaling follows the standard AMBA AXI4-Lite specifications [Ref 4].

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

### ***ECC Enable/Disable***

The ECC\_ON\_OFF register enables/disables the ECC decode functionality. However, encoding is always enabled. The default value at start-up can be parameterized with C\_ECC\_ONOFF\_RESET\_VALUE. Assigning a value of 1 for the ECC\_ON\_OFF bit of this register results in the **correct\_en** signal input into the mem\_intf to be asserted. Writing a value of 0 to the ECC\_ON\_OFF bit of this register results in the **correct\_en** signal to be deasserted. When **correct\_en** is asserted, decoding is enabled, and the opposite is true when this signal is deasserted. ECC\_STATUS/ECC\_CE\_CNT are not updated when ECC\_ON\_OFF = 0. The FI\_D0, FI\_D1, FI\_D2, and FI\_D3 registers are not writable when ECC\_ON\_OFF = 0.

### ***Single Error and Double Error Reporting***

Two vectored signals from the Memory Controller indicate an ECC error: **ecc\_single** and **ecc\_multiple**. The **ecc\_single** signal indicates if there has been a correctable error, and the **ecc\_multiple** signal indicates if there has been an uncorrectable error. The widths of **ecc\_multiple** and **ecc\_single** are based on the C\_NCK\_PER\_CLK parameter.

There can be between 0 and  $C\_NCK\_PER\_CLK \times 2$  errors per cycle with each data beat signaled by one of the vector bits. Multiple bits of the vector can be signaled per cycle indicating that multiple correctable errors or multiple uncorrectable errors have been detected. The **ecc\_err\_addr** signal (discussed in [Fault Collection](#)) is valid during the assertion of either ecc\_single or ecc\_multiple.

The ECC\_STATUS register sets the CE\_STATUS bit and/or UE\_STATUS bit for correctable error detection and uncorrectable error detection, respectively.



**CAUTION!** *Multiple bit error is a serious failure of memory and it is uncorrectable. In such cases, the application cannot rely on the contents of the memory. It is suggested to not perform any further transactions to memory.*

When interrupts are enabled with the CE\_EN\_IRQ and/or UE\_EN\_IRQ bits of the ECC\_EN\_IRQ register, if a correctable error or uncorrectable error occurs, the interrupt signal is asserted.

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

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

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

## ***AXI4-Lite Slave Control/Status Register Interface Parameters***

[Table 1-21](#) lists the AXI4-Lite slave interface parameters.

Table 1-21:

C_S_AXI_CTRL_ADDR_WIDTH	32	32, 64	This is the width of the AXI4-Lite address buses.
C_S_AXI_CTRL_DATA_WIDTH	32	32	This is the width of the AXI4-Lite data buses.
C_ECC_ONOFF_RESET_VALUE	1	0, 1	Controls ECC on/off value at startup/reset.
C_S_AXI_CTRL_BASEADDR	-	Valid Address	This parameter specifies the base address for the AXI4-Lite slave interface.
C_S_AXI_CTRL_HIGHADDR	-	Valid Address	This parameter specifies the high address for the AXI4-Lite slave interface.
C_S_AXI_CTRL_PROTOCOL	AXI4LITE	AXI4LITE	AXI4-Lite protocol

### ***AXI4-Lite Slave Control/Status Register Interface Signals***

Table 1-22 lists the AXI4 slave interface specific signals. Clock/reset to the interface is provided from the Memory Controller.

Table 1-22:

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

Table 1-22:

(Cont'd)

s_axi_ctrl_ready	1	Input		Read ready.
interrupt	1	Output	High	IP Global Interrupt signal

**AXI4-Lite Slave Control/Status Register Map**

ECC register map is shown in [Table 1-23](#). The register map is Little Endian. Write accesses to read-only or reserved values are ignored. Read accesses to write-only or reserved values return the value OxDEADDEAD.

Table 1-23:

0x00	ECC_STATUS	R/W	0x0	ECC Status Register
0x04	ECC_EN_IRQ	R/W	0x0	ECC Enable Interrupt Register
0x08	ECC_ON_OFF	R/W	0x0 or 0x1	ECC On/Off Register. If C_ECC_ONOFF_RESET_VALUE = 1, the default value is 0x1.
0x0C	CE_CNT	R/W	0x0	Correctable Error Count Register
0x100	CE_FFD[31:00]	R	0x0	Correctable Error First Failing Data Register.
0x104	CE_FFD[63:32]	R	0x0	Correctable Error First Failing Data Register
0x108	CE_FFD[95:64] <sup>(1)</sup>	R	0x0	Correctable Error First Failing Data Register.
0x10C	CE_FFD [127:96] <sup>(1)</sup>	R	0x0	Correctable Error First Failing Data Register.
0x180	CE_FFE	R	0x0	Correctable Error First Failing ECC Register.
0x1C0	CE_FFA[31:0]	R	0x0	Correctable Error First Failing Address
0x1C4	CE_FFA[63:32] <sup>(2)</sup>	R	0x0	Correctable Error First Failing Address
0x200	UE_FFD [31:00]	R	0x0	Uncorrectable Error First Failing Data Register
0x204	UE_FFD [63:32]	R	0x0	Uncorrectable Error First Failing Data Register
0x208	UE_FFD [95:64] <sup>(1)</sup>	R	0x0	Uncorrectable Error First Failing Data Register
0x20C	UE_FFD [127:96] <sup>(1)</sup>	R	0x0	Uncorrectable Error First Failing Data Register
0x280	UE_FFE	R	0x0	Uncorrectable Error First Failing ECC Register

Table 1-23:

(Cont'd)

0x2C0	UE_FFA[31:0]	R	0x0	Uncorrectable Error First Failing Address
0x2C4	UE_FFA[63:32] <sup>(2)</sup>	R	0x0	Uncorrectable Error First Failing Address
0x300	FI_D[31:0] <sup>(3)</sup>	W	0x0	Fault Inject Data Register
0x304	FI_D[63:32] <sup>(3)</sup>	W	0x0	Fault Inject Data Register
0x308	FI_D[95:64] <sup>(1)(3)</sup>	W	0x0	Fault Inject Data Register
0x30C	FI_D[127:96] <sup>(1)(3)</sup>	W	0x0	Fault Inject Data Register
0x380	FI_ECC <sup>(3)</sup>	W	0x0	Fault Inject ECC Register

**Notes:**

1. Data bits 64–127 are only enabled if the DQ width is 144 bits.
2. Reporting address bits 63–32 are only available if the address map is > 32 bits.
3. FI\_D\* and FI\_ECC\* are only enabled if ECC\_TEST parameter has been set to 1.

***AXI4-Lite Slave Control/Status Register Map Detailed Descriptions***

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

Table 1-24:

31:2	Reserved	RSVD	-	Reserved
1	CE_STATUS	R/W	0	If 1, a correctable error has occurred. This bit is cleared when a 1 is written to this bit position.
0	UE_STATUS	R/W	0	If 1, an uncorrectable error has occurred. This bit is cleared when a 1 is written to this bit position

This register determines if the values of the CE\_STATUS and UE\_STATUS bits in the ECC Status Register assert the Interrupt output signal (ECC\_INTERRUPT). If both CE\_EN\_IRQ and

UE\_EN IRQ are set to 1 (enabled), the value of the Interrupt signal is the logical OR between the CE\_STATUS and UE\_STATUS bits.

The ECC On/Off Control Register allows the application to enable or disable ECC checking. The design parameter, C\_ECC\_ONOFF\_RESET\_VALUE (default on) determines the reset value for the enable/disable setting of ECC. This facilitates start-u

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

**Table 1-28:**

31:0	CE_FFA[31:0]	R	0	Address (Bits[31:0]) of the first occurrence of a correctable error

**Note:** This register is unused if C\_S\_AXI\_ADDR\_WIDTH < 33.

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

**Table 1-29:**

31:0	CE_FFA[63:32]	R	0	Address (Bits[63:32]) of the first occurrence of a correctable error.

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

**Table 1-30:**

31:0	CE_FFD[31:0]	R	0	Data (Bits[31:0]) of the first occurrence of a correctable error.

This register stores the (corrected) failing data (Bits[63:32]) of the first occurrence of an access with a correctable error. When the CE\_STATUS bit in the ECC Status register is cleared,

this register is re-enabled to store the data of the next correctable error. Storing of the failing data is enabled after reset.

Table 1-31:

31:0	CE_FFD[63:32]	R	0	Data (Bits[63:32]) of the first occurrence of a correctable error.

**Note:** This register is only used when DQ\_WIDTH == 144.

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

Table 1-32:

31:0	CE_FFD[95:64]	R	0	Data (Bits[95:64]) of the first occurrence of a correctable error.

**Note:** This register is only used when DQ\_WIDTH == 144.

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

Table 1-33:

31:0	CE_FFD [127:96]	R	0	Data (Bits[127:96]) of the first occurrence of a correctable error.

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

[Table 1-34](#) describes the register bit usage when DQ\_WIDTH = 72.

*Table 1-34:*

31:8	Reserved	RSVD	-	Reserved
7:0	CE_FFE	R	0	ECC (Bits[7:0]) of the first occurrence of a correctable error.

[Table 1-35](#) describes the register bit usage when DQ\_WIDTH = 144.

*Table 1-35:*

31:16	Reserved	RSVD	-	Reserved
15:0	CE_FFE	R	0	ECC (Bits[15:0]) of the first occurrence of a correctable error.

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

*Table 1-36:*

31:0	UE_FFA [31:0]	R	0	Address (Bits[31:0]) of the first occurrence of an uncorrectable error.

**Note:** This register is unused if C\_S\_AXI\_ADDR\_WIDTH < 33.

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

*Table 1-37:*

31:0	UE_FFA[63:32]	R	0	Address (Bits[63:32]) of the first occurrence of an uncorrectable error

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

*Table 1-38:*

31:0	UE_FFD[31:0]	R	0	Data (Bits[31:0]) of the first occurrence of an uncorrectable error.

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

*Table 1-39:*

31:0	UE_FFD [63:32]	R	0	Data (Bits[63:32]) of the first occurrence of an uncorrectable error.

**Note:** This register is only used when the DQ\_WIDTH == 144.

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

*Table 1-40:*

31:0	UE_FFD[95:64]	R	0	Data (Bits[95:64]) of the first occurrence of an uncorrectable error.

**Note:** This register is only used when the DQ\_WIDTH == 144.

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

*Table 1-41:*

31:0	UE_FFD[127:96]	R	0	Data (Bits[127:96]) of the first occurrence of an uncorrectable error.

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

[Table 1-42](#) describes the register bit usage when DQ\_WIDTH = 72.

*Table 1-42:*

31:8	Reserved	RSVD	-	Reserved
7:0	UE_FFE	R	0	ECC (Bits[7:0]) of the first occurrence of an uncorrectable error.

[Table 1-43](#) describes the register bit usage when DQ\_WIDTH = 144.

*Table 1-43:*

31:16	Reserved	RSVD	-	Reserved
15:0	UE_FFE	R	0	ECC (Bits[15:0]) of the first occurrence of an uncorrectable error.

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

The register is only implemented if C\_ECC\_TEST = "ON" or ECC\_TEST\_FL\_XOR = "ON" and ECC = "ON" in a MIG design in the Vivado IP catalog.

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

*Table 1-44:*

31:0	FI_DO	W	0	Bit positions set to 1 toggle the corresponding Bits[31:0] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

Special consideration must be given across FI\_DO, FI\_D1, FI\_D2, and FI\_D3 such that only a single error condition is introduced.

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

This register is only implemented if C\_ECC\_TEST = "ON" or ECC\_TEST\_FL\_XOR = "ON" and ECC = "ON" in a MIG design in the Vivado IP catalog.

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

*Table 1-45:*

31:0	FI_D1	W	0	Bit positions set to 1 toggle the corresponding Bits[63:32] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

**Note:** This register is only used when DQ\_WIDTH = 144.

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

This register is only implemented if C\_ECC\_TEST = "ON" or ECC\_TEST\_FL\_XOR = "ON" and ECC = "ON" in a MIG design in the Vivado IP catalog.

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

Table 1-46:

31:0	FI_D2	W	0	Bit positions set to 1 toggle the corresponding Bits[95:64] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

Special consideration must be given across FI\_D0, FI\_D1, FI\_D2, and FI\_D3 such that only a single error condition is introduced.

**Note:** This register is only used when DQ\_WIDTH = 144.

This register is used to inject errors in data (Bits[127:96]) written to memory and can be used to test the error correction and error signaling. The bits set in the register toggle the corresponding data bits (word 3 or Bits[127:96]) of the subsequent data written to the memory without affecting the ECC bits written. After the fault has been injected, the Fault Injection Data register is cleared automatically.

The register is only implemented if C\_ECC\_TEST = "ON" or ECC\_TEST\_FL\_XOR = "ON" and ECC = "ON" in a MIG design in the Vivado Design Suite.

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

Table 1-47:

31:0	FI_D3	W	0	Bit positions set to 1 toggle the corresponding Bits[127:96] of the next data word written to the memory. The register is automatically cleared after the fault has been injected.

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

The register is only implemented if C\_ECC\_TEST = "ON" or ECC\_TEST\_FL\_XOR = "ON" and ECC = "ON" in a MIG design in the Vivado IP catalog.

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

[Table 1-48](#) describes the register bit usage when DQ\_WIDTH = 72.

*Table 1-48:*

31:8	Reserved	RSVD	-	Reserved
7:0	FI_ECC	W	0	Bit positions set to 1 toggle the corresponding bit of the next ECC written to the memory. The register is automatically cleared after the fault has been injected.

[Table 1-49](#) describes the register bit usage when DQ\_WIDTH = 144.

*Table 1-49:*

31:16	Reserved	RSVD	-	Reserved
15:0	FI_ECC	W	0	Bit positions set to 1 toggle the corresponding bit of the next ECC written to the memory. The register is automatically cleared after the fault has been injected.

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

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 1-50](#).

*Table 1-50:*

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.

Table 1-50:

(Cont'd)

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>
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 1-51](#).

Table 1-51:

Memory write	000
Memory read	001
Reserved	All other codes

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.

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

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 1-52](#)). These signals connect to the control, address, and data signals of a buffer in the user design.

EVOW WH GO      **l@00** 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.

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.

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.

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 b      p    //      F   f

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 `wr_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 1-53). 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 1-53:*

<code>rd_data[2 × nCK_PER_CLK × PAYLOAD_WIDTH – 1:0]</code>	Output	This is the output data from read commands.
<code>rd_data_addr[DATA_BUF_ADDR_WIDTH – 1:0]</code>	Output	This output provides the base address of the destination buffer for read commands.
<code>rd_data_en</code>	Output	This output indicates that valid read data is available on the <code>rd_data</code> bus.
<code>rd_data_offset[1:0]</code>	Output	This output provides the offset for the destination buffer for read commands.

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.

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.

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.

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 1-54 lists the native interface maintenance command signals.

Table 1-54:

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.

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.

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.

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.

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.

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 PHY also requires one MMCM in the same bank as the PLL. This MMCM compensates for the insertion delay of the BUFG to the PHY.

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

For DDR3 designs, one MMCM is required for IDELAY reference clock generation. If the design frequency is > 667 MHz, then IDELAY reference clock is either 300 MHz or 400 MHz (depending on FPGA speed grade). MIG instantiates one MMCM for 300 MHz and 400 MHz clock generation.

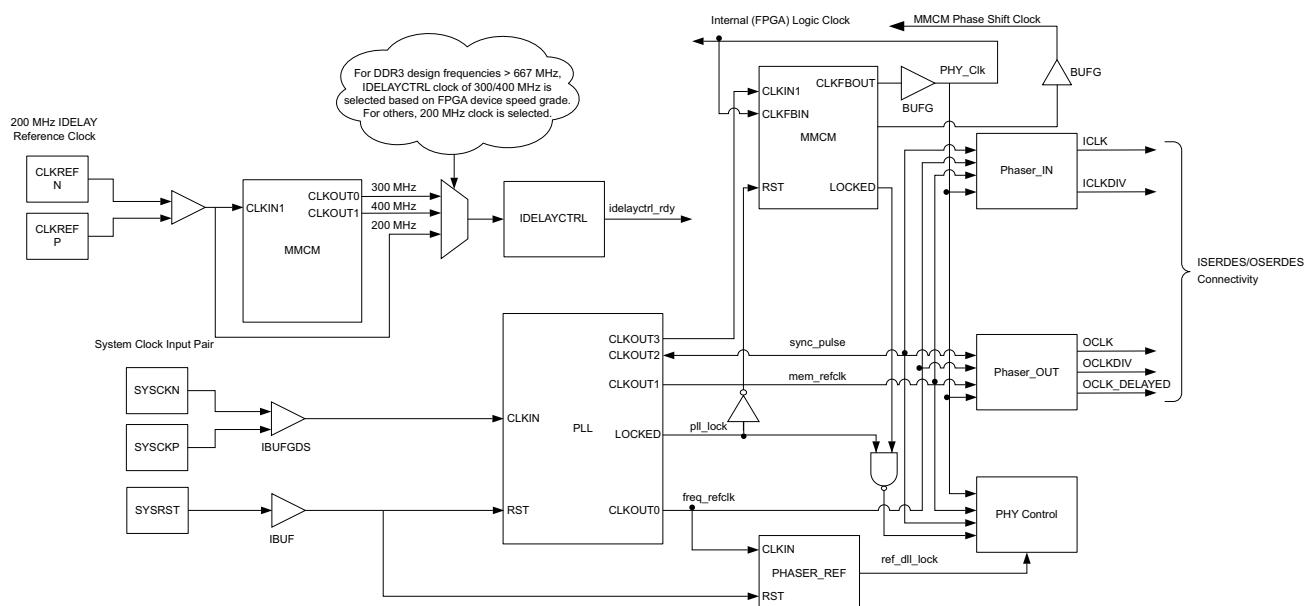
One MMCM and one PLL are required for the PHY. The PLL is used to generate the clocks for most of the internal logic, the frequency reference clocks to the phasers, and a synchronization pulse required for keeping PHY control blocks synchronized in multi-I/O bank implementations.

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 1-52](#). The phase of `freq_refclk` varies based on frequency of operation and banks selected for memory interface pins.

- When HP banks are selected for memory interface pins in GUI and the memory frequencies  $\geq$  400 MHz, the phase is 337.5°.
- When HP banks are selected for memory interface pins in GUI and the memory frequencies are between 200–400 MHz (excluding 400 MHz), the phase is 315°.
- For Low Voltage devices when HP banks are selected for memory interface pins in GUI and the memory frequencies  $\geq$  400 MHz, the phase is 337.5°.
- For Low Voltage devices when HP banks are selected for memory interface pins in GUI and the memory frequencies are between 200–400 MHz (excluding 400 MHz), the phase is 0°.

- When HR banks are selected for memory interface pins in GUI and the memory frequencies  $\geq$  400 MHz, the phase is 337.5°.
  - When HR banks are selected for memory interface pins in GUI and the memory frequencies are between 200–400 MHz (excluding 400 MHz), the phase is 0°.

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 1:1 ratio is not required. The PLL input divider (D) can be any value listed in the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] 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 192](#).



*Figure 1-52:*

The details of the ISERDES/OSERDES connectivity are shown in Figure 1-58, page 142 and Figure 1-60, page 144.

## *Internal (FPGA) Logic Clock*

The internal FPGA logic is clocked by a global clocking resource at a half or quarter frequency of the DDR2 or DDR3 SDRAM clock frequency, which depends on 4:1 or 2:1 mode selected in the MIG tool. 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° phase offset between the write DQS and its associated DQ bits. The PHASER\_OUT also drives the signaling 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 1-58](#) and [Figure 1-60](#).

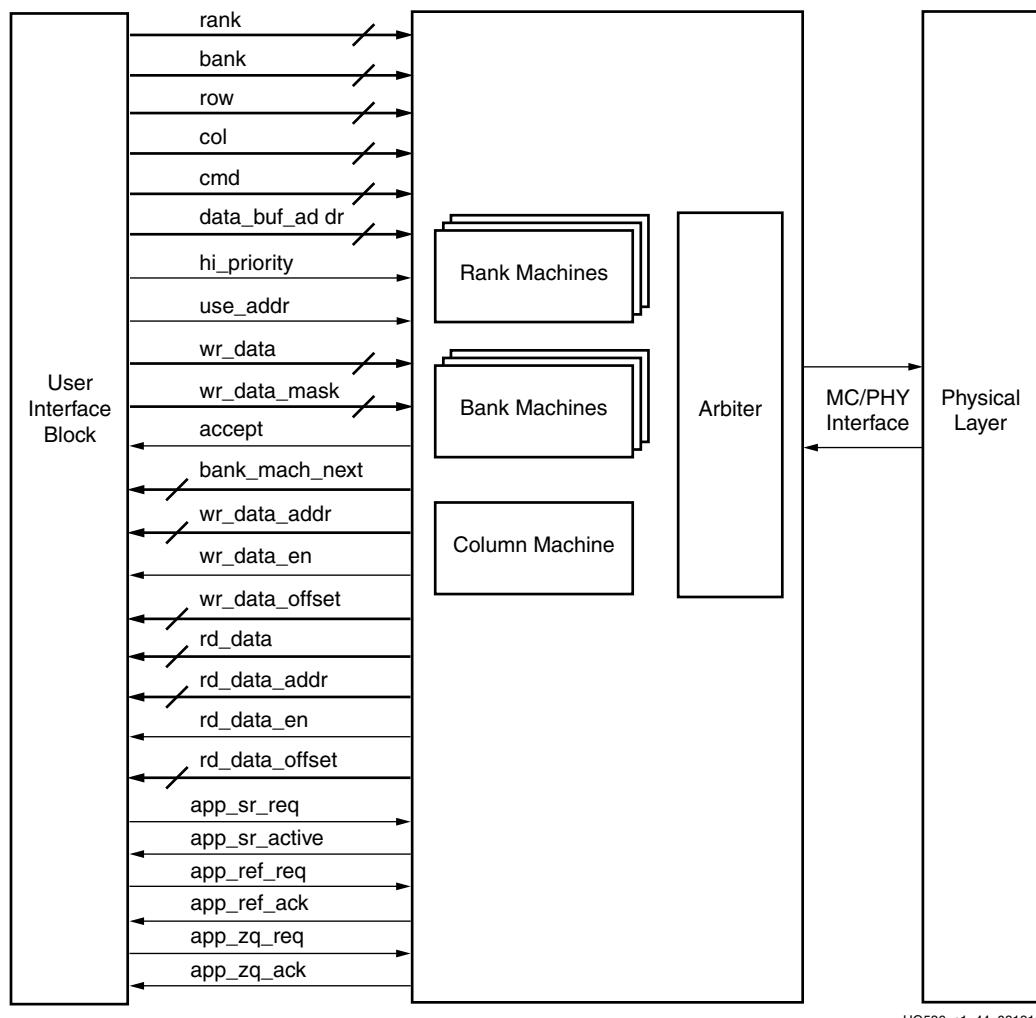
## ***Read Path (Input) I/O Logic Clock***

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 1-60](#).

## ***IDELAY Reference Clock***

You need to always supply a 200 MHz `ref_clk` and then MIG creates the appropriate IDELAYCTRL frequency with an additional MMCM. 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` module. This ensures that the clock is stable before being used.

In the core default configuration, the Memory Controller (MC) resides between the UI block and the physical layer. This is depicted in [Figure 1-53](#).



*Figure 1-53:*

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.

The duration of a bank machine 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 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 your 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.

The controller supports three ordering modes:

- **STRICT** – In this mode the controller always issues commands to the memory in the exact order received at the native interface. This mode can be useful in situations that do not benefit from reordering and the lowest latency is desired. Because the read data comes back in order, the user interface layer might not be needed thus reducing latency. This mode is also useful for debugging.
- **NORM** – In this mode the controller reorders reads but not writes as needed to improve efficiency. All write requests are issued in the request order relative to all other write requests, and requests within a given rank-bank retire in order. This ensures that it is not possible to observe the result of a later write before an earlier write completes.

**Note:** This reordering is only visible at the native interface. The user interface reorders the read requests back into the original request order.

- **RELAXED** – This is the most efficient mode of the controller. Writes and reads can be reordered as needed for maximum efficiency between rank-bank queues. Thus in this mode it is possible to observe the reordering of writes. However, this behavior is not observable at the user interface layer because the requests are retired in order within a rank-bank and the user interface layer returns the read requests in order. Therefore the RELAXED mode is recommended for use with the user interface layer.

**Note:** This option is not selectable in the MIG GUI. To enable, generate the design with the synthesis options "Global" in the **Generate Output Products** settings. After generating the design, the design top-level RTL file should be edited and the ORDERING parameter should be changed to "RELAXED."

## *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, FPGA logic 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 command behavior to determine the optimum setting.

**Note:** The overall read latency of the MIG 7 series DDR3/DDR2 core is dependent on how the Memory Controller is configured, but most critically on the target traffic/access pattern and the number of commands already in the pipeline before the read command is issued. Read latency is measured from the point where the read command is accepted by the user or native interface. Simulation should be run to analyze read latency.

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 1-54](#) shows the ECC block diagram. These blocks are instantiated in the Memory Controller (`mc.v`) module.

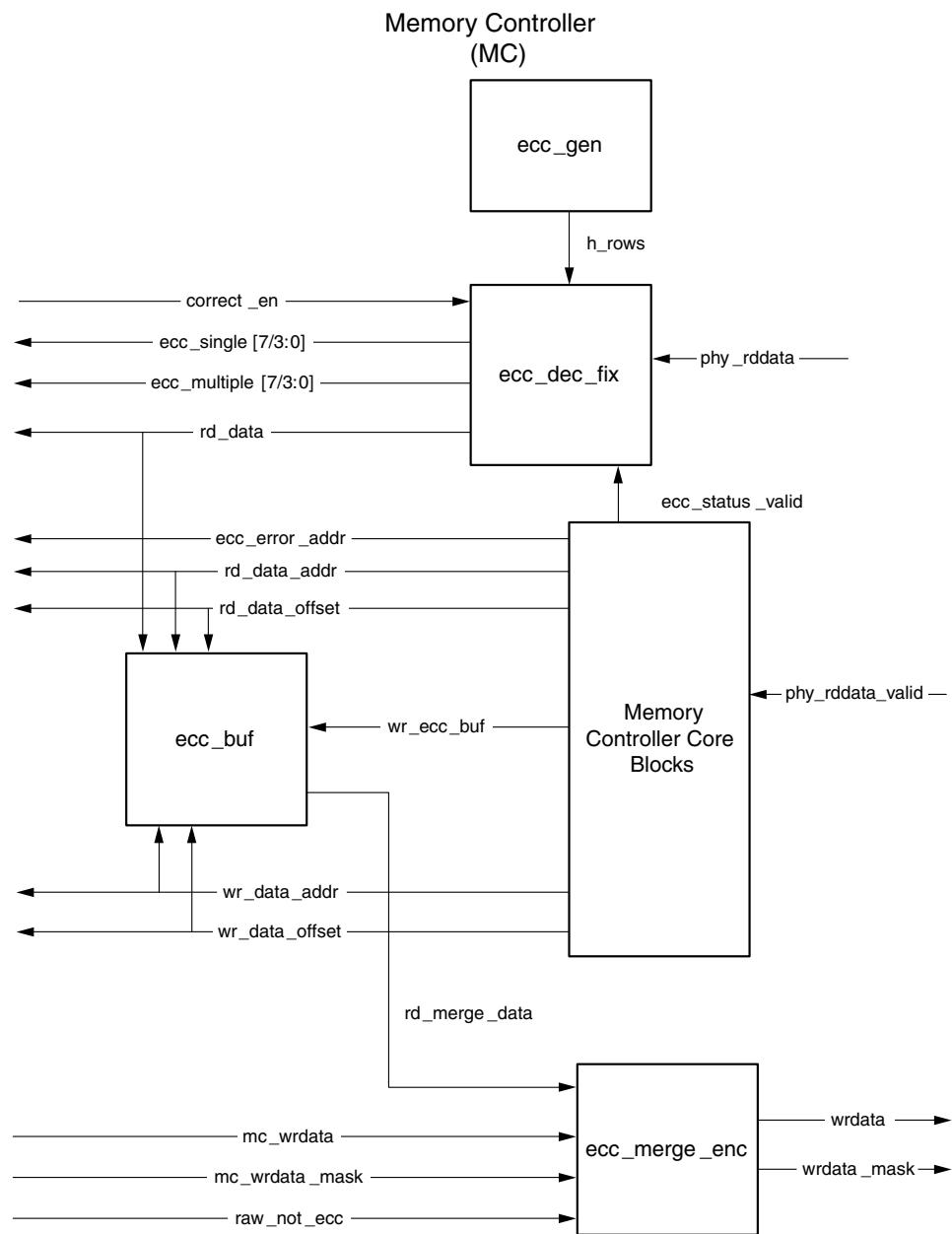


Figure 1-54:

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 1-55](#) shows the available commands when ECC mode is enabled.

**Table 1-55:**

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. Now, 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.




---

**IMPORTANT:** *If performance is important, it is best to avoid the **wr\_bytes** command.*

---

Table 1-56 provides the details of ECC ports at the user interface.

Table 1-56:

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 <b>app_rd_data_valid</b> . The <b>app_ecc_multiple_err</b> 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 four bits wide in 2:1 mode.

Table 1-56:

(Cont'd)

app_ecc_single_err[7:0]	Output	This signal is applicable when ECC is enabled and is valid along with app_rd_data_vali. The app_ecc_single_err signal is non-zero if the read data from the external memory has a single bit error per beat of the read burst.
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 four bits wide in 2:1 mode.
app_wdf_mask[APP_MASK_WIDTH - 1:0]	Input	This signal provides the mask for app_wdf_data[].

**Note:** The MIG generated sim\_tb\_top.v module has error injection logic on DQ[0] bit. Whenever ECC is enabled, the DQ[0] bit is corrupted with error injection. This results in app\_ecc\_single\_err bits toggling for each read data transaction.

### 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. The **app\_raw\_not\_ecc\_i** is seven bits wide (four 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 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.

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 XDC file.

### ***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 1-55](#) 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.

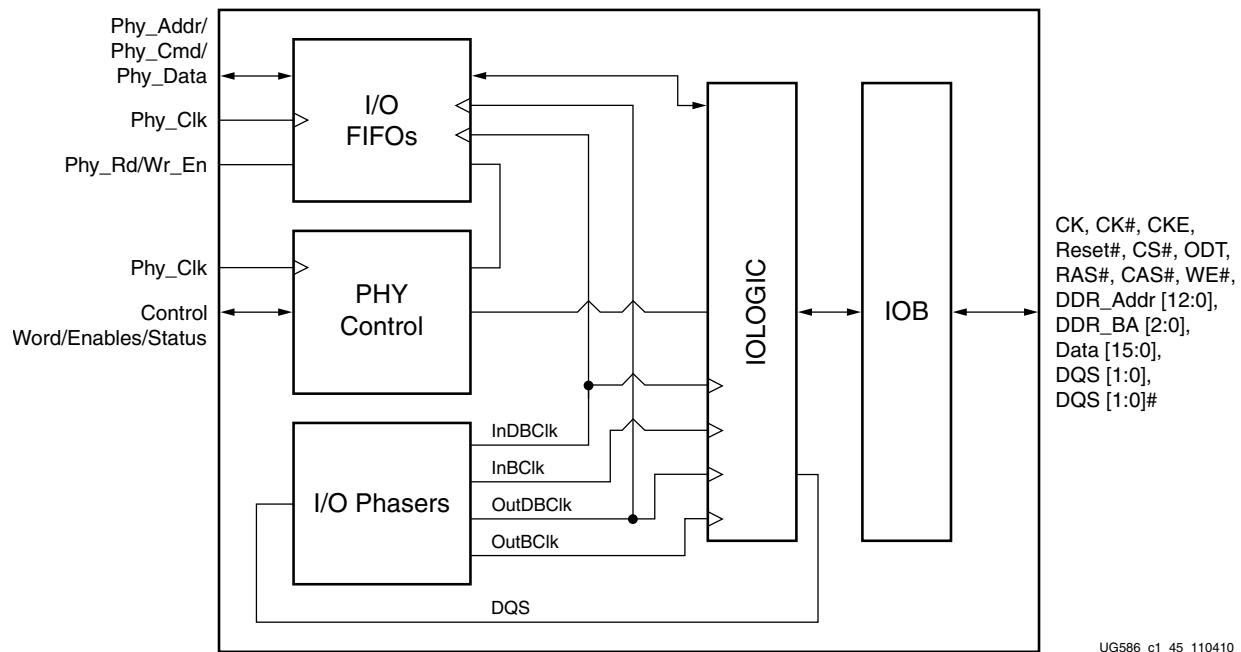
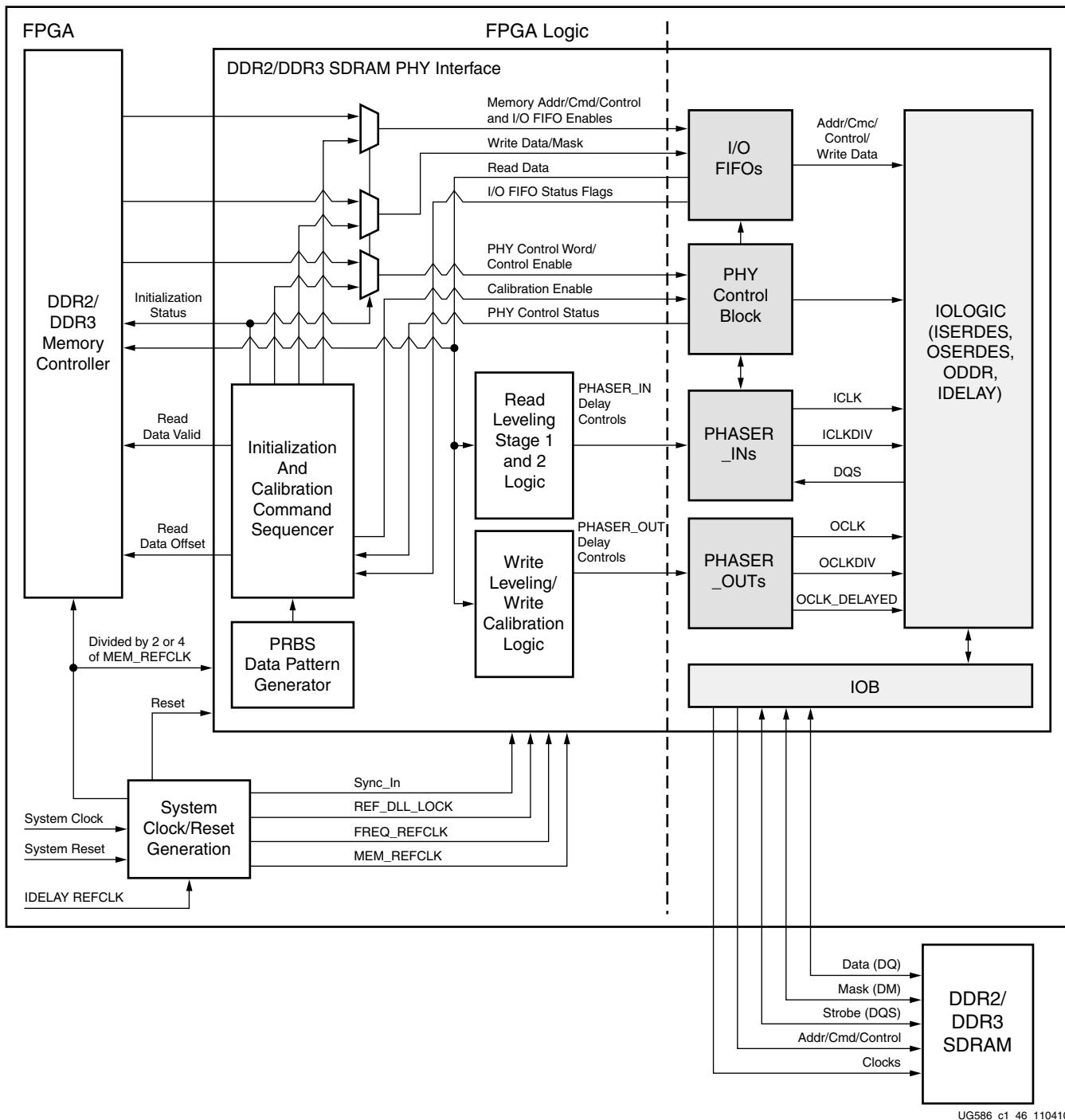


Figure 1-55:

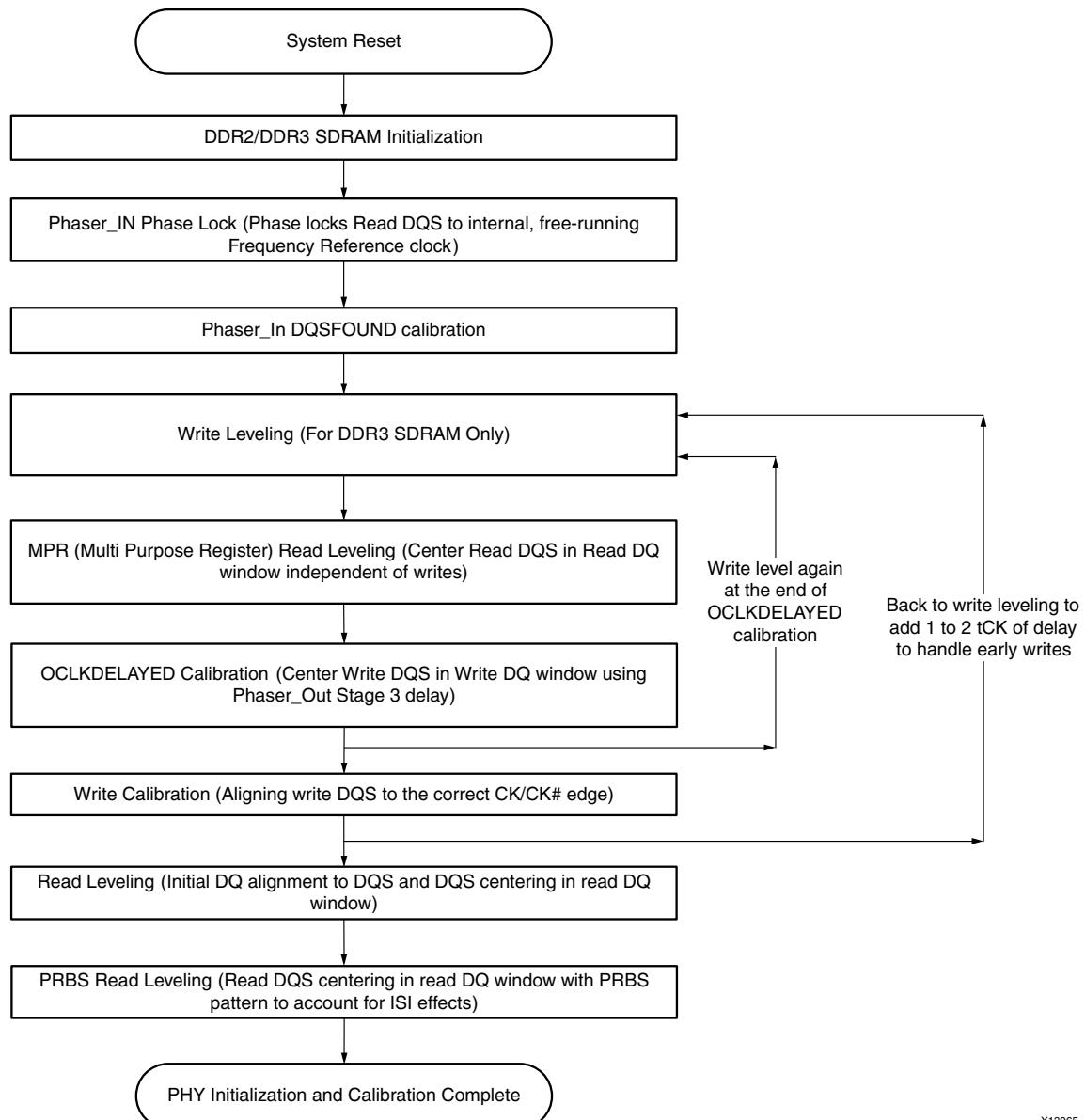
The Memory Controller and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is either a divided by 4 or divided by 2 version of the DDR2 or DDR3 memory clock. A block diagram of the PHY design is shown in [Figure 1-56](#).


*Figure 1-56:*

## 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 1-57 shows the overall flow of memory initialization and the different stages of calibration.



X13065

Figure 1-57:

The calibration stages in [Figure 1-57](#) correspond to these sections:

- [Memory Initialization, page 145](#)
- [PHASER\\_IN Phase Lock, page 146](#)
- [PHASER\\_IN DQSFOUND Calibration, page 146](#)
- [Write Leveling, page 147](#)
- [Multi-Purpose Register Read Leveling, page 150](#)
- [OCLKDELAYED Calibration, page 151](#)
- [Write Calibration, page 153](#)
- [Read Leveling, page 155](#)
- [PRBS Read Leveling, page 158](#)
- [Dynamic Calibration and Periodic Read Behavior, page 158](#)

## ***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.

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 1-57](#). 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 1-57:*

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 1-58</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 anymore 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 1-58](#).

*Table 1-58:*

3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9 9	8 8	7 7	6 6	5 5	4 4	3 3	2 2	1 1	0 0
Act Pre	Event Delay		CAS Slot	Seq	Data Offset			Reser ved	Low Index		Aux_Out		Control Offset		PHY Cmd																

- **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 1-59, page 139](#). These outputs are not used by the DDR2 and DDR3 interfaces generated by the MIG tool; they are set to 0.
- **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.
- **Reserved** – This field must always be set to 2'b00.
- **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 1-59:

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.
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.

The PHY control block has several 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 increased 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 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.

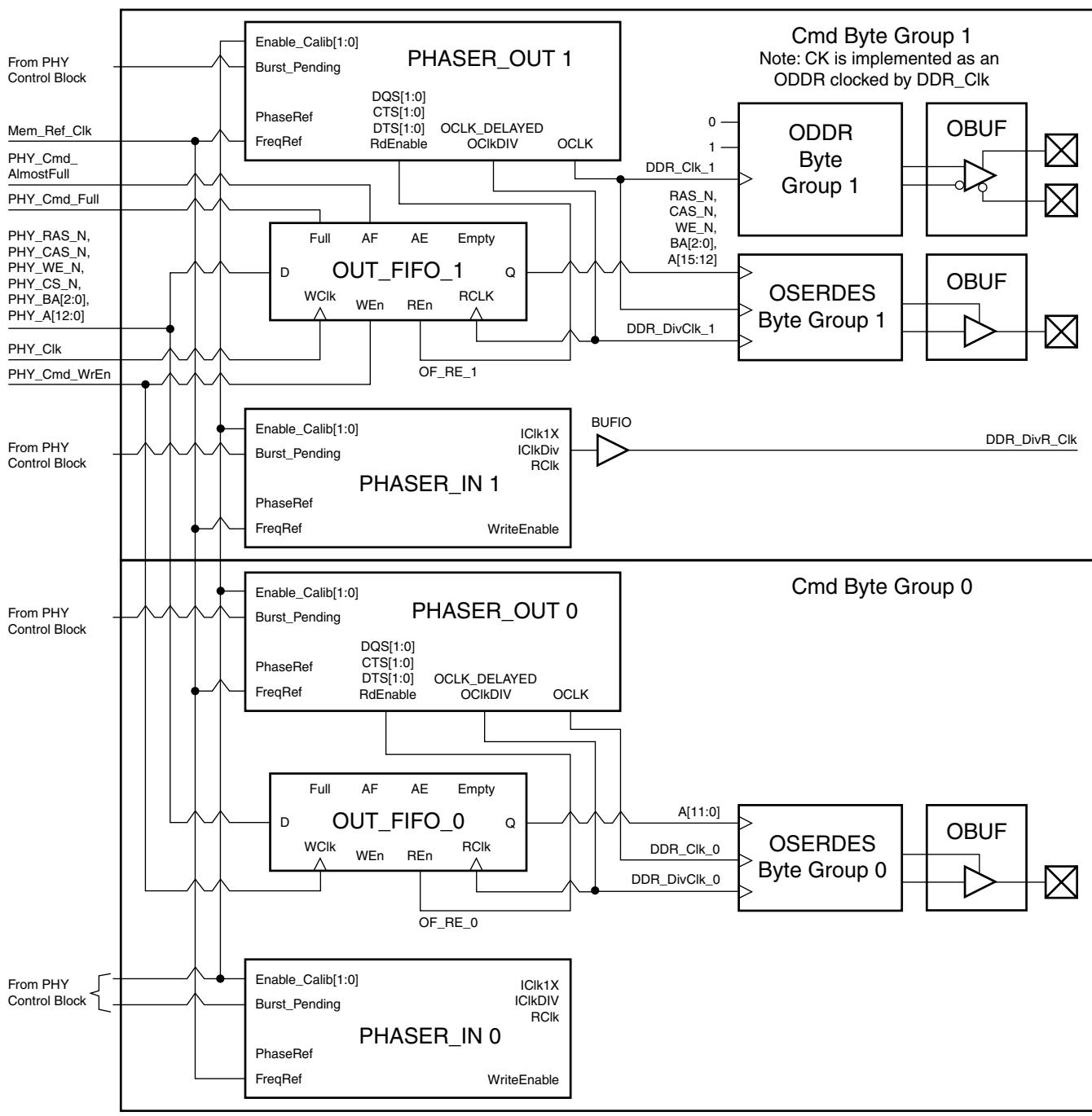
The PHY cmd field is set based on whether the sequence of four commands has either a write, a read, or neither. The PHY cmd field is set to write if there is a write request in the command sequence. It is set to read if there is a read request in the command sequence, and it is set to non-data if there 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

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 1-58 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\_030412

*Figure 1-58:*

The timing diagram of the address/command path from the output of the OUT\_FIFO to the FPGA pins is shown in [Figure 1-59](#).

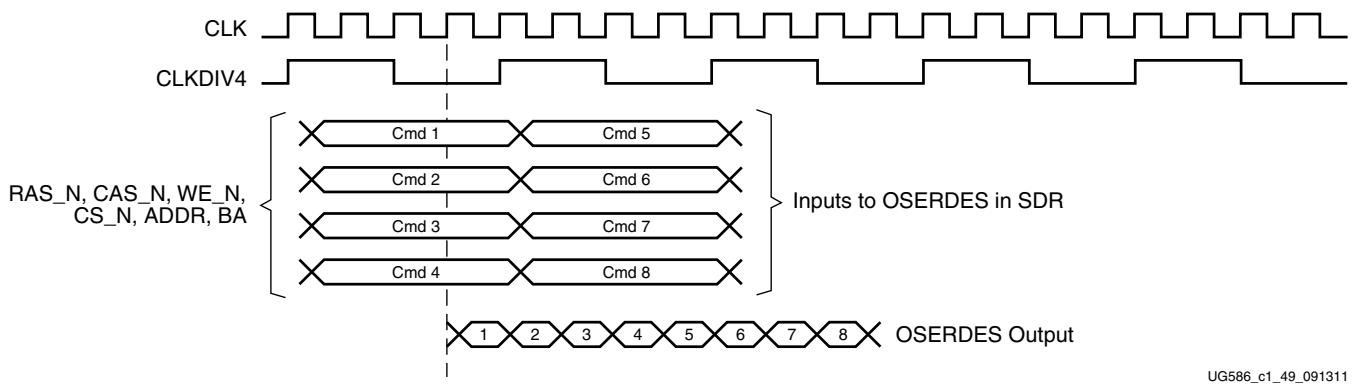


Figure 1-59:

UG586\_c1\_49\_091311

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 1-60](#) shows the block diagram of the datapath.

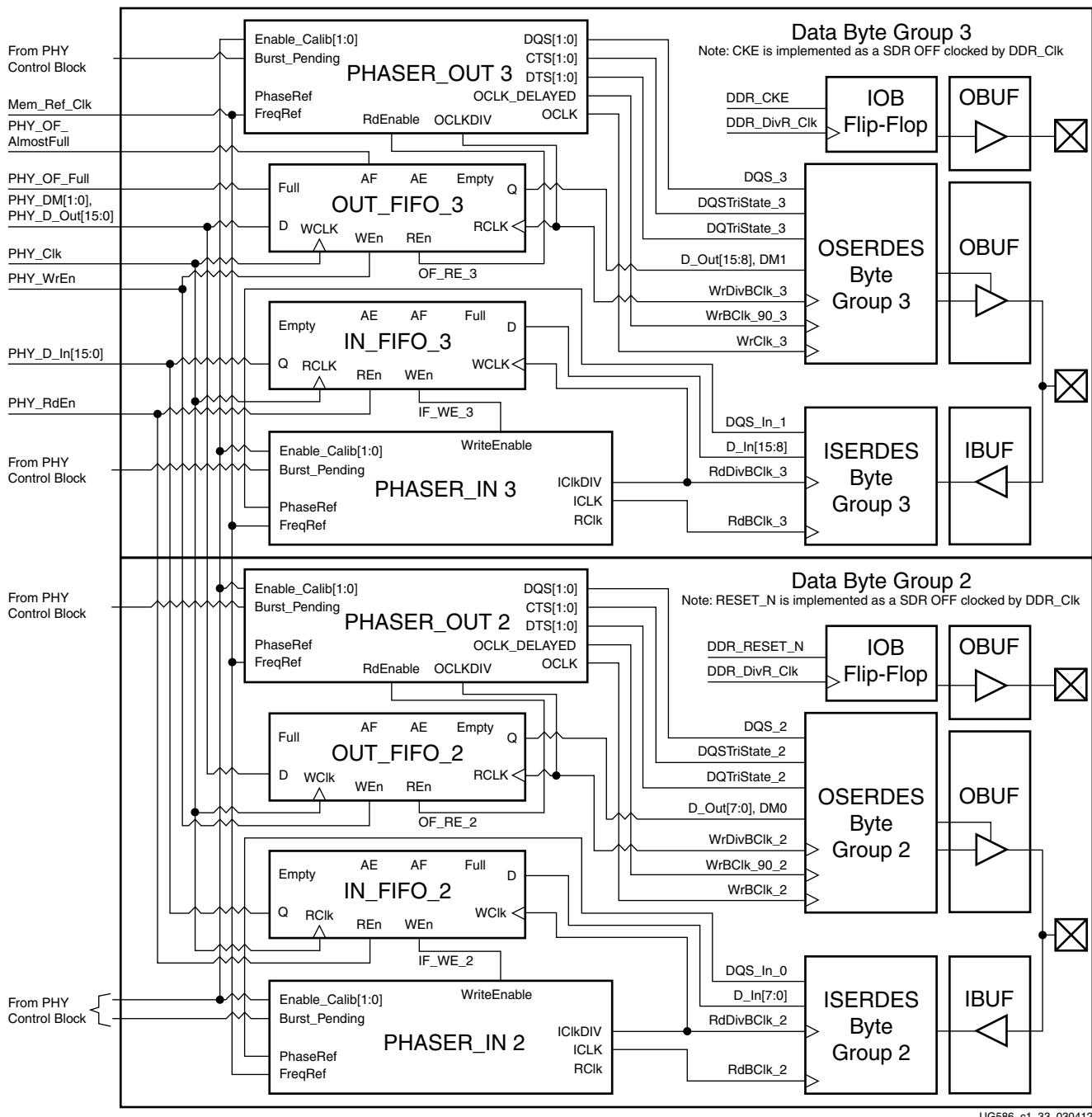


Figure 1-60:

Each IN/OUT\_FIFO has a storage array of memory elements arranged as 10 groups eight bits wide and eight entries deep. During a write, the OUT\_FIFO receives eight 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 eight bits to four 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° 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 1-59](#) 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.

### ***Power-Saving Features***

Designs generated by the MIG tool use the SSTL T\_DCI standards, which save power by turning off the DCI when the FPGA output driver is active. Also, by using the IOBUF\_DCEN (High Performance banks) and IOBUF\_INTERMDISABLE (High Range banks) primitives, designs automatically disable the IBUF when the output is active. The controller uses these primitives to disable both DCI/IN\_TERM and the IBUF when the controller is idle.

For more information on the IOBUF\_DCEN and IOBUF\_INTERMDISABLE primitives, see [7 Series FPGAs SelectIO™ Resources User Guide \(UG471\)](#) [[Ref 2](#)].

### ***Calibration and Initialization Stages***

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 through 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), write recovery for auto precharge (tWR), on-die termination resistor values (RTT\_NOM and RTT\_WR), and output driver strength (OUTPUT\_DRV).

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.

This calibration stage is required to align the different **DQS** groups to the same PHY\_CLK clock edge in an I/O bank. Different **DQS** groups have different skews with respect to each other because of the 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 per I/O bank.

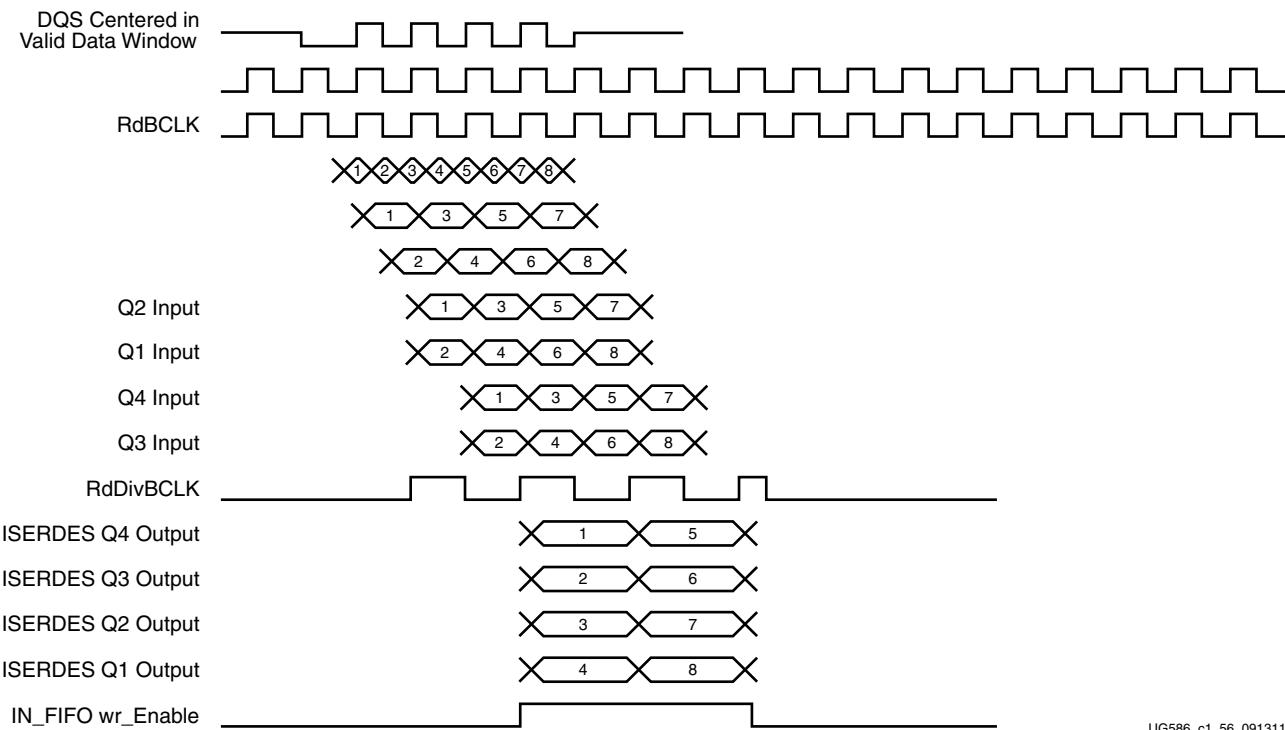
In this stage of calibration, the PHASER\_IN block is in normal operation mode and the calibration logic issues a set of four back-to-back read commands with gaps in between. The **data\_offset** associated with the first read command is not accurate because the round-trip delays are unknown.

For interfaces using HP I/O banks, the **data\_offset** for the first set of read commands is set to CL + 13. The **data\_offset** value for the subsequent set of reads is decreased 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 of the bytes, the **CK** delay stage begins.

For interfaces using HR I/O banks, the **data\_offset** for the first set of read commands is set to CL - 2. The **data\_offset** value for the subsequent set of reads is increased 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 of the bytes, the **CK** delay stage begins.

In the **CK** delay stage, the PHASER\_OUT stage 2 delay tap is increased one at a time starting from 0 to 63 for **CK**/Address/Command/Control byte lanes. This effectively moves where the read **DQS** preamble begins and causes the **DQSFOUND** to fail. Any one **DQSFOUND** failure of the entire interface is considered a failure. A passing window is determined by recording the taps where the **DQSFOUND** failures occur. The final tap value for **CK**/Address/Command/Control byte lanes is set to the center of the passing window. If no failing edges are found the final tap is set to 32.

Each byte group can be read out of the IN\_FIFO on different **PHY\_clk** cycles due to fly-by routing 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 1-61](#) shows the read data capture timing diagram.



UG586\_c1\_56\_091311

Figure 1-61:

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 1-62](#) shows the write leveling block diagram.

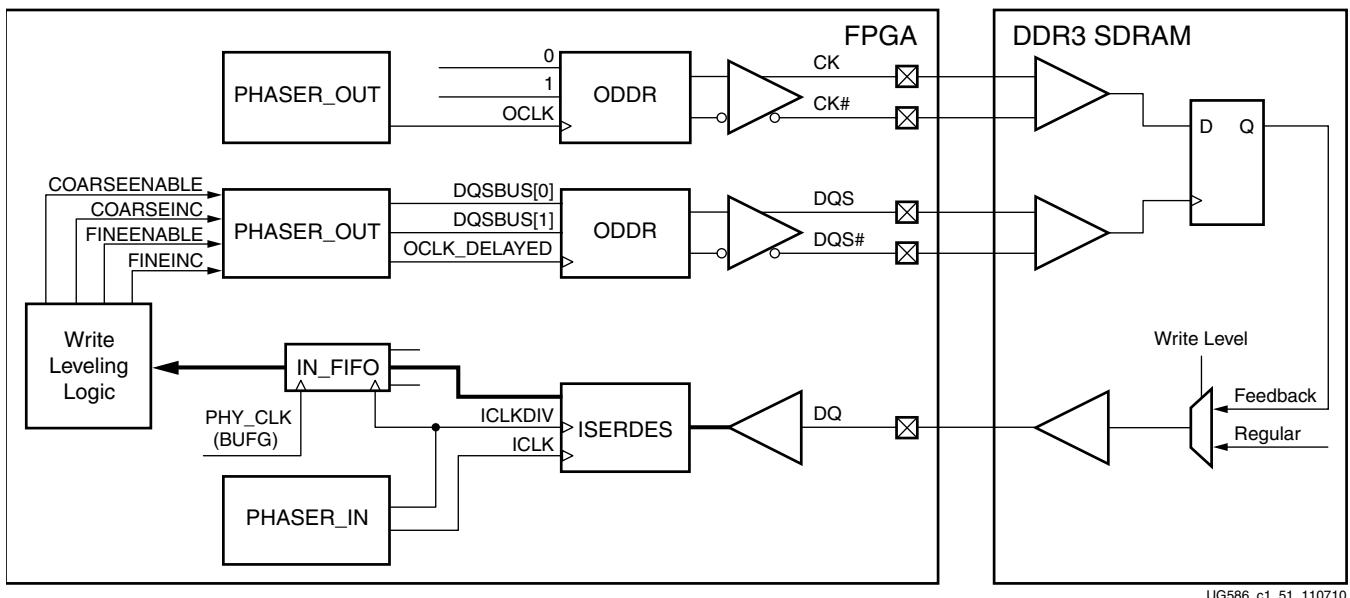


Figure 1-62:

The timing diagram for write leveling is shown in Figure 1-63. 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.

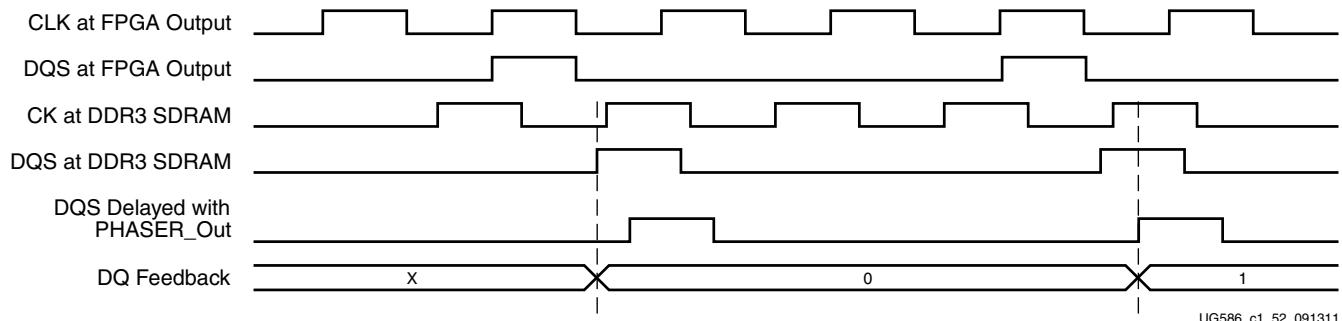


Figure 1-63:

Figure 1-64 shows that the worst-case delay required during write leveling can be one tCK (DDR3 SDRAM clock period).

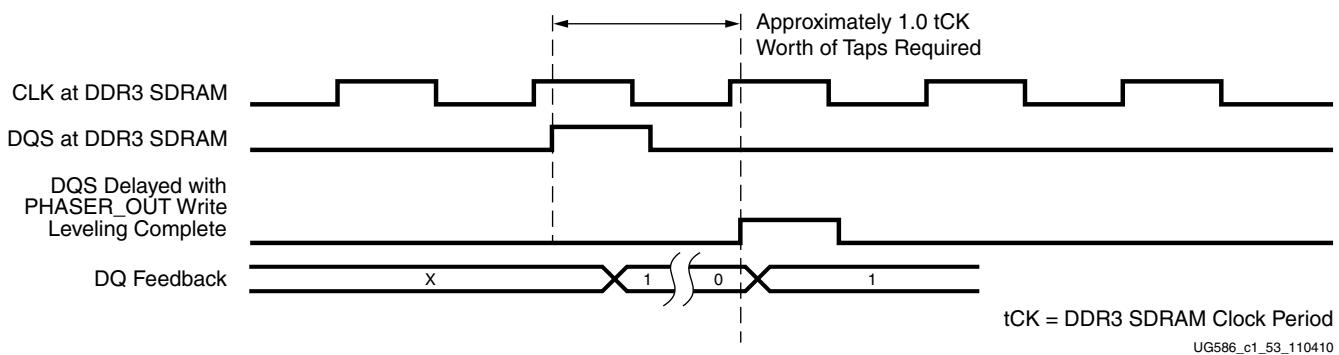


Figure 1-64:

### Implementation Details

The `write_calib_n` signal indicating the start of write leveling mode is input to the PHY control block after `tWLQSEN` 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 1-65 shows the flow diagram of the sequence of commands during write leveling. The `PHASER_OUT` fine phase shift taps are increased 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 three means that the sampled data value was constant for three 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 `DQSs` in all ranks.

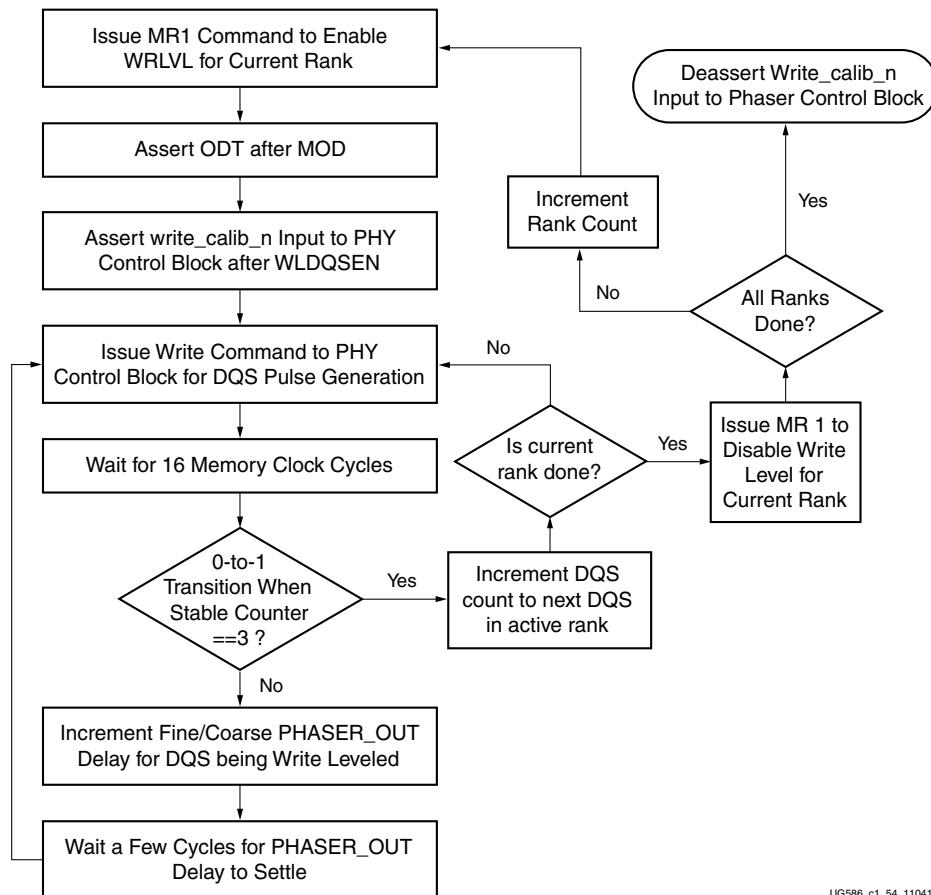


Figure 1-65:

At this stage of calibration, the write DQS is not centered in the write DQ window nor is the read DQS centered in the read DQ window. The Multi-Purpose Register (MPR) is used to center the read DQS in the read DQ window. The MPR has a predefined **01010101** pattern that is read back during this stage of calibration. Read DQS centering is required for the next stage of calibration.

MPR read leveling is performed on a per byte basis and it is a two step process.

- The first step is to delay all the DQ bits in a byte using IDELAY taps by monitoring Bit[0] in that byte. The DQ bits are moved to place the first valid rising edge data after the first rising edge of DQS.
- The second step is to sweep DQS across the entire byte window using PHASER\_IN fine taps to detect two edges. The entire DQ byte lane (Bits[7:0]) is monitored while sweeping the byte window to find the aggregate eye. Note that although the JEDEC standard states it is only required to send the MPR pattern on Bit[0] of a byte, all vendors tested sent the MPR pattern on the entire byte.
- Minimum data window (MIN\_EYE\_SIZE) must be met for two edges to be found.

- When second edge is not found, second\_edge\_taps are set to zero. However, the algorithm computes the midpoint of the data window using 63 as the second edge tap position because 63 is the maximum fine tap value and no edge was detected.

Write **DQS** is centered in the write **DQ** window using the PHASER\_OUT stage 3 delay in this stage of calibration. The starting stage 3 tap value ranges from 28 to 34 depending on the memory clock frequency. There are three substages in this calibration stage performed on a per byte basis:

- Stage 3 tap limit determination
- Detection of write **DQ** valid window edges
- Write **DQS** centering in the write data valid window

The DDR3 SDRAM JEDEC specification requires the write **DQS** to be within  $\pm 90^\circ$  of **CK** defined by the **t<sub>DQSS</sub>** specification. To avoid **t<sub>DQSS</sub>** violation during the edge detection, stage left and right limits of stage 3 tap movement are determined in this substage. These limits are calibrated using MMCM phase shift taps to optimize the calibration center point. The start of this substage is triggered by **lim\_start**. The output signals **lim2local\_stg3\_left\_lim** and **lim2local\_stg3\_right\_lim** validated by **lim\_done** are input to the edge detection substage.

In the edge detection substage, the first step is decrementing stage 3 taps until either one or more edges are found or the tap value reaches **lim2local\_stg3\_left\_lim**. The stage 3 taps are then increased until one or more edges are found or the tap value reaches **lim2local\_stg3\_right\_lim**.

At the end of edge detection stage, the following signals indicate which edges are detected. [Figure 1-66](#) shows the names associated with the different edges.

- f2z** – If asserted, this indicates that the left-edge of the rise window was detected and it validates fuzz2zero as the tap value of the left-edge of the rise window.
- z2f** – If asserted, this indicates that the right-edge of the rise window is detected and it validates zero2fuzz as the tap value of the right-edge of the rise window.
- f2o** – If asserted, this indicates that the left-edge of the fall window was detected and it validates fuzz2oneeighty as the tap value of the left-edge of the fall window.
- o2f** – If asserted, this indicates that the right-edge of the fall window was detected and it validates oneeighty2fuzz as the tap value of the right-edge of the fall window.

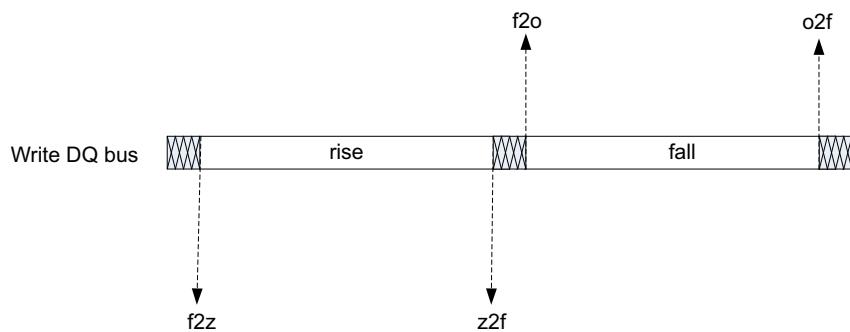


Figure 1-66:

The stage 3 start tap value places the **DQS** anywhere between 90° and 225° with respect to **DQ**. The number of edges detected depends on the write level taps value after the initial write leveling and the starting **DQS** position with respect to **DQ**. If **DQS** starts in the rise **DQ** window between 90° and 180°, then the following can occur:

- Either f2z, z2f, and f2o edges can be detected
- Or f2z and z2f edges of the rise window can be detected
- Or only f2z edge can be detected or only z2f edge can be detected

If DQS starts in the noise/jitter region around 180°, then the following can occur:

- Either z2f, f2o, and o2f edges can be detected
- Or f2z, z2f and f2o edges can be detected
- Or z2f and f2o edges of the noise can be detected

Finally, if **DQS** starts in the fall window between 180° and 225°, then the following can occur:

- Either z2f, f2o, and o2f edges can be detected
- Or f2o and o2f edges can be detected
- Or only f2o edge can be detected

**Table 1-60** describes the different starting scenarios of write **DQS** with respect to **DQ**, the possible edges that can be detected, and the equation used to determine the center value.

Table 1-60:

Case 1	Found	Found	Found	Not Found	$(\text{fuzz2zero} + \text{zero2fuzz})/2$
Case 2	Found	Found	Not Found	Not Found	$(\text{fuzz2zero} + \text{zero2fuzz})/2$

Table 1-60:

(Cont'd)

Case 3	Found	Not Found	Not Found	Not Found	(fuzz2zero + lim2ocal_stg3_right_lim)/2
Case 4	Not Found	Found	Not Found	Not Found	(lim2ocal_stg3_left_lim + zero2fuzz)/2
<hr/>					
Case 1	Found	Found	Found	Not Found	(fuzz2zero + zero2fuzz)/2
Case 2	Not Found	Found	Found	Found	(zero2fuzz + fuzz2oneeighty)/2 - 90°
Case 3	Not Found	Found	Found	Not Found	(zero2fuzz + fuzz2oneeighty)/2 - 90°
<hr/>					
Case 1	Not Found	Found	Found	Found	(zero2fuzz + fuzz2oneeighty)/2 - 90°
Case 2	Not Found	Not Found	Found	Found	(fuzz2oneeighty + oneeighty2fuzz)/2 - 180°
Case 3	Not Found	Not Found	Found	Not Found	(fuzz2oneeighty + lim2ocal_stg3_right_lim)/2 - 180°

During the centering substage the write **DQS** is centered in the write **DQ** window based on the edges found during the edge detection stage. At the end of this stage, write **DQS** should be centered in the write **DQ** window. **DQS** to **CK** are not correct therefore write leveling is performed at the end of this stage of calibration.

With every stage 3 tap decrease, the stage 2 taps are increased by 2 to maintain the **DQS** to **CK** relationship established during write leveling. Similarly, with every stage 3 tap increment, the stage 2 taps are decreased by 2. If stage 2 taps reach 0 or 63, stage 3 tap increment/decrement is allowed to proceed up to the left and right limit values to avoid **t<sub>DQSS</sub>** violation. At the end of this stage of calibration, write leveling is redone to align **DQS** and **CK** using stage 2 taps.

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 1-67](#) shows several different scenarios based on the initial phase relationship between **DQS** and **CK** for a UDIMM or RDIMM interface.

Figure 1-68 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.

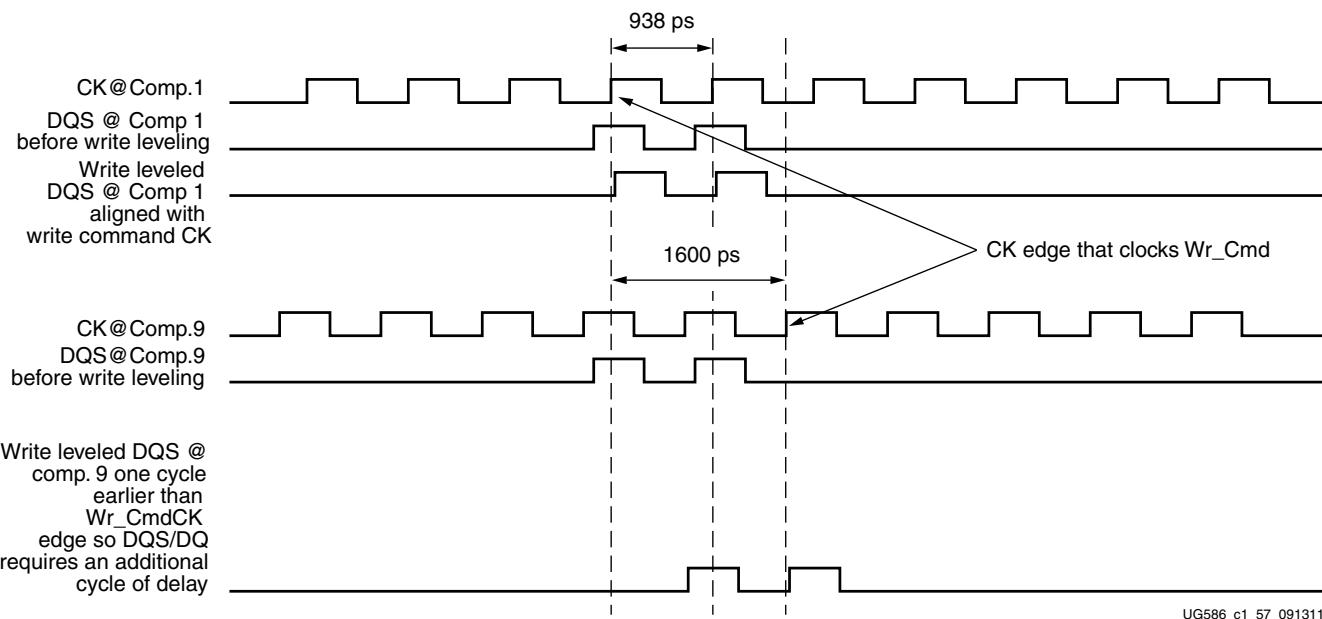


Figure 1-67:

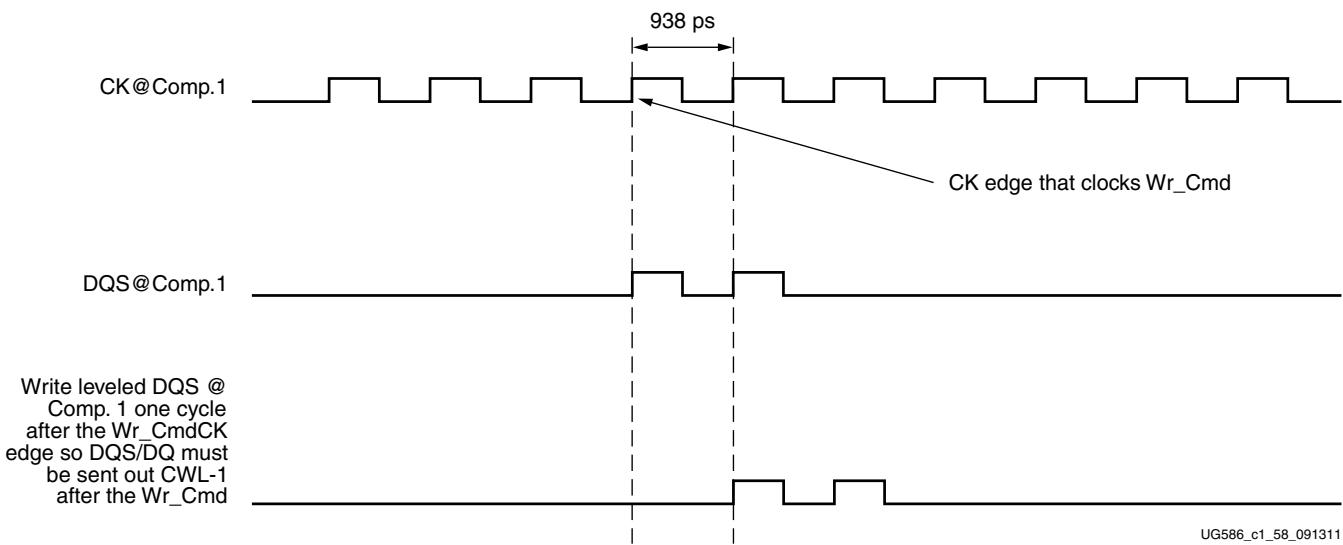
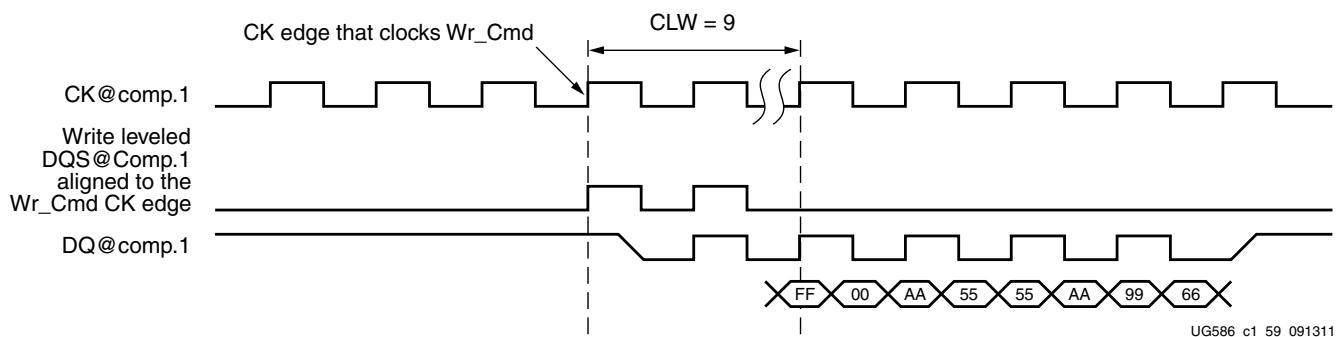


Figure 1-68:

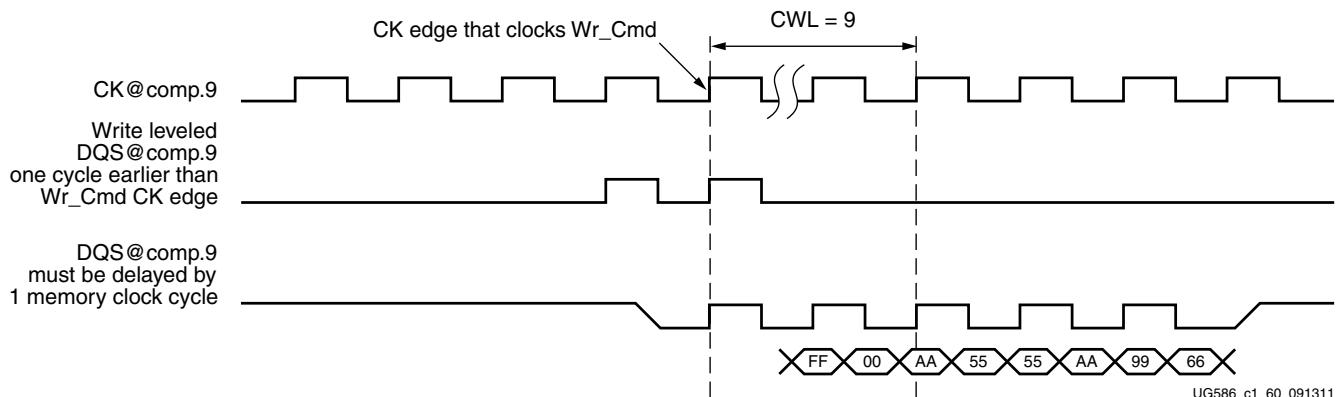
The PHASER\_OUT fine and coarse delay provides  $1 t_{CK}$  worth of delay for write leveling. The additional clock cycle of delay required to align to the correct **CK** edge is achieved using the coarse delay line. If the total delay required is over one clock cycle, the **div\_cycle\_delay** input 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 1-69](#). 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 1-70](#). 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 1-69:*



*Figure 1-70:*

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 **DQS** 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 1-71](#) shows the timing diagram for **DQS** center alignment in the data valid window.

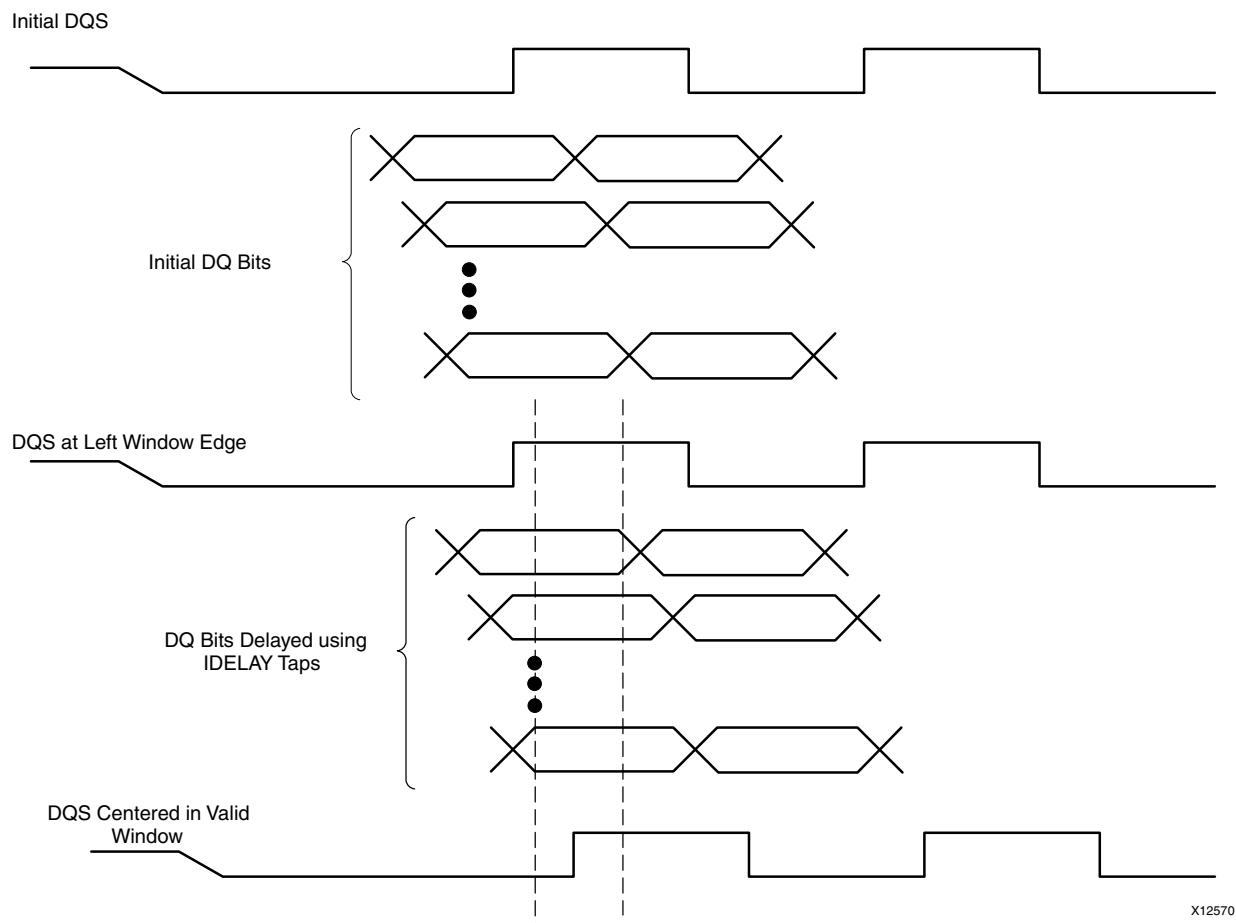


Figure 1-71:

This stage of read calibration follows the Read Leveling calibration stage. The **DQS** PHASER\_IN fine tap setting determined during the Read Leveling calibration stage is used as the starting point for this stage of calibration. The PRBS read leveling stage does not change the **DQ** IDELAY tap settings determined during the Read Leveling calibration stage.

Complex pattern pre-loaded in a block RAM is written to the DDR3 SDRAM at the start of this calibration stage. This sequence is then read back non-continuously to determine the read data valid window. For each phaser in tap, read back happens multiple times determined by internal sample count setting. The algorithm starts at the **DQS** PHASER\_IN fine tap setting determined during the Read Leveling calibration stage (initial tap value) and decrements one tap at time until a data mismatch is found when comparing read data with the expected data. Per-bit deskew scheme using FINEDELAY is also added to increase the read valid margin. FINEDELAY is a part of IDELAY primitive. The algorithm starts edge detection at PHASER\_IN tap 0. It increments until it detects the valid pattern and records the tap as left edge. The PHASER\_IN taps are further increased until a data pattern mismatch is found or tap value is 63 and this tap is recorded as right edge. The algorithm then computes the center of the read data valid window based on the detected edges.

The PHASER\_IN performs two dynamic adjustments during reads. The first is within the PHASER\_IN DLL which needs to see **DQS** edges to keep the free-running frequency reference clock phase align locked to the associated read **DQS**. This dynamic adjustment only looks at the **DQS** edges and makes adjustments as required. The internal clock is used at the end of the burst when there are no more **DQS** edges, but clocks are needed to get the final data through the ISERDES.

The second dynamic adjustment is performed within the PHASER\_IN to fine tune the position of the **DQS** preamble for the subsequent read. This dynamic adjustment only looks for the **DQS** preamble. It is needed to account for drift in the system which can move the **DQS** with respect to the internal clock.

Both of these PHASER\_IN dynamic adjustments require periodic reads to ensure the PHASER\_IN is continually adjusted and ready for reads. Because of this, the MIG 7 series DDR2/DDR3 controller sends periodic reads every 1  $\mu$ s when the bus is idle or performing writes. The PHASER\_IN only requires read **DQS**. Therefore, if reads are being performed as requested from the user interface, the controller does not send the periodic reads.

When the controller is writing and the 1  $\mu$ s periodic reads are due, the reads are sent to the address of the next read/write in the queue. When the controller is idle and no reads or writes are requested, the periodic reads use the last address accessed. If this address has been closed, an activate is required. Two back-to-back BL8 reads are required for the dynamic alignment.

All of the dynamic adjustment is hard logic. However, the periodic reads sent to look at **DQS** is soft logic controlled by the MIG 7 series DDR2/DDR3 controller.



The second module, `ddr_phy_tempmon`, resides in the top-level calibration module, `calib_top`. It receives the `device_temp[11:0]` from the `tempmon` module and an enable signal from the Memory Controller. The enable signal is set by the Memory Controller whenever a REF or ZQ command has been sent to the DRAM and all pending transactions have cleared the `DQ` bus. The temperature value is sampled on the clock when the enable transitions from Low to High.

User designs utilizing the PHY-only design must drive the `tempmon_sample_en` input every time a ZQ or REF is sent. It should be brought High after all pending reads have been received through the ISERDES and held until the REF or ZQ has completed and an ACT is ready to be sent. After calibration has completed and the enable signal is set, the `ddr_phy_tempmon` samples the `device_temp[11:0]` bus and establishes a baseline temperature.

After each subsequent enable, the current temperature is compared to the baseline temperature. If the temperature change is sufficient, the module adjusts the PHASER\_IN fine delay to mitigate temperature drift. This process continues throughout normal operation.

### ***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 1-61](#) describes the Memory Controller to PHY interface signals. These signals are synchronous to the FPGA logic clock.

*Table 1-61:*

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.

Table 1-61:

(Cont'd)

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.
mmcm_ps_clk	1	Input	-	MMCM phase shifted clock used in OCLKDELAYED calibration stage to optimize calibration center point.
poc_sample_pd	1	Input	-	Input to phase detector in OCLKDELAYED calibration logic used for optimization of center point.
iddr_rst	1	Input	Active-High	Reset input to phase detector in OCLKDELAYED calibration logic.
psen	1	Output	Active-High	When psen is asserted for one mmcm_ps_clk clock period, a phase shift increment or decrement is initiated.
psincdec	1	Output	Active-High	A High on psincdec initiates a phase increment by 1/56th of the VCO period. A Low on psincdec initiates a phase decrement by 1/56th of the VCO period.
psdone	1	Input	Active-High	The MMCM asserted this signal for one mmcm_ps_clk period when phase shift is completed.
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_odt	[1:0]	Input	-	mc_odt [1:0] is the ODT driven by the controller based on the RTT_NOM and RTT_WR values. This signal is valid when the CKE_ODT_AUX parameter is set to FALSE.

Table 1-61:

(Cont'd)

mc_cke	[nCK_PER_CLK – 1:0]	Input	-	mc_cke [nCK_PER_CLK – 1:0] is the CKE associated with the DRAM interface. This signal is valid when the CKE_ODT_AUX parameter is set to FALSE.
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-High	This signal is the write enable input of the address/command OUT_FIFOS.
mc_ctl_wren	1	Input	Active-High	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: <b>0x04</b> : Non-data command (No column command in the sequence of commands) <b>0x01</b> : Write command <b>0x03</b> : Read command
mc_data_offset	[5:0]	Input	-	This signal is used for PHY_Ctl_Wd configuration: <b>0x00</b> : Non-data command (No column command in the sequence of commands) CWL + COL cmd position + 2 (for nCK_PER_CLK = 4) or CWL + COL cmd position - 2 (for nCK_PER_CLK = 2): 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.

Table 1-61:

(Cont'd)

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.
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.
- 

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 181](#) for supported configuration parameters.

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.

**Note:** For a multi-ported memory interface or an interface that is sending requests faster than the MIG can consume, putting a packet (store and forward) FIFO on the input side of the user logic side of the crossbar is necessary. This allows it to buffer the requests and grants bursts to come out as soon as it is ready.

The AXI4 slave interface follows the AXI4 memory-mapped slave protocol specification as described in the Arm AMBA open specifications. See this specification [\[Ref 4\]](#) for the signaling details of the AXI4 slave interface.

The AXI address from the AXI master is a TRUE byte address. The AXI shim converts the address from the AXI master to the memory based on AXI SIZE and memory data width. The LSBs of the AXI byte address are masked to 0, depending on the data width of the memory array. If the memory array is 64 bits (8 bytes) wide, AXI address[2:0] are ignored and treated as 0. If the memory array is 16 bits (2 bytes) wide, AXI address[0] is ignored and treated as 0.

DDR3 DRAM is accessed in blocks of eight DRAM words for a burst length of 8. The UI data port is as wide as eight DRAM words for 4:1 PHY to Memory Controller (MC) clock ratio mode and four DRAM words for 2:1 PHY to MC clock ratio.

Table 1-62:

4:1	64	8	A[7:0]
	128	16	A[7:1], 1'b0
	256	32	A[7:2], 2'b00
	512	64	A[7:3], 3'b000

*Table 1-62:*

*(Cont'd)*

2:1	32	8	A[7:0]
	64	16	A[7:1], 1'b0
	128	32	A[7:2], 2'b00
	256	64	A[7:3], 3'b000

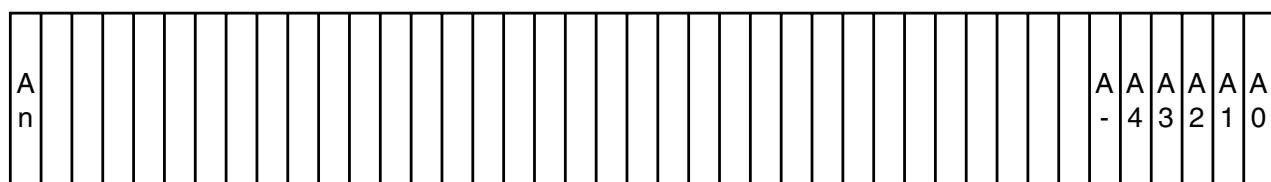
When the data width on the User Interface side is wider than that on the AXI Interface side, upsizing is performed in the AXI Shim interface. Data packing is performed for INCR and WRAP bursts.

In the resulting transaction issued to the user interface side, the number of data beats is reduced accordingly:

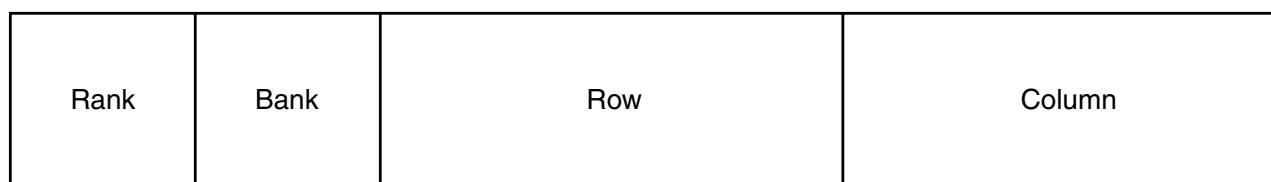
- For writes, data merging occurs.
  - For reads, data serialization occurs.

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 1-72](#) and [Figure 1-73](#).

### User Address



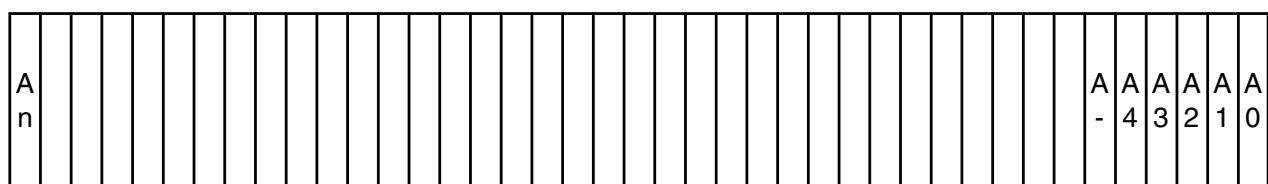
## Memory



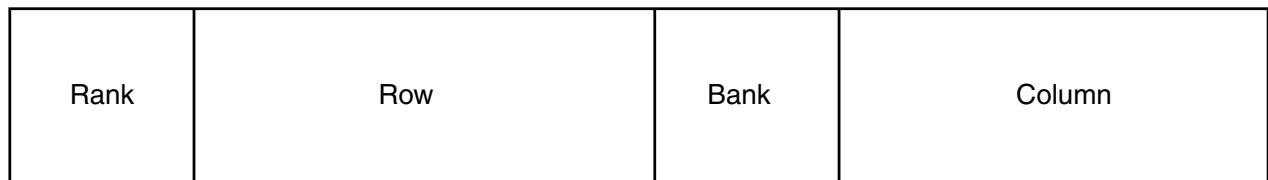
UG586 c1 61 091410

*Figure 1-72:*

User Address



Memory



UG586\_c1\_61a\_012411

*Figure 1-73:*

[Figure 1-72](#) and [Figure 1-73](#) show that the address map is controlled by the string parameter `MEM_ADDR_ORDER`. This parameter can take the following values:

- **BANK\_ROW\_COLUMN** – Address map is as shown in [Figure 1-72](#).
- **ROW\_BANK\_COLUMN** – Address map is as shown in [Figure 1-73](#).
- **TG\_TEST** – Address map is used for testing purpose only. It enables the address remap to test address access to different portions of the DRAM. It remaps the address as explained in the following examples. The remap is done within the UI portion of the controller.

**Note:** The row width, column width, and bank width value settings are assumed for the following examples:

- **Row Width** – 15
- **Bank Width** – 3
- **Column Width** – 10

**Example (1)** – When the selected option in the MIG GUI is `BANK_ROW_COLUMN` and the address to the controller is mapped accordingly.

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
B2	B1	B0	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	RO	C9	C8	C7	C6	C5	C4	C3	C2	C1	CO

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
RO	C9	C8	R4	R3	B2	B1	BO	R14	R13	R12	R11	R10	R9	R8	C7	C6	C5	R2	R1	R7	R6	R5	C4	C3	C2	C1	CO

**Example (2)** – When the selected option in the MIG GUI is ROW\_BANK\_COLUMN and the address to the controller is mapped accordingly.

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
R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	RO	B2	B1	BO	C9	C8	C7	C6	C5	C4	C3	C2	C1	CO
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
RO	C9	C8	R4	R3	B2	B1	BO	R14	R13	R12	R11	R10	R9	R8	C7	C6	C5	R2	R1	R7	R6	R5	C4	C3	C2	C1	CO

### *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 1-74.

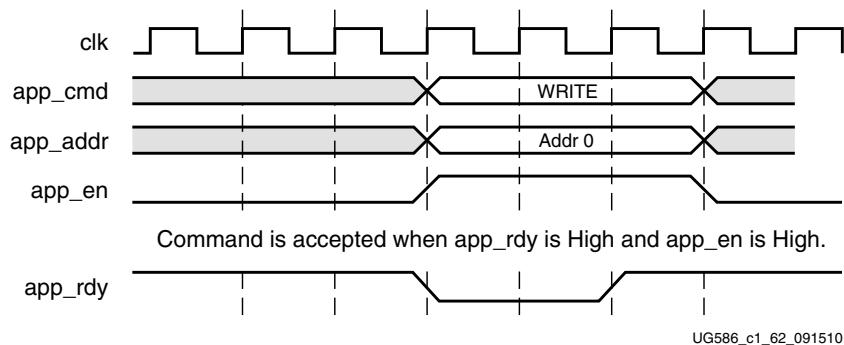


Figure 1-74:

A non back-to-back write command can be issued as shown in [Figure 1-75](#). 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.

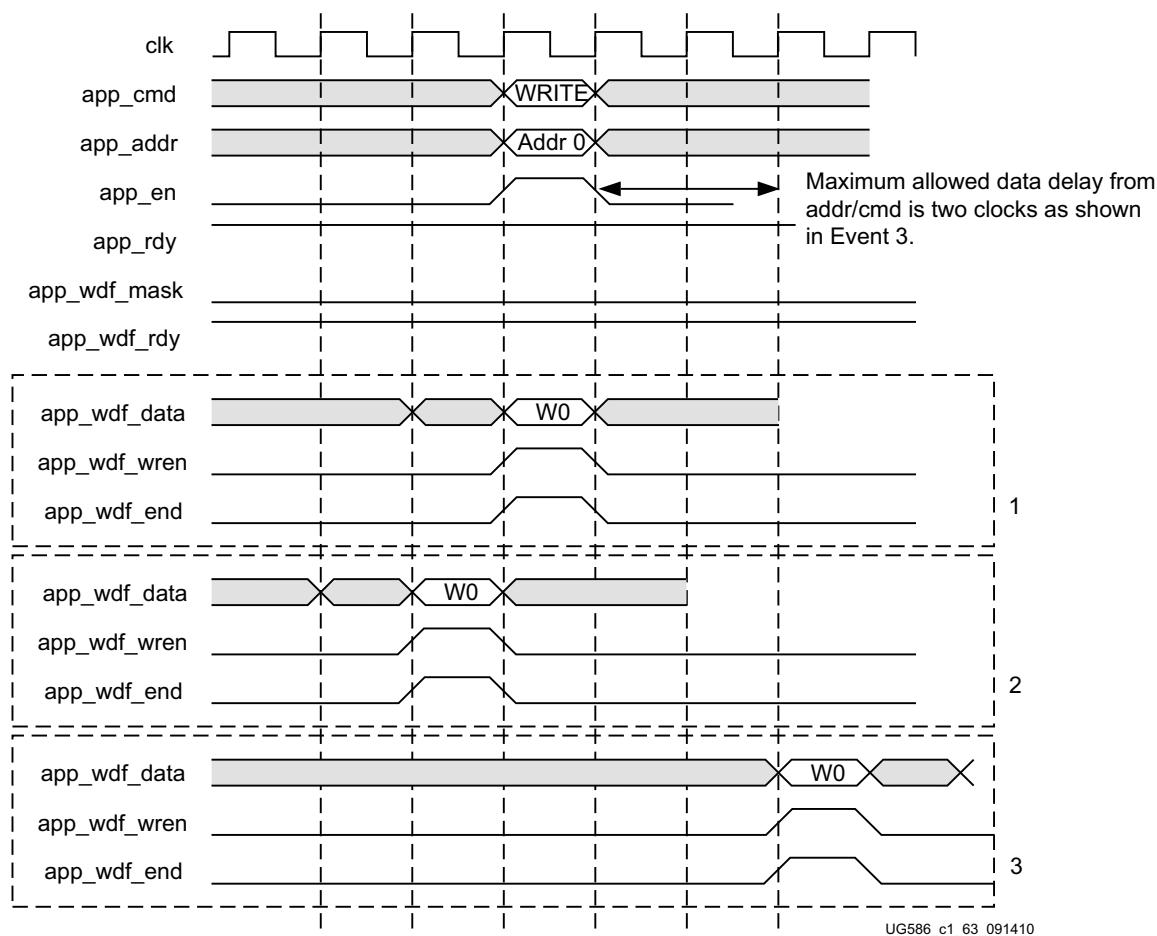
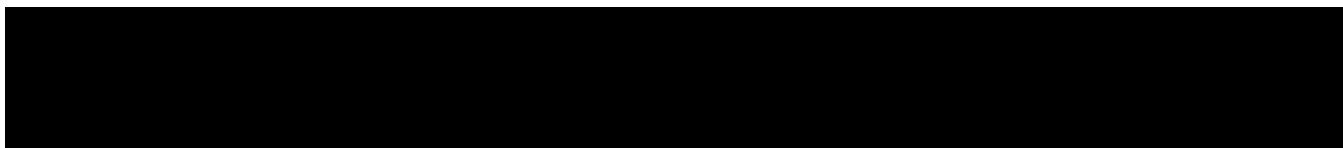


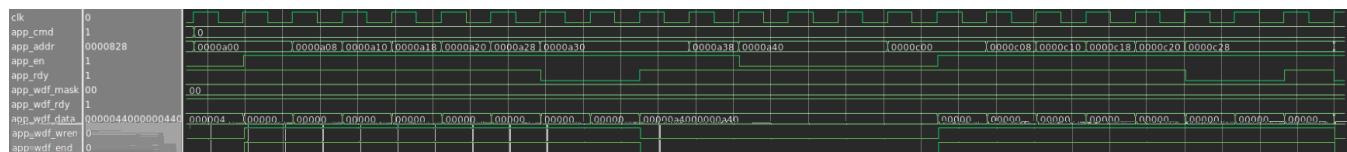
Figure 1-75:

## *Write Path*

The write data is registered in the write FIFO when **app\_wdf\_wren** is asserted and **app\_wdf\_rdy** is High (Figure 1-76). 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. **app\_wdf\_data** data can be pushed even before **app\_cmd** "write command" is asserted. The only condition is that for every **app\_cmd** "write command," the associated **app\_wdf\_data** "write data" must be present. The **app\_wdf\_mask** signal can be used to mask out the bytes to write to external memory.



*Figure 1-76:*



*Figure 1-77:*

As shown in Figure 1-75, page 168, the maximum delay for a single write between the write data and the associated write command is two clock cycles.

The **app\_wdf\_end** signal must be used to indicate the end of a memory write burst. For memory burst types of eight in 2:1 mode, 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 1-78](#). This is for a Burst Length 8 (BL8) transaction.

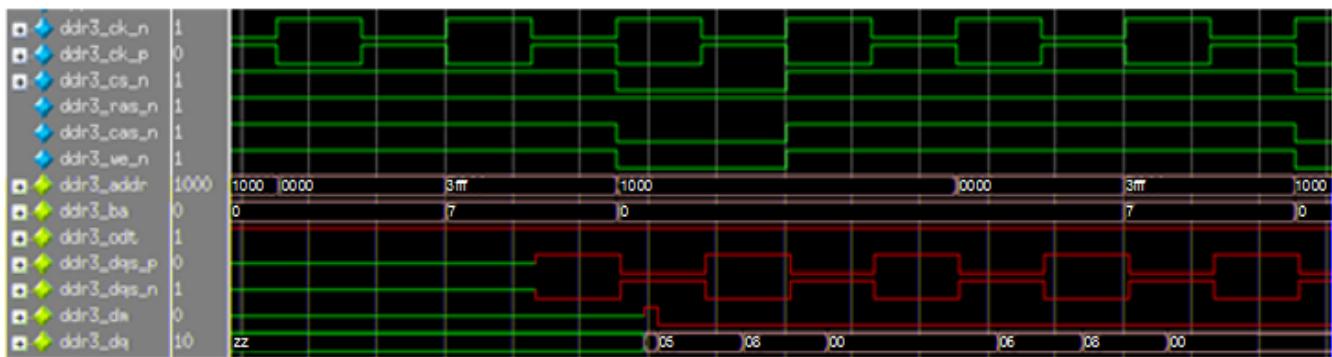


Figure 1-78:

The data values at different clock edges are as shown in Table 1-63.

Table 1-63:

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 1-79. 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 1-79:

Figure 1-80 shows the corresponding data at the DRAM interface.

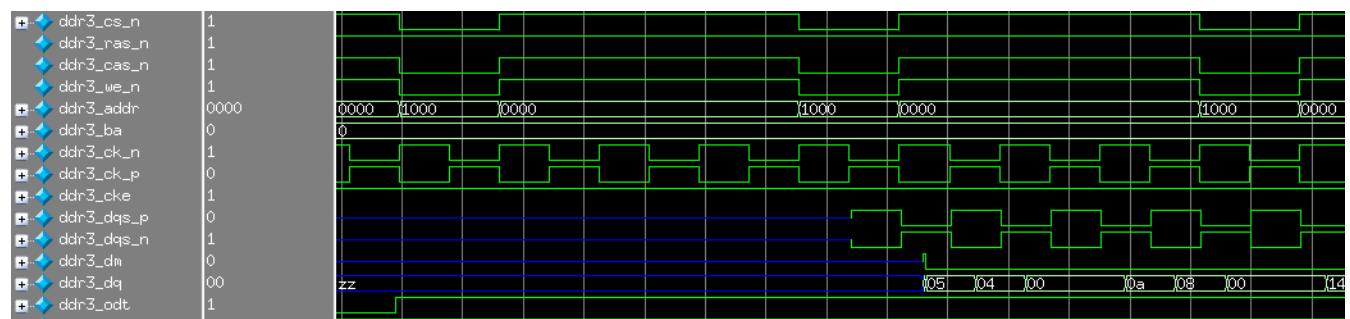


Figure 1-80:

## 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 1-81 and Figure 1-82). The `app_rd_data_end` signal indicates the end of each read command burst and is not needed in user logic.

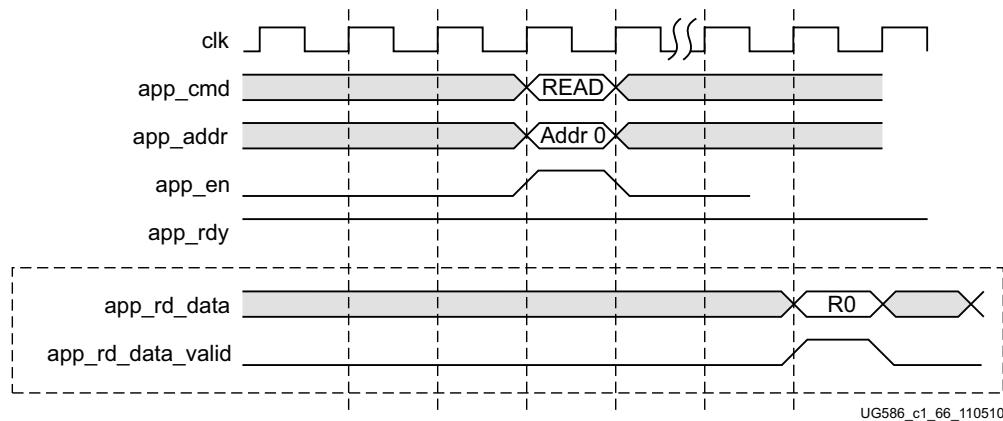


Figure 1-81:

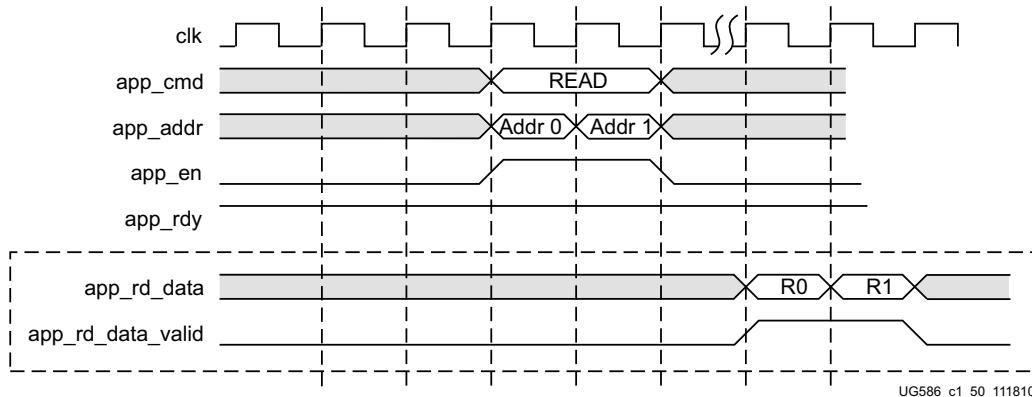


Figure 1-82:

In Figure 1-82, 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 strobos `app_ref_ack` for one cycle, after which another request can be sent. Figure 1-83 illustrates the interface.

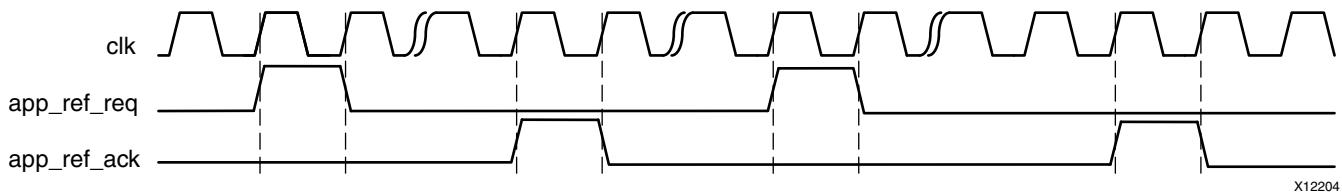


Figure 1-83:

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 strobos **app\_zq\_ack** for one cycle, after which another request can be sent. [Figure 1-84](#) illustrates the interface.

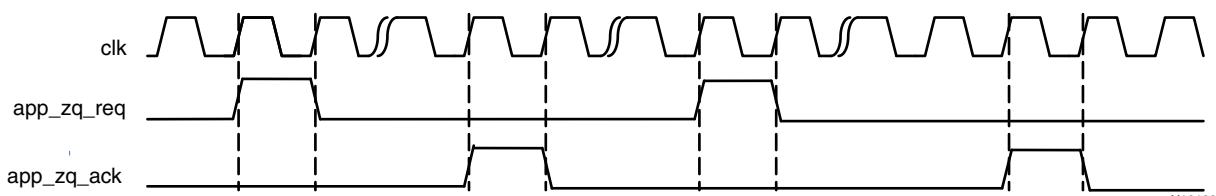


Figure 1-84:

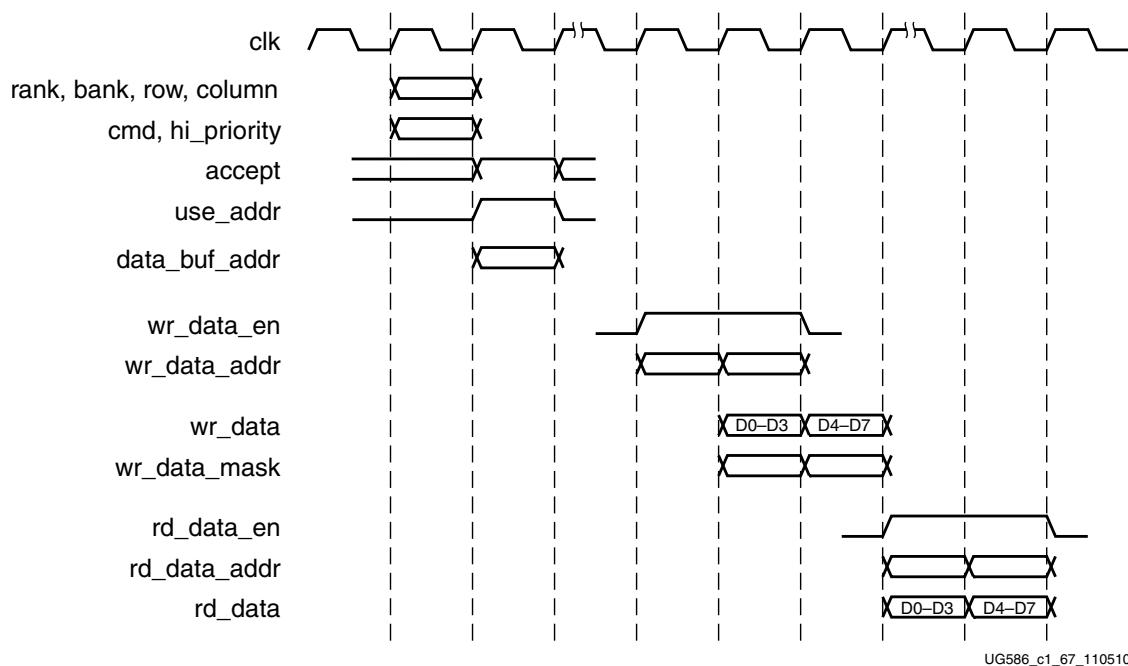
A user ZQ operation can be performed any time provided the handshake defined above is followed. There are no additional interfacing requirements with respect to other commands. However, pending requests affect when the operation goes out. The Memory Controller fulfills all pending data requests before 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.

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

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

The native interface protocol is shown in [Figure 1-85](#).



[Figure 1-85](#):

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 1-86](#).

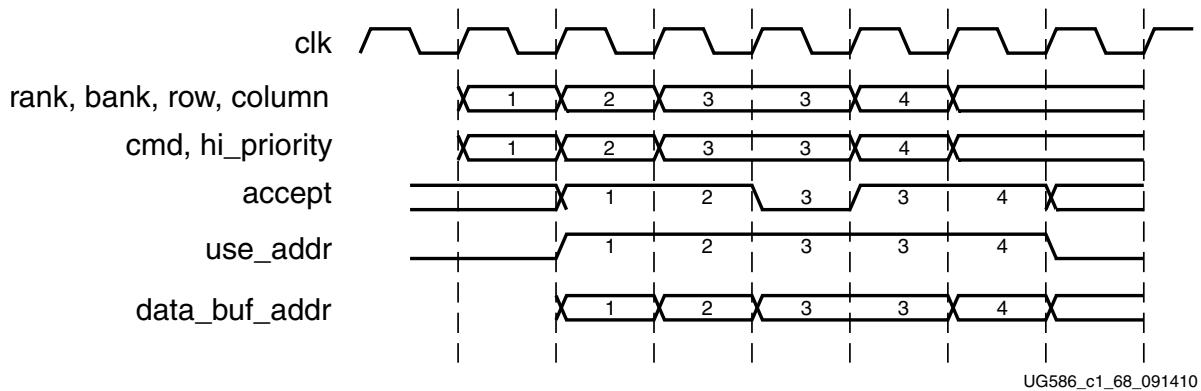


Figure 1-86:

In Figure 1-86, 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 by **wr\_data\_addr** for write commands and **rd\_data\_addr** for read commands. This behavior is shown in Figure 1-87. Write data must be supplied in the same clock cycle that **wr\_data\_en** is asserted.

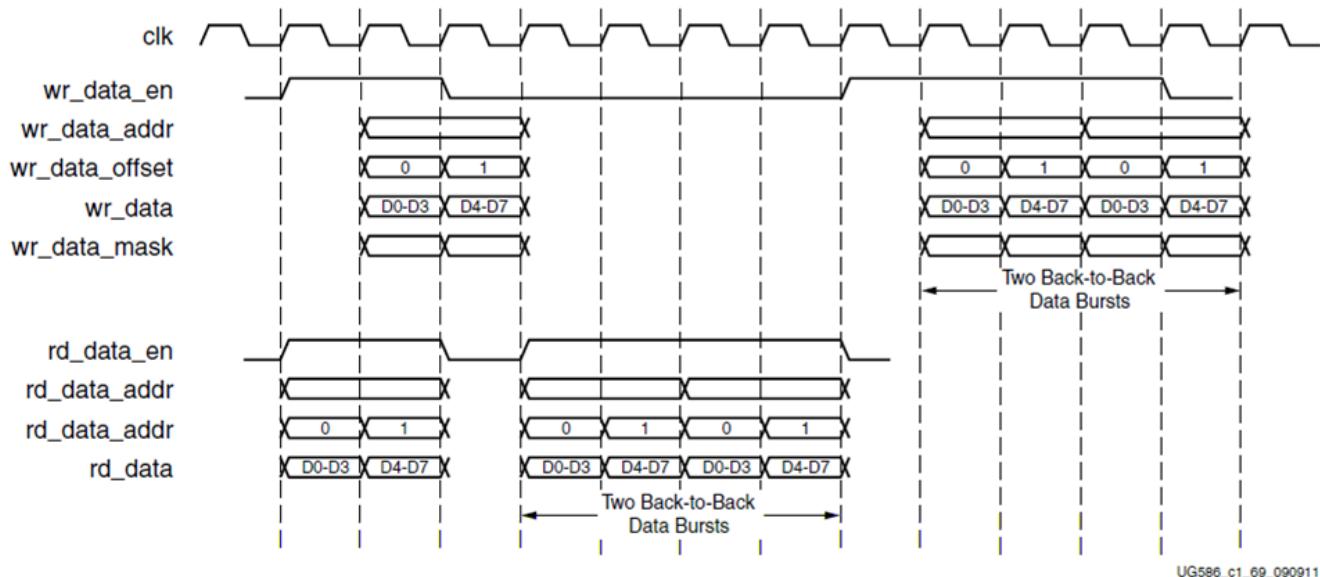


Figure 1-87:

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 1-87](#).

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.

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 132](#). The signals required for the Memory Controller to interface to the PHY are listed in [Table 1-61](#).

For clocking requirements, see [Clocking Architecture, page 120](#). You 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 your system design.

In 2014.4, the OCLKDELAYED calibration stage was enhanced to optimize the calibration center point using MMCM phase shift taps. This resulted in the addition of a BUFG for the distribution of the phase shifted clock output and a few input and output ports between the PHY and the infrastructure module. These additional ports are listed in [Table 1-61](#).

The `tempmon` module in the clocking directory should be used to supply the temperature data from the XADC to the `ddr_phy_tempmon` module. For more information, see [Temperature Monitor, page 159](#).

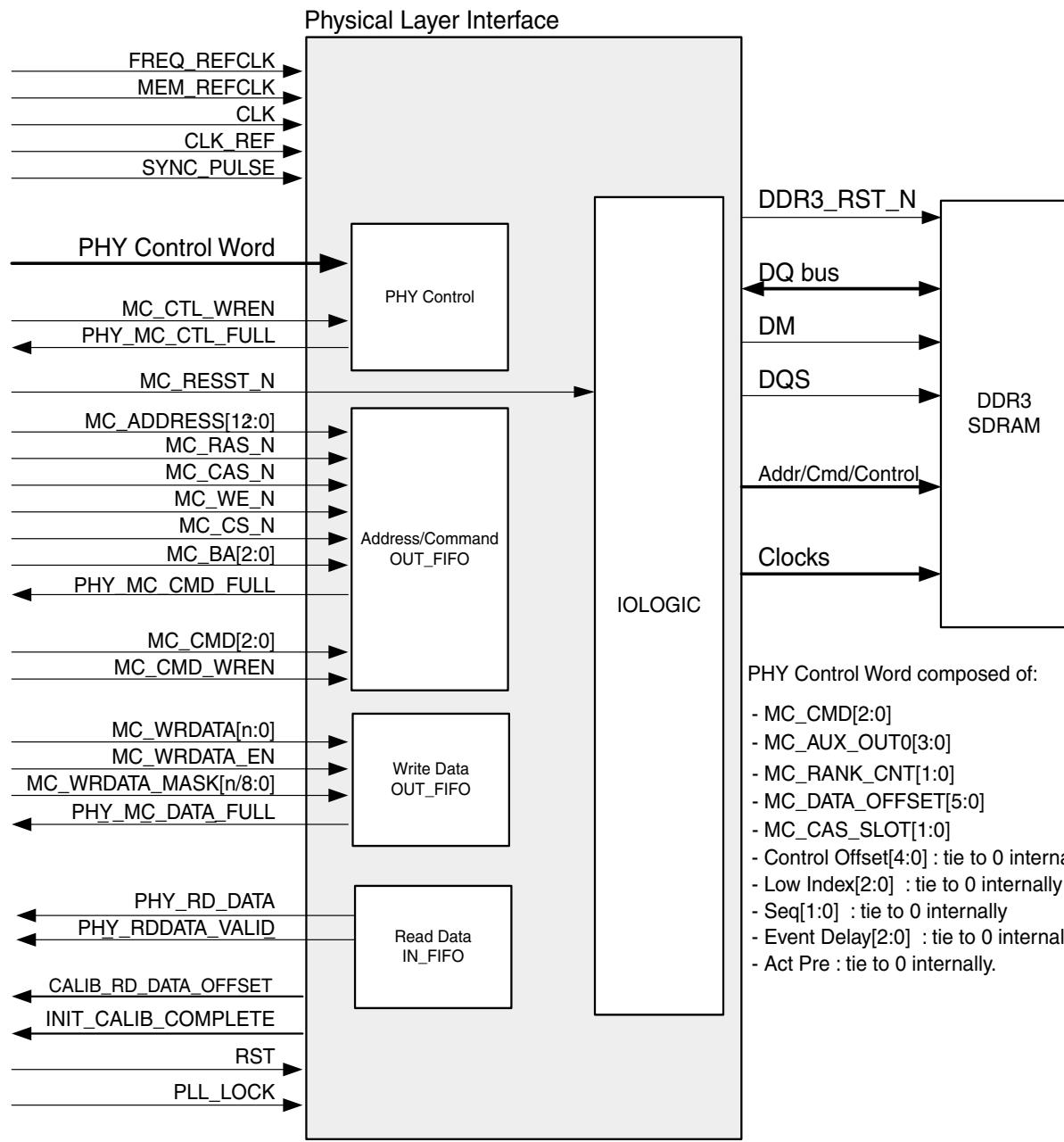
The PHY Control FIFO, command OUT\_FIFOs, and write data OUT\_FIFOs are all in asynchronous operation mode. The read clock and the write clock to these FIFOs differ in frequency and phase. Therefore all three OUT\_FIFO FULL flags (`phy_mc_ctl_full`, `phy_mc_cmd_full`, and `phy_mc_data_full`) described in [Table 1-61, page 160](#) 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 1-88](#), 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.



**IMPORTANT:** Note that the following inputs have to be tied to logic "1."

```
assign mc_reset_n = 1'b1;  
assign mc_cmd_wren = 1'b1;  
assign mc_ctl_wren = 1'b1;
```



X12189

Figure 1-88:

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

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

- For nCK\_PER\_CLK = 2

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

The write waveform shown in [Figure 1-89](#) illustrates an example with DDR3 SDRAM CWL = 7 and nCK\_PER\_CLK = 4. The selected slot number can be 1 or 3.

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

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

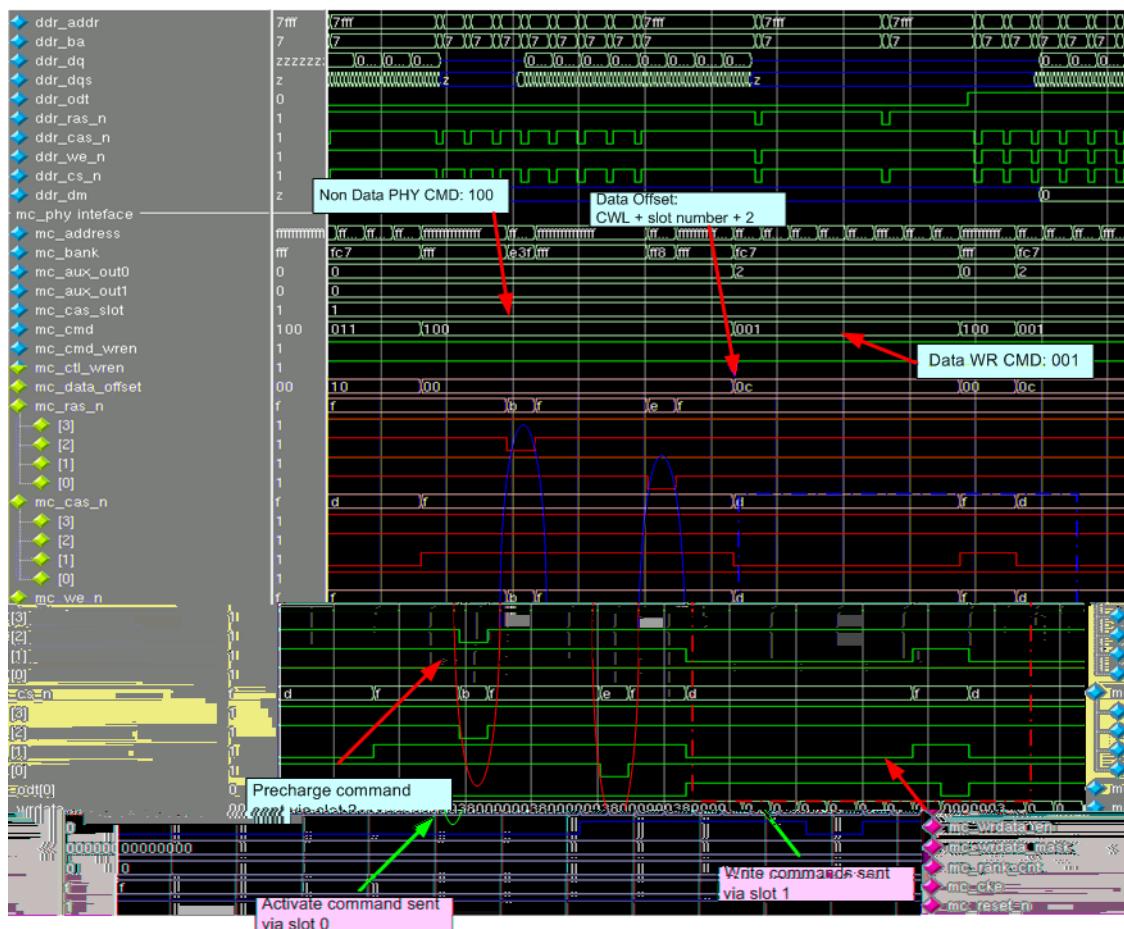


Figure 1-89:

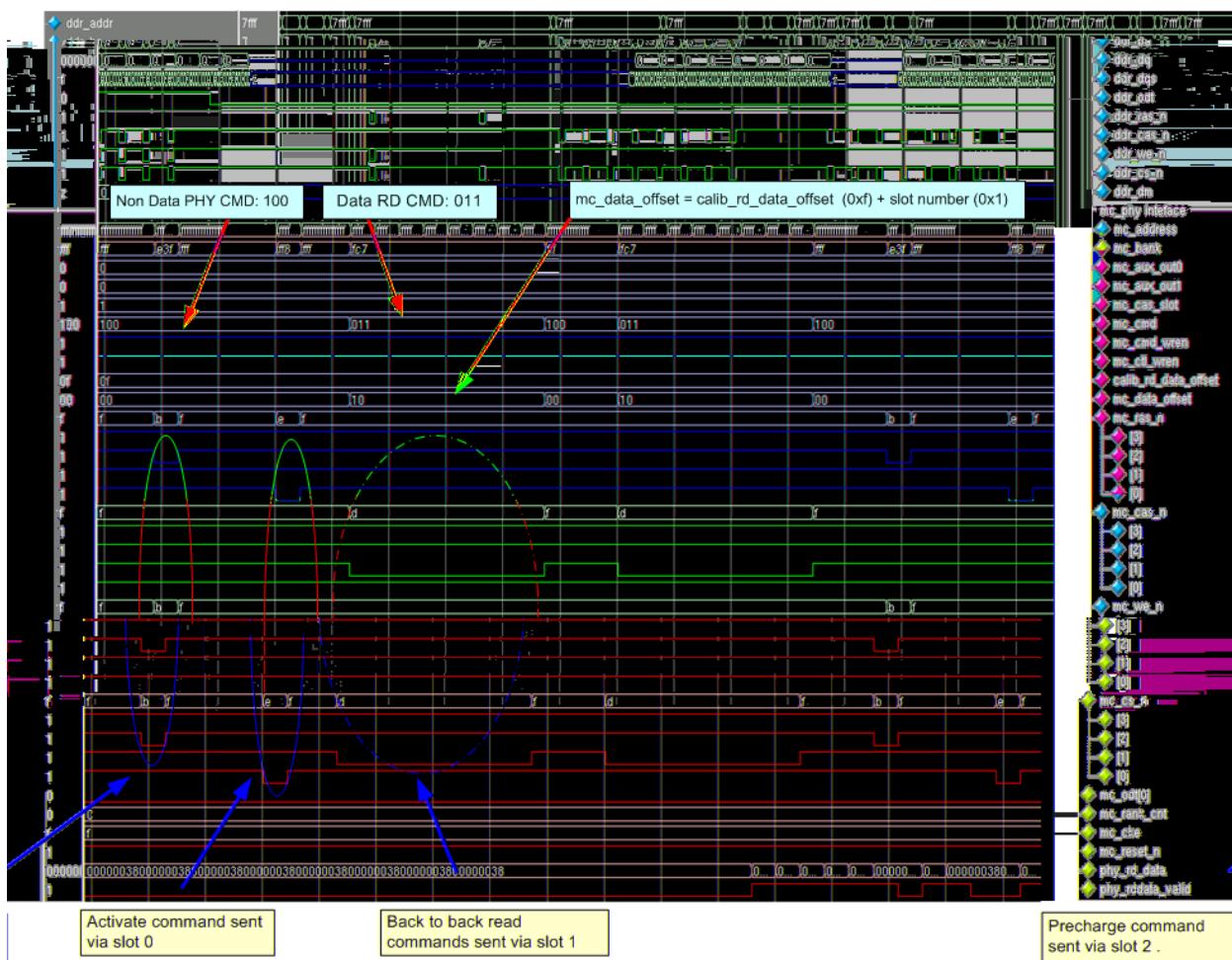
**IMPORTANT:** Bits[31:27, 24:23, 14:12, and 7:3] in [Table 1-58, page 137](#) are not used in this example.



The write waveform shown in [Figure 1-90](#) 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 1-90:*

**IMPORTANT:** Bits[31:27, 24:23, 14:12, and 7:3] in Table 1-58, page 137 are not used in this example.



The PHY calibration operates with additive latency AL) equal to 0. If a non-zero additive latency (CL - 1 or CL - 2) is preferred after the completion of calibration, the controller must issue the appropriate MRS command. Furthermore, the mentioned data offset must be recalculated with the addition of the AL value.

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. As per the OOC flow, none of the parameter values are passed down to the user design RTL file from the example design top RTL file. So, any design related parameter change is not reflected in the user design logic. 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 1-64](#), [Table 1-65](#), and [Table 1-66](#).

**Table 1-64:**

REFCLK_FREQ <sup>(1)</sup>	This is the reference clock frequency for IDELAYCTRLs. This can be set to 200.0 for any speed grade device. For DDR3 SDRAM designs, the frequency value is dependent on memory design frequency and FPGA speed grade. For more information, see the IDELAYE2 (IDELAY) and ODELAYE2 (ODELAY) Attribute Summary table in the <i>7 Series FPGAs SelectIO™ Resources User Guide</i> [Ref 2]. This parameter should <b>not</b> be changed.	200.0, 300.0, and 400.0
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, 2 (depends on the PHY to Controller Clock ratio chosen in the GUI)
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.

Table 1-64:

(Cont'd)

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.	This parameter value is 1 for both Single and Dual rank devices.
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
DO_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 $\text{ceil}(\log_2(nBANK\_MACHS))$ .	
ADDR_CMD_MODE	This parameter is used by the controller to calculate timing on the memory addr/cmd bus. This parameter should <b>not</b> be changed.	"1T"
ORDERING <sup>(3)</sup>	This option reorders received requests to optimize data throughput and latency.	"NORM": Allows the Memory Controller to reorder read but not write commands to the memory. "RELAXED": Allows the Memory Controller to reorder commands to the memory for maximum efficiency. Strong ordering is not preserved at the native interface in this mode. "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

Table 1-64:

(Cont'd)

WRLVL	This option enables write leveling calibration in DDR3 designs. This parameter must always be "ON" for DDR3 and "OFF" for DDR2. This parameter should <b>not</b> be changed.	DDR3: "ON" 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.  RTT_WR should always be set to "OFF" since Dynamic ODT is not supported.	DDR3_SDRAM: "OFF": RTT_WR disabled. "120": RZQ/2 "60": RZQ/4
OUTPUT_DRV	This is the DRAM reduced output drive option.	"HIGH" "LOW"
REG_CTRL	This is the option for DIMM or unbuffered DIMM selection. This parameter should <b>not</b> be changed.	"ON": Registered DIMM "OFF": Components, SODIMMs, UDIMMs.
IODELAY_GRP <sup>(4)</sup>	This is an ASCII character string to define an IDELAY group used in a memory design. This is used by the Vivado Design Suite 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.

Table 1-64:

(Cont'd)

DIFF_TERM_REFCLK	"TRUE," "FALSE"	Differential termination for IDELAY reference clock input pins.
TEMP_MON_CONTROL	This option selects the device temperature source for the Temperature Monitor feature (see <a href="#">Temperature Monitor</a> ). Select "INTERNAL" to direct the MIG tool to instantiate the XADC and temperature polling circuit in the memory interface top-level user design module. Select "EXTERNAL" if the XADC is already instantiated elsewhere in the design. In this case, the device temperature must be periodically sampled and driven onto the device_temp_i bus in the memory interface top-level user design module.	"INTERNAL" "EXTERNAL"

**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. 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 or RELAXED, ORDERING enables the reordering algorithm in the Memory Controller. When set to STRICT, request reordering is disabled, which might limit 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.
4. This parameter is prefixed with the module name entered in the MIG tool during design generation. If the design is generated with the module name as mig\_7series\_0, then IDELAY\_GRP parameter name is "mig\_7series\_0\_IDELAY\_MIG."

The parameters listed in [Table 1-65](#) 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_top` IP core and should not be modified in the top-level.



**RECOMMENDED:** Xilinx strongly recommends that the MIG tool be rerun for different configurations.

Table 1-65:

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.

(Cont'd)

Table 1-65:

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 ZQ short calibration interval. This value is system dependent and should be based on the expected rate of change of voltage and temperature in the system. Consult the memory vendor for more information on ZQ calibration.	This value is set in nanoseconds. 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, 10, 11 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"
RST_ACT_LOW	This is the active-Low or active-High reset. This is set to 1 when System Reset Polarity option is selected as active-Low and set to 0 when the option is selected as active-High.	0, 1
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"

(Cont'd)

Table 1-65:

DATA_IO_PRIM_TYPE	This option instantiates IBUF primitives for Data (DQ) and Strobe (DQS) as per banks selected for the interface and also depends on the I/O Power Reduction option in the MIG tool.	"HP_LP" "HR_LP" "DEFAULT"
DATA_IO_IDLE_PWRDWN	This option is set to ON valid when I/O Power reduction option is enabled.	"ON," "OFF"
CA_MIRROR	This option enables Address mirroring on second rank when it is enabled. This is valid for DDR3 SDRAM dual rank UDIMMs only. This parameter should <b>not</b> be changed.	"ON," "OFF"
SYSCLK_TYPE	This parameter indicates whether the system uses single-ended system clocks, differential system clocks, or is driven from an internal clock (No Buffer). 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. For the No Buffer option, sys_clk_i, which appears in port list, needs to be driven from an internal clock.	DIFFERENTIAL SINGLE_ENDED NO_BUFFER
REFCLK_TYPE	This parameter indicates whether the system uses single-ended reference clocks, differential reference clocks, is driven from an internal clock (No Buffer), or can connect system clock inputs only (Use System Clock). Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, clk_ref_p/clk_ref_n must be used. For single-ended clocks, clk_ref_i must be used. For the No Buffer option, clk_ref_i, which appears in the port list, needs to be driven from an internal clock. For the Use System Clock option, clk_ref_i is connected to the system clock in the user design top module.	DIFFERENTIAL SINGLE_ENDED NO_BUFFER USE_SYSTEM_CLOCK
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.
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.

Table 1-65:

(Cont'd)

CLKOUTO_PHASE	-	Phase of PLL output CLKOUTO. This value is set by the MIG tool based on the banks selected for memory interface pins and the frequency of operation.
DIVCLK_DIVIDE	-	PLLE2 VCO divisor. This value is set by the MIG tool based on the frequency of operation.
USE_DM_PORT	This is the enable data mask option used during memory write operations.	0 = Disable 1 = Enable
CK_WIDTH	This is the number of CK/CK# outputs to memory.	
DQ_CNT_WIDTH	This is $\text{ceil}(\log_2(\text{DQ\_WIDTH}))$ .	
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 <sup>(1)</sup>	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. This parameter should <b>not</b> be changed.	5, 4
SLOT_O_CONFIG	This is the rank mapping. This parameter should <b>not</b> be changed.	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.	
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_INTC parameter specifies the payload data width in the UI.	APP_DATA_WIDTH = 2 x nCK_PER_CLK x PAYLOAD_WIDTH
APP_MASK_WIDTH	This UI_INTC parameter specifies the payload mask width in the UI.	

Table 1-65:

(Cont'd)

USER_REFRESH	This parameter indicates if the user manages refresh commands. Can be set for either the User or Native interface.	"ON," "OFF"
REF_CLK_MMCM_IODELAY_CTRL	This parameter value determines the instantiation of MMCM. This MMCM is used to generate 300 MHz and 400 MHz clock for IDELAY CTRL module.	"TRUE," "FALSE"

**Notes:**

1. nBANK\_MACHS parameter values can be changed in user\_design top-level RTL file (<module name>\_mig.v/vhd: This RTL file is used as user design top RTL file for synthesis and implementation. <module name>\_mig\_sim.v/vhd: This RTL file is used as user design top RTL file for simulation.). Note that the parameter value can be updated only in non-OOC MIG designs.

Table 1-66 contains parameters set up by the MIG tool based on the pinout selected. When making pinout changes, Xilinx recommends rerunning the MIG tool to set up the parameters properly. See [Bank and Pin Selection Guides for DDR3 Designs, page 193](#) and [Bank and Pin Selection Guides for DDR2 Designs, page 203](#).

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 1-66:

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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	4'b1100: With respect to the BYTE_LANE example, two byte lanes are used for Data and one for Address/Control.

Table 1-66:

(Cont'd)

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 CK/CK# and RESET_N pins, all Data and Address/Control pins are considered for this parameter generation. DQS pins are excluded when used for DQS pins in data byte groups. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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_BYTE_MAP	Bank and byte lane location information for the CK/CK#. An 8-bit parameter is provided per pair of signals. [7:4] – Bank position. Values of 0, 1, or 2 are supported [3:0] – Byte lane position within a bank. Values of 0, 1, 2, and 3 are supported. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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_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 0. 8'h20: CK/CK# placed in bank 2, byte lane 3.

Table 1-66:

(Cont'd)

ADDR_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>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<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>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 with in a byte group by excluding DQS I/Os. DQS_N and DQS_P pins of the byte group are numbered as A and B, respectively.</p> <p>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).</p> <p>12'h02B: Address pin placed in bank 0, byte lane 1, at location B.</p> <p>12'h235: Address pin placed in bank 2, byte lane 0, at location 5.</p>
BANK_MAP	<p>Bank and byte lane position information for the bank address. See the <a href="#">ADDR_MAP</a> description.</p> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	See the <a href="#">ADDR_MAP</a> example.
CAS_MAP	<p>Bank and byte lane position information for the CAS command. See the <a href="#">ADDR_MAP</a> description.</p> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	See the <a href="#">ADDR_MAP</a> example.
CKE_MAP	<p>Bank and byte lane position information for the CKE. This parameter is referred to as one of the Address/Control byte groups. See <a href="#">ADDR_MAP</a> description. This parameter varies based on the pinout and should not be changed manually in generated design.</p>	See the <a href="#">ADDR_MAP</a> example.

Table 1-66:

(Cont'd)

ODT_MAP	Bank and byte lane position information for the ODT. This parameter is referred to as one of the Address/Control byte groups. See <a href="#">ADDR_MAP</a> description. This parameter varies based on the pinout and should not be changed manually in generated design.	See the <a href="#">ADDR_MAP</a> example.
CS_MAP	Bank and byte lane position information for the chip select. See the <a href="#">ADDR_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">ADDR_MAP</a> example.

Table 1-66:

(Cont'd)

DQS_BYTE_MAP	Bank and byte lane position information for the strobe. See the <a href="#">CK_BYTE_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">ADDR_MAP</a> example.

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

For general PCB routing guidelines, see [Appendix A, General Memory Routing Guidelines](#).

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.

## Bank and Pin Selection Guides for DDR3 Designs

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

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 10 associated I/Os.

Several times in this document byte groups are referenced for address and control as well, this refers to the 12 associated groups. In a typical DDR3 data bus configuration, eight of these 10 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.

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

- 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.



---

**RECOMMENDED:** *Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.*

---

- **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 **DQS** pins.
- If multiple **CK** outputs are used, such as for dual rank, all **CK** outputs must come from the same byte lane.
- **DQS** signals for a byte group must be connected to a designated **DQS** pair in the bank due to the dedicated strobe connections for DDR2 and DDR3 SDRAM. For more information, see *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10].
- **DQ** and **DM** (if used) signals must be connected to the byte group pins associated with the corresponding **DQS**.
- **VRN** and **VRP** are used for the digitally controlled impedance (DCI) reference for banks that support DCI.
- The non-byte groups pins (that is, **VRN/VRP** pins in HP banks and top/bottom most pins in HR banks) can be used for an address/control pin, if the following conditions are met:
  - For HP banks, 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.

- An unused pin exists in the adjacent byte group (T0/T3) or the **CK** output is contained in the adjacent byte group.
- No more than three vertical banks from a die perspective can be used for a single interface.
- The address/control must be in the middle I/O bank of interfaces that span three I/O banks. All address/control must be in the same I/O bank. Address/control cannot be split between banks.
- 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.
- **RESET\_N** can be connected to any available pin within the device, including the **VRN/VRP** pins if DCI cascade is used, as long as timing is met and an appropriate I/O voltage standard is used. The GUI restricts this pin to the banks used for the interface to help with timing, but this is not a requirement.
- 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.

### ***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.

### ***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. With the exception of the shared address and control in the dual controller supported in the MIG core.

### ***System Clock, PLL and MMCM Locations, and Constraints***

The PLL and MMCM are 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. If the clock came 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 120](#) for information on allowed PLL parameters.

### ***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. 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.

### ***V<sub>REF</sub>***

The V<sub>REF</sub> includes internal and external:

- **Internal V<sub>REF</sub>** – Only be used for data rates of 800 Mb/s or below.
- **External V<sub>REF</sub> and V<sub>REF</sub> Tracking** – For the maximum specified data rate in a given FPGA speed grade, external V<sub>REF</sub> must track the midpoint of the VDD supplied to the DRAM and ground. V<sub>REF</sub> tracking can be done with a resistive divider or by a regulator that tracks this midpoint. Regulators that supply a fixed reference voltage irrespective of the VDD voltage should not be used at these data rates. V<sub>REF</sub> traces need to have a larger than the minimum spacing to reduce coupling from other intrusive signals. See [7 Series FPGAs PCB Design and Pin Planning Guide \(UG483\) \[Ref 12\]](#), "V<sub>REF</sub> Stabilization Capacitors" section.

### ***VCCAUX\_IO***

VCCAUX\_IO has two values that can be set to 1.8V or 2.0V depending on memory performance. If migration occurs between different memory performance or FPGA speed grades, VCCAUX\_IO might need to be its own supply that can be adjusted. For performance information, see the [7 Series FPGAs Data Sheets \[Ref 13\]](#).

For more information on VCCAUX\_IO, see *7 Series SelectIO™ Resources User Guide* (UG471) [Ref 2], "VCCAUX\_IO" section.

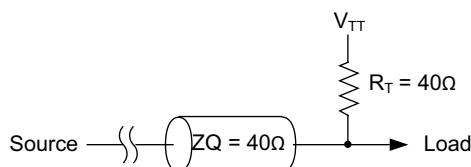
## ***Power System and Plane Discontinuities***

See *7 Series FPGAs PCB Design and Pin Planning Guide* (UG483) [Ref 12].

### ***Termination***

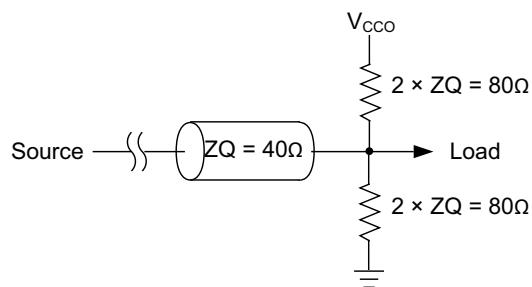
These rules apply to termination for DDR3 SDRAM:

- 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.
- Single ended  $40\Omega$  traces and termination are required for operation at 1,333 Mb/s and higher.  $50\Omega$  is acceptable below 1,333 Mb/s. [Figure 1-91](#) and [Figure 1-92](#) are for 1,333 Mb/s and higher.
- Differential  $80\Omega$  traces and termination are required for operation at 1,333 Mb/s and higher.  $100\Omega$  is acceptable below 1,333 Mb/s. [Figure 1-93](#) is for 1,333 Mb/s and higher.
- When using a  $V_{TT}$  supply, care must be taken to manage the high frequency currents from the terminations. Bypass caps recommendation 1  $\mu$ F for every four terminations and 100  $\mu$ F for every 25 terminations evenly distributed relative to the terminations. A planelet should also be used to distribute power to the terminations.
- Address and control signals (**A**, **BA**, **RAS\_N**, **CAS\_N**, **WE\_N**, **CS\_N**, **CK**, **CK\_N**, **ODT**) are to be terminated with the onboard DIMM termination. If DIMM termination does not exist or a component is being used, a  $40\Omega$  pull-up to  $V_{TT}$  at the far end of the line should be used ([Figure 1-91](#)). Except for the **CK**/**CK\_N** which requires a differential termination as shown in [Figure 1-93](#).
- A split  $80\Omega$  termination to  $V_{CCO}$  and a  $80\Omega$  termination to GND can be used ([Figure 1-92](#)), 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.



X18716-012917

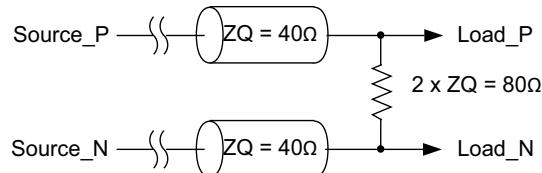
*Figure 1-91:*  $\Omega$



X18717-012917

*Figure 1-92:*  $\Omega$

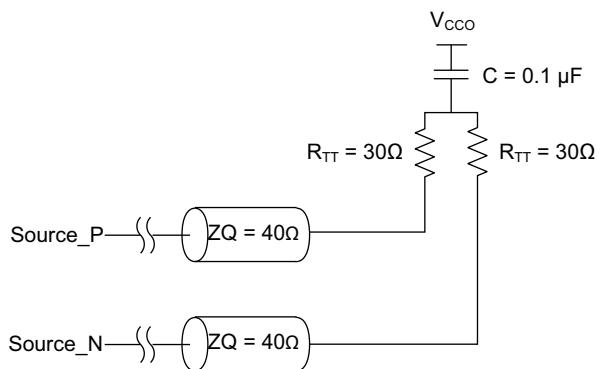
- Differential signals should be terminated with the memory device internal termination or an  $80\Omega$  differential termination at the load (Figure 1-93). For bidirectional signals, termination is needed at both ends of the signal. ODT should be used on the memory side. For best performance in HP banks, DCI should be used. For best performance in HR banks, IN\_TERM (internal termination) should be used.



X16403-012917

*Figure 1-93:*  $\Omega$

**Note:** For CK\_P/CK\_N differential signals, the termination method mentioned in Figure 1-94 is recommended.



X16404-012917

*Figure 1-94:*  $\Omega$

- 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 (HP banks) or IN\_TERM (HR banks) is required at the FPGA to meet the specified performance.
- The **RESET\_N** signal is not terminated. This signal should be pulled down during memory initialization with a  $4.7\text{ k}\Omega$  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 [\[Ref 14\]](#) for additional details on **ODT**.
- **DM** should be pulled to GND if **DM** is not driven by the FPGA (data mask not used or data mask disabled scenarios). The value of the pull-down resistor used for **DM** in this case should be no larger than four times the **ODT** value. Check with the memory vendor for further information.

## 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.

Another method is to generate the package lengths using Vivado Design Suite. The following commands generate a **csv** file that contains the package delay values for every pin of the device under consideration.

```
link_design -part <part_number>
write_csv <file_name>
```

For example, to obtain the package delay information for the 7 series FPGA XC7K160T-FF676, this command should be issued:

```
link_design -part xc7k160tfg676
write_csv flight_time
```

This generates a file named **flight\_time.csv** 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.

When migrating between different die sizes in the same package, there might be different delays for the same package pin. The delay values for each of the devices must be accounted for and the mid-range should be used for each pin. This might decrease the maximum possible performance for the target device. See [Table 1-67](#) for exact degradation.

These rules indicate the maximum electrical delays between DDR3 SDRAM signals:

- The maximum electrical delay between any **DQ** or **DM** and its associated **DQS/DQS#** must be  $\leq \pm 5\text{ ps}$ .

- The maximum electrical delay between any address and control signals and the corresponding **CK/CK#** must be  $\leq \pm 25$  ps, with 8 ps being the optimal target.
- CK/CK#** signals must arrive at each memory device after the **DQS/DQS#** signals. The skew allowed between **CK/CK#** and **DQS/DQS#** must be bounded between 0 and 1,600 ps. The recommended skew between **CK/CK#** and **DQS/DQS#** is 150 ps to 1,600 ps for components/UDIMMs and for RDIMMs it is 450 ps to 750 ps. For DIMM modules, the total **CK/CK#** and **DQS/DQS#** propagation delays from the FPGA to the memory components on the DIMM must be accounted for when designing to this requirement.
- CK/CK#** must arrive after **DQS/DQS#** at each memory component to ensure calibration can align **DQS/DQS#** to the correct **CK/CK#** clock cycle. Write Calibration failures are seen if this specification is violated. See [Debugging Write Calibration Failures \(dbg\\_wrcal\\_err = 1\)](#), page 253 in the [Debugging DDR3/DDR2 Designs](#), page 228.

The specified **DQ** to **DQS** skew limit can be increased if the memory interface is not operated at the maximum frequency. [Table 1-67](#) indicates the relaxed skew limit ( $\pm$ ) for these cases. The vertical axis is the bit rate in Mb/s. The first column is the FPGA maximum rate, check the data sheet to determine this maximum rate. The second column is the actual speed the memory system is operating at. The horizontal axis is the DDR3 SDRAM component speed rating.

*Table 1-67:*

1,866	1,866	18.0	5.0	-	-	-	-
	1,600	62.6	49.5	31.3	-	-	-
	1,333	125.2	112.1	93.9	66.4	-	-
	1,066	150.0	150.0	150.0	150.0	125.4	-
	800	150.0	150.0	150.0	150.0	150.0	150.0
1,600	1,866	-	-	-	-	-	-
	1,600	36.2	23.2	5.0	-	-	-
	1,333	98.8	85.8	67.6	40.1	-	-
	1,066	150.0	150.0	150.0	134.0	99.0	-
	800	150.0	150.0	150.0	150.0	150.0	150.0
1,333	1,866	-	-	-	-	-	-
	1,600	-	-	-	-	-	-
	1,333	63.7	50.7	32.5	5.0	-	-
	1,066	150.0	144.7	126.4	98.9	63.9	-
	800	150.0	150.0	150.0	150.0	150.0	150.0

Table 1-67: (Cont'd)

1,066	1,866	-	-	-	-	-	-	-
	1,600	-	-	-	-	-	-	-
	1,333	-	-	-	-	-	-	-
	1,066	98.7	85.7	67.5	40.0	5.0	-	-
	800	150.0	150.0	150.0	150.0	150.0	98.5	-
800	1,866	-	-	-	-	-	-	-
	1,600	-	-	-	-	-	-	-
	1,333	-	-	-	-	-	-	-
	1,066	-	-	-	-	-	-	-
	800	150.0	148.2	130.0	102.5	67.0	5.0	-

For example, if an 1,866 rated –3 FPGA operates at 1,600 Mb/s with a 1,600 rated DDR3 component, the **DQ to DQS** skew limit is  $\pm 31.3$  ps. If the interface operates at 1,066 with a 1,333 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 1-68 indicates the relaxed skew limit ( $\pm$ ) 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 1,867 Mb/s rated FPGAs, while the lower portion is for the 1,600 Mb/s rated FPGAs.

Table 1-68:

1,866	1,866	35.0	25.0	-	-	-	-	-
	1,600	124.1	114.1	94.1	-	-	-	-
	1,333	150.0	150.0	150.0	150.0	-	-	-
	1,066	150.0	150.0	150.0	150.0	150.0	-	-
	800	150.0	150.0	150.0	150.0	150.0	150.0	150.0
1,600	1,866	-	-	-	-	-	-	-
	1,600	55.0	45.0	25.0	-	-	-	-
	1,333	150.0	150.0	150.0	130.2	-	-	-
	1,066	150.0	150.0	150.0	150.0	150.0	-	-
	800	150.0	150.0	150.0	150.0	150.0	150.0	150.0

Table 1-68:

(Cont'd)

1,333	1,866	-	-	-	-	-	-	-
	1,600	-	-	-	-	-	-	-
	1,333	75.0	65.0	45.0	25.0	-	-	-
	1,066	150.0	150.0	150.0	150.0	140.4	-	-
	800	150.0	150.0	150.0	150.0	150.0	150.0	150.0
1,066	1,866	-	-	-	-	-	-	-
	1,600	-	-	-	-	-	-	-
	1,333	-	-	-	-	-	-	-
	1,066	147.5	137.5	117.5	97.5	25.0	-	-
	800	150.0	150.0	150.0	150.0	150.0	150.0	150.0
800	1,866	-	-	-	-	-	-	-
	1,600	-	-	-	-	-	-	-
	1,333	-	-	-	-	-	-	-
	1,066	-	-	-	-	-	-	-
	800	150.0	150.0	150.0	150.0	100.0	25.0	-

For example, if an 1,863 Mb/s rated FPGA operates at 1600 Mb/s with a 1,600 rated DDR3 component, the CK to address/control skew limit is  $\pm 94.1$  ps. If a 1,600 Mb/s rated FPGA operates at 1,066 with a 1,333 rated DDR3 component, the skew limit is  $\pm 150$  ps.

The skew between bytes in an I/O bank must be 1 ns or less.

## Configuration

The XDC 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. This must be rerun if this needs to be altered, because other internal parameters are affected. For example:

```
create_clock -period 1.875 [get_ports sys_clk_p]
```

The **clk\_ref** constraint sets the frequency for the IDELAY reference clock, which is typically 200 MHz. For example:

```
create_clock -period 5 [get_ports clk_ref_p]
```

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 (asynchronous). The reset signal must be applied for a minimum pulse width of 5 ns.
- **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 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 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, the MIG tool puts an additional constraint in the XDC. An example is shown here:

```
set_property CLOCK_DEDICATED_ROUTE_BACKBONE [get_nets sys_clk_p]
set_property CLOCK_DEDICATED_ROUTE_BACKBONE [get_pins -hierarchical *pll*CLKIN1]
```

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 I/O 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 might want to analyze why this issue 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 might lead to very poor timing results. It is recommended that this
error condition be corrected in the design.
```

Do not drive user clocks through 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. For more information, see the *7 Series FPGAs Clocking Resources User Guide* (UG472) [\[Ref 10\]](#).

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

```
# PadFunction: IO_L1P_T0_39
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[0]}]
set_property SLEW FAST [get_ports {ddr3_dq[0]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[0]}]
set_property PACKAGE_PIN A9 [get_ports {ddr3_dq[0]}]

# PadFunction: IO_L1N_T0_39
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[1]}]
set_property SLEW FAST [get_ports {ddr3_dq[1]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[1]}]
set_property PACKAGE_PIN A8 [get_ports {ddr3_dq[1]}]
```

Consult the Constraints Guide for more information.

For DDR3 SDRAM interfaces that have the memory system input clock (`sys_clk_p`/`sys_clk_n`) placed on CCIO pins within one of the memory banks, the MIG tool assigns the DIFF\_SSTL15 I/O standard (VCCO = 1.5V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_SSTL15 and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_SSTL15 CCIO pins. For more details on usage and required circuitry for LVDS and LVDS\_25 I/O Standards, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2].

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.
- LVCMS15 is used for the `RESET_N` signal driven to the DDR3 memory.

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

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, eight 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 non-byte groups pins (that is, VRN/VRP pins in HP banks and top/bottom most pins in HR banks) can be used for an address/control pin, if the following conditions are met:
  - For HP banks, 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.
  - An unused 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 DQS 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. **CK** pairs are generated for each component, and a maximum of four pairs only are allowed due to I/O pin limitations. This varies based on **Memory Clock Selection** in the Memory Options page in the MIG GUI.
- **CS\_N** pins are generated for each component and a maximum of four ports/pairs only are allowed due to I/O pin limitations.
- 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.



**RECOMMENDED:** *Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.*

- 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.

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.

- 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 V<sub>REF</sub>***

Internal V<sub>REF</sub> 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 120](#) for information on allowed PLL parameters.

## *Configuration*

The XDC 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:

```
create_clock -period 1.875 [get_ports sys_clk_p]
```

The **clk\_ref** constraint sets the frequency for the IDELAY reference clock, which is typically 200 MHz. For example:

```
create_clock -period 5 [get_ports clk_ref_p]
```

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 (asynchronous). The reset signal must be applied for a minimum pulse width of 5 ns.
- **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 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 XDC. An example is shown here:

```
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_nets sys_clk_p]
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hierarchical *pll*CLKIN1]
```

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 I/O 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 might want to analyze why this issue 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 might lead to very poor timing results. It is recommended that this error condition be corrected in the design.*

Do not drive user clocks through 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. For more information, see the *7 Series FPGAs Clocking Resources User Guide* (UG472) [\[Ref 10\]](#).

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

```
# PadFunction: IO_L14P_T2_SRCC_36
set_property VCCAUX_IO NORMAL [get_ports {ddr2_dq[0]}]
set_property SLEW FAST [get_ports {ddr2_dq[0]}]
set_property IOSTANDARD SSTL18_II_T_DCI [get_ports {ddr2_dq[0]}]
set_property PACKAGE_PIN AJ12 [get_ports {ddr2_dq[0]}]

# PadFunction: IO_L14N_T2_SRCC_36
set_property VCCAUX_IO NORMAL [get_ports {ddr2_dq[1]}]
set_property SLEW FAST [get_ports {ddr2_dq[1]}]
set_property IOSTANDARD SSTL18_II_T_DCI [get_ports {ddr2_dq[1]}]
set_property PACKAGE_PIN AK12 [get_ports {ddr2_dq[1]}]
```

Consult the *Xilinx Timing Constraints User Guide* (UG612) [\[Ref 15\]](#) for more information.

For DDR2 SDRAM interfaces that have the memory system input clock (**sys\_clk\_p/sys\_clk\_n**) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_SSTL18\_II I/O standard (VCCO = 1.8V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_SSTL18\_II and LVDS inputs, an LVDS clock

source can be connected directly to the DIFF\_SSTL18\_II CCIO pins. For more details on usage and required circuitry for LVDS and LVDS\_25 I/O Standards, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2].

## 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 internal termination or a pull-up of  $50\Omega$  to VTT at the load (Figure 1-91 with  $50\Omega$  instead of  $40\Omega$ ). A split  $100\Omega$  termination to  $V_{CCO}$  and a  $100\Omega$  termination to GND can be used (Figure 1-92 with  $100\Omega$  instead of  $80\Omega$ ), 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 internal termination or a  $100\Omega$  differential termination at the load (Figure 1-93). 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 (HP banks) or IN\_TERM (HR banks) is required at the FPGA to meet the specified performance.
- Address (**A**, **BA**) and control signals (**RAS\_N**, **CAS\_N**, **WE\_N**, **CS\_N**, **ODT**) are to be terminated with the onboard DIMM termination. If DIMM termination does not exist or a component is being used, a  $50\Omega$  pull-up to  $V_{TT}$  at the far end of the line should be used except for the **CK/CK\_N** which requires a differential termination.
- The **CKE** signal is not terminated. This signal should be pulled down during memory initialization with a  $4.7\text{ k}\Omega$  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 14] for additional details on ODT.
- **ODT** applies to the **DQ**, **DQS**, and **DM** signals only. If **ODT** is used, the mode register must be set appropriately to enable **ODT** at the memory.

- **DM** should be pulled to GND if **ODT** is used but **DM** is not driven by the FPGA (for scenarios where the data mask 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 SSTL18\_II\_T\_DCI and DIFF\_SSTL18\_II\_T\_DCI standards for all bidirectional I/O (**DQ**, **DQS**) in the High-Performance banks. In the High-Range banks, the tool uses the SSTL18\_II and DIFF\_SSTL18\_II standard with the internal termination (IN\_TERM) attribute chosen in the GUI.
- The SSTL18\_II and DIFF\_SSTL18\_II standards are used for unidirectional outputs, such as control/address and forward memory clocks.
- LVCMOS18 is used for the **RESET\_N** signal driven to the DDR2 memory RDIMM interfaces. The MIG tool creates the XDC 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 generate the package lengths using Vivado. The following commands generate a **csv** file that contains the package delay values for every pin of the device under consideration.

```
link_design -part <part_number>
write_csv <file_name>
```

For example, to obtain the package delay information for the 7 series FPGA XC7K160T-FF676, this command should be issued:

```
link_design -part xc7k160tfgb676
write_csv flight_time
```

This generates a file named **flight\_time.csv** 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** or **DM** and its associated **DQS/DQS#** must be  $\leq \pm 5$  ps.
- The maximum electrical delay between any address and control signals and the corresponding **CK/CK#** must be  $\leq \pm 25$  ps.
- The maximum electrical delay between any **DQS/DQS#** and **CK/CK#** must be  $< \pm 25$  ps.

The 7 series FPGA MIG DDR3/DDR2 design has two clock inputs, the reference clock and the system clock. The reference clock drives the IODELAYCTRL components in the design, while the system clock input is used to create all MIG design clocks that are used to clock 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 more information on clocking architecture, see [Clocking Architecture, page 120](#).

The MIG tool allows you to input the Memory Clock Period and then lists available Input Clock Periods that follow the supported clocking guidelines. Based on these two clock periods selections, the generated MIG core appropriately sets the PLL parameters. The MIG tool enables automatic generation of all supported clocking structures. For information on how to use the MIG tool to set up the desired clocking structure including input clock placement, input clock frequency, and IDELAYCTRL **ref\_clk** generation, see [Creating 7 Series FPGA DDR3 Memory Controller Block Design, page 32](#).

## ***Input Clock Guidelines***



---

**IMPORTANT:** *The input system clock cannot be generated internally.*

---

- PLL Guidelines
  - CLKFBOUT\_MULT\_F (M) must be between 1 and 16 inclusive.
  - DIVCLK\_DIVIDE (D, Input Divider) can be any value supported by the PLLE2 parameter.
  - CLKOUT\_DIVIDE (O, Output Divider) must be 2 for 400 MHz and up operation and 4 for below 400 MHz operation.
  - The above settings must ensure the minimum PLL VCO frequency (FVCOMIN) is met. For specifications, see the appropriate DC and Switching Characteristics Data Sheet. The *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] includes the equation for calculating FVCO.
  - The relationship between the input period and the memory period is  $\text{InputPeriod} = (\text{MemoryPeriod} \times M) / (D \times D1)$ .

- The clock input (**sys\_clk**) can be input on any CCIO in the column where the memory interface is located; this includes CCIO in banks that do not contain the memory interface, but must be in the same column as the memory interface. The PLL must be located in the bank containing the clock sent to the memory. To route the input clock to the memory interface PLL, the CMT backbone must be used. With the MIG implementation, one spare interconnect on the backbone is available that can be used for this purpose.
- MIG core versions 1.4 and later allow this input clocking setup and properly drive the CMT backbone.
- **CLOCK\_DEDICATED\_ROUTE = BACKBONE** constraint is used to implement CMT backbone, following warning message is expected. It can be ignored safely.

**WARNING:** [Place 30-172] Sub-optimal placement for a clock-capable IO pin and PLL pair. The flow will continue as the **CLOCK\_DEDICATED\_ROUTE** constraint is set to **BACKBONE**.

```
u_mig_7series_0/c0_u_ddr3_clk_ibuf/diff_input_clk.u_ibufg_sys_clk (IBUFDS_O) is
locked to IOB_X0Y176
u_mig_7series_0/c0_u_ddr3_infrastructure/plle2_i (PLLE2_ADV.CLKIN1) is locked to
PLLE2_ADV_X0Y1
u_mig_7series_0/c1_u_ddr3_infrastructure/plle2_i (PLLE2_ADV.CLKIN1) is locked to
PLLE2_ADV_X0Y5
....
```

- For DDR3 interfaces that have the memory system input clock (**sys\_clk**) placed on CCIO pins within one of the memory banks, the MIG tool assigns the DIFF\_SSTL15 I/O standard (VCCO = 1.5V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_SSTL15 and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_SSTL15 CCIO pins.
- It is acceptable to have differential inputs such as LVDS and LVDS\_25 in I/O banks that are powered at voltage levels other than the nominal voltages required for the outputs of those standards (1.8V for LVDS outputs, and 2.5V for LVDS\_25 outputs). However, these criteria must be met:
  - a. The optional internal differential termination is not used (DIFF\_TERM = FALSE, which is the default value).  
**Note:** This might require manually changing DIFF\_TERM parameter located in the top-level module or setting this in the UCF or XDC.
  - b. The differential signals at the input pins meet the VIN requirements in the Recommended Operating Conditions table of the specific device family data sheet.
  - c. The differential signals at the input pins meet the VIDIFF (min) requirements in the corresponding LVDS or LVDS\_25 DC specifications tables of the specific device family data sheet.

One way to accomplish the above criteria is to use an external circuit that both AC-couples and DC-biases the input signals. The figure shows an example circuit for providing an AC-coupled and DC-biased circuit for a differential clock input. RDIFF

provides the  $100\Omega$  differential receiver termination because the internal DIFF\_TERM is set to FALSE. To maximize the input noise margin, all R<sub>BIAST</sub> resistors should be the same value, essentially creating a V<sub>CM</sub> level of V<sub>CCO</sub>/2. Resistors in the 10k to 100 k $\Omega$  range are recommended. The typical values for the AC coupling capacitors C<sub>AC</sub> are in the range of 100 nF. All components should be placed physically close to the FPGA inputs.

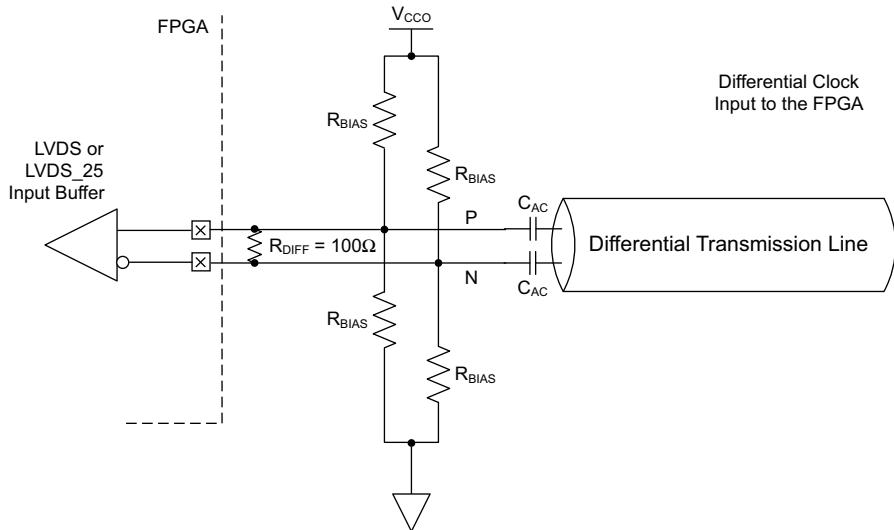


Figure 1-95:

**Note:** The last set of guidelines on differential LVDS inputs are added within the LVDS and LVDS\_25 (Low Voltage Differential Signaling) section of the *7 Series SelectIO Resources User Guide* (UG471) [Ref 2] in the next release of the document.

These guidelines are irrespective of Package, Column (HR/HP), or I/O Voltage.

### Sharing sys\_clk between Controllers

The MIG 7 series FPGA designs require **sys\_clk** to be in the same I/O bank column as the memory interface to minimize jitter.

- **Interfaces Spanning I/O Columns** – A single **sys\_clk** input cannot drive memory interfaces spanning multiple I/O columns. The input clock input must be in the same column as the memory interface to drive the PLL using the CMT Backbone, which minimizes jitter.
- **Interfaces in Single I/O Column** – If the memory interfaces are entirely contained within the same I/O column, a common **sys\_clk** can be shared among the interfaces. The **sys\_clk** can be input on any CCIO in the column where the memory interfaces are located. This includes CCIO in banks that do not contain the memory interfaces, but must be in the same column as the memory interfaces.

## *Information on Sharing BUFG Clock (phy\_clk)*

The MIG 7 series DDR3 design includes an MMCM which outputs the **phy\_clk** on a BUFG route. It is not possible to share this clock amongst multiple controllers to synchronize the user interfaces. This is not allowed because the timing from the FPGA logic to the PHY Control block must be controlled. This is not possible when the clock is shared amongst multiple controllers. The only option for synchronizing user interfaces amongst multiple controllers is to create an asynchronous FIFO for clock domain transfer.

## *Information on Sync\_Pulse*

The MIG 7 series DDR3/DDR2 design includes one PLL that generates the necessary design clocks. One of these outputs is the **sync\_pulse**. The sync pulse clock is 1/16 of the **mem\_refclk** frequency and must have a duty cycle distortion of 1/16 or 6.25%. This clock is distributed across the low skew clock backbone and keeps all PHASER\_IN/\_OUT and PHY\_Control blocks in sync with each other. The signal is sampled by the **mem\_refclk** in both the PHASER\_INs/\_OUTs and PHY\_Control blocks. The phase, frequency, and duty cycle of the **sync\_pulse** is chosen to provide the greatest setup and hold margin across PVT.

[Table 1-69](#) 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 V<sub>REF</sub> is used in this example.

*Table 1-69:*

1	VRP	-	SE	49	-
1	DQ15	D_11	P	48	-
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	DQS-P

*Table 1-69:*
*(Cont'd)*

1	DQS1_N	D_06	N	43	DQS-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	DQSO_P	C_07	P	32	DQS-P
1	DQSO_N	C_06	N	31	DQS-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	DQS-P
1	CK_N	B_06	N	19	DQS-N
1	BA1	B_05	P	18	-
1	BA0	B_04	N	17	-
1	CS_N	B_03	P	16	-
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	-

**Table 1-69:**
*(Cont'd)*

1	A8	A_08	N	9	-
1	A7	A_07	P	8	DQS-P
1	A6	A_06	N	7	DQS-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	AO	A_00	N	1	-
1	VRN	-	SE	0	-

**Table 1-70** shows an example of a 32-bit DDR3 interface contained within two banks. This example uses 2 Gb x8 components.

**Table 1-70:**

1	VRP	-	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	DQS-P
1	-	D_06	N	43	DQS-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	-
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	DQS-P

1	-	C_06	N	31	DQS-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	DQS-P
1	CK_N	B_06	N	19	DQS-N
1	BA1	B_05	P	18	-
1	BAO	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	DQS-P
1	A6	A_06	N	7	DQS-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	AO	A_00	N	1	-
1	VN	1	A		Q

*Table 1-70:*
*(Cont'd)*

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	DQS-P
2	DQS3_N	D_06	N	43	DQS-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	DQS-P
2	DQS2_N	C_06	N	31	DQS-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
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	DQS-P
2	DQS1_N	B_06	N	19	DQS-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	-

*Table 1-70:*
*(Cont'd)*

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	DQSO_P	A_07	P	8	DQS-P
2	DQSO_N	A_06	N	7	DQS-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	-	SE	0	-

*Table 1-71* shows an example of a 64-bit DDR3 interface contained within three banks. This example uses four 2 Gb x16 components.

*Table 1-71:*

1	VRP	-	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	DQS-P
1	DQS7_N	D_06	N	43	DQS-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	-

*Table 1-71:*
*(Cont'd)*

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	DQS-P
1	DQS6_N	C_06	N	31	DQS-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	DQS-P
1	DQS5_N	B_06	N	19	DQS-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	-
1	DQ36	A_08	N	9	-
1	DQS4_P	A_07	P	8	DQS-P
1	DQS4_N	A_06	N	7	DQS-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	-

*Table 1-71:*
*(Cont'd)*

1	-	A_00	N	1	-
1	VRN	-	SE	0	-
2	VRP	-	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	DQS-P
2	-	D_06	N	43	DQS-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	DQS-P
2	-	C_06	N	31	DQS-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	DQS-P
2	CK_N	B_06	N	19	DQS-N
2	BA1	B_05	P	18	-

*Chapter 1:*

*Table 1-71:*
*(Cont'd)*

3	DQ20	C_08	N	33	-
3	DQS2_P	C_07	P	32	DQS-P
3	DQS2_N	C_06	N	31	DQS-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	DQS-P
3	DQS1_N	B_06	N	19	DQS-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	-
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	DQSO_P	A_07	P	8	DQS-P
3	DQSO_N	A_06	N	7	DQS-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	-	SE	0	-

[Table 1-72](#) 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 **SO#** as **CS\_N** for consistency with the component design examples in [Table 1-69, page 213](#), [Table 1-70, page 215](#), and [Table 1-71, page 218](#).

*Table 1-72:*

1	VRP	-	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	DQS-P
1	DQS7_N	D_06	N	43	DQS-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	DQS-P
1	DQS6_N	C_06	N	31	DQS-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



*Table 1-72:*
*(Cont'd)*

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	DQS-P
2	-	C_06	N	31	DQS-N
2	-	C_05	P	30	-
2	-	C_04	N	29	-
2	-	C_03	P	28	CCIO-P
2	ODTO	C_02	N	27	CCIO-N
2	CKEO	C_01	P	26	CCIO-P
2	CS_NO	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	DQS-P
2	CK_N	B_06	N	19	DQS-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	DQS-P
2	A6	A_06	N	7	DQS-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	-

*Table 1-72:*
*(Cont'd)*

2	A1	A_01	P	2	-
2	AO	A_00	N	1	-
2	VRN	-	SE	0	-
3	VRP	-	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	DQS-P
3	DQS3_N	D_06	N	43	DQS-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	DQS-P
3	DQS2_N	C_06	N	31	DQS-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	DQS-P
3	DQS1_N	B_06	N	19	DQS-N

*Table 1-72:*
*(Cont'd)*

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	-
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	DQSO_P	A_07	P	8	DQS-P
3	DQSO_N	A_06	N	7	DQS-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	-	SE	0	-

---

Calibration failures and data errors can occur for many reasons and the debug of these errors can be time consuming. This section is intended to provide a clear step-by-step debug process to quickly identify the root cause of the failure and move to resolution.

To focus the debug of calibration or data errors, use the provided MIG Example Design on the targeted board with the Debug Feature enabled through the MIG 7 series GUI. The latest MIG 7 series release should be used to generate the Example Design.

To help in the design and debug process when using the MIG IP core, the [Xilinx Support web page](#) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

### ***Documentation***

This product guide is the main document associated with the MIG IP core. This guide, along with documentation related to all products that aid in the design process, can be found on the [Xilinx Support web page](#) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the [Downloads page](#). For more information about this tool and the features available, open the online help after installation.

### ***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 the MIG IP core core is located at the [Xilinx MIG Solution Center](#).

### ***Answer Records***

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can also be located by using the Search Support box on the main [Xilinx support web page](#). To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

### **Answer Record for the DDR2/DDR3 Cores Generated by MIG IP core**

AR: [54025](#) for Vivado

### ***Technical Support***

Xilinx provides technical support at [Xilinx support web page](#) for this product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the [Xilinx Support web page](#).

**Note:** Access to WebCase is not available in all cases. Log in to the WebCase tool to see your specific support options.

There are many tools available to address MIG IP core design issues. It is important to know which tools are useful for debugging various situations.

### ***Example Design***

Generation of a DDR2 or DDR3 design through the MIG 7 series tool produces an example design and a user design. The example design includes a synthesizable test bench with a traffic generator that is fully verified in simulation and hardware. This example design can be used to observe the behavior of the MIG 7 series design and can also aid in identifying board-related issues. For complete details on the example design, see the [Quick Start Example Design, page 65](#). This section describes using the example design to perform hardware validation.

## Debug Signals

The MIG 7 series 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 Vivado logic analyzer feature. Selecting this option port maps the debug signals to ILA VIO cores in the design top module. For details on enabling this debug feature, see the “[Using MIG in the Vivado Design Suite, page 21](#)”. The debug port is disabled for functional simulation and can only be enabled if the signals are actively driven by the user design.

**Note:** “Debug Signals” are not available when using IP integrator but Integrated Logic Analyzer (ILA) insertion is still available on the synthesized DCP. For more information, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [\[Ref 16\]](#).

## Vivado Design Suite Debug Feature

The Vivado Design Suite debug feature inserts ILA 3.0 and VIO 3.0 directly into your design. The debug feature also allows you to set trigger conditions to capture application and MIG debug signals in hardware. Captured signals can be analyzed through the Vivado logic analyzer feature. For more information about the Vivado logic see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [\[Ref 16\]](#).



**IMPORTANT:** *The ILA operates on a synchronous clock and cannot be triggered during reset. Instead, set the trigger on an ILA signal to look for a rising edge (“R”) or falling edge (“F”) with the radix value of the signal set to “Binary.” With this trigger setting, the trigger can be armed. When the reset is applied and released, the trigger captures the desired ILA results.*

The Vivado logic analyzer feature snapshot is shown in [Figure 1-96](#). The [Hardware Debug](#) section has a snapshot of the older analyzer version but the debugging steps and data to be captured remain the same.

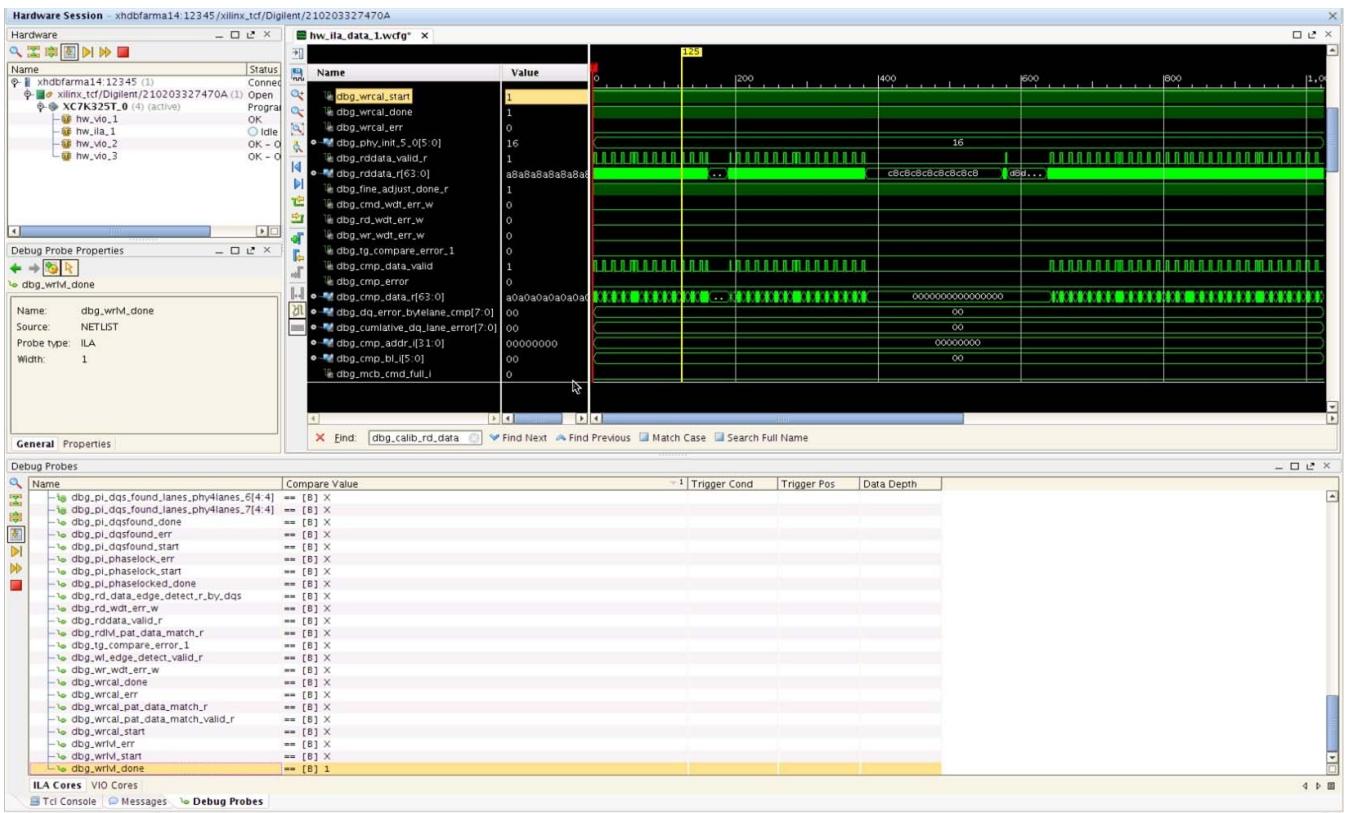


Figure 1-96:

## Reference Boards

Various Xilinx development boards support MIG IP core that include FPGA interfaces to a DDR SODIMM. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation boards
  - VC707
  - KC705
  - AC701

Hardware issues can range from calibration failures to issues seen after hours of testing. This section provides debug steps for common issues. The Vivado logic analyzer feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado logic analyzer feature for debugging the specific issues.

Many of these common issues can also be applied to debugging design simulations. Details are provided on:

- [General Checks](#)
- [Calibration Stages](#)
- [Determine the Failing Calibration Stage](#)
- [Debug Signals](#)
- [Debugging PHASER\\_IN PHASELOCKED Calibration Failures \(dbg\\_pi\\_phaselock\\_err = 1\)](#)
- [Debugging PHASER\\_IN DQSFOUND Calibration Failures \(dbg\\_pi\\_dqsfound\\_err = 1\)](#)
- [Debugging Write Leveling Failures \(dbg\\_wrlvl\\_err = 1\)](#)
- [Debugging MPR Read Leveling Failures – DDR3 Only \(dbg\\_rdlvl\\_err\[1\] = 1\)](#)
- [Debugging OCLKDELAYED Calibration Failures](#)
- [Debugging Write Calibration Failures \(dbg\\_wrcal\\_err = 1\)](#)
- [Debugging Read Leveling Failures \(dbg\\_rdlvl\\_err\[0\] = 1\)](#)
- [Debugging PRBS Read Leveling Failures](#)
- [Calibration Times](#)
- [Debugging Data Errors](#)

### ***General Checks***

This section details the list of general checks, primarily board level, which need to be verified before moving forward with the debug process. Strict adherence to the proper board design is critical in working with high speed memory interfaces.

- Ensure all guidelines referenced in the [Design Guidelines](#) have been followed. The [Design Guidelines](#) section includes information on trace matching, PCB Routing, noise, termination, I/O Standards, and pin/bank requirements. Adherence to these guidelines, along with proper board design and signal integrity analysis, is critical to the success of high-speed memory interfaces.
- Measure all voltages on the board during idle and non-idle times to ensure the voltages are set appropriately and noise is within specifications.
  - Ensure the termination voltage regulator ( $V_{tt}$ ) is turned on (set to 0.75V).
  - Ensure  $V_{REF}$  is measured.
- When applicable, check VRN/VRP resistors. Note the values are not the same as Virtex-6 FPGA.
- Look at the clock inputs to ensure they are clean.

- Check the reset to ensure the polarity is correct and the signal is clean. The reset signal must be applied for a minimum pulse width of 5 ns.
- Check terminations by using this user guide as a guideline.
- Perform general signal integrity analysis.
  - IBIS simulations should run to ensure terminations, ODT, and output drive strength settings are appropriate.
  - Observe **DQ/DQS** on a scope at the memory. View the alignment of the signals and analyze the signal integrity during both writes and reads.
  - Observe the Address and Command signals on a scope at the memory. View the alignment and analyze the signal integrity.
- Verify the memory parts on the board(s) in test are the correct part(s) set through the MIG tool. The timing parameters and signals widths (that is, address, bank address) must match between the RTL and physical parts. Read/write failures can occur due to a mismatch.
- Verify SDRAM pins are behaving correctly. Look for floating or grounded signals. It is rare, but manufacturing issues with the memory devices can occur and result in calibration failures.
- If Data Mask (DM) is not being used, ensure DM is tied Low at the memory with the appropriate termination as noted in the memory data sheet.
- Measure the **CK/CK\_n**, **DQS/DQS\_n**, and system clocks for duty cycle distortion and general signal integrity.
- If internal V<sub>REF</sub> is used, ensure the constraints are set appropriately according to the Xilinx Constraints Guide. When the constraints are applied properly, a note similar to the following appears in the **.bgn** BitGen report file:
  - There were two CONFIG constraint(s) processed from example\_top.pcf.  
**CONFIG INTERNAL\_VREF\_BANK12 = 0.75**  
**CONFIG INTERNAL\_VREF\_BANK14 = 0.75**
- Check the **iodelay\_ctrl** ready signal.
- Check the PLL lock.
- Check the **phaser\_ref** lock signal.
- Bring the **init\_calib\_complete**

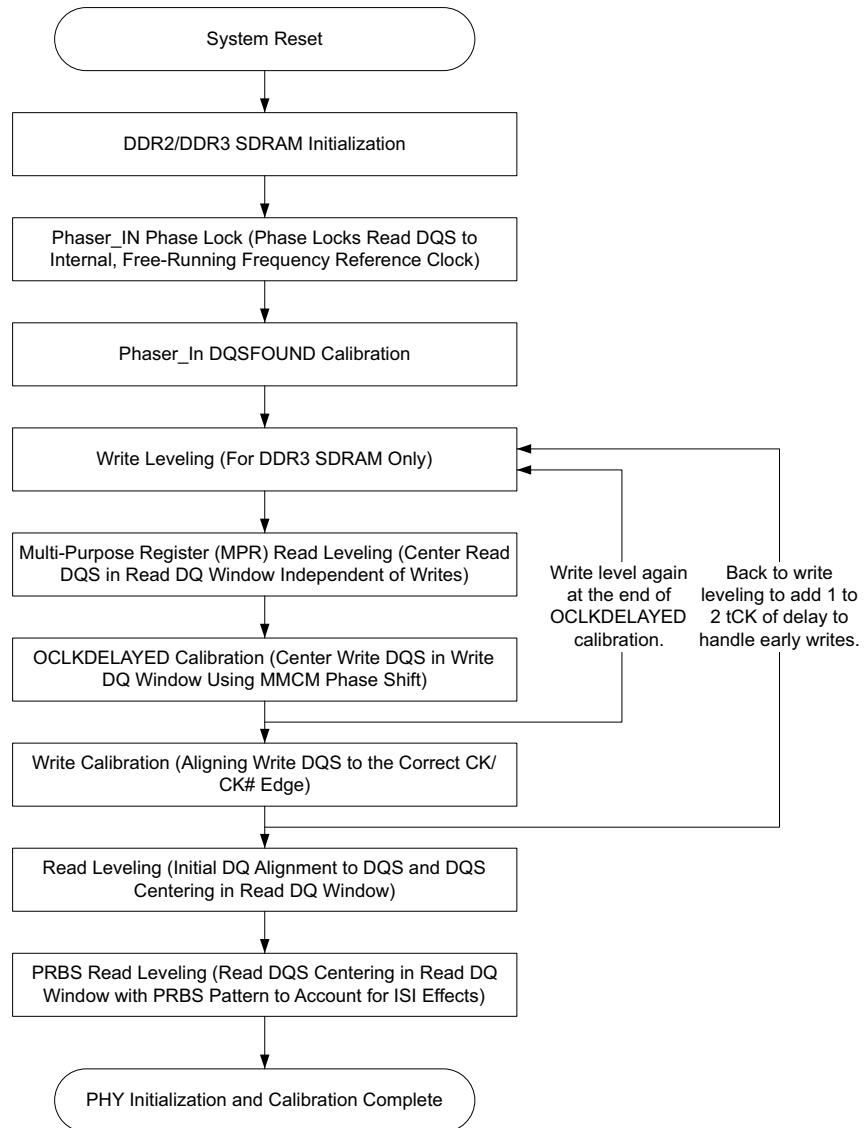
*Calibration Stages*

Figure 1-97:

The PHY executes a JEDEC-compliant DDR2 or DDR3 initialization sequence following the deassertion of system reset. Each DDR2 or DDR3 SDRAM has a series of mode registers accessed through 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 MIG 7 series designs does not issue a calibration failure during Memory Initialization.

All other initialization/calibration stages are reviewed in the following Debugging Calibration Stages section.

## Determine the Failing Calibration Stage

Using the Vivado logic analyzer feature, configure the device along with `debug_nets.ltx` file. This file can be found in the `example.runs\impl_1\` directory after implementation completes with `Debug_port` feature enabled. Observe the following debug signals in the provided ILA core. This indicates which calibration stage failed:

*Table 1-73:*

init_calib_complete	Signifies memory initialization and calibration have completed successfully.
dbg_wrlvl_start	Signifies the start of the Write Leveling stage of calibration.
dbg_wrlvl_done	Signifies successful completion of the Write Leveling stage of calibration.
dbg_wrlvl_err	Signifies the Write Leveling stage of calibration exhibited errors and did not complete.
dbg_pi_phaselock_start	Signifies the start of the PHASELOCK stage of calibration.
dbg_pi_phaselock_done	Signifies successful completion of the PHASELOCK stage of calibration.
dbg_pi_phaselock_err	Signifies the PHASELOCK stage of calibration exhibited errors and did not complete.
dbg_pi_dqsfound_start	Signifies the start of the DQSFOUND stage of calibration.
dbg_pi_dqsfound_done	Signifies successful completion of the DQSFOUND stage of calibration.
dbg_pi_dqsfound_err	Signifies the DQSFOUND stage of calibration exhibited errors and did not complete.
dbg_rdlvl_start[0]	Signifies the start of Read Leveling Stage 1 calibration.
dbg_rdlvl_start[1]	Signifies the start of the MPR stage of calibration.
dbg_rdlvl_done[0]	Signifies the successful completion of Read Leveling Stage 1 calibration.
dbg_rdlvl_done[1]	Signifies the successful completion of the MPR Stage of calibration.
dbg_rdlvl_err[0]	Signifies Read Leveling Stage 1 calibration exhibited errors and did not complete.
dbg_rdlvl_err[1]	Signifies the MPR stage of calibration exhibited errors and did not complete.
dbg_ockdelay_calib_start	Signifies the start of the OCLKDELAY stage of calibration.
dbg_ockdelay_calib_done	Signifies successful completion of the OCLKDELAY stage of calibration.
dbg_wrcal_start	Signifies the start of the Write Calibration stage of calibration.
dbg_wrcal_done	Signifies successful completion of the Write Calibration stage of calibration.
dbg_wrcal_err	Signifies Write Calibration exhibited errors and did not complete.

*Debug Signals*

Table 1-74:

dbg_init_calib_complete	Signifies memory initialization and calibration have completed successfully.
dbg_wrlvl_start	Signifies the start of the Write Leveling stage of calibration.
dbg_wrlvl_done	Signifies successful completion of the Write Leveling stage of calibration.
dbg_wrlvl_err	Signifies the Write Leveling stage of calibration exhibited errors and did not complete.
dbg_pi_phaselock_start	Signifies the start of the PHASELOCK stage of calibration.
dbg_pi_phaselocked_done	Signifies successful completion of the PHASELOCK stage of calibration.
dbg_pi_phaselock_err	Signifies the PHASELOCK stage of calibration exhibited errors and did not complete.
dbg_pi_dqsfound_start	Signifies the start of the DQSFOUND stage of calibration.
dbg_pi_dqsfound_done	Signifies successful completion of the DQSFOUND stage of calibration.
dbg_pi_dqsfound_err	Signifies the DQSFOUND stage of calibration exhibited errors and did not complete.
dbg_rdlvl_start[1]	Signifies the start of the MPR stage of calibration.
dbg_rdlvl_start[0]	Signifies the start of Read Leveling Stage 1 calibration.
dbg_rdlvl_done[1]	Signifies the successful completion of the MPR Stage of calibration.
dbg_rdlvl_done[0]	Signifies the successful completion of Read Leveling Stage 1 calibration.
dbg_rdlvl_err[1]	Signifies Read Leveling Stage 1 calibration exhibited errors and did not complete
dbg_rdlvl_err[0]	Signifies the MPR stage of calibration exhibited errors and did not complete.
dbg_ockdelay_calib_start	Signifies the start of the OCLKDELAY stage of calibration.
dbg_ockdelay_calib_done	Signifies successful completion of the OCLKDELAY stage of calibration.
dbg_wrcal_start	Signifies the start of the Write Calibration stage of calibration.
dbg_wrcal_done	Signifies successful completion of the write calibration stage of calibration.
dbg_wrcal_err	Signifies write calibration exhibited errors and did not complete.
dbg_phy_init_5_0	State variable for the PHY Init state machine. States can be decoded in the ddr_phy_init module.
dbg_rddata_valid_r	Asserts when the read data (dbg_rddata_r) is valid.

Table 1-74:

(Cont'd)

dbg_rddata_r	Read data read out of the IN_FIFO for the DQS group selected through dbg_dqs on the VIO. This is a 64-bit bus. This debug port does not capture ECC data.
dbg_fine_adjust_done_r	Asserts after fine adjustment is completed in DQS found calibration.
dbg_cmd_wdt_err_w	Watch dog timeout error from Traffic Generator when the user interface is not processing the command from traffic generator.
dbg_rd_wdt_err_w	Watch dog timeout error from Traffic Generator when no read data is available from the user interface.
dbg_wr_wdt_err_w	Watch dog timeout error from Traffic Generator when no write data is taken by the user interface.
dbg_tg_compare_error	Sticky bit from the internal Traffic Generator asserted when a data error is found after calibration is completed.
dbg_cmp_data_valid	Valid signal showing that the compare data from Traffic Generator is valid.
dbg_cmp_error	Asserts when compare data is not matching the read data from User Interface.
dbg_cmp_data_r	Register version of compare data from the Traffic Generator.
dbg_dq_error_bytelane_cmp	Indicates which byte has data comparison error for the Traffic Generator.
dbg_cumulative_dq_lane_error	Indicates which byte has data comparison error for the Traffic Generator. This is a sticky status signal and stays asserted until cleared manually using the dbg_clear_error.
dbg_cmp_addr_i	The start address of the burst for which the first data error occurred.
dbg_cmp_bl_i	Burst length of the burst for which the first data error occurred.
dbg_mcb_cmd_full_i	Memory Controller command FIFO full status when the first data error occurred
dbg_mcb_wr_full_i	Memory Controller write data FIFO full status when the first data error occurred.
dbg_mcb_rd_empty_i	Memory Controller read data FIFO empty status when the first data error occurred.
dbg_ddrx ila_rdpAth_765_764[0]	Signifies PRBS Read Level Stage Start
dbg_ddrx ila_rdpAth_765_764[1]	Signifies PRBS Read Level Stage Done
dbg_wl_state_r	State variable for the Write Leveling State Machine. States can be decoded in the ddr_phy_wrlvl.v module.
dbg_dqs_cnt_r	Signifies the DQS byte group being calibrated during Write Leveling. The algorithm sequentially steps through the DQS byte groups until Write Leveling completes successfully or a data byte group fails due a 0 to 1 transition not being detected on DQ.
dbg_wl_edge_detect_valid_r	Signifies valid time Write Leveling algorithm is searching for edge.
dbg_rd_data_edge_detect_r_by_dqs	Signifies Write Leveling calibration found the 0 to 1 edge transition.

Table 1-74:

(Cont'd)

dbg_wl_po_fine_cnt_by_dqs	PHASER_OUT Fine Taps found during Write Leveling. Byte capture based on VIO dbg_dqs setting.
dbg_wl_po_coarse_cnt_by_dqs	PHASER_OUT Coarse Taps found during Write Leveling. Byte capture based on VIO dbg_dqs setting.
dbg_phy_oclkdelay_zfo	OCLKDELAY edge found indicator. {z2f, o2f, f2z, f2o}.
dbg_ocal_fuzz2oneeighty	Stage 3 tap value of the left-edge of the fall window.
dbg_ocal_fuzz2zero	fuzz2zero Stage 3 tap value of the left-edge of the rise window.
dbg_ocal_oneeighty2fuzz	oneeighty2fuzz Stage 3 tap value of the right-edge of the fall window.
dbg_ocal_zero2fuzz	Stage 3 tap value of the right-edge of the rise window.
dbg_ocal_oclkdelay_calib_cnt	DQS group counter indicates which DQS group is in OCLKDELAY calibration.
dbg_ocal_scan_win_not_found	Indicator that window is not found during OCLKDELAY calibration.
dbg_wrcal_pat_data_match_r	Asserts when the data pattern match is found during Write Calibration stage.
dbg_wrcal_pat_data_match_valid_r	Acts as a qualifier and asserts when the data pattern match is valid during Write Calibration stage.
dbg_wrcal_dqs_cnt_r	Current DQS group being calibrated in Write Calibration. When wrcal_start asserts, wrcal_dqs_cnt_r is 0. The algorithm sequentially steps through the DQS byte groups checking to see if the read data pattern matches the expected FFOOAA5555AA9966 pattern. If the pattern matches, wrcal_dqs_cnt increments by 1. The algorithm then starts looking for the correct data pattern on the next byte until it reaches DQS_WIDTH – 1 or a data byte group fails due to the data pattern not being detected properly. The wrcal_dqs_cnt stays at DQS_WIDTH – 1 after wrcal_done signal is asserted.
cal2_state_r	Write Calibration state machine variable. States can be decoded in the ddr_phy_wrcal.v module.
not_empty_wait_cnt	Count value during Write Calibration pattern detection. Maximum count is 0x1F. If count reaches 0x1F, write calibration fails with the assertion of dbg_wrcal_err.
dbg_early1_data	Asserts when the pattern detected is one CK clock cycle early. When this is asserted, the Write Leveling algorithm moves the CK clock one cycle. After CK is moved, the Write Calibration algorithm restarts pattern detection.
dbg_early2_data	Asserts when the pattern detected is two CK clock cycles early. When this is asserted, the Write Leveling algorithm moves the CK clock two cycles. After CK is moved, the Write Calibration algorithm restarts pattern detection.
dbg_phy_oclkdelay_cal_57_54	Current DQS group being calibrated from OCLK_DELAY calibration stage.
dbg_phy_wrlvl_128_75	PHASER_OUT Fine Taps found during Write Leveling for all bytes
dbg_phy_wrlvl_155_129	PHASER_OUT Coarse Taps found during Write Leveling for all bytes.

Table 1-74:

(Cont'd)

dbg_pi_phase_locked_phy4lanes	<p>Signifies which of the PHASER_IN lanes has achieved lock. It is a 12-bit bus, three nibble data.</p> <p>Each nibble corresponds to a bank information. Uppermost Data or Address/Control byte group selected bank is referred to as Bank0, this corresponds to nibble 0 or Bits[3:0] of the bus. Numbering of banks is 0, 1, and 2 from top to bottom. Bank1 corresponds to nibble 1 or Bits[7:4] of the bus. Bank2 corresponds to nibble 2 or Bits[11:8] of the bus.</p> <p>LSB to MSB bits in each nibble corresponds to T3 to T0 byte lane information of the corresponding bank.</p> <p>For example, Nibble 0, Bit[3] corresponds to T0 byte lane, Bit[2] corresponds to T1 byte lane, Bit[1] corresponds to T2 byte lane, Bit[0] corresponds to T3 byte lane information.</p>
dbg_pi_dqs_found_lanes_phy4lanes	<p>Signifies which of the PHASER_IN lanes is able to find the DQS. It is a 12-bit bus, three nibble data.</p> <p>Each nibble corresponds to a bank information. Uppermost Data or Address/Control byte group selected bank is referred to as Bank0, this corresponds to nibble 0 or Bits[3:0] of the bus. Numbering of banks is 0, 1, and 2 from top to bottom. Bank1 corresponds to nibble 1 or Bits[7:4] of the bus. Bank2 corresponds to nibble 2 or Bits[11:8] of the bus.</p> <p>LSB to MSB bits in each nibble corresponds to T3 to T0 byte lane information of the corresponding bank.</p> <p>For example, Nibble 0, Bit[3] corresponds to T0 byte lane, Bit[2] corresponds to T1 byte lane, Bit[1] corresponds to T2 byte lane, Bit[0] corresponds to T3 byte lane information.</p>
dbg_rd_data_offset	Read data offset found during calibration.
dbg_cal1_state_r	State machine variable for MPR and Read Leveling Stage 1. States can be decoded in the ddr_phy_rdlvl.v module.
dbg_cal1_cnt_cpt_r	Signifies the byte that failed MPR Read Leveling or Read Leveling Stage 1.
dbg_mux_rd_rise0_r	Data pattern received on rising edge 0.
dbg_mux_rd_fall0_r	Data pattern received on falling edge 0.
dbg_mux_rd_rise1_r	Data pattern received on rising edge 1.
dbg_mux_rd_fall1_r	Data pattern received on falling edge 1.
dbg_mux_rd_rise2_r	Data pattern received on rising edge 2.
dbg_mux_rd_fall2_r	Data pattern received on falling edge 2.
dbg_mux_rd_rise3_r	Data pattern received on rising edge 3.
dbg_mux_rd_fall3_r	Data pattern received on falling edge 3.
dbg_rdlvl_pat_data_match_r	Asserts when the valid pattern is detected on the data and is found to match with the expected pattern sent during read leveling.

Table 1-74:

(Cont'd)

dbg_mux_rd_valid_r	Asserts when the valid pattern is detected on dbg_mux_rd_rise0_r, dbg_mux_rd_fall0_r, dbg_mux_rd_rise1_r, dbg_mux_rd_fall1_r, dbg_mux_rd_rise2_r, dbg_mux_rd_fall2_r, dbg_mux_rd_rise3_r, and dbg_mux_rd_fall3_r.
dbg_cpt_first_edge_cnt_by_dqs	Signifies PHASER_IN fine tap count when the first edge in MPR and Read Leveling Stage 1 is found. Byte capture based on VIO dbg_dqs setting.
dbg_cpt_second_edge_cnt_by_dqs	Signifies PHASER_IN fine tap count when the second edge in MPR and Read Leveling Stage 1 is found. Byte capture based on VIO dbg_dqs setting.
dbg_cpt_tap_cnt_by_dqs	Signifies the center tap moved to based on when the first and second edges were found. Byte capture based on VIO dbg_dqs setting.
dbg_dq_idelay_tap_cnt_by_dqs	IDELAY tap value for MPR and Read Leveling Stage 1. This should be within 2 to 3 taps across all DQS byte groups. Byte capture based on VIO dbg_dqs setting.
dbg_dbg_calib_rd_data_offset_1	Read data offset found during calibration.
dbg_dbg_calib_rd_data_offset_2	Read data offset found during calibration.
dbg_data_offset	Data Offset used during normal operation. Value changes during writes, reads, and idle. During writes, it is CWL + 2+slot#. During non-data commands, it is 0. During reads, it should match what was found during DQSFOUND calibration (rd_data_offset_ranks).
dbg_data_offset_1	Data Offset used during normal operation. Value changes during writes, reads, and idle. During writes, it is CWL + 2+slot#. During non-data commands, it is 0. During reads, it should match what was found during DQSFOUND calibration (rd_data_offset_ranks).
dbg_data_offset_2	Data Offset used during normal operation. Value changes during writes, reads, and idle. During writes, it is CWL + 2+slot#. During non-data commands, it is 0. During reads, it should match what was found during DQSFOUND calibration (rd_data_offset_ranks).
dbg_cpt_first_edge_cnt	Signifies PHASER_IN fine tap count when the first edge in MPR and Read Leveling Stage 1 is found.
dbg_cpt_second_edge_cnt	Signifies PHASER_IN fine tap count when the second edge in MPR and Read Leveling Stage 1 is found.
dbg_cpt_tap_cnt	Center PHASER_IN fine tap value in MPR or Read Leveling Stage 1 is found.
dbg_dq_idelay_tap_cnt	IDELAY tap value for MPR and Read Leveling Stage 1. This should be within 2 to 3 taps across all DQS byte groups. Byte capture based on VIO dbg_dqs setting.
dbg_prbs_rdlvl	Debug signals of PRBS Read Level Stage.
dbg_ocal_lim_done	Indicates that stage 3 lower and upper limits have been determined.
dbg_ocal_stg3_lim_left	Stage 3 lower limit.
dbg_ocal_stg3_lim_right	Stage 3 upper limit.
dbg_ocal_center_calib_start	OCLKDELAY center calibration start indicator.

Table 1-74:

(Cont'd)

dbg_wcal_mux_rd_rise0_r	Data pattern received on the Write Calibration stage MUX on rising edge 0.
dbg_wcal_mux_rd_fall0_r	Data pattern received on the Write Calibration stage MUX on falling edge 0.
dbg_wcal_mux_rd_rise1_r	Data pattern received on the Write Calibration stage MUX on rising edge 1.
dbg_wcal_mux_rd_fall1_r	Data pattern received on the Write Calibration stage MUX on falling edge 1.
dbg_wcal_mux_rd_rise2_r	Data pattern received on the Write Calibration stage MUX on rising edge 2.
dbg_wcal_mux_rd_fall2_r	Data pattern received on the Write Calibration stage MUX on falling edge 2.
dbg_wcal_mux_rd_rise3_r	Data pattern received on the Write Calibration stage MUX on rising edge 3.
dbg_wcal_mux_rd_fall3_r	Data pattern received on the Write Calibration stage MUX on falling edge 3.
dbg_early1_data_match_r	Asserts when the pattern detected is one CK clock cycle early and a match is found during Write Calibration.
dbg_early2_data_match_r	Asserts when the pattern detected is two CK clock cycle early and a match is found during Write Calibration.
dbg_wcal_sanity_pat_data_match_valid_r	Asserts when the valid pattern is detected on the data and is found to match with the expected pattern sent during Write Calibration sanity check.
dbg_prbs_final_dqs_tap_cnt_r	Tap values set at the end of PRBS Read Leveling stage.
dbg_prbs_first_edge_taps	Tap value when the first edge is found during PRBS Read Leveling stage calibration.
dbg_prbs_second_edge_taps	Tap value when the second edge is found during PRBS Read Leveling stage calibration.
dbg_ocal_center_calib_done	OCLKDELAY center calibration completing indicator.
dbg_phy_oclkdelay_cal_taps	Final stage 3 tap values for all the bytes in the interface. Bits[5:0] for byte 0 and Bits[11:6] for byte 1.
dbg_ocal_tap_cnt	Stage 3 tap value during calibration for each group.
dbg_bit	Currently Unused
dbg_dqs	Input to select DQS byte for which the ILA displays the tap counts of PHASER_OUT. For example, set to 4'b0000 to view the results on DQS[0].
vio_modify_enable	See <a href="#">Table 1-83</a> .
vio_data_mode_value	See <a href="#">Table 1-83</a> .
vio_addr_mode_value	See <a href="#">Table 1-83</a> .

**Table 1-74:**
*(Cont'd)*

vio_instr_mode_value	See <a href="#">Table 1-83</a> .
vio_bl_mode_value	See <a href="#">Table 1-83</a> .
vio_fixed_bl_value	See <a href="#">Table 1-83</a> .
vio_data_mask_gen	Enables Traffic Generator Data Mask Generation
vio_pause_traffic	See <a href="#">Table 1-83</a> .
vio_fixed_instr_value	See <a href="#">Table 1-83</a> .
dbg_clear_error	See <a href="#">Table 1-83</a> .
vio_tg_RST	Currently Unused
wdt_en_W	Enable Watch Dog Timer in Traffic Generator
win_start	See <a href="#">Table 1-85</a> .
win_sel_pi_pon	See <a href="#">Table 1-85</a> .
vio_dbg_sel_pi_jncdec	See <a href="#">Table 1-85</a> .
vio_dbg_sel_po_incdec	See <a href="#">Table 1-85</a> .
vio_dbg_pi_f_inc	See <a href="#">Table 1-85</a> .
vio_dbg_pi_f_dec	See <a href="#">Table 1-85</a> .
vio_dbg_po_f_inc	See <a href="#">Table 1-85</a> .
vio_dbg_po_f_dec	See <a href="#">Table 1-85</a> .
vio_dbg_po_f_stg23_sel	See <a href="#">Table 1-85</a> .
vio_win_byte_select_inc	See <a href="#">Table 1-85</a> .
vio_win_byte_select_dec	See <a href="#">Table 1-85</a> .
vio_sel_mux_rdd[3:0]	See <a href="#">Table 1-85</a> .
vio_tg_simple_data_sel	Currently Unused
win_start	Status signal reflecting the logic level of win_start input.
dbg_pi_counter_read_val	See <a href="#">Table 1-85</a> .
pi_win_left_ram_out	See <a href="#">Table 1-85</a> .
pi_win_right_ram_out	See <a href="#">Table 1-85</a> .
win_active	See <a href="#">Table 1-85</a> .
dbg_win_chk	Debug signals from window margin check module. See _chk_win.v file for details.
win_current_bit	Unused for DDR3 Interface
win_current_byte[3:0]	See <a href="#">Table 1-85</a> .
win_byte_select	See <a href="#">Table 1-85</a> .
po_win_left_ram_out	See <a href="#">Table 1-85</a> .
po_win_right_ram_out	See <a href="#">Table 1-85</a> .

Table 1-74:

(Cont'd)

dbg_po_counter_read_val	See <a href="#">Table 1-85</a> .
dbg_mem_pattern_init_done	Signal that indicates initial write to the memory is completed.
dbg_tg_compare_error	Sticky bit indicating the error in data transfer after calibration is done.
dbg_tg_wr_data_counts	Counter for the number of bytes written by the Traffic Generator.
dbg_tg_rd_data_counts	Counter for the number of bytes read by the Traffic Generator.

***Debugging PHASER\_IN PHASELOCKED Calibration Failures  
(dbg\_pi\_phaselock\_err = 1)***

During this stage of calibration, each 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. Each DQS has an associated PHASER\_IN block. **dbg\_pi\_phase\_locked** asserts when all PHASER\_INs have achieved lock and the PHASER\_INs are then placed in normal operation mode.

If PHASER\_IN PHASELOCKED calibration failed, probe the DQS at the memory. A continuous stream of DQS pulses must be seen for lock to occur. Verify the signal integrity of the DQS pulses.

***Debugging PHASER\_IN DQSFOUND Calibration Failures  
(dbg\_pi\_dqsfound\_err = 1)***

In this stage of calibration, the different DQS groups in an I/O bank are aligned to the same **PHY\_CLK** and the optimal read data offset position is found with respect to the read command. The calibration logic issues a set of four back-to-back reads with gaps in between. Each PHASER\_IN detects the read DQS preamble. A single read data offset value is determined for all DQS groups in an I/O bank. The PHASER\_OUT stage 2 delay for CK/Address/Command/Control byte lanes are increased and decreased to improve margin on the read DQS preamble detected. This read data offset is then used during read requests to the PHY\_CONTROL block.

- If the **DQSFOUND** stage fails, probe **DQS** at the memory. Sets of four back-to-back reads should be seen. Read **DQS**(s) is required by the PHASER\_IN(s) to establish the **read\_data\_offset** value. If the design is stuck in the **DQSFOUND** stage, start observing the quality of **DQS** at the memory.
- Look at the **read\_data\_offset** values. There are two sets of **read\_data\_offset** values that need to be compared.
  - To determine the read data offset found at the end of **DQSFOUND** calibration, look at **dbg\_rd\_data\_offset\_0**, **dbg\_calib\_data\_offset\_1** (only when more than one bank is used), **dbg\_calib\_data\_offset\_2** (only when three banks are used).
  - To determine the data offset used during normal operation reads, look at **dbg\_data\_offset**, **dbg\_data\_offset\_1** (only when more than one bank is used), and **dbg\_data\_offset\_2** (only when three banks are used).
    - These signals change between reads, writes, and non-data commands. During writes, the value is CWL + 2 + slot#. During non-data commands, the value is 0. During reads, the value should match what was found during **DQSFOUND** calibration (**dbg\_rd\_data\_offset\_0**, **dbg\_rd\_data\_offset\_1**, and **dbg\_rd\_data\_offset\_2**).
- Compare the read data offset values used during calibration and normal operation reads. These values should match for reads with even CWL and be off by 1 for reads with odd CWL. One additional offset is added for odd CWL values because reads/writes are assigned to slot1 by the Memory Controller, whereas slot0 is used for even CWL.
- The read data offset should be equal to or greater than CL (Read Latency) + 4 or 5 memory cycles of round trip delay on the PCB. For DDR2 interfaces at lower frequencies, it is possible for read data offset to equal CL (Read Latency).
- The PHASER\_OUT stage 2 delay for **CK**/Address/Command/Control byte lanes should also be observed for differences between passing and failing cases. The **CK** PHASER\_OUT stage 2 delay can be observed in Vivado logic analyzer using the **dbg\_po\_counter\_read\_val** signal with **dbg\_pi\_dqsfound\_done** as the trigger.
- When this stage fails (**pi\_dqsfound\_err** = 1), look to see if any of the **dbg\_calib\_rd\_data\_offset/\_1/\_2** have calculated offsets. If not, focus on the **DQS** signals associated with the failing bank by probing each and analyzing the signal integrity.

If **pi\_dqsfound\_err** asserted, denoting a failure during **DQSFOUND** calibration, use **pi\_dqsfound\_err** = R as the trigger. If this stage completed successfully with the asserting of **pi\_dqsfound\_done** = 1, use **pi\_dqsfound\_done** = R as the trigger to analyze how the stage completed.

Look at `dbg_rd_data_offset`, `dbg_calib_rd_data_offset_1`, and `dbg_calib_rd_data_offset_2`, these values should vary by one at the most. Next, compare these values to the values used during normal operation reads on the `dbg_data_offset`, `dbg_data_offset_1` and `dbg_data_offset_2` signals. Record the results in the "7 Series DDR3 Calibration Results" spreadsheet.

Table 1-75:

<code>dbg_pi_dqsfound_start</code>	Signifies the start of the DQSFOUND stage of calibration.
<code>dbg_pi_dqsfound_done</code>	Signifies successful completion of the DQSFOUND stage of calibration.
<code>dbg_pi_dqsfound_err</code>	Signifies the DQSFOUND stage of calibration exhibited errors and did not complete.
<code>dbg_rd_data_offset_0</code>	Read Data Offset found during calibration.
<code>dbg_calib_rd_data_offset_1</code>	Read Data Offset found during calibration.
<code>dbg_calib_rd_data_offset_2</code>	Read Data Offset found during calibration.
<code>dbg_data_offset</code>	Data Offset used during normal operation. Value changes during writes, reads, and idle. During writes, it is CWL+ 2+ slot#. During non-data commands, it is 0. During reads, it should match what was found during DQSFOUND calibration ( <code>rd_data_offset_ranks</code> ).
<code>dbg_data_offset_1</code>	Data Offset used during normal operation. Value changes during writes, reads, and idle. During writes, it is CWL+ 2+ slot#. During non-data commands, it is 0. During reads, it should match what was found during DQSFOUND calibration ( <code>rd_data_offset_ranks</code> ).
<code>dbg_data_offset_2</code>	Data Offset used during normal operation. Value changes during writes, reads, and idle. During writes, it is CWL+ 2+ slot#. During non-data commands, it is 0. During reads, it should match what was found during DQSFOUND calibration ( <code>rd_data_offset_ranks</code> ).

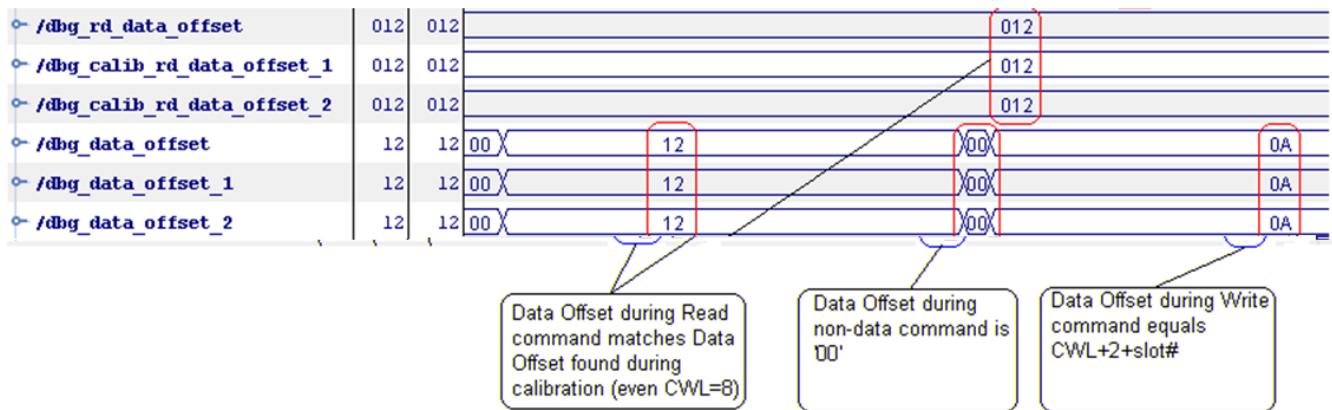


Figure 1-98:

### Debugging Write Leveling Failures (`dbg_wrlvl_err = 1`)

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  $t_{DQSS}$  specification. During this stage, the PHY logic asserts the **Write\_Calib\_N** input to the PHY Control block to indicate the start of write leveling. Periodic write requests are issued to the PHY Control block to generate periodic **DQS** pulses. The **PHASER\_IN** outputs a free-running clock to capture the **DQ** feedback into the **DQ\_IN\_FIFOS**. The **PHASER\_OUT** fine and coarse taps are used to phase shift **DQS** one tap at a time until a 0-to-1 transition is seen on the feedback **DQ**.

Write Leveling is performed at three different points during the calibration process. After memory initialization completes, the **PHASER\_OUT** fine and coarse taps are set to zero. Write Leveling is then initially performed to align **DQS** to **CK**. After OCLKDELAYED calibration completes, the coarse tap values found during the initial Write Leveling are carried over and the fine taps are reset to zero. Write Leveling is performed again to ensure the **DQS**-to-**CK** relationship is still correct.

Finally, during Write Calibration both the fine and coarse delays are carried over and final adjustments are made when necessary. During Write Calibration, the appropriate pattern must be detected. If Write Leveling aligned **DQS** to the wrong **CK** clock, final **PHASER\_OUT** fine/coarse delay adjustments are required to move **DQS** up to two **CK** clock cycles. This section shows how to capture the Write Leveling results after each of these adjustments.

- Verify **DQS** is toggling on the board. The FPGA sends **DQS** during Write Leveling. If **DQS** is not toggling, something is wrong with the setup and the General Checks section of this answer record should be thoroughly reviewed.
- Verify fly-by-routing is implemented correctly on the board.
- Verify **CK** to **DQS** trace routing. The **CK** clocks should be longer than **DQS**. The recommended value for additional total electrical delay on **CK/CK#** relative to **DQS/DQS#** is 150 ps, but any value greater than 0 ps is acceptable.
- The Mode Registers must be properly set up to enable Write Leveling. Specifically, address bit A7 must be correct. If the part chosen in the MIG tool is not accurate or there is an issue with the connection of the address bits on the board, this could be an issue. If the Mode Registers are not set up to enable Write Leveling, the 0-to-1 transition is not seen.

**Note:** For dual rank design when address mirroring is used, address bit A7 is not the same between the two ranks.

- When **dbg\_wrlvl\_err** asserts (equals 1), users must determine during which of the three different stages write leveling is performed the failure occurred. Set the ILA trigger to **dbg\_wrlvl\_err = R** and look at the other "DDR Basic" signals to see which stages completed.
  - a. If only PHASELOCK and **DQSFOUND** completed, the write leveling failure occurred during the initial run through.
  - b. If **dbg\_wrcal\_start** did not assert, the write leveling failure occurred after OCLKDELAYED calibration.
  - c. If **dbg\_wrcal\_start** asserted but **dbg\_wrcal\_done** did not, the write leveling failure occurred during the final run through during Write Calibration.
- When **dbg\_wrlvl\_done** asserts (equals 1) and the results of each Write Leveling stage is of interest, separately use the following three ILA triggers to capture the Write Leveling tap results for each stage. Seeing how Write Leveling completed is useful to see how far apart the taps are for different **DQS** byte groups.
  - a. **dbg\_wrlvl\_done = R**
  - b. **dbg\_wrcal\_start = R**
  - c. **init\_calib\_complete = R**
- To capture the write leveling results at each stage, change/increment **dbg\_dqs** on the VIO and set the appropriate trigger as noted above. Look at the taps results and record in the "7 Series DDR3 Calibration Results" spreadsheet. Later releases of the MIG tool include results for all **DQS** byte groups removing the need to use **dbg\_dqs**.

**Note:** The tap variance across DQS byte groups is quite different due to fly-by routing.

Table 1-76:

dbg_wrlvl_start	Signifies the start of the Write Leveling stage of calibration.
dbg_wrlvl_done	Signifies successful completion of the Write Leveling stage of calibration.
dbg_wrlvl_err	Signifies the Write Leveling stage of calibration exhibited errors and did not complete.
wl_state_r	State variable for the Write Leveling State Machine. States can be decoded in the ddr_phy_wrlvl.v module.
dbg_dqs_cnt_r	Signifies the DQS byte group being calibrated during Write Leveling. The algorithm sequentially steps through the DQS byte groups until write leveling completes successfully or a data byte group fails due a 0 to 1 transition not being detected on DQ.
dbg_wl_edge_detect_valid_r	Signifies valid time Write Leveling algorithm is searching for edge.
dbg_rd_data_edge_detect_r_by_dqs	Signifies Write Leveling calibration found the 0-to-1 edge transition.
dbg_wl_po_fine_cnt_by_dqs	PHASER_OUT Fine Taps found during Write Leveling. Byte capture based on VIO dbg_dqs setting.
dbg_phy_wrlvl_128_75	PHASER_OUT Fine Taps found during Write Leveling.
dbg_wl_po_coarse_cnt_by_dqs	PHASER_OUT Coarse Taps found during Write Leveling. Byte capture based on VIO dbg_dqs setting.
dbg_phy_wrlvl_155_129	PHASER_OUT Coarse Taps found during Write Leveling.

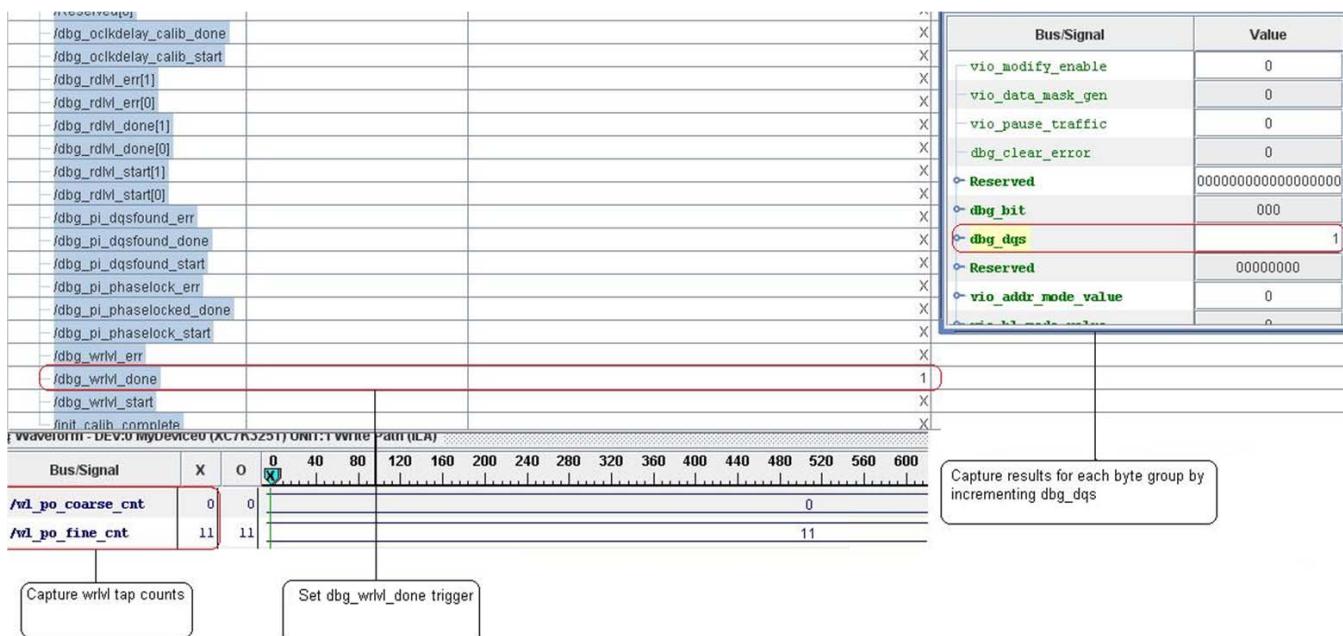


Figure 1-99:

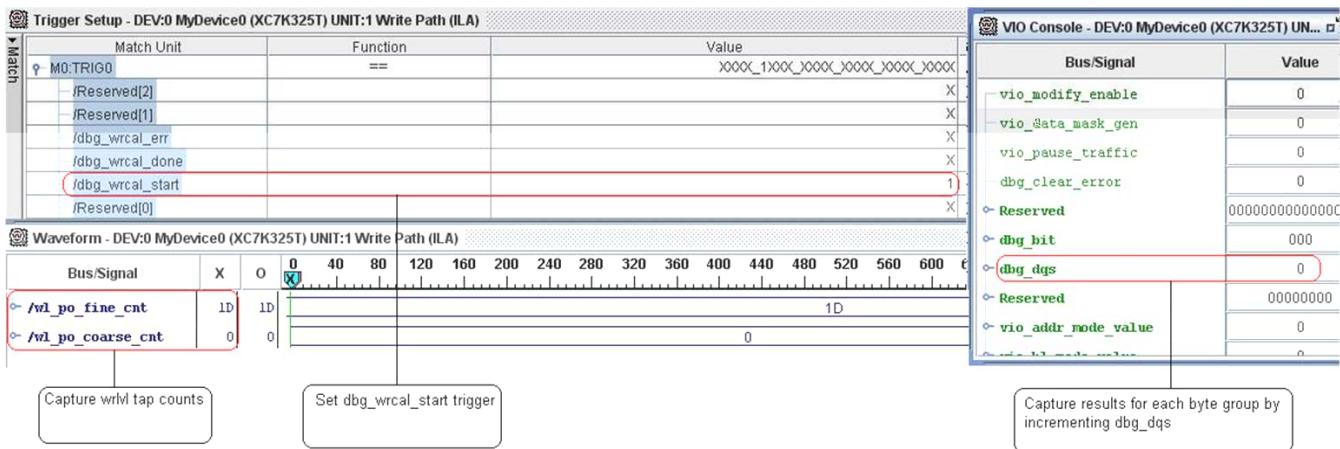


Figure 1-100:

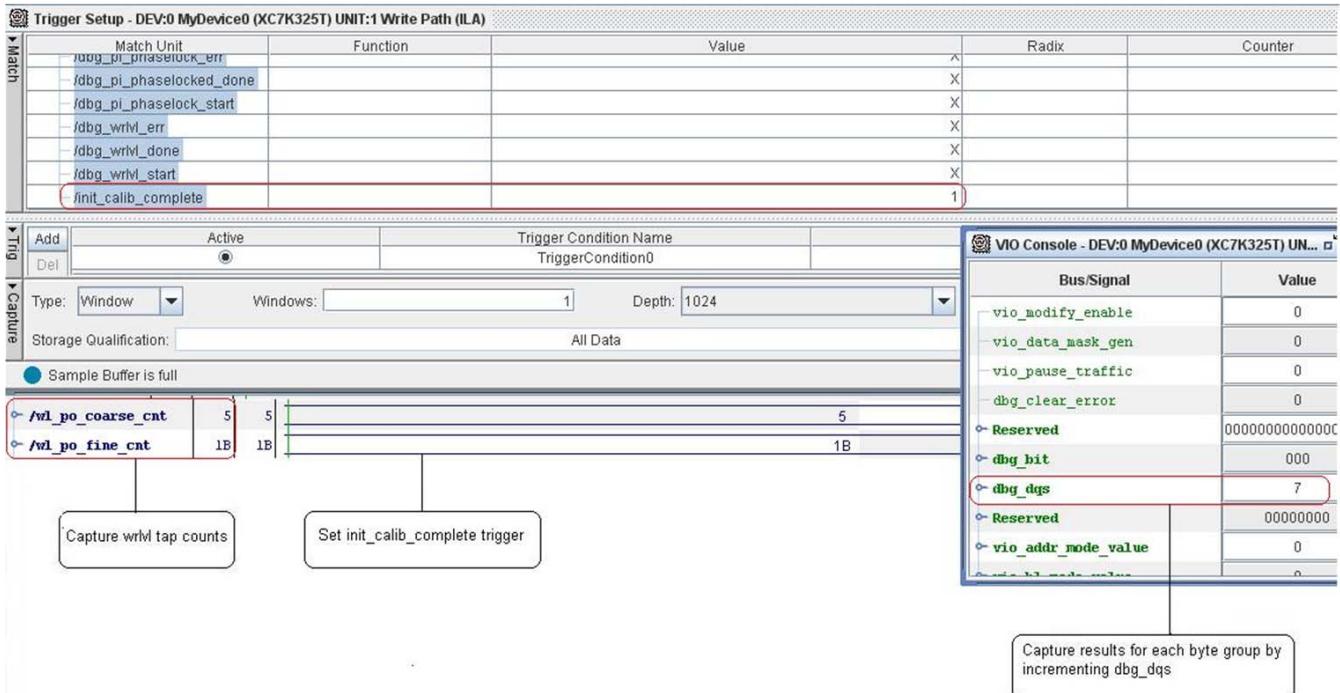


Figure 1-101:

### Debugging MPR Read Leveling Failures – DDR3 Only ( $\text{dbg_rdlvl\_err}[1] = 1$ )

At this stage of calibration, the write DQS is not centered in the write DQ window, nor is the read DQS centered in the read DQ window. The DDR3 Multi-Purpose Register (MPR) is used to center the read DQS in the read DQ window. The MPR has a pre-defined "01010101" or "10101010" pattern that is read back during this stage of calibration. The read DQS centering is required for the next stage of calibration, OCLKDELAYED calibration.

- If this stage of calibration failed with the assertion of `dbg_rdlvl_err[1]`, set the ILA trigger to `dbg_rdlvl_err[1]`.
- If this stage of calibration was successful and the results need to be analyzed, use the trigger `dbg_rdlvl_done[1] = R`.
- Set the VIO `dbg_dqs` for each byte and capture the following signals; the results for each byte should be captured in the "7 Series DDR3 Calibration Results" spreadsheet. Later releases of the MIG tool include results for all `DQS` byte groups removing the need to use `dbg_dqs`

Table 1-77:

<code>dbg_rdlvl_start[1]</code>	Signifies the start of the MPR stage of calibration.
<code>dbg_rdlvl_done[1]</code>	Signifies the successful completion of the MPR Stage of calibration.
<code>dbg_rdlvl_err[1]</code>	Signifies the MPR stage of calibration exhibited errors and did not complete.
<code>cal1_state_r</code>	State machine variable for MPR and Read Leveling Stage 1. States can be decoded in the <code>ddr_phy_rdlvl.v</code> module.
<code>cal1_cnt_cpt_r</code>	Signifies the byte that failed MPR read leveling or read leveling stage 1.
<code>dbg_cpt_first_edge_cnt_by_dqs</code>	Signifies PHASER_IN fine tap count when the first edge in MPR and Read Leveling Stage 1 is found. Byte capture based on VIO <code>dbg_dqs</code> setting.
<code>dbg_cpt_first_edge_cnt</code>	Signifies PHASER_IN fine tap count when the first edge in MPR and Read Leveling Stage 1 is found.
<code>dbg_cpt_second_edge_cnt_by_dqs</code>	Signifies PHASER_IN fine tap count when the second edge in MPR and Read Leveling Stage 1 is found. Byte capture based on VIO <code>dbg_dqs</code> setting.
<code>dbg_cpt_second_edge_cnt</code>	Signifies PHASER_IN fine tap count when the second edge in MPR and Read Leveling Stage 1 is found.
<code>dbg_cpt_tap_cnt_by_dqs</code>	Signifies the center tap moved to based on when the first and second edges were found. Byte capture based on VIO <code>dbg_dqs</code> setting.
<code>dbg_cpt_tap_cnt</code>	Signifies the center tap moved to based on when the first and second edges were found.
<code>dbg_dq_idelay_tap_cnt_by_dqs</code>	IDELAY tap value for MPR and Read Leveling Stage 1. This should be within 2 to 3 taps across all DQS byte groups. Byte capture based on VIO <code>dbg_dqs</code> setting.
<code>dbg_dq_idelay_tap_cnt</code>	IDELAY tap value for MPR and Read Leveling Stage 1. This should be within 2 to 3 taps across all DQS byte groups.

- Always look at `DQ[0]` for each component. Memory devices either send the "01010101" or "10101010" pattern on all `DQ` bits or on `DQ[0]` as specified by the JEDEC standard.

The MIG design only looks at **DQ[0]**. If there is an issue with **DQ[0]**, the MPR calibration stage would fail.

- If a **DQS** byte group failed this stage of calibration, **call1\_cnt\_cpt\_r** would equal the byte number that is failing as no further progress or increment on **call1\_cnt\_cpt\_r** occurred.
- Check if the failing **DQS** byte has an **dq\_idelay\_tap\_cnt** value of 31. This means the algorithm ran out of taps searching for the capture edges.
- Check and compare the **dq\_idelay\_tap\_cnt**, **cpt\_first\_edge\_cnt**, **cpt\_second\_edge\_cnt**, and **cpt\_tap\_cnt** values across bytes during MPR read leveling.
- Look at **idelay\_tap\_cnt** for each byte group. The **idelay\_tap\_cnt** across the **DQS** byte groups should only vary by 2 to 3 taps
- Look at how many edges (up to two) were found. Less than two edges can be found when running around or below 400 MHz. Otherwise, two edges should always be found.
- Using high quality probes and scope, probe the address/command to ensure the load register command to the DRAM that enables MPR was correct. To enable the MPR, a MODE Register Set (MRS) command is issued to the MR3 Register with bit A2 = 1. To make this measurement, bring **mpc\_rdlvl\_start** to an I/O pin and use as the trigger to capture A2 (must be 1) and **WE\_N** (must be 0).

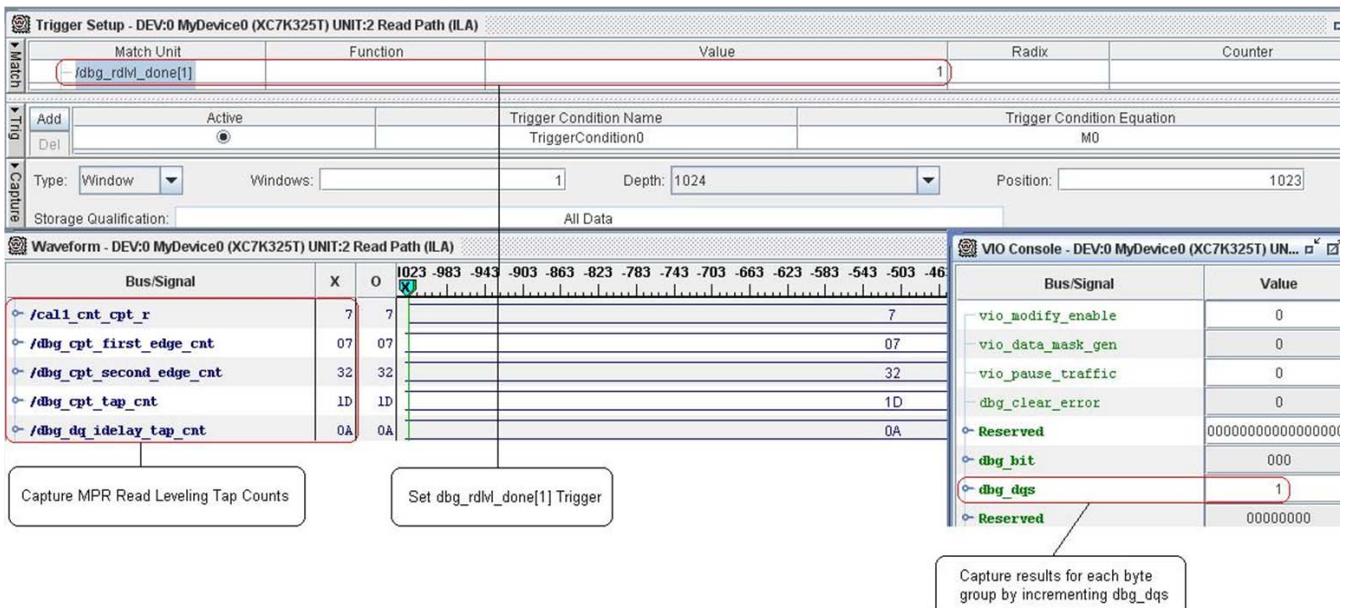


Figure 1-102:

## Debugging OCLKDELAYED Calibration Failures

The 7 series MMCM has outputs with "fine phase shift" capability. This fine phase shift capability is relatively linear and with fairly high resolution. The algorithm finds at least two edges that are either the edges of the data valid window or the edges of the noise region using Phaser\_Out stage 3 taps. The MMCM fine phase shift is used to align the MMCM clock to these detected edges to determine the center of the data valid window using MMCM taps. Finally, the write DQS is edge-aligned with the already centered MMCM clock using Phaser\_Out stage 3 taps.

There are three substages in this step:

1. Set Phaser\_Out stage 3 limit for OCLKDELAY calibration using MMCM per byte. This stage determines the limits of stage 3 tap movement during the edge detection substage.
2. Detect edges of the write DQ window using simple pattern. These modules perform edge detection by scanning the write DQ window using stage 3 within the limits determined by the limit module.
3. Set DQS to the center of write DQ window using MMCM phaser shift. Centering stage during which the write DQS is centered in the write DQ window based on the edges found during the edge detection stage.

This stage of calibration can fail if no edges are detected (highly unlikely). Sub-optimal OCLKDELAYED calibration can result in data bit errors during normal operation. This occurs because the DQS to DQ 90° relationship is not correct. Full analysis of this calibration stage is critical.

- Probe the DQS to DQ phase relationship at the memory. DQS should be center aligned to DQ.
- Using `dbg_oclkdely_calib_done = R` as the ILA trigger, capture the below signals and record the results in the "7 Series DDR3 Calibration Results" spreadsheet.
- Look at how many edges (up to three) were found. Less than three edges can be found when running around 400 MHz or at higher frequencies when the write level tap values are around 56 taps.

Table 1-78:

<code>dbg_oclkdely_calib_start</code>	Signifies the start of the OCLKDELAY stage of calibration.
<code>dbg_oclkdely_calib_done</code>	Signifies the end of the OCLKDELAY stage of calibration.

Table 1-78:

(Cont'd)

dbg_phy_ockdelay_zfo[0]	1 indicates that the left-edge of the fall window was detected and it validates fuzz2oneeighty as the tap value of the left-edge of the fall window.
dbg_phy_ockdelay_zfo[1]	1 indicates that the left-edge of the rise window was detected and it validates fuzz2zero as the tap value of the left-edge of the rise window.
dbg_phy_ockdelay_zfo[2]	1 indicates that the right-edge of the fall window was detected and it validates oneeighty2fuzz as the tap value of the right-edge of the fall window.
dbg_phy_ockdelay_zfo[3]	1 indicates that the right-edge of the rise window was detected and it validates zero2fuzz as the tap value of the right-edge of the rise window.
dbg_ocal_fuzz2oneeighty	Stage 3 tap value of the left-edge of the fall window.
dbg_ocal_fuzz2zero	Stage 3 tap value of the left-edge of the rise window.
dbg_ocal_oneeighty2fuzz	Stage 3 tap value of the right-edge of the fall window.
dbg_ocal_zero2fuzz	Stage 3 tap value of the right-edge of the rise window.
dbg_ocal_ockdelay_calib_cnt	Byte count indicating the byte being calibrated.
dbg_ocal_lim_done	Indicates that stage 3 lower and upper limits have been determined.
dbg_ocal_stg3_lim_left	Stage 3 lower limit
dbg_ocal_stg3_lim_right	Stage 3 upper limit
phy_ockdelay_cal_taps	Final stage 3 tap values for all the bytes in the interface. Bits[5:0] for byte 0 and Bits[11:6] for byte 1.
dbg_ocal_center_calib_start	Indicates end of edge detection and start of centering in valid window.
dbg_ocal_center_calib_done	Indicates end of the centering stage of calibration.
dbg_ocal_tap_cnt	Stage 3 tap value during calibration for each group.
dbg_ocal_scan_win_not_found	1 indicates that window edge is not found.

### Debugging Write Calibration Failures (`dbg_wrcal_err = 1`)

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, two **CK** cycles earlier than, or aligned to the **CK** edge that captures the write command.

This is a pattern based calibration; hence, multiple writes followed by a single read are issued during this stage. The following data patterns might be seen:

- On-time write pattern read back – FFOOAA5555AA9966
- One **CK** early write pattern read back – AA5555AA9966BB11

- Two **CK** early write pattern read back – 55AA 9966BB11EE44
- One **CK** late write pattern read back – XXXXFFOOAA5555AA
  - Calibration cannot correct for this pattern. This pattern indicates that the trace delays are incorrect where

Table 1-79:

(Cont'd)

dbg_wcal_mux_rd_fall1_r	Data pattern received on falling edge 1.
dbg_wcal_mux_rd_rise2_r	Data pattern received on rising edge 2.
dbg_wcal_mux_rd_fall2_r	Data pattern received on falling edge 2.
dbg_wcal_mux_rd_rise3_r	Data pattern received on rising edge 3.
dbg_wcal_mux_rd_fall3_r	Data pattern received on falling edge 3.

1. The number on **wrcal\_dqs\_cnt** when **dbg\_wrcal\_err** asserts signifies the byte that failed write calibration. Debug should be focused on this byte group.
2. Observe the **rddata** bus or the **mux\_rd\_fall/riseX\_r** buses and look for the appropriate data pattern. Note, **mux\_rd\_fall/rise\_2/3\_r** is not used with the half-rate controller and it is always zero. Again, the three scenarios that allow write calibration to continue are:
  - On-time write expected pattern – FFOOAA5555AA 9966
  - One cycle early write expected pattern – AA5555AA 9966BB11
  - Two cycles early expected pattern – 55AA 9966BB11EE44
3. If none of these three patterns are observed on a failing byte, look at the failing pattern and determine how the pattern is failing. Look if there are failing **DQ** bit(s) within a byte, failing bytes, and others. If the late write pattern noted above was detected, there is most likely a trace length issue between **DQS** and **CK** where **CK** is not longer than **DQS** as required.
4. If the design is stuck in the Write Calibration stage, the issue could be related to either the write or the read. Determining whether the write or read is causing the failure is critical. The following steps should be completed using **dbg\_wrcal\_start** as the scope trigger. To perform this, **dbg\_wrcal\_start** must be brought out to an I/O. For additional details and example Read and Write scope shots, review the [Determining If a Data Error is Due to the Write or Read](#).
  - a. To ensure the writes are correct, observe the write **DQS** to write **DQ** relationship at the memory using high quality scope and probes. During write calibration, a write is followed by a read so care needs to be taken to ensure the write is captured. See the [Determining If a Data Error is Due to the Write or Read](#) section for details. If there is a failing bit, determining the write **DQS** to write **DQ** relationship for the specific **DQ** bit is critical. The write ideally has the **DQS** center aligned in the **DQ** window. Misalignment between **DQS** and **DQ** during Write Calibration points to an issue with OCLKDELAY calibration. Review the [Debugging OCLKDELAYED Calibration Failures](#) section.
  - b. If the **DQ-DQS** alignment looks correct, next observe the **WE\_N** to **DQS** relationship at the memory during a write again using high quality scope and probes. The **WE\_N** to **DQS** delay must equal the CAS Write Latency (CWL).

- c. Using high quality scope and probes, verify the expected pattern (FF00AA5555AA9966) is being written to the DRAM during a write and that the expected pattern is being read back during the first Write Calibration read. If the pattern is correct during write and read at the DRAM, verify the **DQS-CK** alignment. During Write Calibration, these two signals should be aligned. Write Leveling aligned these two signals which has successfully completed before Write Calibration.
  - d. Probe **ODT** and **WE\_N** during a write command. In order for **ODT** to be properly turned on in the memory, **ODT** must assert before the write command.
  - e. Probe DM to ensure it is held Low during calibration. If a board issue exists causing DM to improperly assert, incorrect data is read back during calibration causing a write calibration failure. An example of a board issue on DM is when DM is not used and tied Low at the memory with improper termination.
- 5. It is possible for write calibration to fail due to rare manufacturing issues with the memory device. Verify SDRAM pins are behaving correctly. Look for floating or grounded signals. The debug signals should be used to determine which byte group is failing and if specific pin(s) within that byte group are causing the incorrect data pattern. These pins should be the focus at the memory device.
  - 6. If the **DQS**-to-**DQ**, **CWL**, and **DQS**-to-**CK** look correct, review the above [Debugging MPR Read Leveling Failures – DDR3 Only \(dbg\\_rdlvl\\_err\[1\] = 1\)](#) section.

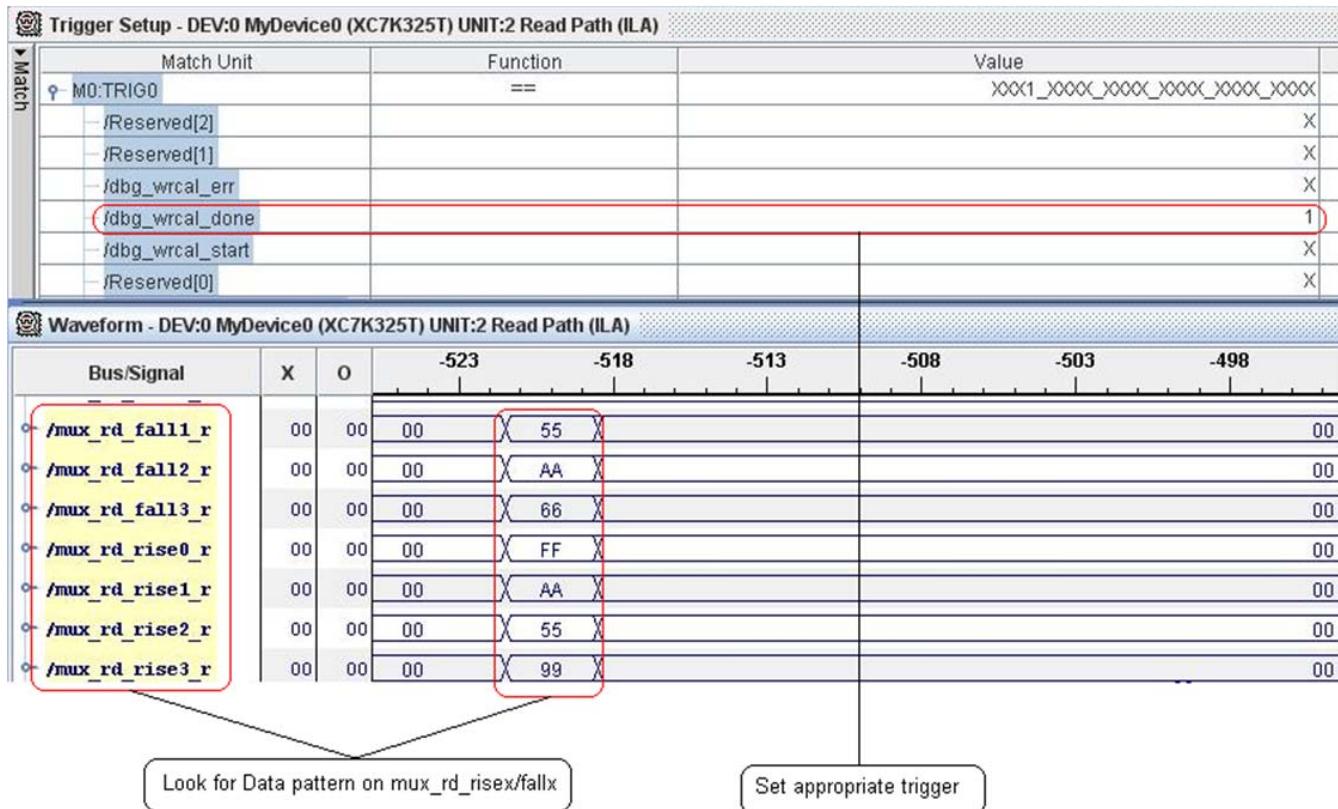


Figure 1-103:

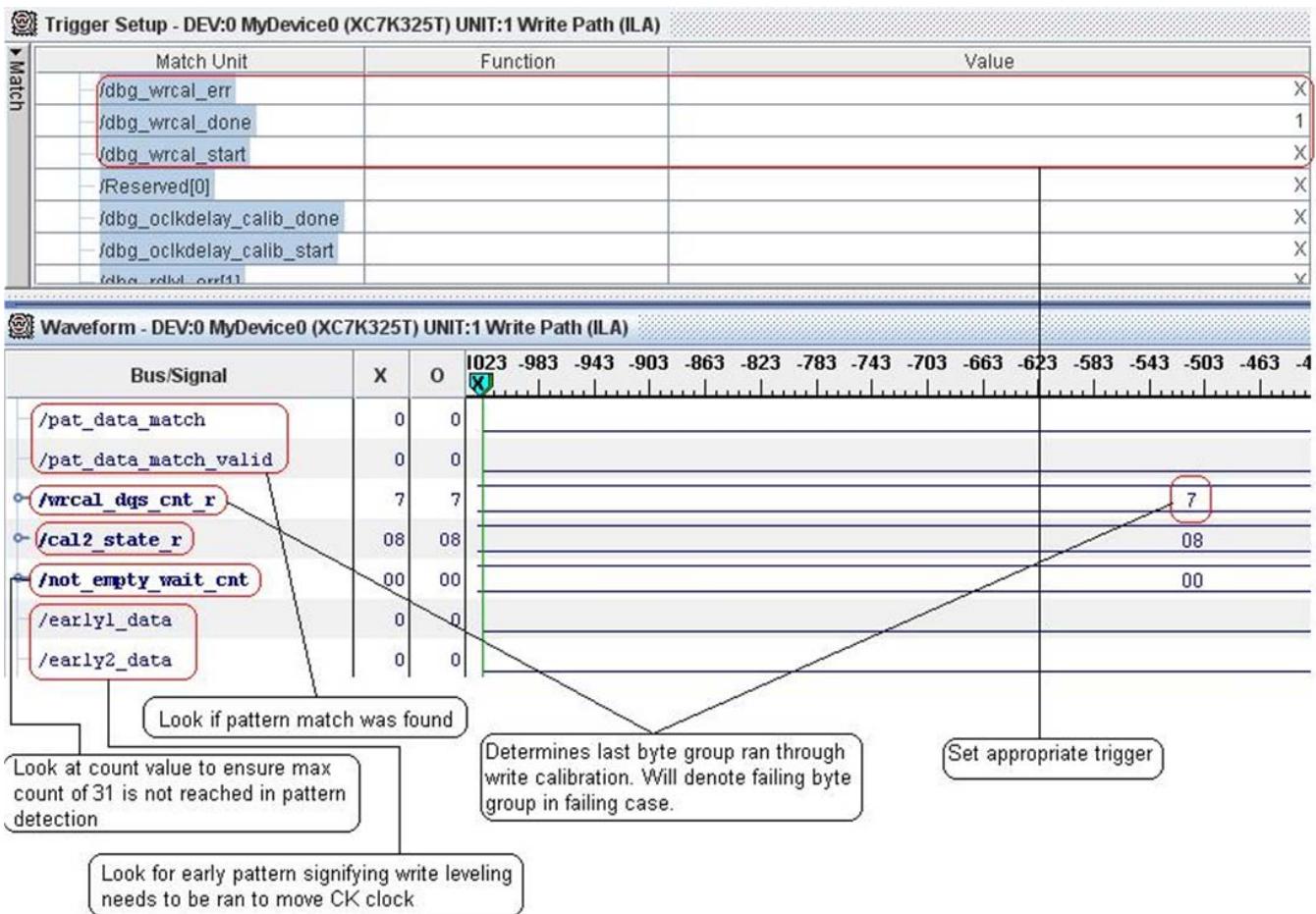


Figure 1-104:

### Debugging Read Leveling Failures (`dbg_rdlvl_err[0] = 1`)

For memory clock frequencies of 400 MHz and above, Read Leveling is performed after Write Calibration.

The final read **DQS** to read **DQ** centering is done in this stage of calibration. The first step in this stage is to decrease the IDELAY and PHASER\_IN stage 2 taps values to zero to undo MPR read leveling. MPR read leveling was only required for OCLKDELAYED calibration. This stage of read leveling accurately centers the read **DQS** in the read **DQ** window using a 993377EECC992244 data pattern. If this stage calibrates successfully, the **init\_calib\_complete** signal is asserted and calibration is complete.

- If this stage of calibration failed with the assertion of `dbg_rdlvl_err[0]`, set the ILA trigger to `dbg_rdlvl_err[0]`.

- If this stage of calibration was successful and the results need to be analyzed, set the ILA trigger to **dbg\_rdlvl\_done[0]** = R.
- Set the VIO **dbg\_dqs** for each byte and capture the following signals. The results for each byte should be captured in the “7 Series DDR3 Calibration Results” spreadsheet. Later releases of the MIG tool include results for all DQS byte groups removing the need to use **dbg\_dqs**.

Table 1-80:

dbg_rdlvl_start[0]	Signifies the start of Read Leveling Stage 1 of calibration.
dbg_rdlvl_done[0]	Signifies the successful completion of Read Leveling Stage 1 of calibration.
dbg_rdlvl_err[0]	Signifies Read Leveling Stage 1 of calibration exhibited errors and did not complete.
cal1_state_r	State machine variable for MPR and Read Leveling Stage 1. States can be decoded in the <code>ddr_phy_rdlvl.v</code> module.
cal1_cnt_cpt_r	Signifies the byte that failed MPR read leveling or read leveling stage 1.
dbg_cpt_first_edge_cnt_by_dqs	Signifies PHASER_IN fine tap count when the first edge in MPR and Read Leveling Stage 1 is found. Byte capture based on VIO <code>dbg_dqs</code> setting.
dbg_cpt_first_edge_cnt	Signifies PHASER_IN fine tap count when the first edge in MPR and Read Leveling Stage 1 is found.
dbg_cpt_second_edge_cnt_by_dqs	Signifies PHASER_IN fine tap count when the second edge in MPR and Read Leveling Stage 1 is found. Byte capture based on VIO <code>dbg_dqs</code> setting.
dbg_cpt_second_edge_cnt	Signifies PHASER_IN fine tap count when the second edge in MPR and Read Leveling Stage 1 is found.
dbg_cpt_tap_cnt_by_dqs	Signifies the center tap moved to based on when the first and second edges were found. Byte capture based on VIO <code>dbg_dqs</code> setting.
dbg_cpt_tap_cnt	Signifies the center tap moved to based on when the first and second edges were found.
dbg_dq_idelay_tap_cnt_by_dqs	IDELAY tap value for MPR and Read Leveling Stage 1. This should be within 2 to 3 taps across all DQS byte groups. Byte capture based on VIO <code>dbg_dqs</code> setting.
dbg_dq_idelay_tap_cnt	IDELAY tap value for MPR and Read Leveling Stage 1. This should be within 2 to 3 taps across all DQS byte groups.

- Determine which stage is failing by observing **cal1\_state\_r**.
- Look at **idelay\_tap\_cnt** for each byte group. The **idelay\_tap\_cnt** across the DQS byte groups should only vary by 2 to 3 taps.

- Look at how many edges (up to two) were found. Less than two edges can be found when running around or below 400 MHz. Otherwise, two edges should always be found to then center the IDELAY taps.
- Determine if any bytes completed successfully. The read leveling algorithm sequentially steps through each DQS byte group detecting the capture edges. When the failure occurs, the value on `call1_cnt_cpt_r` indicates the byte that failed edge detection.
- If the incorrect data pattern is detected, determine if the error is due to the write access or the read access. See the [Determining If a Data Error is Due to the Write or Read](#) section.
- If the `dbg_rdlvl_err[0]` is asserted (read leveling failure), use high quality probes and scope observe the DQS-to-DQ phase relationship during a write. The scope trigger should be `dbg_rdlvl_start[0]`. The alignment should be approximately 90°.
- If the DQS-to-DQ alignment is correct, observe the WE\_N-to-DQS relationship to see if it meets CWL again using `dbg_rdlvl_start[0]` as a trigger.

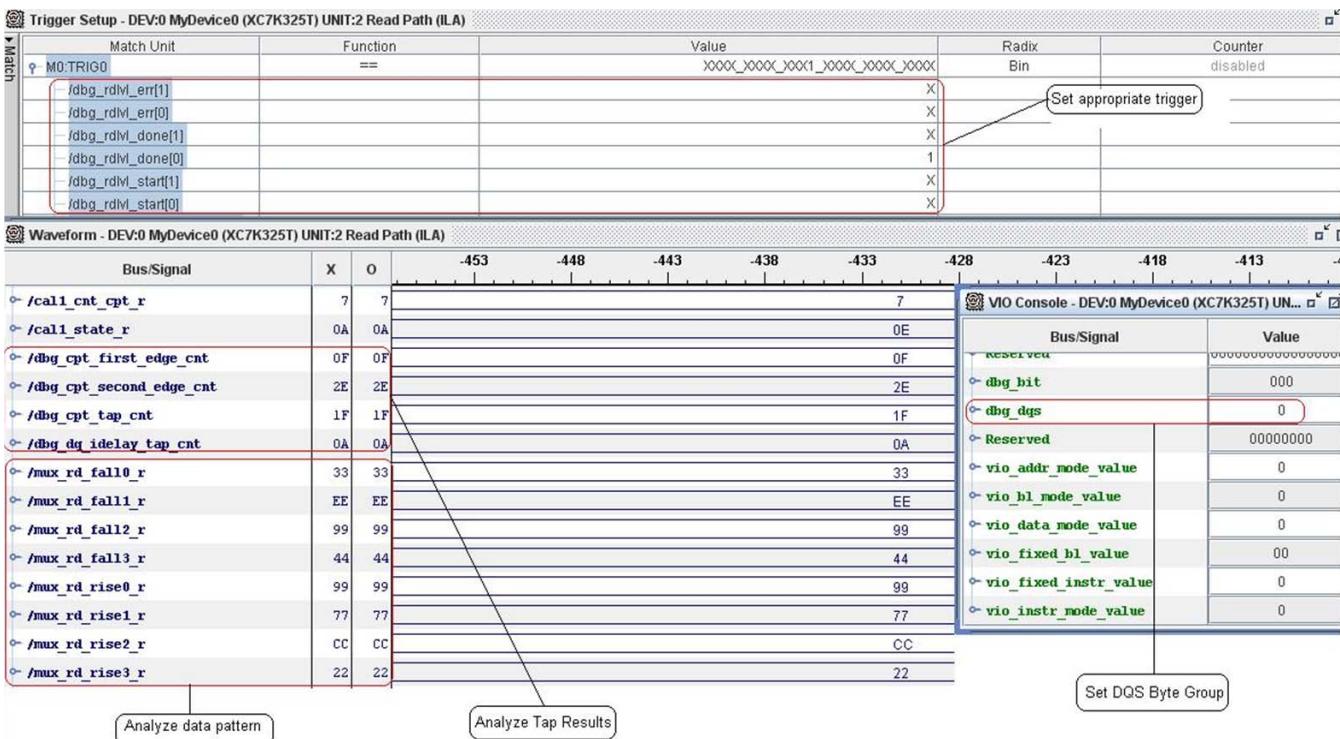


Figure 1-105:

*Debugging PRBS Read Leveling Failures*

This stage of calibration was added in MIG 7 series v1.7 and determines the read data valid window using complex pattern that is written once and read back from the DDR3 SDRAM.

*Table 1-81:*

left_edge_pb	Signifies PHASER_IN fine tap value of starting valid read window (left edge) for each bit in a byte. left_edge_pb[5:0] is the left edge of Bit[0] and left_edge_pb[47:42] is the left edge of Bit[7].
left_loss_pb	Signifies the loss in aggregate window size caused by left edge change for each bit in a byte. If left edge change of the bit does not affect the valid window, it is set to "0," left_loss_pb[1:0] is for Bit[0], and left_loss_pb[15:14] is for Bit[7].
right_edge_pb	Signifies PHASER_IN fine tap value of ending valid read window (right edge) for each bit in a byte. right_edge_pb[5:0] is right edge of Bit[0] and right_edge_pb[47:42] is the right edge of Bit[7].
right_gain_pb	Signifies the gain in aggregate valid window caused by right edge change for each bit in a byte. If right edge change of the bit does not affect the valid window, it is set to "0," right_gain_pb[1:0] is for Bit[0], and right_gain_pb[15:14] is for Bit[7].
prbs_dqs_cnt_r	Signifies the current DQS byte group being calibrated during PRBS Read Leveling. Use VIO dbg_dqs to select the byte group.
prbs_rdlvl_start	Signifies the start of PRBS Read Leveling calibration.
prbs_rdlvl_done	Signifies the successful completion of PRBS Read Leveling.
compare_err_r0	Signifies data mismatch on first rising edge data comparison.
compare_err_r1	Signifies data mismatch on second rising edge data comparison.
compare_err_r2	Signifies data mismatch on third rising edge data comparison.
compare_err_r3	Signifies data mismatch on forth rising edge data comparison.
compare_err_f0	Signifies data mismatch on first falling edge data comparison.
compare_err_f1	Signifies data mismatch on second falling edge data comparison.
compare_err_f2	Signifies data mismatch on third falling edge data comparison.
compare_err_f3	Signifies data mismatch on forth falling edge data comparison.
compare_err	Signifies data comparison failure due to a read data pattern
prbs_dqs_tap_cnt_r	Signifies the internal counter which tracks PHASER_IN fine tap movement.
pi_counter_read_val	Signifies DQS PHASER_IN fine tap setting.

**Table 1-81:**
*(Cont'd)*

ref_bit	Signifies that reference bit of the byte which has largest left edge PHASER_IN tap value.
complex_victim_inc	Indicates the victim increment for internal calibration pattern change.
rd_victim_sel	Signifies the victim selection to get the correct data pattern to compare during read back.
prbs_state_r1	Signifies the state of PRBS Read Leveling state machine.
rd_valid_r2	Indicates the read out data is valid to use for comparison.
left_edge_found_pb	Indicates left edge found for each bit in a byte. Left_edge_found_pb[0] is for Bit[0] and left_edge_found_pb[7] is for Bit[7].
right_edge_found_pb	Indicates right edge found for each bit in a byte. right_edge_found_pb[0] is for Bit[0] and right_edge_found_pb[7] is for Bit[7].
largest_left_edge	Signifies the left edge tap value of the byte.
smallest_right_edge	Signifies the right edge tap value of the byte.
fine_delay_incdec_pb	Indicates the increment of FINEDELAY tap in IDELAY primitive for each bit in a byte. fine_delay_incdec_pb[0] is for Bit[0] and fine_delay_incdec_pb[7] is for Bit[7].
fine_delay_sel	Indicates fine_delay_incdec_pb is applied for FINEDELAY in IDELAY primitive.
compare_err_pb_latch_r	Indicates that data mismatch happened for each bit in a byte at a specific PHASER_IN tap setting. compare_err_pb_latch_r[0] is for Bit[0] and compare_err_pb_latch_r[7] is for Bit[7].
fine_pi_dec_cnt	Signifies the computed decrease value of PHASER_IN tap.
match_flag_and	Indicates all bits have data mismatch. Recoded for five consecutive PHASER_IN tap setting.
stage_cnt	Signifies the number of scans with different setting of FINEDELAY in IDELAY primitive.
fine_inc_stage	Indicates that FINEDELAY in IDELAY primitive is in the increment stage.
compare_err_pb_and	Signifies the data mismatch happened for all bits in a byte.
right_edge_found	Indicates the right edge of the byte is found.

## Calibration Times

For Initial ES (IES) with extended calibration, completing calibration in hardware should take about 30 seconds.

For General ES (GES) and Production, hardware calibration time scales with interface data width and data rate. [Table 1-82](#) lists the calibration times for 32-bit and 72-bit interfaces at 800 Mb/s and 1,600 Mb/s. Calibration times are faster at higher frequencies because the required number of reads and writes to perform the phase adjustment can complete faster than at a lower frequency. Calibration is completed on a per-byte basis. Therefore, larger interface widths result in longer calibration times.

**Note:** These are typical values. Calibration times can vary significantly depending on the board signal integrity, the FPGA, and the frequency of operation.

MIG release 2014.2 includes updates to the read calibration algorithm resulting in calibration time increase compared to the previous release. MIG release 2014.4 includes updates to the write calibration algorithm using MMCM for better write DQS to write data centering which resulted in increased calibration time.

*Table 1-82:*

2014.1	< 1 second	< 1 second	< 1 second	< 1 second
2014.2	< 1 second	~1 second	< 1 second	Little over 1 second
2014.4	< 1 second	~2 seconds	< 1 second	~1 second
2015.1	~1 second	~2 seconds	< 1 second	~1 second

## Debugging Data Errors

As with calibration error debug, the General Checks section of this answer record should be reviewed. Strict adherence to proper board design is critical in working with high speed memory interfaces. Violation of these general checks is often the root cause of data errors.

When data errors are seen during normal operation, the MIG 7 series Example Design (Traffic Generator) should be used to replicate the error. The Traffic Generator can be configured to send a wide range of data, address, and command patterns allowing customers to test their target traffic pattern on a verified solution. The Traffic Generator stores the write data and compares it to the read data. This allows comparison of expected and actual data when errors occur. The following section details the critical step in Data Error debug.

Table 1-83:

vio_modify_enable	Set to 1 to vary the command Traffic Generator command pattern.
vio_data_mask_gen	Traffic generator Data Mask generation.
vio_pause_traffic	Set to 1 to pause the Traffic Generator.
dbg_clear_error	Set to clear Traffic Generator errors. This signal can be used in checking for single bit errors or measuring a read window.
vio_addr_mode_value	Valid settings for this signal are: <ul style="list-style-type: none"><li>• 0x1 = FIXED address mode <a href="#">(1)</a></li><li>• 0x2 = PRBS address mode</li><li>• 0x3 = SEQUENTIAL address mode</li></ul>
vio_bl_mode_value	Valid settings for this signal are: <ul style="list-style-type: none"><li>• 0x1 = FIXED burst length <a href="#">(1)</a></li><li>• 0x2 = PRBS burst length</li></ul>
vio_fixed_bl_value	Valid settings are 1 to 256.
vio_fixed_instr_value	Valid settings are: <ul style="list-style-type: none"><li>• 0x0 = Write instruction</li><li>• 0x1 = Read instruction</li></ul>
vio_instr_mode_value	Valid settings for this signal are: <ul style="list-style-type: none"><li>• 0x1 = Command type (read/write) as defined by fixed_instr_ <a href="#">(1)</a></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>

Table 1-83:

(Cont'd)

	Valid settings for this signal are: <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.<sup>(1)</sup></li> <li>• 0x2 = ADDRESS – 32 bits address as data. Data is generated based on the logical address space. If a design has a 256-bit user data bus, each write beat in the user bus would have a 256/8 address increment in byte boundary. If the starting address is 1300, the data is 1300, followed by 1320 in the next cycle. To simplify the logic, the user data pattern is a repeat of the increment of the address value Bits[31:0].</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_data" inputs.<sup>(1)</sup></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>
--	---

**Notes:**

1. This setting does not work by default and additional RTL modifications are required.

Table 1-84:

dbg_rddata_r	Read data read out of the IN_FIFO for the DQS group selected through dbg_dqs on the VIO. This is a 64-bit bus. This debug port does not capture ECC data.
cmp_data_r	Expected data to be compared with read back data from memory. <sup>(1)</sup>
dbg_rddata_valid	Signifies that the read data is valid.
cmp_data_valid	Signifies the compare data is valid.
cmp_error	Signifies the cmp_data is not the same as the readback data from memory.

Table 1-84:

(Cont'd)

error_status[n:0]	<p>This signal latches these values when the error signal is asserted:</p> <ul style="list-style-type: none"> <li>• [42] = mcb_rd_empty</li> <li>• [41] = mcb_wr_full</li> <li>• [37:32] = cmp_bl_i</li> <li>• [31:0] = cmp_addr_i</li> </ul>

**Notes:**

1. Cmp\_data\_r is not cycle aligned with dbg\_rddata\_r and might vary from 1 burst before to 3 bursts after dbg\_rddata\_r.

Using either the MIG 7 series Traffic Generator or the user design, the first step in data error debug is to isolate when and where the data errors occur. To perform this, the expected data and actual data must be known and compared. Looking at the data errors, the following should be identified:

- Are the errors bit or byte errors?
  - Are errors seen on data bits belonging to certain **DQS** groups?
  - Are errors seen on specific **DQ** bits?
- Is the data shifted, garbage, swapped, and others?
- Are errors seen on accesses to certain addresses, banks, or ranks of memory?
  - Designs that can support multiple varieties of DIMM modules, all possible address and bank bit combinations should be supported.
- Do the errors only occur for certain data patterns or sequences?
  - This can indicate a shorted or open connection on the PCB. It can also indicate an SSO or crosstalk issue.
- Determine the frequency and reproducibility of the error.
  - Does the error occur on every calibration/reset?
  - Does the error occur at specific temperature or voltage conditions?
- Determine if the error is correctable.
  - Rewriting, rereading, resetting, and recalibrating.

To isolate the data error using the MIG 7 series Example Design Traffic Generator, use the following steps.

- Determine what type of data error is being seen (bit or byte errors).
  - a. Set the ILA trigger to **cmp\_error** = 1.

- b. Observe the `dbg_rddata_r` and `cmp_data_r` signals in Vivado logic analyzer feature.
  - Are errors seen on a data bit/s belonging to a certain `DQS` group(s)?
  - Does the data appear shifted, garbage, swapped, and others?
- Determine if errors are seen on accesses to a certain address, bank, or rank of the memory.
  - a. Set the ILA trigger to `cmp_error = 1`.
  - b. Set the VIO cores.

```
vio_modify_enable = 1

vio_data_mode_value = 2

vio_addr_mode_value = 3
```
- c. Observe the `cmp_addr_i` bits of the `error_status[31:0]` in Vivado logic analyzer.
- Determine if errors only occur for certain data patterns or sequences. This can indicate a shorted or open connection on the PCB or can also indicate an SSO or crosstalk issue.
  - a. Set the LA trigger to `cmp_error = 1`.
  - b. Set the VIO cores.

```
vio_modify_enable = 1

vio_instr_mode_value = 2

vio_data_mode_value = 2

vio_addr_mode_value = 3
```
- c. Observe the `dbg_rddata_r` and `cmp_data_r` signals and the `cmp_addr_i` bits of the `error_status[31:0]` bus in the Vivado logic analyzer feature.
- d. Repeat steps 1 to 3 with setting `vio_data_mode_value` to values varying from 3-F.
- Determine the frequency and reproducibility of the error.
  - Does the error occur after every calibration or reset?
  - Does the error occur at specific temperature or voltage conditions?
- Determine if the error is correctable.
  - Rewriting, rereading, resetting, and recalibrating.

**Note:** `vio_pause_traffic` should be asserted and deasserted each time the VIO inputs are changed.

Determining whether a data error is due to the write or the read can be difficult because if writes are the cause, read back of data is bad as well. In addition, issues with control or address timing affect both writes and reads. Some experiments that can help to isolate the issue are:

- If errors are intermittent, 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 issue. If the reads always yield the same (wrong) data, there is a write issue.

Determine if this is a Write or Read issue using the MIG 7 series Example Design Traffic Generator within the Vivado logic analyzer feature:

1. Set up all the FIXED parameter values in the RTL:

- a. Open `example_top.v` and change `fixed_data_i` and `fixed_addr_i` under the `traffic_gen_top` instantiation.

- **fixed\_addr\_i** (32'b00000000000000000000000000000000)
  - **fixed\_data\_i** (32'b11111111111111111111111111111111)

- b. Regenerate bitstream.

2. Set the ILA trigger to cmp\_error = 1.

- ### 3. Set VIO cores to:

```
vio modify enable = 1
```

```
vio pause traffic = 1
```

```
vio addr mode value = 1
```

vio bl mode value = 1

vio fixed bl value = 8

vio instr mode value = 1

```
vio_fixed_instr_value =
```

vio data mode value =

```
vio pause traffic = 0
```

- #### 4. Set the VIO cores to:

vio pause traffic = 1

```
vio_fixed_instr_value = 1 (Read Only)
```

```
vio_pause_traffic = 0
```

5. Observe the **dbg\_rddata\_r** and **cmp\_data\_r** signals in Vivado logic analyzer feature.

This can also be done using high quality probes and a scope using the Traffic Generator or your own user design.

1. Capture the write at the memory and the read at the FPGA to view data accuracy, appropriate **DQS**-to-**DQ**.
2. Look at the initial transition on **DQS** from 3-state to active.
3. During Write, **DQS** does not have a preamble.
4. During Read, the **DQS** has a Low preamble that is 1 clock cycle long.
5. The following is an example of a Read and a Write to illustrate the difference.

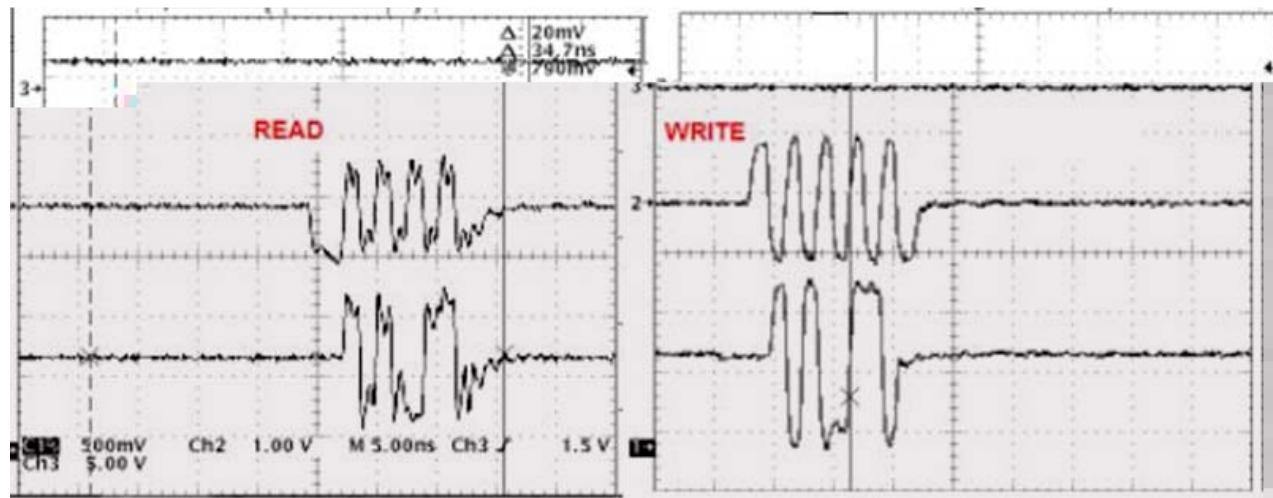


Figure 1-106:

Analyze write timing:

- If on-die termination (ODT) is used, check that the correct value is enabled in the DDR2/DDR3 device and that the timing on the **ODT** signal relative to the write burst is correct.
- Measure the phase of **DQ** relative to **DQS**. During a Write, **DQS** should be center aligned to **DQ**. If the alignment is not correct, focus on the debugging [OCLKDELAYED Calibration, page 151](#).
- For debugging purposes only, use ODELAY to vary the phase of **DQ** relative to **DQS**.

Analyze read timing:

- Check the IDELAY values after calibration. Look for variations between IDELAY values. IDELAY values should be very similar for DQS in the same DQS group.
- For debugging purposes only, vary the IDELAY taps after calibration for the bits that are returning bad data.

Debug signals are provided to verify read window margin on a per-byte basis and should be used for debugging purposes only. Determining if sufficient margin is available for reliable operation can be useful for debugging purposes if data errors are seen after calibration.

There is an automated window check flow that can be used to step through the entire interface and provides the # of PHASER taps required to reach the left edge and right edge of the data window. The window checking can also be manually verified by manually incrementing and decrementing the PHASER taps to verify how much window margin is available.

*Table 1-85:*

win_start	Single pulse that starts the window check logic.
win_sel_pi_pon	Controls window check logic on read path. Valid settings are: <ul style="list-style-type: none"><li>• 0x1 = Enables Read Path</li></ul>
vio_dbg_sel_pi_jncdec	Enables manual incrementing and decrementing of the PHASER_IN taps.
vio_dbg_pi_f_inc	Increments PHASER_IN fine taps when win_sel_pi_pon = 0x1.
vio_dbg_pi_f_dec	Decrements PHASER_IN fine taps when win_sel_pi_pon = 0x1.
vio_win_byte_select_inc	Increments the byte group being checked by the window margin check module.
vio_win_byte_select_dec	Decrements the byte group being checked by the window margin check module.
dbg_pi_counter_read_val	Current PHASER_IN tap count corresponding to current byte being checked.
pi_win_left_ram_out	PHASER_IN tap count to reach the left edge of the read window for a given byte.
pi_win_right_ram_out	PHASER_IN tap count to reach the right edge of the read window for a given byte.
win_active	Flag to indicate the Window check logic is active and measuring window margins. While active, the other VIOs should not be changed.
win_current_byte	Feedback to indicate which byte is currently being monitored.
win_byte_select	Selects which byte group to display the measured results for.
dbg_clear_error	Clears error in Traffic Generator as a result of changing tap values.
vio_sel_mux_rdd[3:0]	Selects the byte for which the phaser increments or decrements are applied.

The automated window checking is enabled by asserting **win\_start** with a single pulse. **win\_active** should then assert until all byte groups have been measured. **win\_sel\_pi\_pon** must be set to 0x1 to enable Read window measurement. and

`win_byte_select` can be used to select between each byte groups measured results and display them to the Vivado logic analyzer feature waveform window. To calculate the total data valid window use the following equation:

$$(\text{Total # of taps} \times \text{CLK_PERIOD})/128 = \text{Total Valid Data Window}$$

**Note:** The Read window measurement results are stored in a block RAM after `win_start` is asserted.

To manually measure the data window margin, follow these steps:

1. Enable the manual window check by asserting `dbg_sel_pi_incdec`.

**Note:** When `dbg_sel_pi_incdec` is enabled, `dbg_pi_counter_read_cal` does not represent the true centered PHASER\_IN tap value.

2. Set the ILA trigger to `cmp_error = 1`.
3. Manually increment/decrement the taps using the `dbg_pi_f_inc` or `dbg_pi_f_dec` an event is triggered indicating a left or right edge was found. Note the number of taps that occurred until event triggered.
4. Manually increment/decrement the taps back the same # of taps.
5. Issue a single pulse event to `dbg_clear_error`, and reset the ILA trigger.
6. Manually increment/decrement the taps in the other direction using the `dbg_pi_f_inc` or `dbg_pi_f_dec` an event is triggered indicating a left or right edge was found. Note the number of taps that occurred until event triggered.
7. Add up the left and right tap values determined and calculate the total data valid window using the following equation:

$$(\text{Total # of taps} \times \text{CLK_PERIOD})/128 = \text{Total Valid Data Window}$$

When data errors occur, the results of calibration should be analyzed to ensure the results are expected and accurate. Each of the above debugging calibration sections notes what the expected results are such as how many edges should be found, how much variance across byte groups should exist, and others. Follow these sections to capture and then analyze the calibration results.



## *System Clock*

If the SRCC/MRCC I/O pin and PLL are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint must be set to BACKBONE. DDR3/DDR2 SDRAM manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on the SRCC/MRCC I/O and PLL allocation needs to be handled manually for the IP flow. DDR3/DDR2 SDRAM does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

## *Reference Clock*

If the SRCC/MRCC I/O pin and MMCM are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint is set to FALSE. Reference clock is a 200 MHz clock source used to drive IODELAY CTRL logic (through an additional MMCM). This clock does not utilize CLOCK\_DEDICATED\_ROUTE (as they are limited in number), hence the FALSE value is set. DDR3/DDR2 SDRAM manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on SRCC/MRCC I/O and MMCM allocation needs to be handled manually for the IP flow. DDR3/DDR2 SDRAM does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

# QDR II+ Memory Interface Solution

---

The QDR II+ SRAM Memory Interface Solution (MIS) is a physical layer for interfacing Xilinx® 7 series FPGAs user designs to QDR II+ SRAM devices. QDR II+ SRAM capabilities 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 QDR II+ SRAM memory solutions core is a PHY that takes simple user commands, converts them to the QDR II+ protocol, and provides the converted commands to the memory. The PHY half-frequency design enables you 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 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 QDR II+ SRAM MIS 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 the PCB design need to be followed to have the best possible solution. For detailed board design guidelines, see [Design Guidelines, page 343](#).



**IMPORTANT:** *QDR II+ SRAM designs currently do not support memory-mapped AXI4 interfaces.*

---

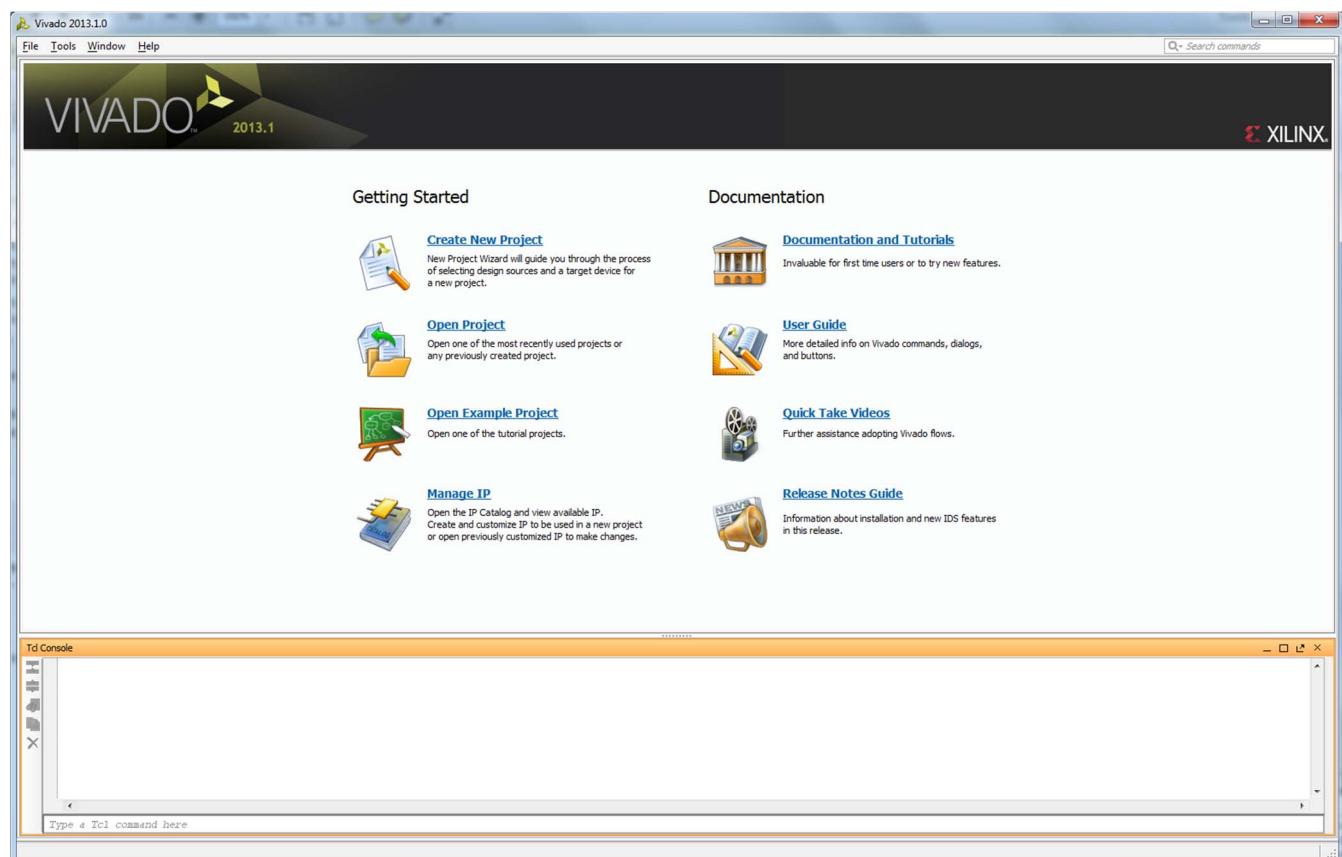
For detailed information and updates about the 7 series FPGAs QDR II+ SRAM MIS core, see the *Xilinx 7 Series FPGA Data Sheets* [\[Ref 13\]](#) and the *Zynq-7000 SoC and 7 Series FPGAs Memory Interface Solutions Data Sheet* (DS176) [\[Ref 1\]](#).



**IMPORTANT:** *Memory Interface Solutions v4.2 only supports the Vivado® Design Suite. The ISE® Design Suite is not supported in this version.*

This section provides the steps to generate the Memory Interface Generator (MIG) IP core using the Vivado Design Suite and run implementation.

1. Start the Vivado Design Suite (see [Figure 2-1](#)).



*Figure 2-1:*

2. To create a new project, click the **Create New Project** option shown in [Figure 2-1](#) to open the page as shown in [Figure 2-2](#).

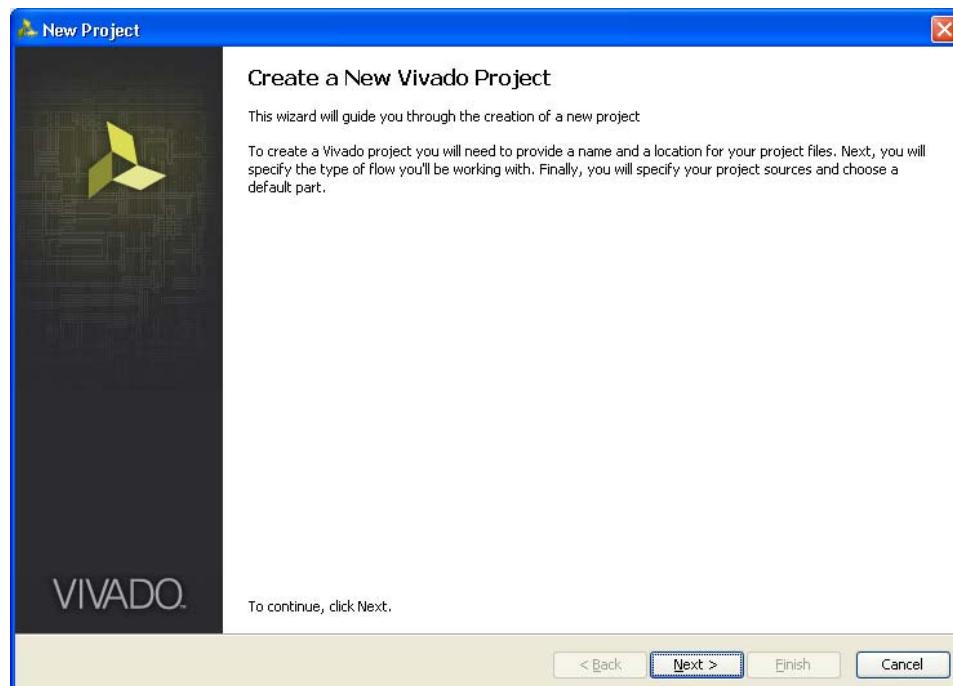


Figure 2-2:

3. Click **Next** to proceed to the **Project Name** page (Figure 2-3). Enter the **Project Name** and **Project Location**. Based on the details provided, the project is saved in the directory.

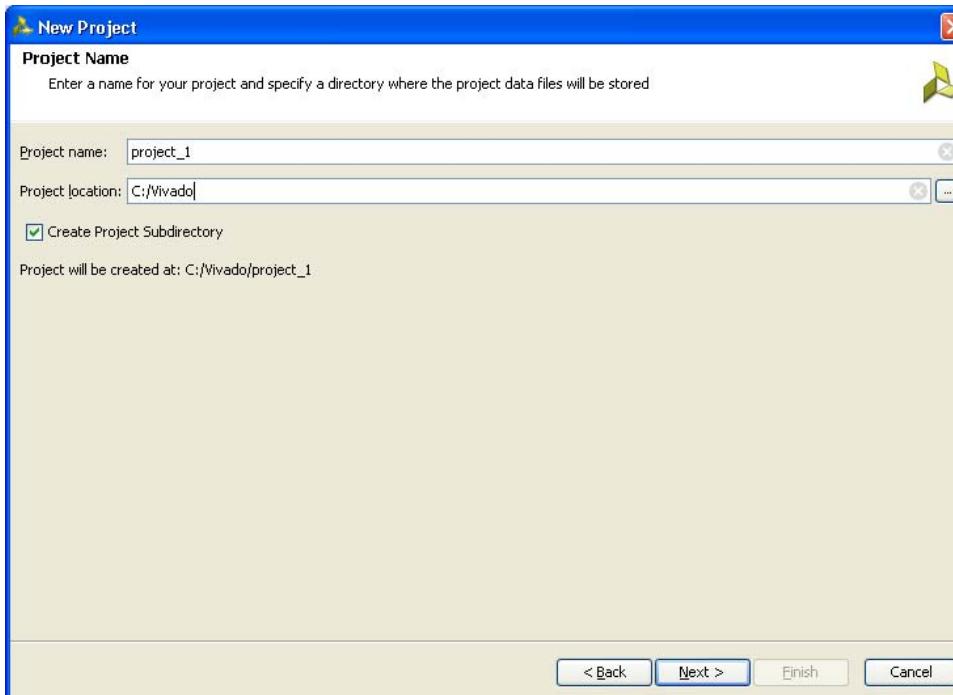


Figure 2-3:

4. Click **Next** to proceed to the **Project Type** page (Figure 2-4). Select the **Project Type** as **RTL Project** because MIG deliverables are RTL files.

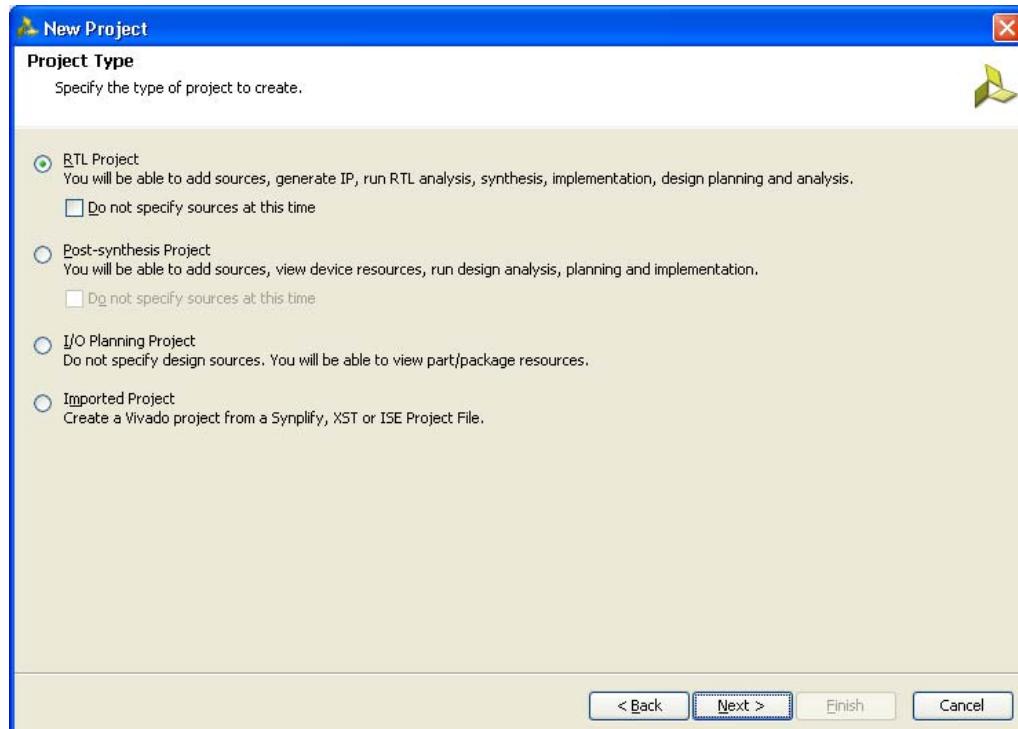


Figure 2-4:

5. Click **Next** to proceed to the **Add Sources** page (Figure 2-5). RTL files can be added to the project in this page. If the project was not created earlier, proceed to the next page.

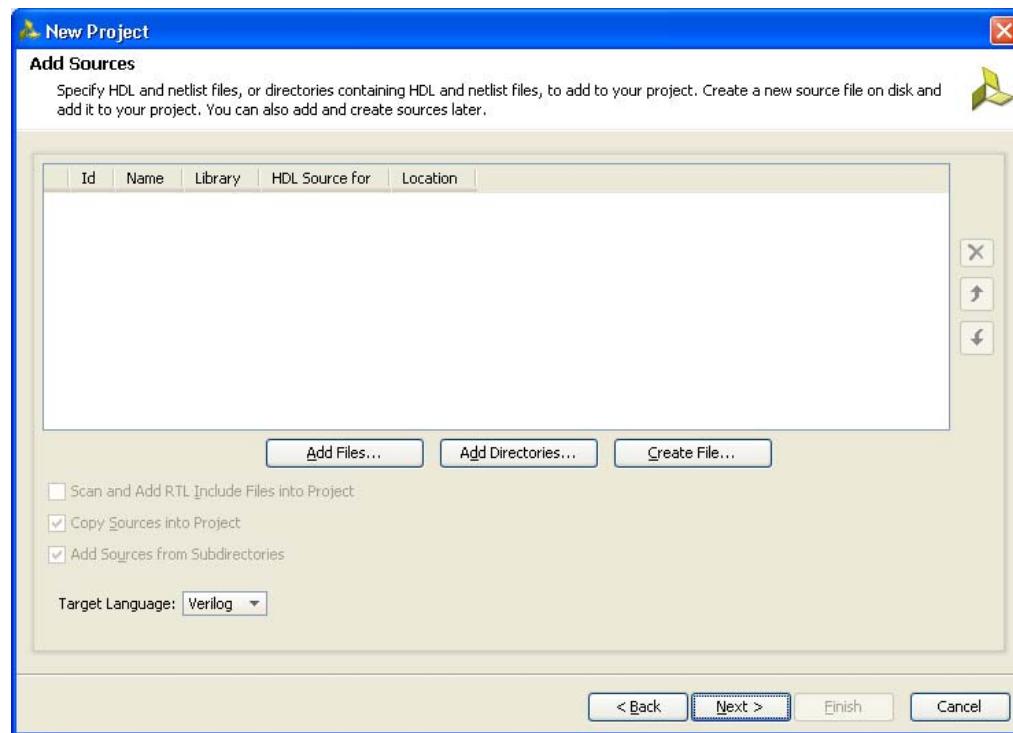


Figure 2-5:

6. Click **Next** to open the **Add Existing IP (Optional)** page (Figure 2-6). If the IP is already created, the XCI file generated by the IP can be added to the project and the previous created IP files are automatically added to the project. If the IP was not created earlier, proceed to the next page.

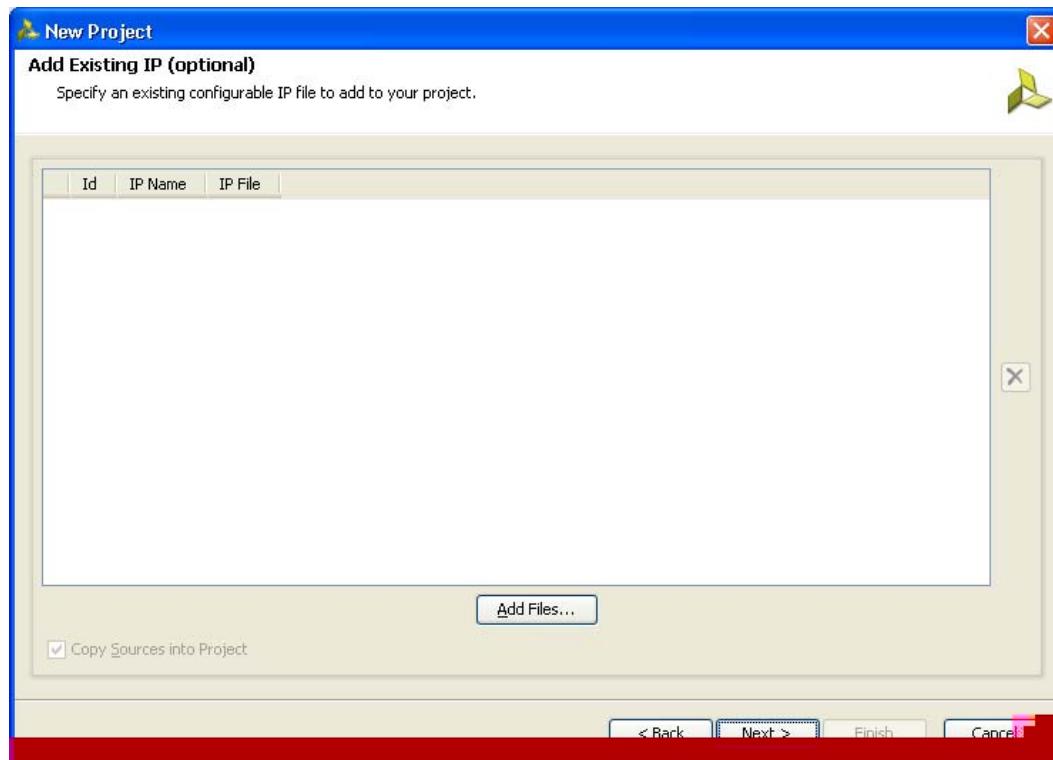


Figure 2-6:

7. Click **Next** to open the **Add Constraints (Optional)** page (Figure 2-7). If the constraints file exists in the repository, it can be added to the project. Proceed to the next page if the constraints file does not exist.

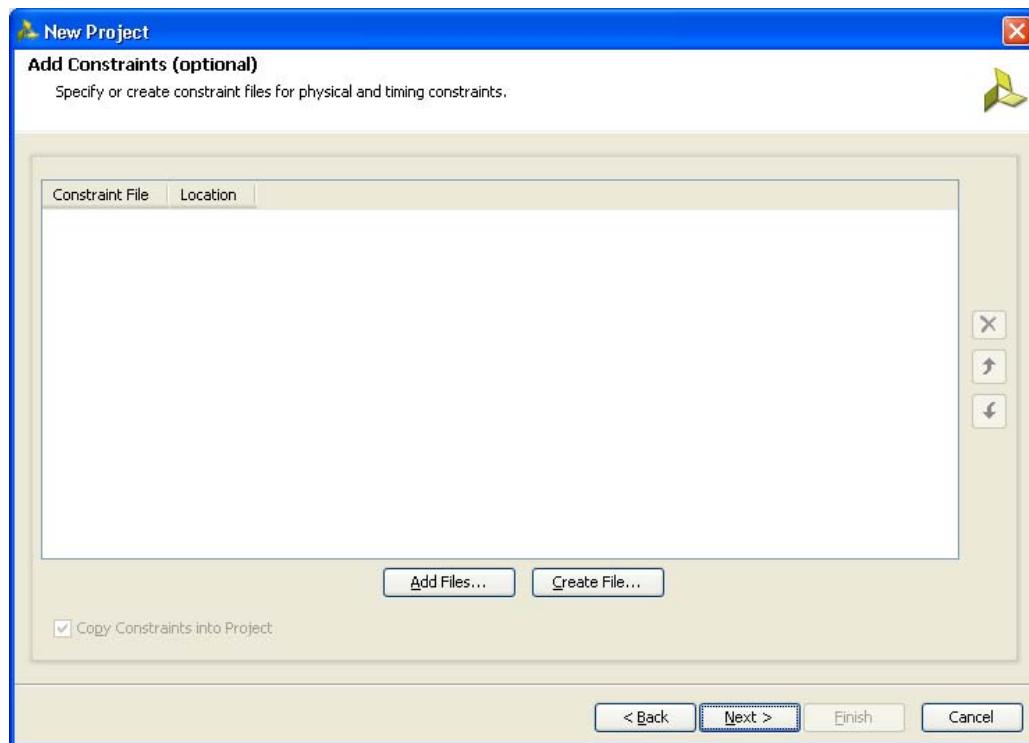


Figure 2-7:

8. Click **Next** to proceed to the **Default Part** page (Figure 2-8) where the device that needs to be targeted can be selected. The **Default Part** page appears as shown in Figure 2-8.

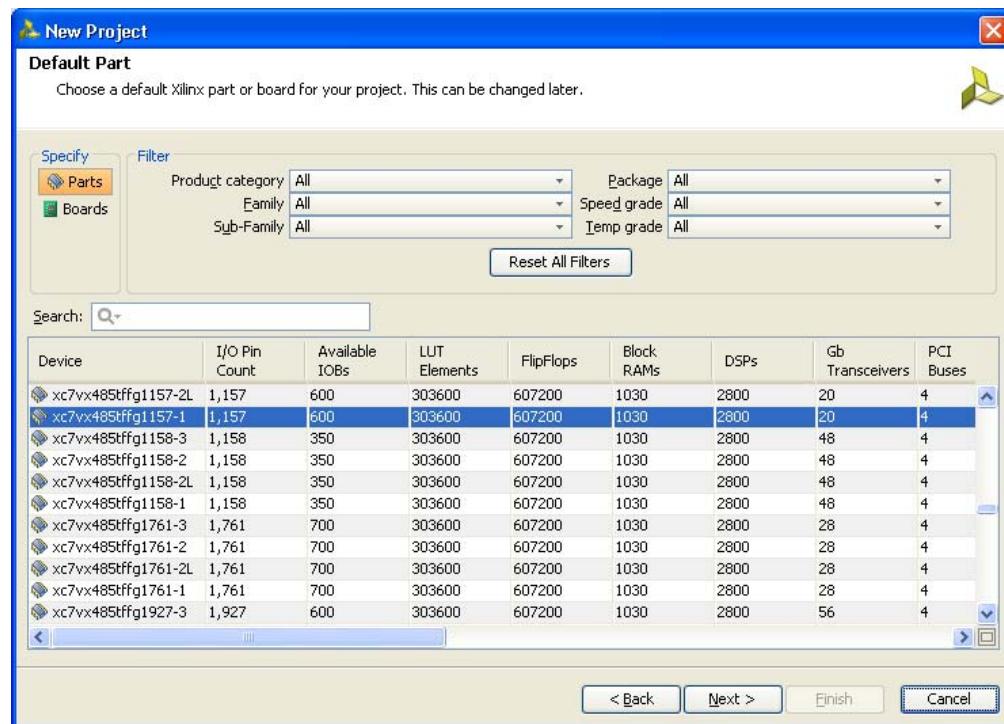
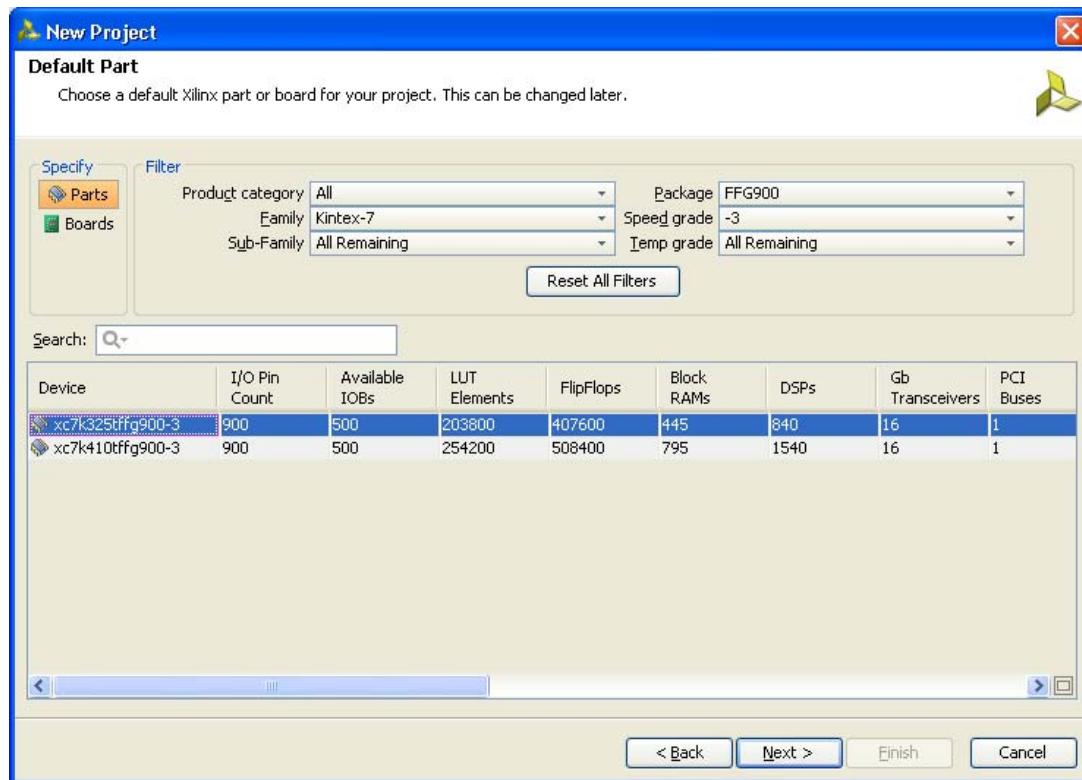


Figure 2-8:

Select the target **Family**, **Package**, and **Speed Grade**. The valid devices are displayed in the same page, and the device can be selected based on the targeted device ([Figure 2-9](#)).



*Figure 2-9:*

Apart from selecting the parts by using the **Parts** option, parts can be selected by choosing the **Boards** option, which brings up the evaluation boards supported by Xilinx ([Figure 2-10](#)). With this option, a design can be targeted for the various evaluation boards. If the XCI file of an existing IP was selected in an earlier step, the same part should be selected here.

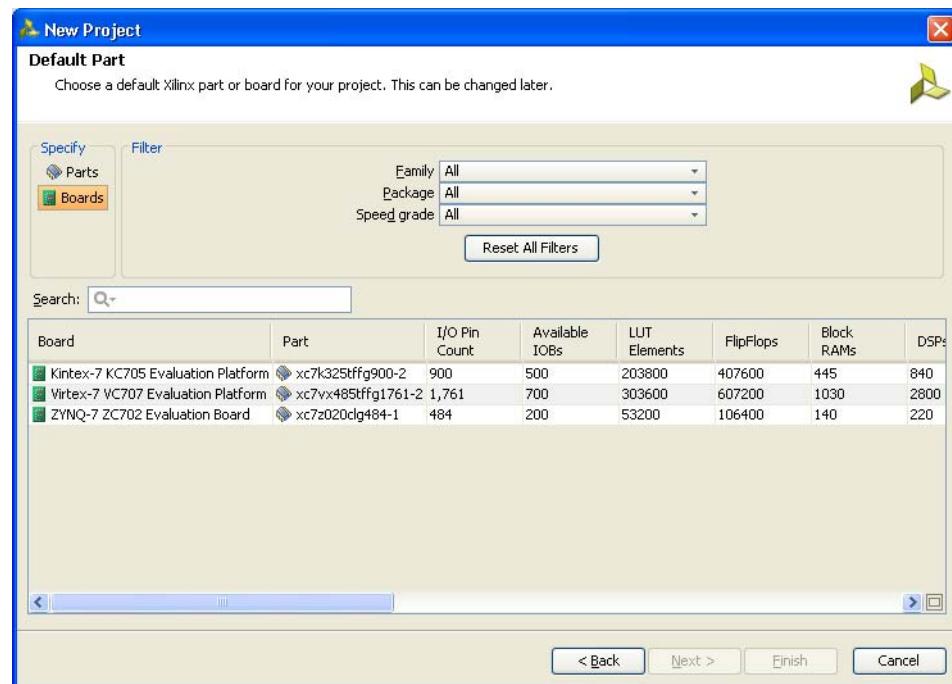


Figure 2-10:

- Click **Next** to open the **New Project Summary** page (Figure 2-11). This includes the summary of selected project details.

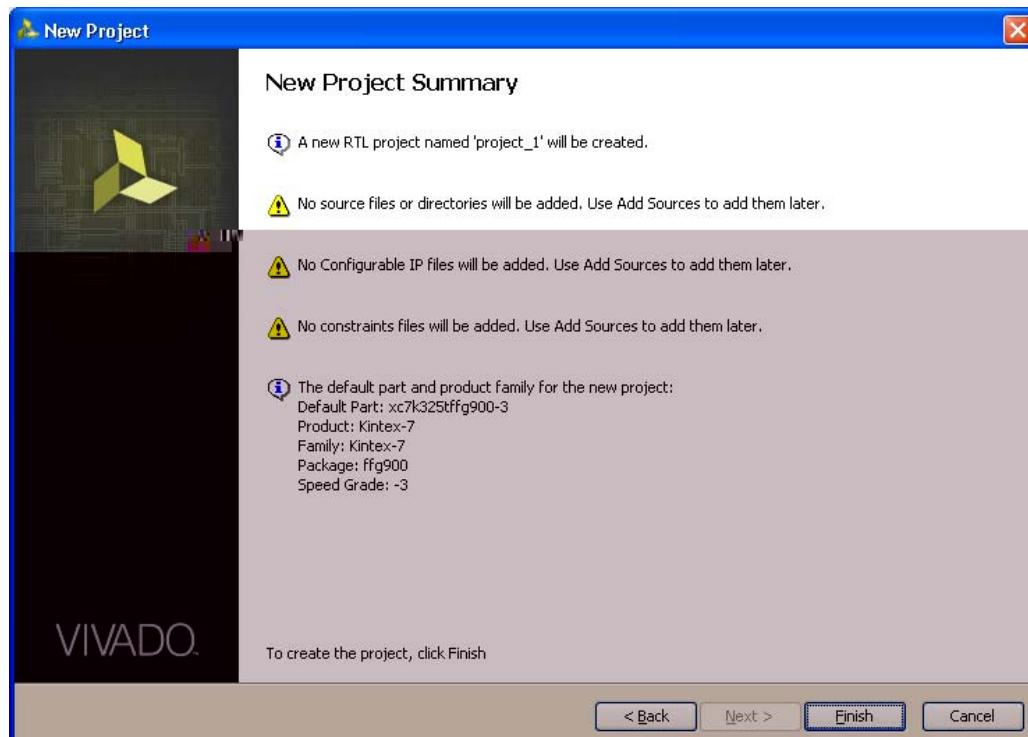
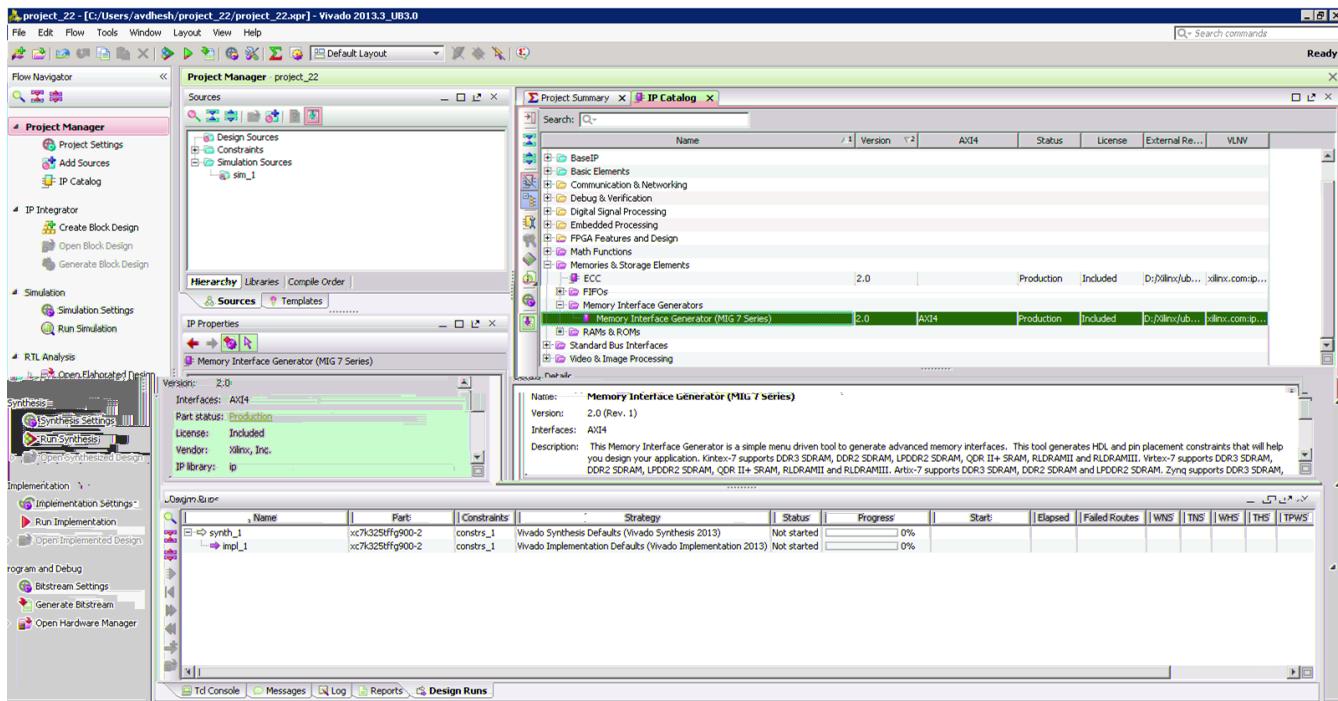


Figure 2-11:

- Click **Finish** to complete the project creation.

11. Click **IP Catalog** on the **Project Manager** window to open the Vivado IP catalog window. The IP catalog window appears on the right side panel (see [Figure 2-12](#), highlighted in a red circle).
12. The MIG tool exists in the **Memories & Storage Elements > Memory Interface Generators** section of the IP catalog window ([Figure 2-12](#)) or you can search from the Search tool bar for the string “MIG.”



*Figure 2-12:*

13. Select **MIG 7 Series** to open the MIG tool ([Figure 2-13](#)).

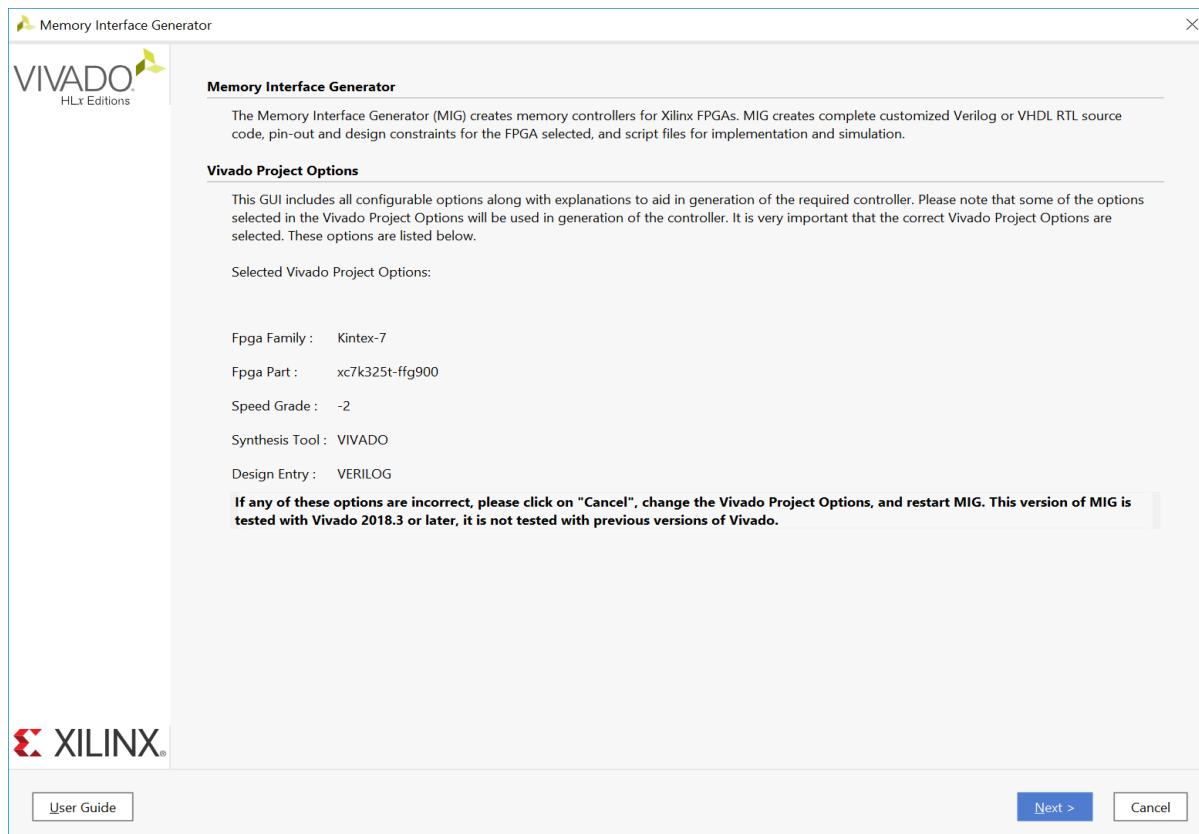


Figure 2-13:

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



**CAUTION!** *The Windows operating system has a 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs.*

## MIG Output Options

1. Select **Create Design** to create a new Memory Controller design. Enter a component name in the Component Name field ([Figure 2-14](#)).
2. Choose the number of controllers to be generated. This selection determines the replication of further pages.

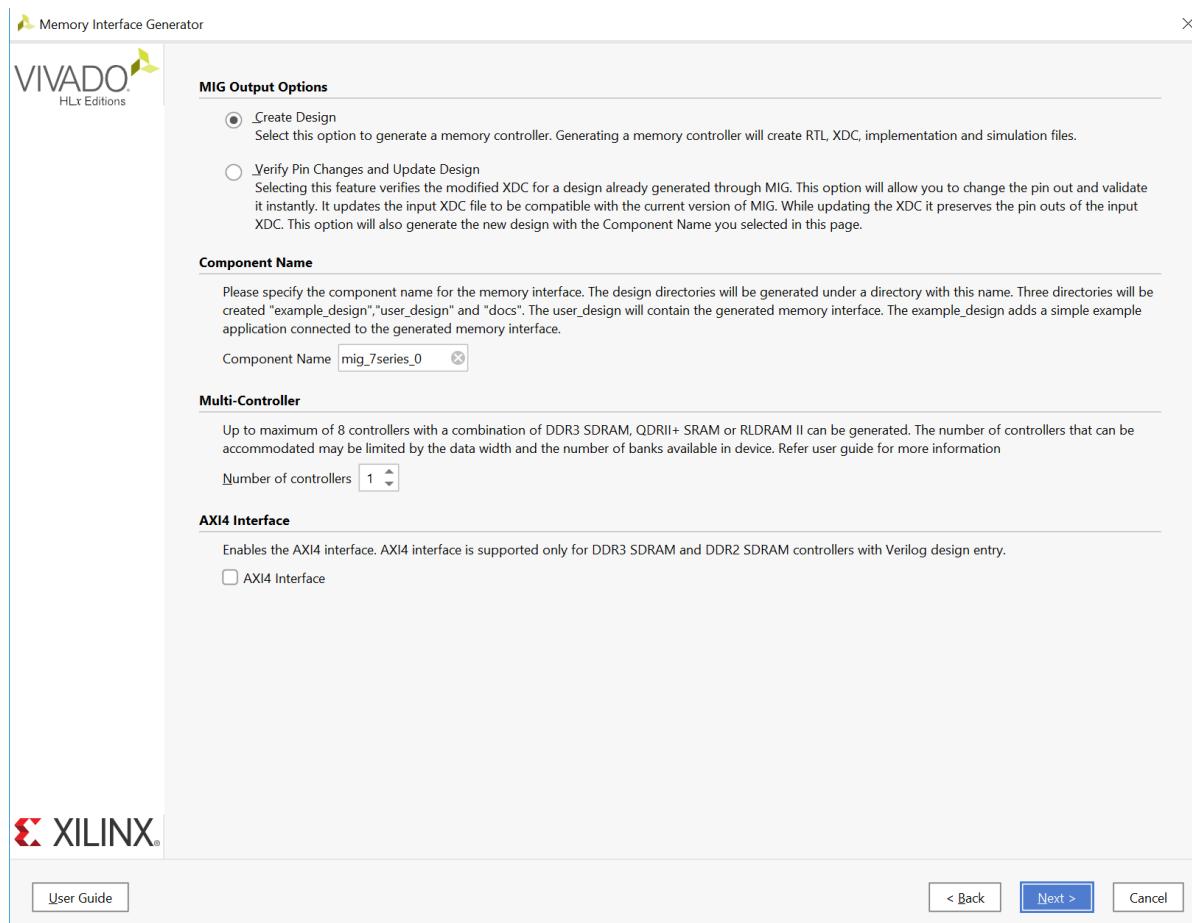


Figure 2-14:

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



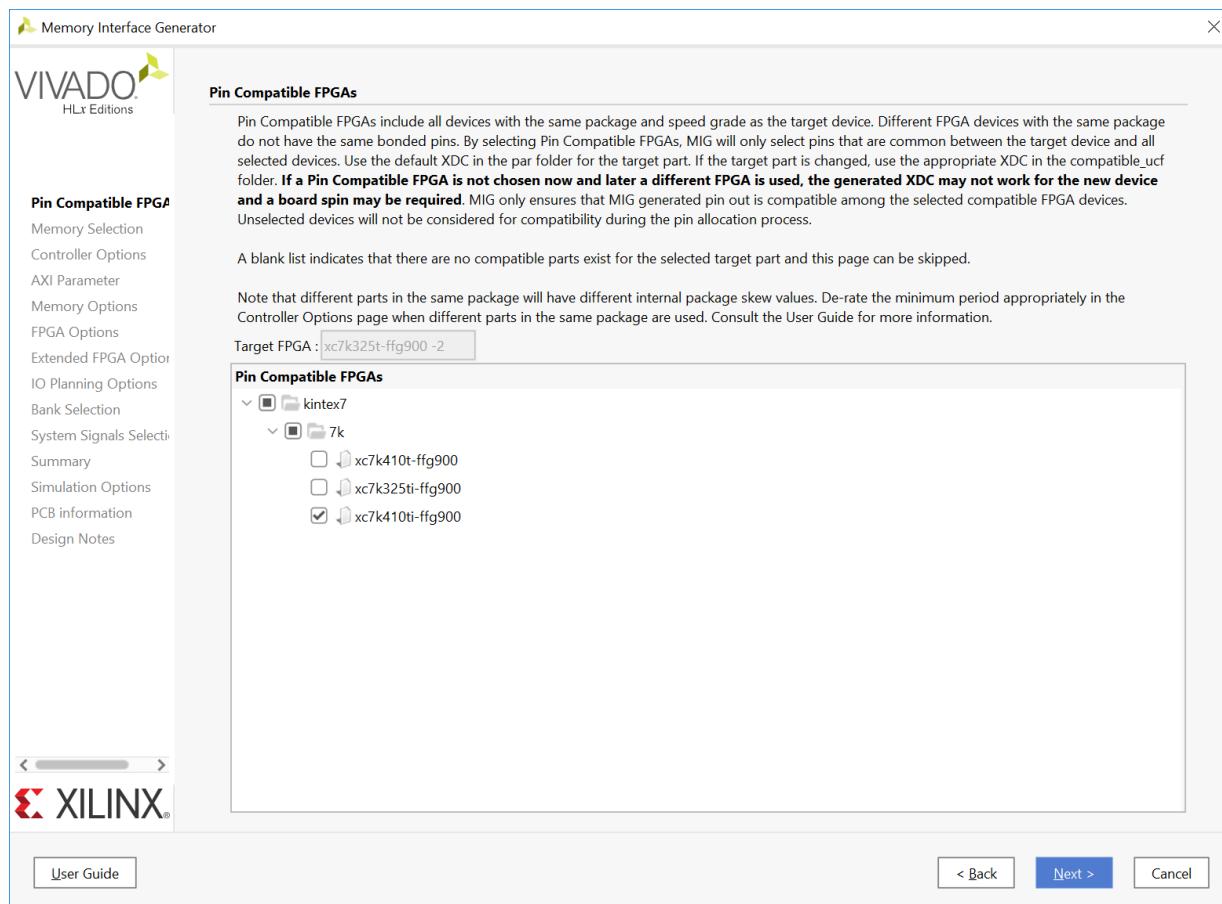
**IMPORTANT:** Only alphanumeric characters can be used for <component name>. Special characters cannot be used. This name should always start with an alphabetical character and can end with an alphanumeric character.

When invoked from Xilinx Platform Studio (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 2-15).



*Figure 2-15:*

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 QDR II+ SRAM Design*

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

1. Select the QDR II+ SRAM controller type.
2. Click **Next** to display the **Controller Options** page.

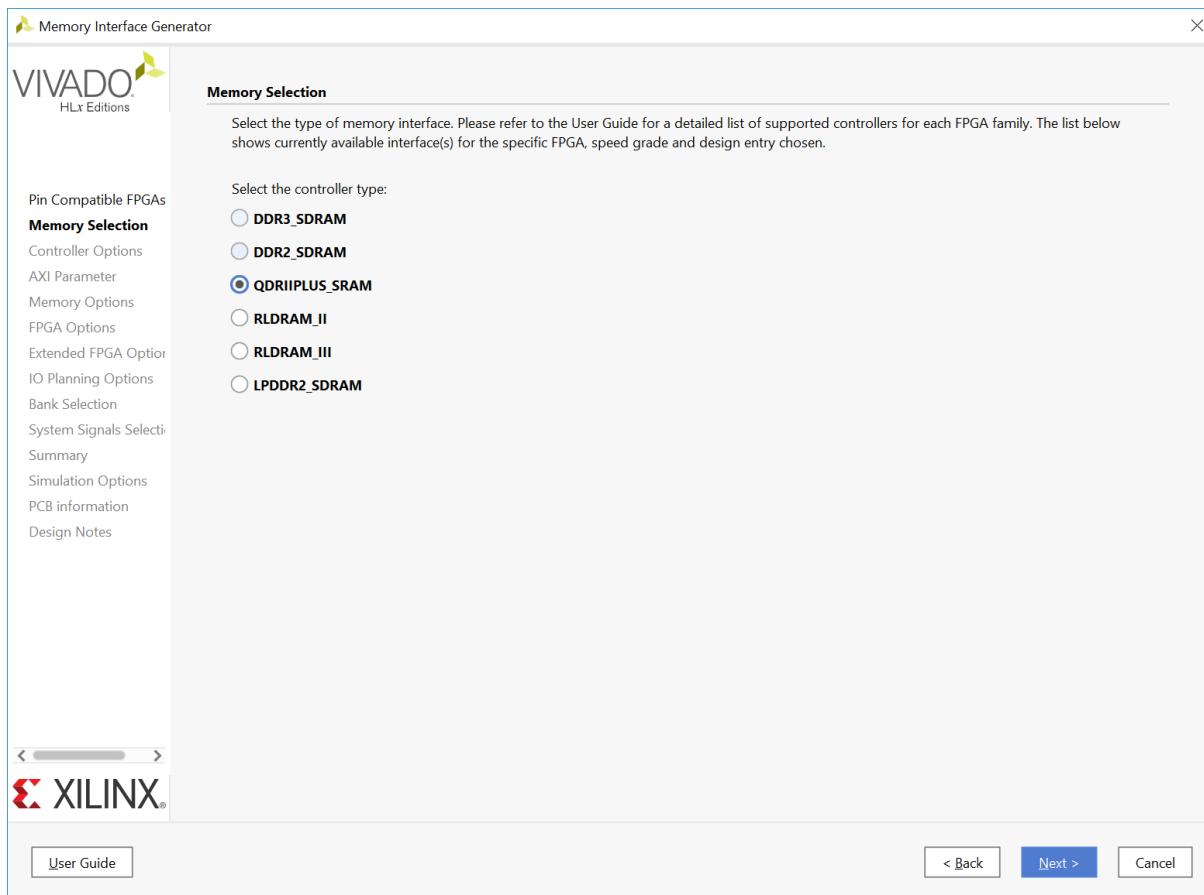


Figure 2-16:

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

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

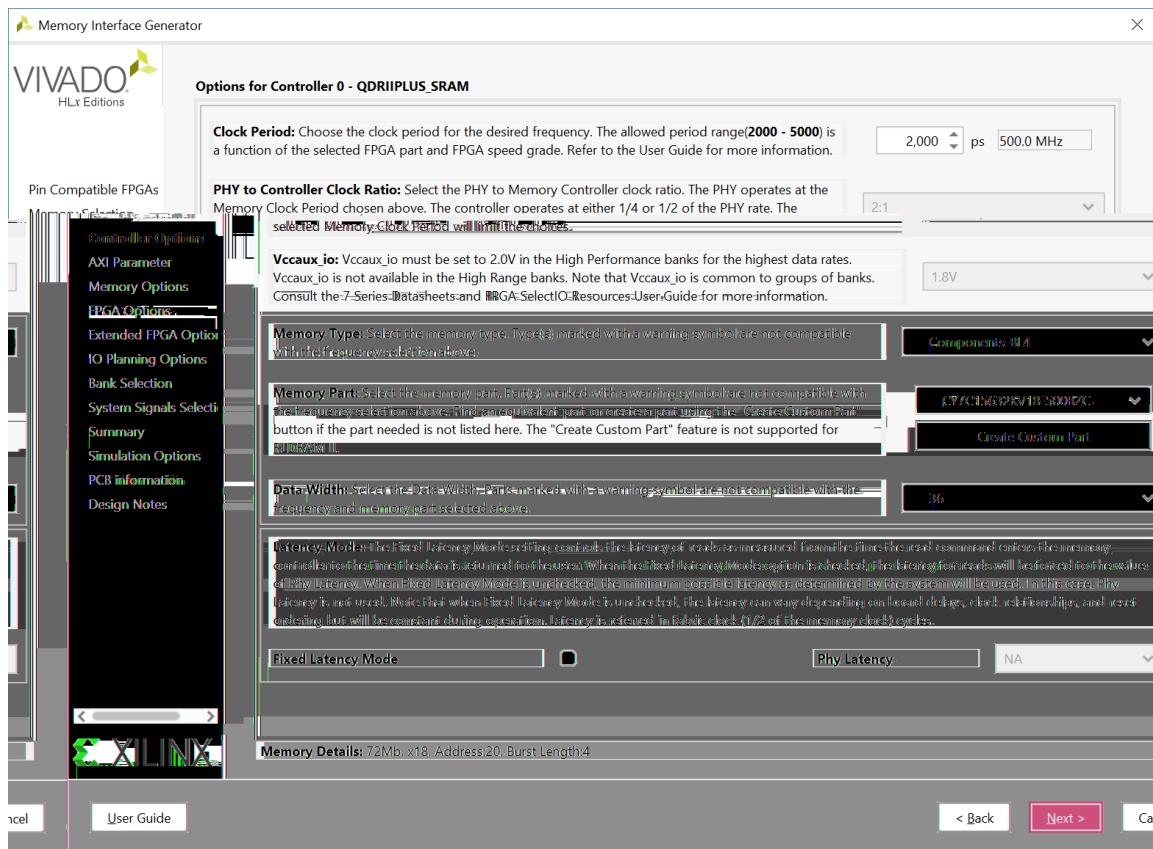


Figure 2-17:

- **Clock Period** – 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** – 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. For more information, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2].
- **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**). The QDR II+ SRAM devices with read latency 2.0 and 2.5 clock cycles are supported by the design. If a desired part is not available in the list, you 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 you 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 you to select the number of cycles until the read response is returned to you. This value ranges from 21 to 30 cycles. Based on actual hardware conditions, if the latency seen through the system is higher, you need to modify this value accordingly in the top-level RTL file.

When **Fixed Latency Mode** is enabled, failures can occur if the actual read latency is larger than the specified **Fixed Latency** value. **Read Latency** can vary across byte lanes by as much as five clock cycles because of the command output path in the PHY control block and data input path across asynchronous **IN\_FIFO**.

**Note:** Xilinx recommends adding five additional clocks to the minimum latency measured to determine the actual fixed latency value to be used. 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 2-18](#) displays the details for the selected memory configuration.

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

*Figure 2-18:*

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** below the **Memory Part** pull-down menu. A new page appears, as shown in [Figure 2-19](#).

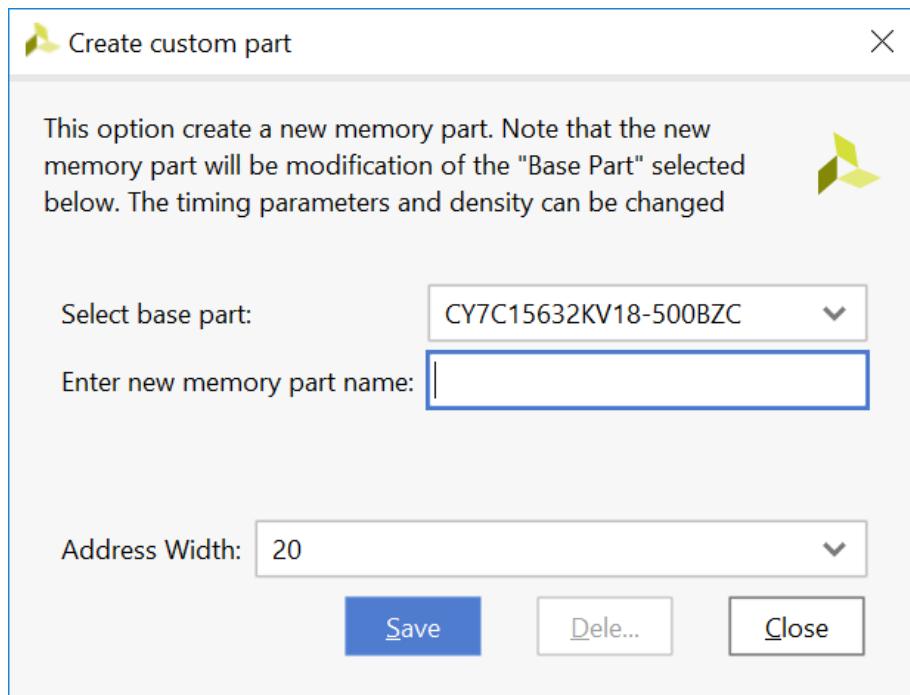


Figure 2-19:

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

1. Select the suitable base part from the **Select Base Part** list.
2. Enter the appropriate **Memory Part Name** in the text box.
3. Select a suitable value for the Row Address.
4. After editing the required fields, click **Save**. 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.

Figure 2-20 shows the **Memory Options** page.

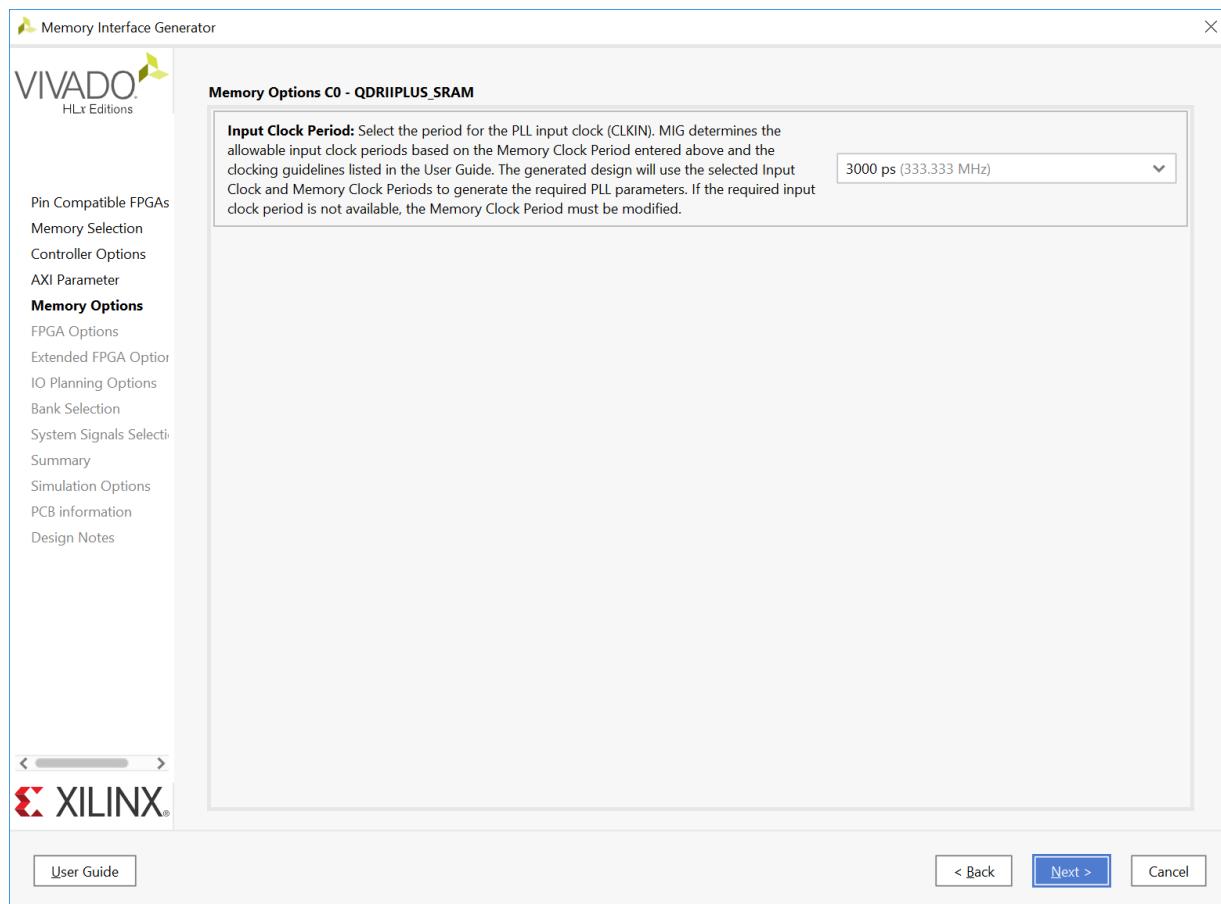


Figure 2-20:

- **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 323](#) for more information on the PLL parameter limits.

Figure 2-21 shows the **FPGA Options** page.

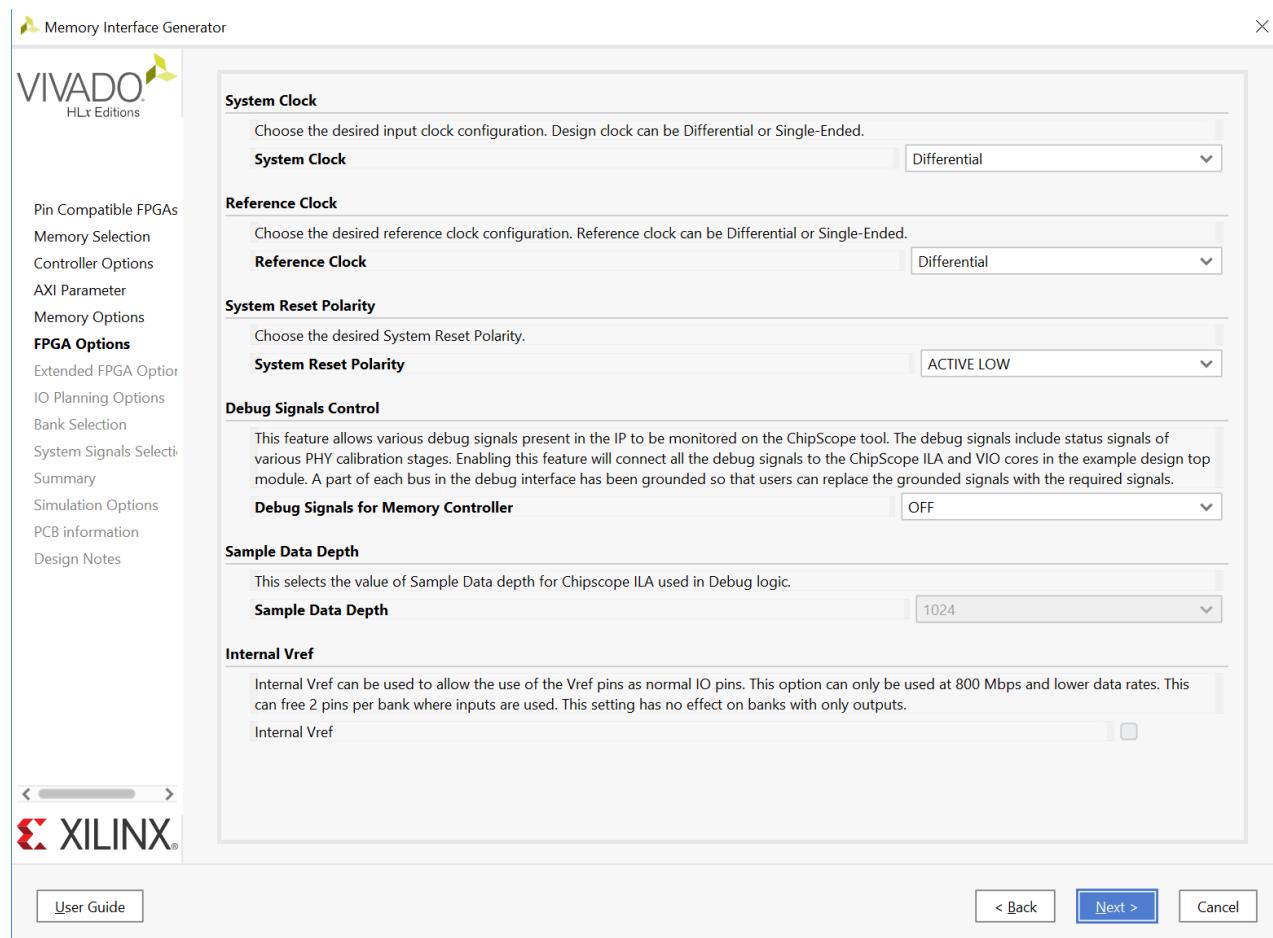


Figure 2-21:

- **System Clock** – This option selects the clock type (Single-Ended, Differential or No Buffer) for the **sys\_clk** signal pair. When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the system clock.

If the designs generated from MIG tool for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the **sys\_clk\_i** signal. So for **No Buffer** scenarios, **sys\_clk\_i** signal needs to be connected to an internal clock.

- **Reference Clock** – This option selects the clock type (Single-Ended, Differential, No Buffer, or Use System Clock) for the `clk_ref` signal pair. The **Use System Clock** option appears when the input frequency is between 199 and 200 MHz (that is, the Input Clock Period is between 5,025 ps (199 MHz) and 4,975 ps (201 MHz). When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the reference clock.

If the designs generated from MIG tool for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the `ref_clk_i` signal. So for **No Buffer** scenarios, `ref_clk_i` signal needs to be connected to an internal clock.

- **System Reset Polarity** – The polarity for system reset (`sys_rst`) can be selected. If the option is selected as active-Low, the parameter `RST_ACT_LOW` is set to 1 and if set to active-High the parameter `RST_ACT_LOW` is set to 0.
- **Debug Signals Control** – Selecting this option enables calibration status and user port signals to be port mapped to the ILA and VIO in the `example_top` module. This helps in monitoring traffic on the user interface port with the Vivado Design Suite debug feature. Deselecting the **Debug Signals Control** option leaves the debug signals unconnected in the `example_top` module and no ILA/VIO modules are generated by the IP catalog. Additionally, the debug port is always disabled for functional simulations.
- **Sample Data Depth** – This option selects the Sample Data depth for the ILA module used in the Vivado debug logic. This option can be selected when the **Debug Signals for Memory Controller** option is ON.
- **Internal V<sub>REF</sub> Selection** – Internal V<sub>REF</sub> can be used for data group bytes to allow the use of the V<sub>REF</sub> pins for normal I/O usage. Internal V<sub>REF</sub> should only be used for data rates of 800 Mb/s or below.

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

Figure 2-22 shows the Extended FPGA Options page.

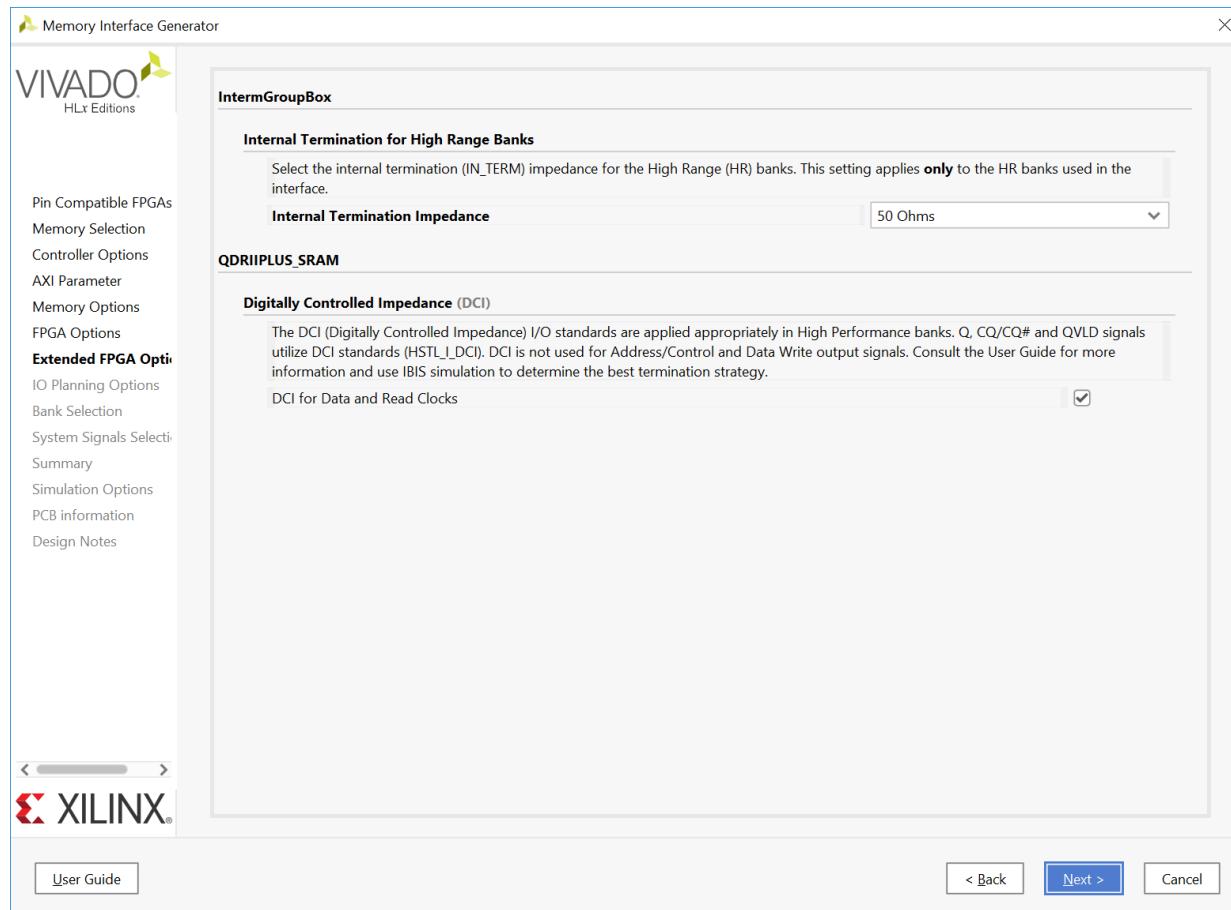


Figure 2-22:

- **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 read datapath from the QDR II+ SRAM. This selection is only for High Range banks.
- **Digitally Controlled Impedance (DCI)** – When selected, this option internally terminates the signals from the QDR II+ SRAM read path. DCI is available in the High Performance Banks.

Figure 2-23 shows the I/O Planning Options page.

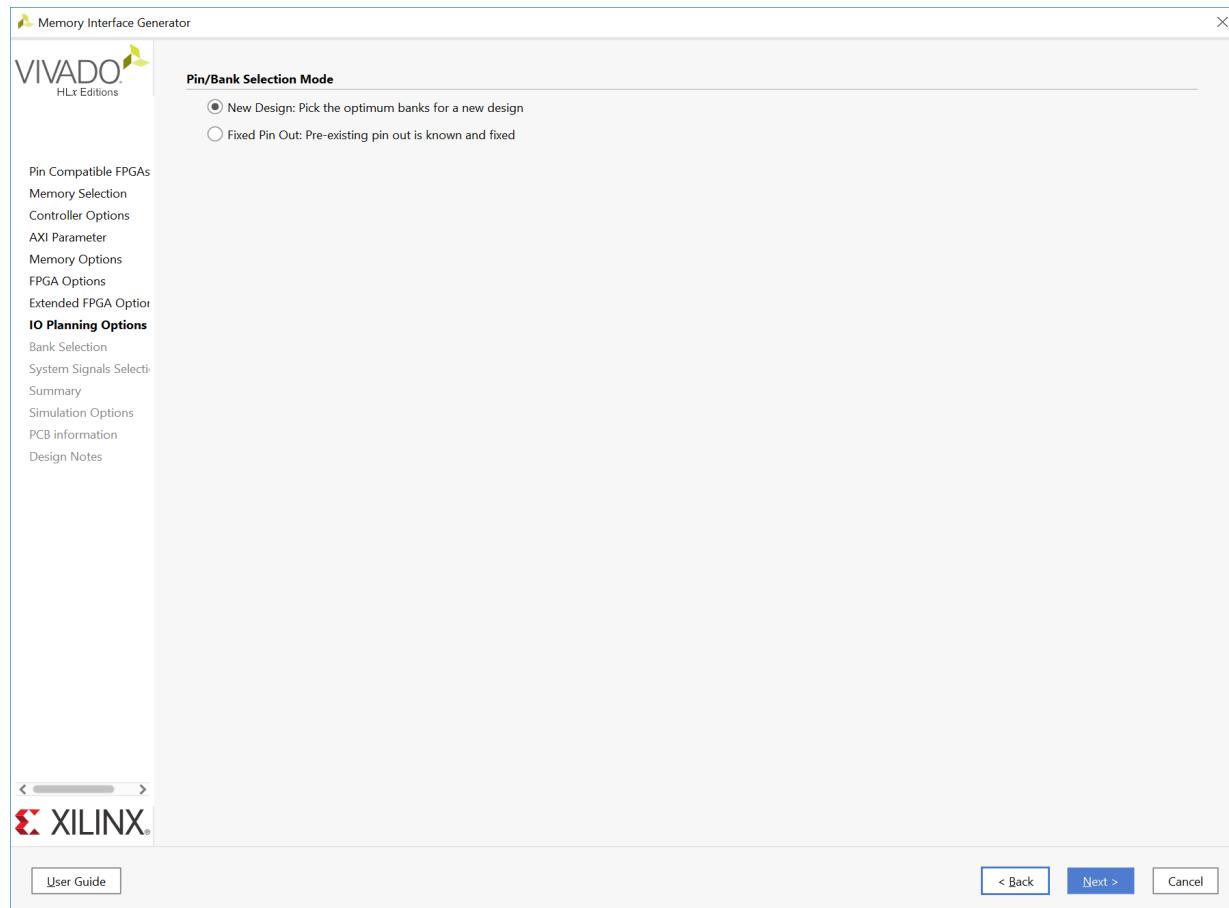


Figure 2-23:

- **Pin/Bank Selection Mode** – This allows you to specify an existing pinout and generate the RTL for this pinout or pick banks for a new design. Figure 2-24 shows the options for using an existing pinout. You 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. You cannot proceed until the MIG DRC has been validated by clicking **Validate**.

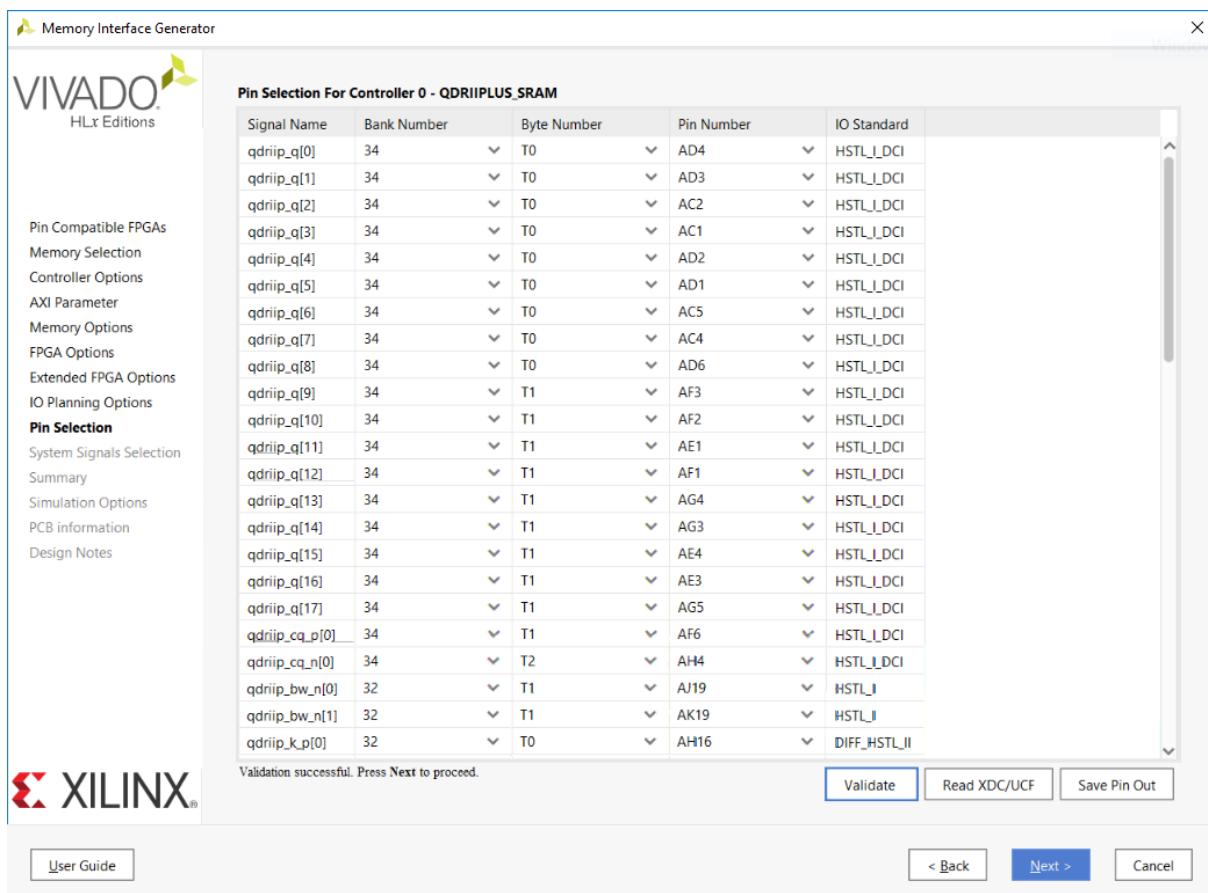


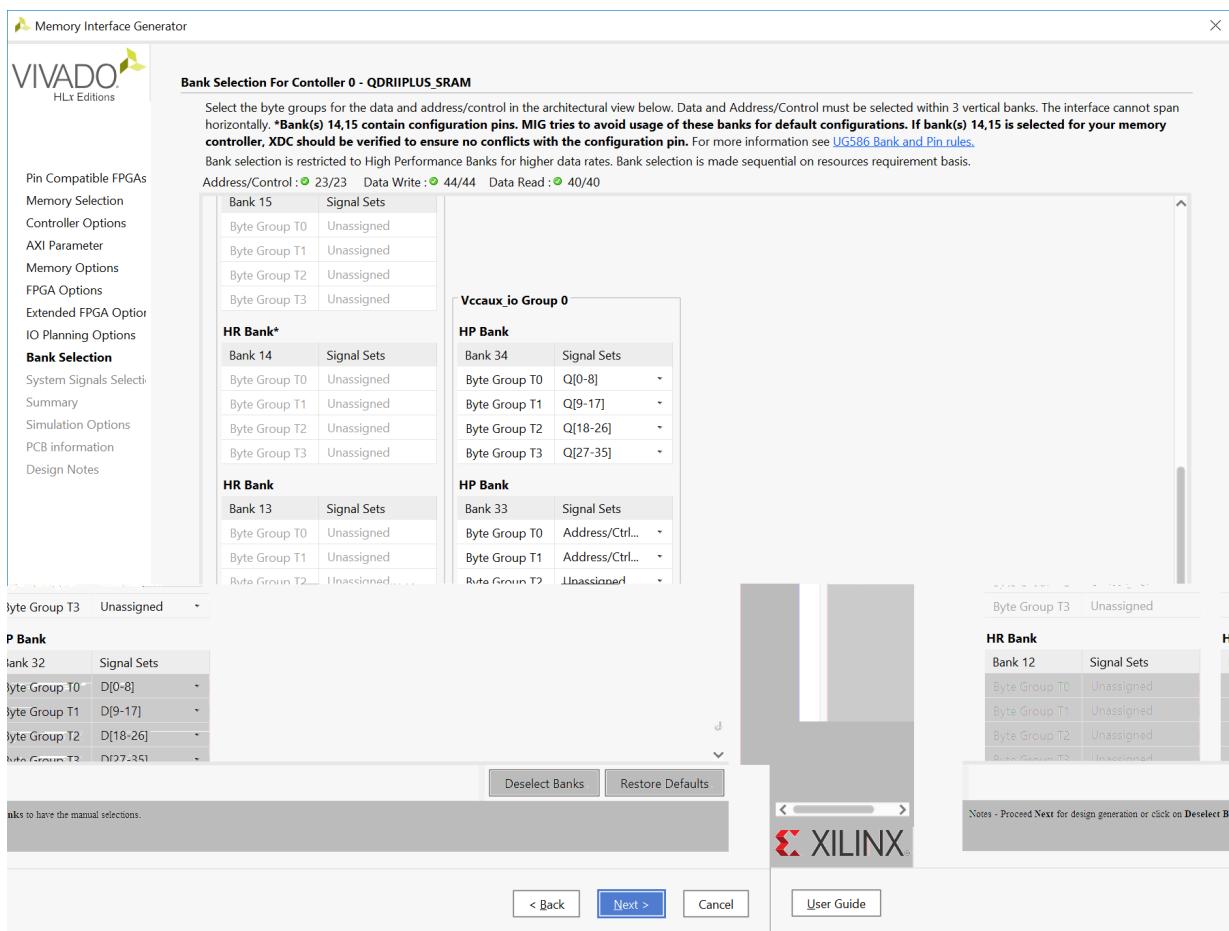
Figure 2-24:

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 **Deselect Banks**. To restore the defaults, click **Restore Defaults**. 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 2-25:*

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.

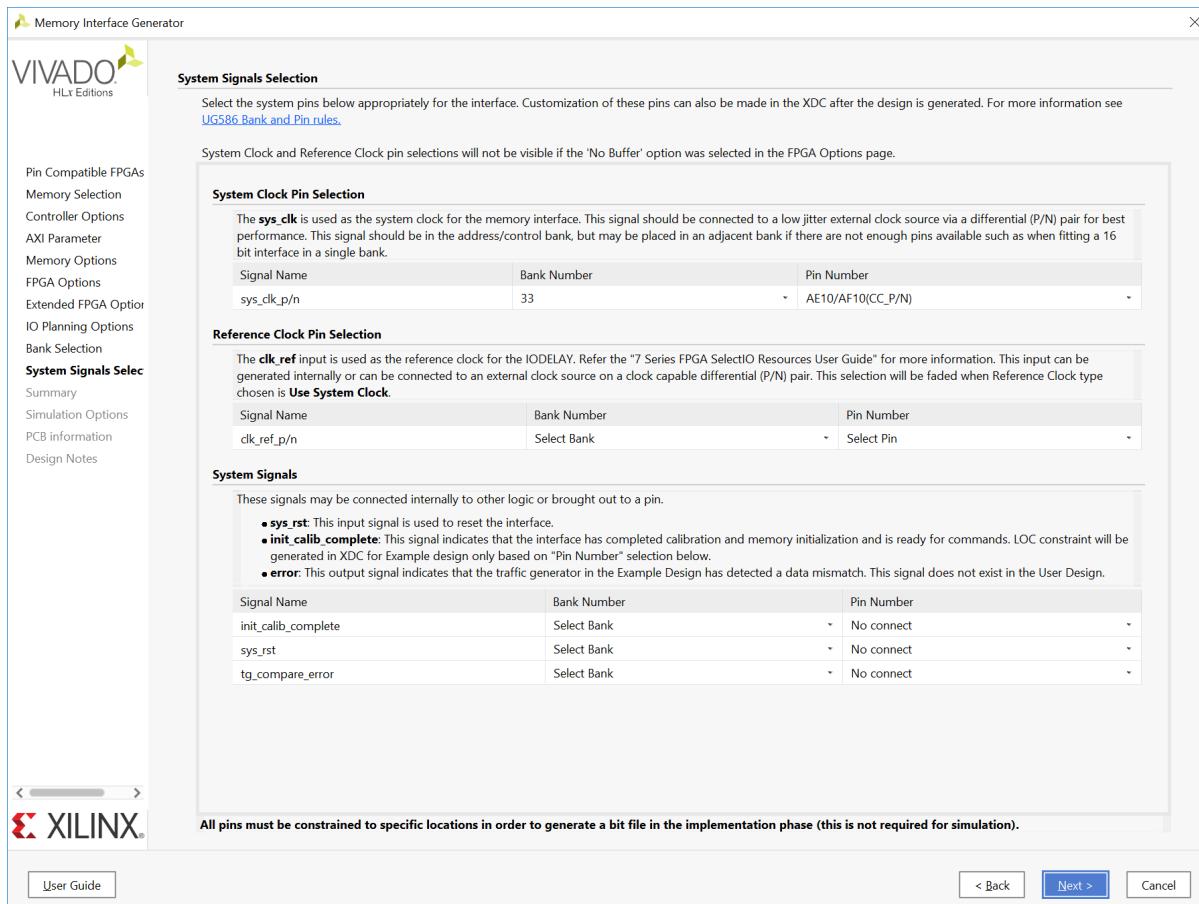


Figure 2-26:

- **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 2-21). 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\_HSTL\_I or HSTL\_I. If **sys\_clk** is not connected in a memory interface bank, the MIG tool selects an appropriate standard such as LVCMOS18 or LVDS. The XDC 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 2-21). The I/O standard is selected in a similar way as **sys\_clk** above.

- **sys\_rst** – This is the asynchronous 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 LVCMOS18 and LVCMOS25 for HP and HR banks, respectively. The default polarity of **sys\_rst** pin is active-Low. The polarity of **sys\_rst** pin varies based on the **System Reset Polarity** option chosen in **FPGA Options** page ([Figure 2-21](#)).
- **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.

This page provides the complete details about the 7 series FPGA memory core selection, interface parameters, Vivado IP catalog options, and FPGA options of the active project.

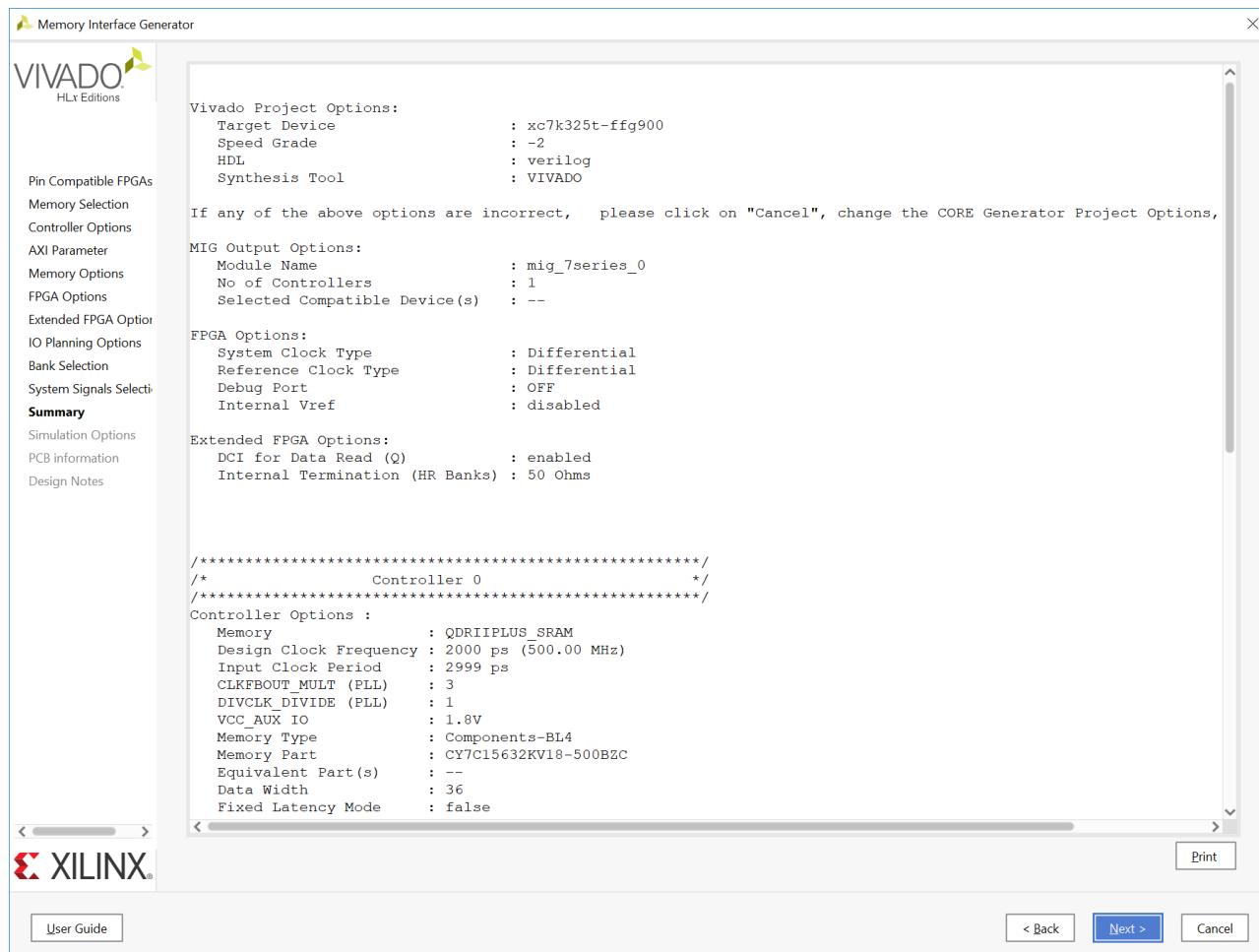


Figure 2-27:

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

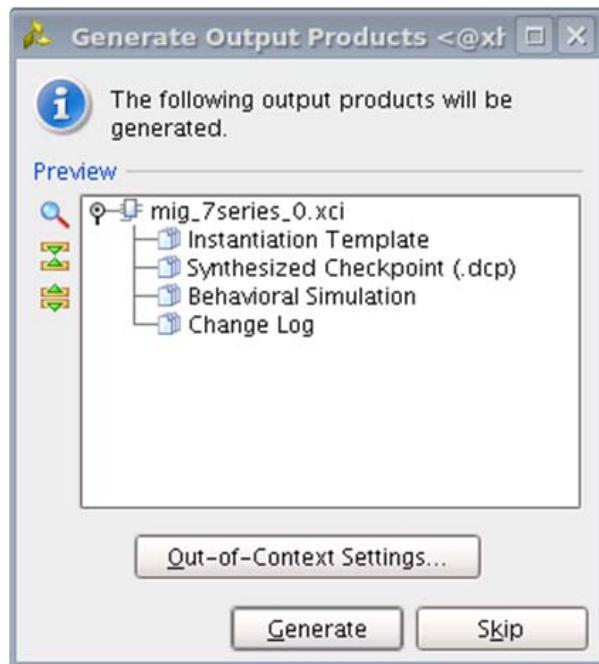
This page displays the PCB-related information to be considered while designing the board that uses the MIG tool generated designs. Click **Next** to move to the **Design Notes** page.

Click **Generate** 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.

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

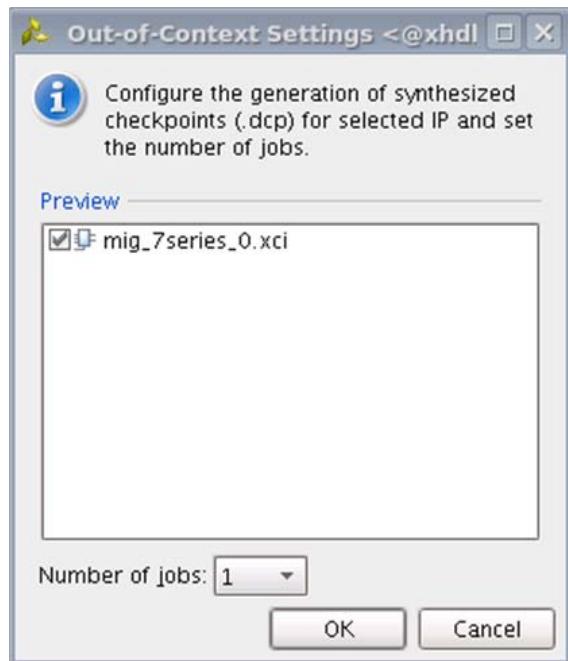
Click **Close** to complete the MIG tool flow.

1. After clicking **Generate**, the **Generate Output Products** window appears. This window has the **Out-of-Context Settings** as shown in [Figure 2-28](#).



*Figure 2-28:*

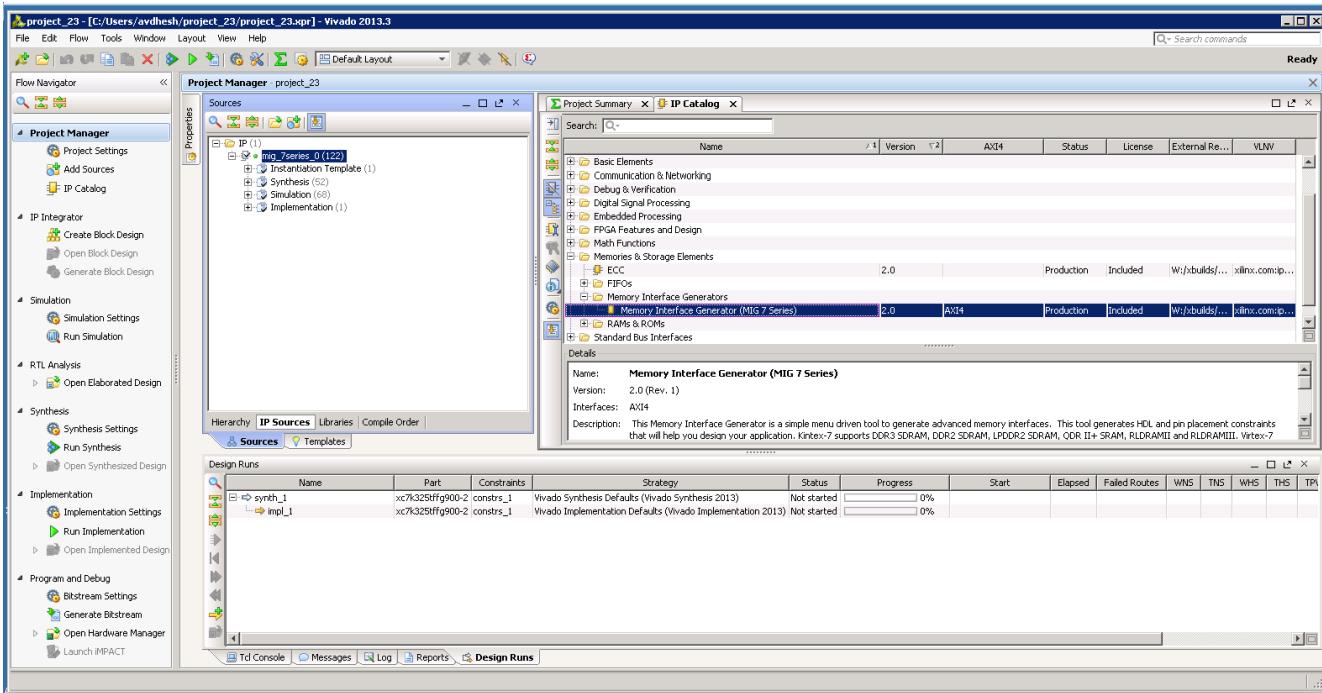
2. Click **Out-of-Context Settings** to configure generation of synthesized checkpoints. To enable the **Out-of-Context** flow, enable the check box. To disable the **Out-of-Context** flow, disable the check box. The default option is "enable" as shown in [Figure 2-29](#).



*Figure 2-29:*

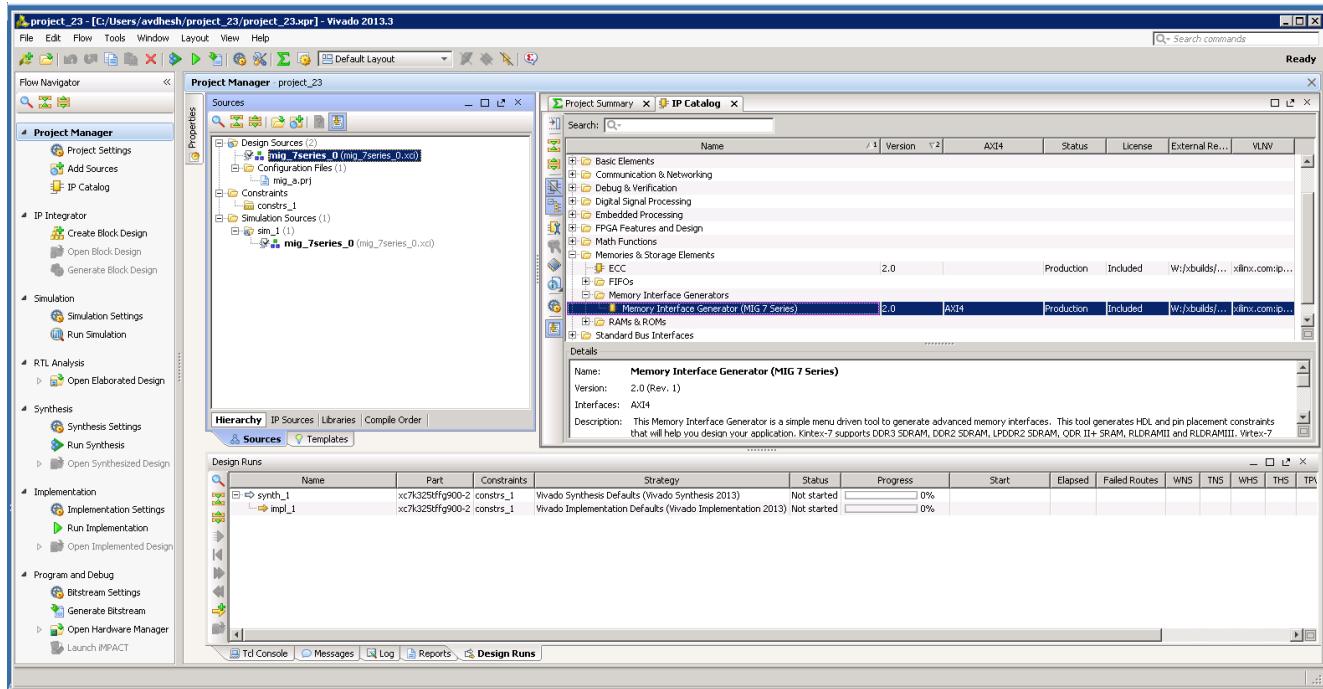
3. MIG designs comply with "Hierarchical Design" flow in Vivado. For more information, see the *Vivado Design Suite User Guide: Hierarchical Design* (UG905) [\[Ref 5\]](#) and the *Vivado Design Suite Tutorial: Hierarchical Design* (UG946) [\[Ref 6\]](#).

4. After generating the MIG design, the project window appears as shown in [Figure 2-30](#).



*Figure 2-30:*

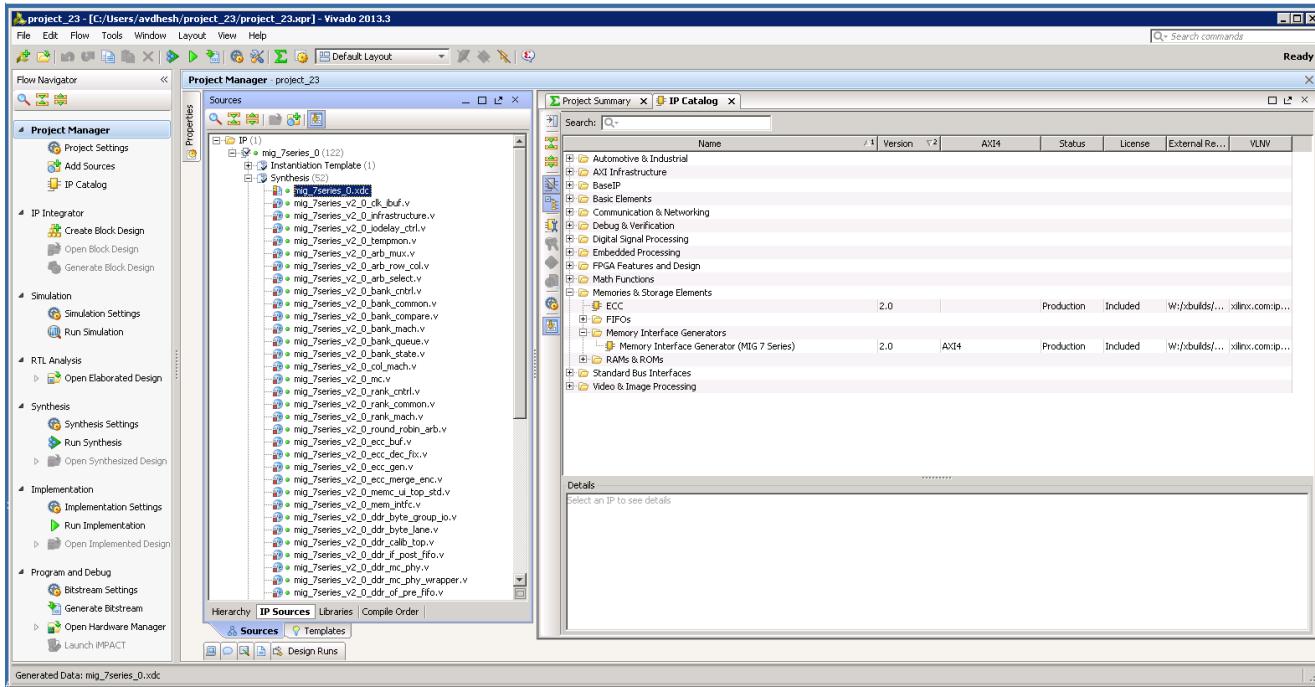
5. After project creation, the XCI file is added to the Project Hierarchy. The same view also displays the module hierarchies of the user design. The list of HDL and XDC files is available in the **IP Sources** view in the **Sources** window. Double-clicking on any module or file opens the file in the Vivado Editor. These files are read only.



*Figure 2-31:*

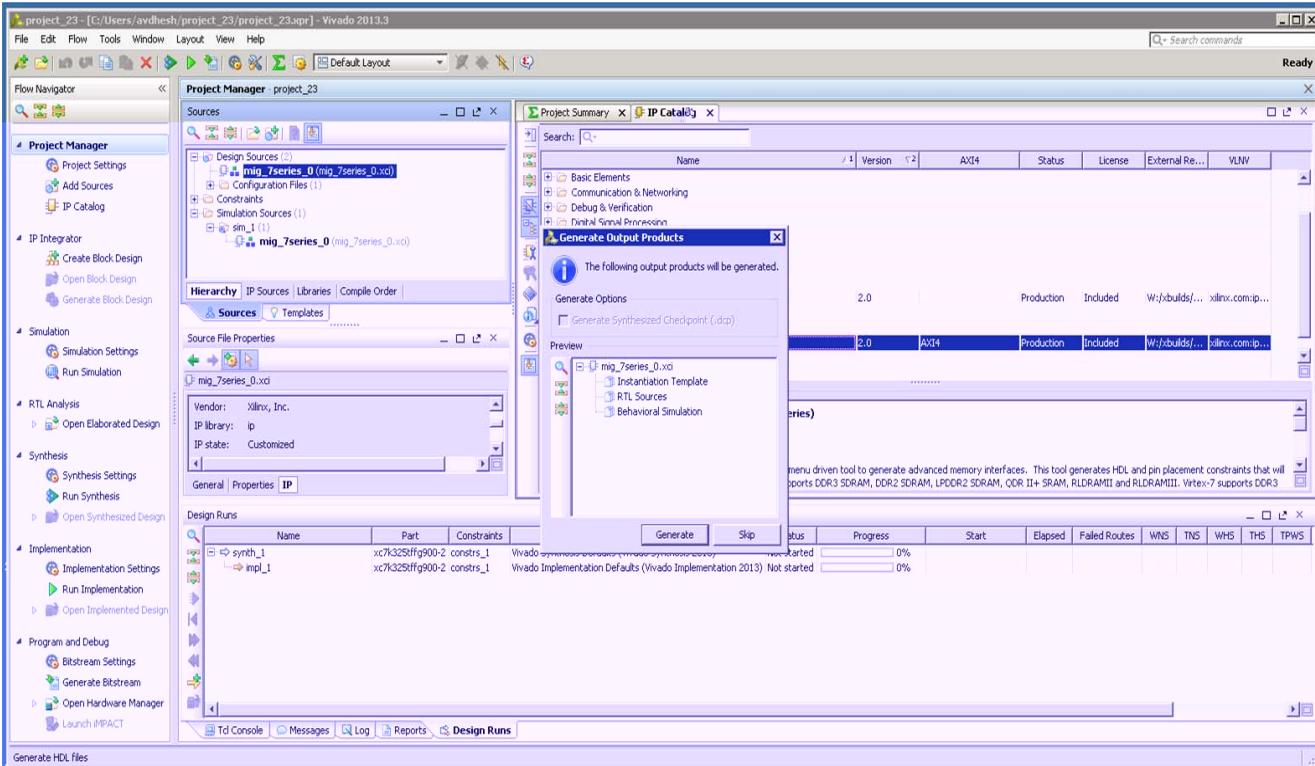
Design generation from MIG can be generated using the **Create Design** flow or the **Verify Pin Changes** and **Update Design** flows. There is no difference between the flow when generating the design from the MIG tool. Irrespective of the flow by which designs are generated from the MIG tool, the XCI file is added to the Vivado tool project. The implementation flow is the same for all scenarios because the flow depends on the XCI file added to the project.

6. All MIG generated user design RTL and XDC files are automatically added to the project. If files are modified and you wish to regenerate them, right-click the XCI file and select **Generate Output Products** (Figure 2-32).



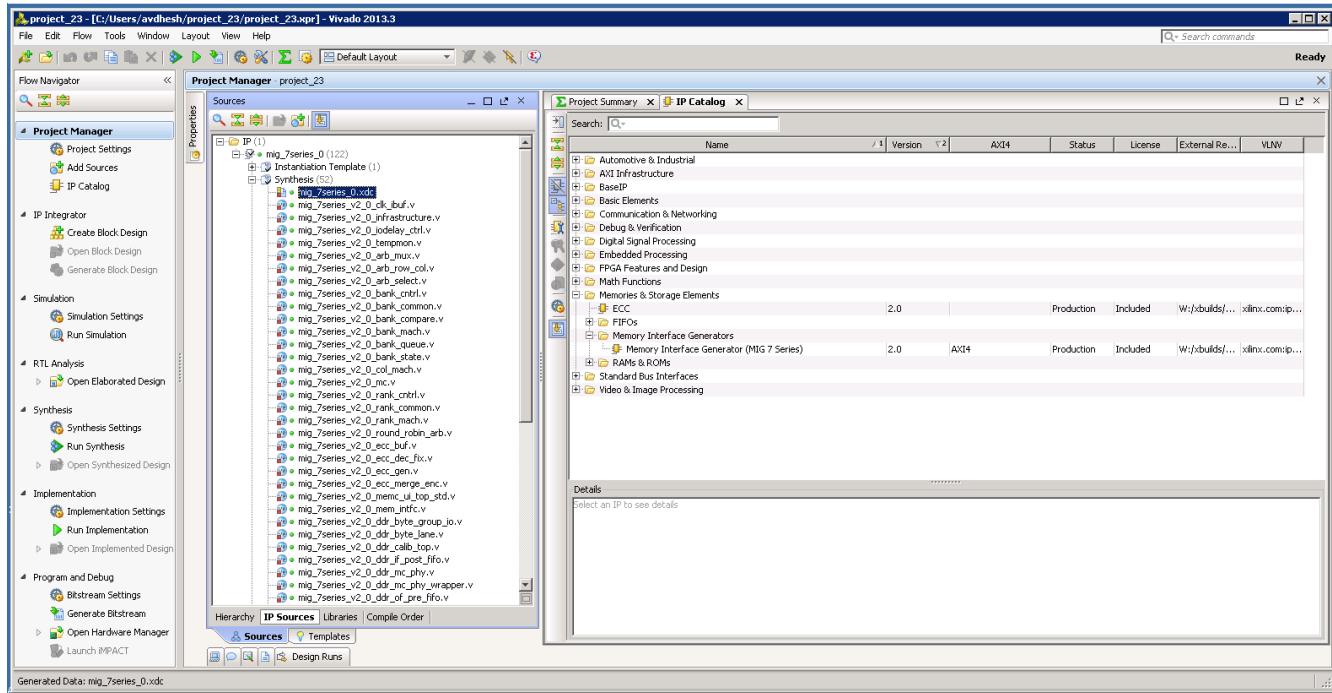
*Figure 2-32:*

- Clicking the **Generate Output Products** option brings up the **Manage Outputs** window (Figure 2-33).



*Figure 2-33:*

8. All user-design RTL and constraints files (XDC files) can be viewed in the **Sources > Libraries** tab ([Figure 2-34](#)).



*Figure 2-34:*

9. The Vivado Design Suite supports the **Open IP Example Design** flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in [Figure 2-35](#) and select.

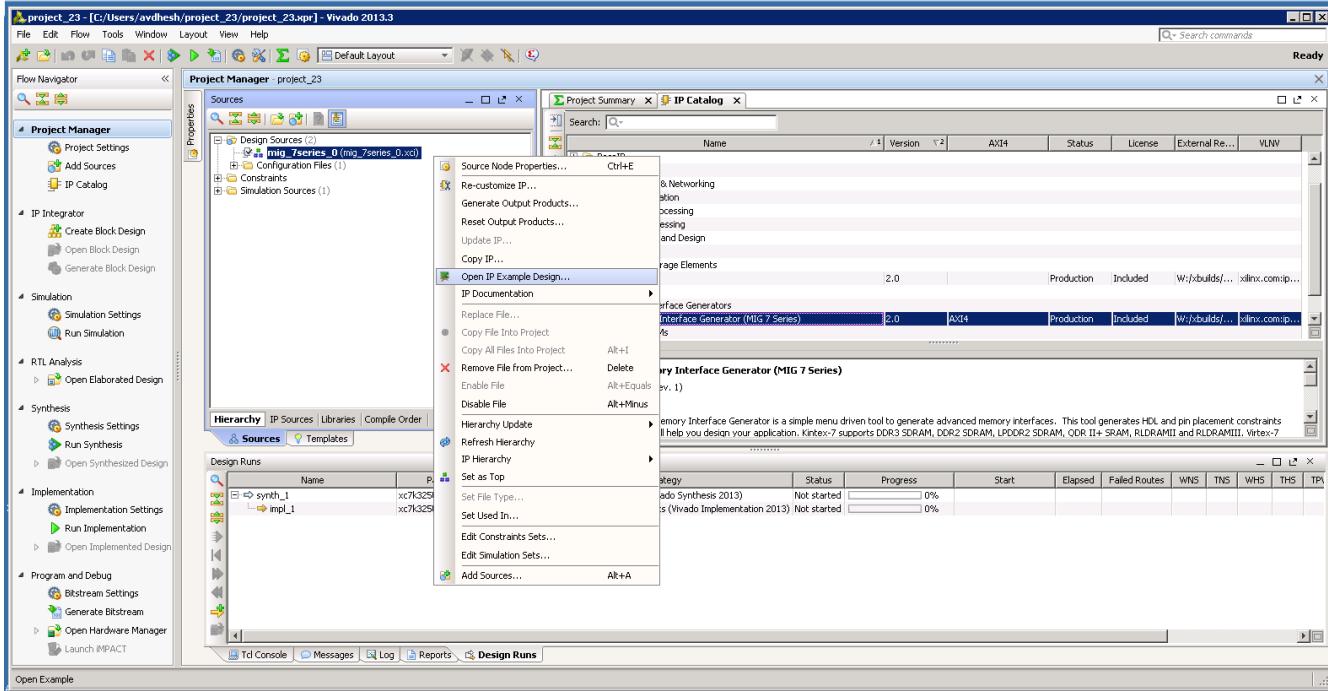


Figure 2-35:

10. This option creates a new Vivado project. Selecting the menu brings up a dialog box, which guides you to the directory for a new design project. Select a directory (or use the defaults) and click **OK**.

This launches a new Vivado project with all example design files and a copy of the IP. This project has **example\_top** as the Implementation top directory, and **sim\_tb\_top** as the Simulation top directory, as shown in Figure 2-36.

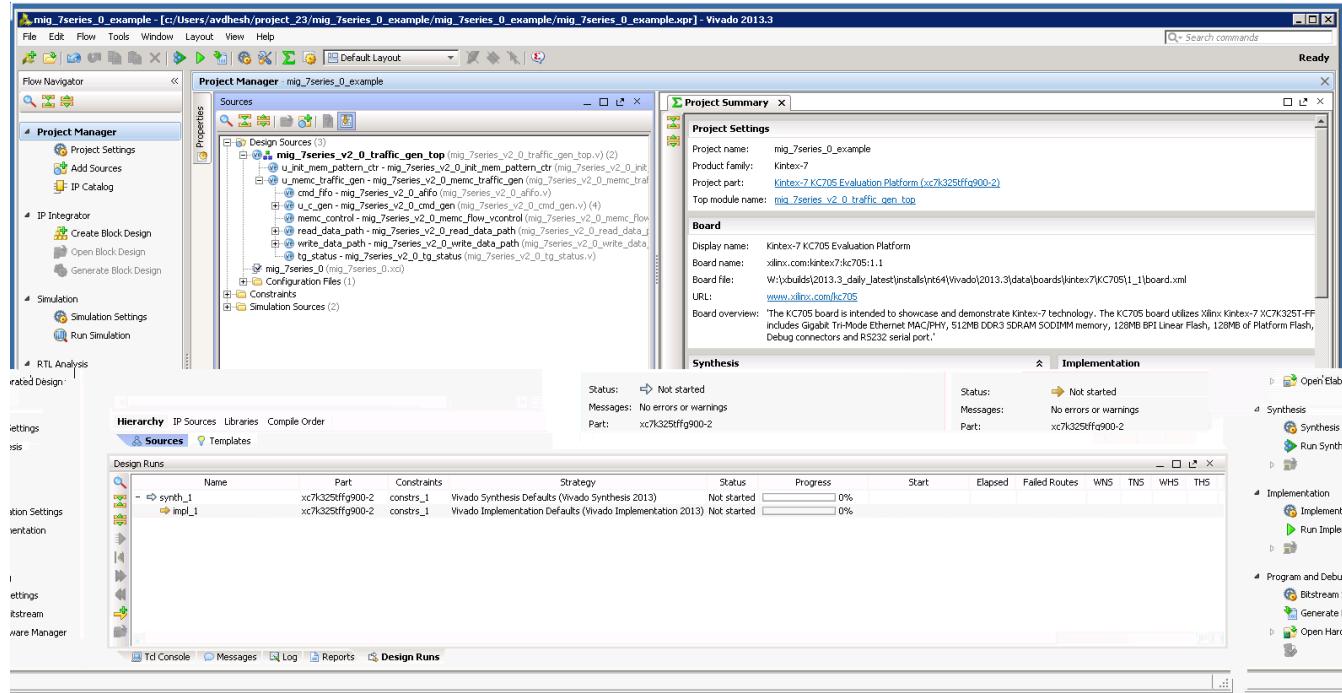


Figure 2-36:

- Click Generate Bitstream under Project Manager > Program and Debug to generate the BIT file for the generated design.

The `<project_directory>/<project_directory>.runs/ impl_1` directory includes all report files generated for the project after running the implementation. It is also possible to run the simulation in this project.

- Recustomization of the MIG IP core can be done by using the Recustomize IP option. It is not recommended to recustomize the IP in the `example_design` project. The correct solution is to close the `example_design` project, go back to original project and customize there. Right-click the XCI file and click Recustomize IP (Figure 2-37) to open the MIG GUI and regenerate the design with the preferred options.

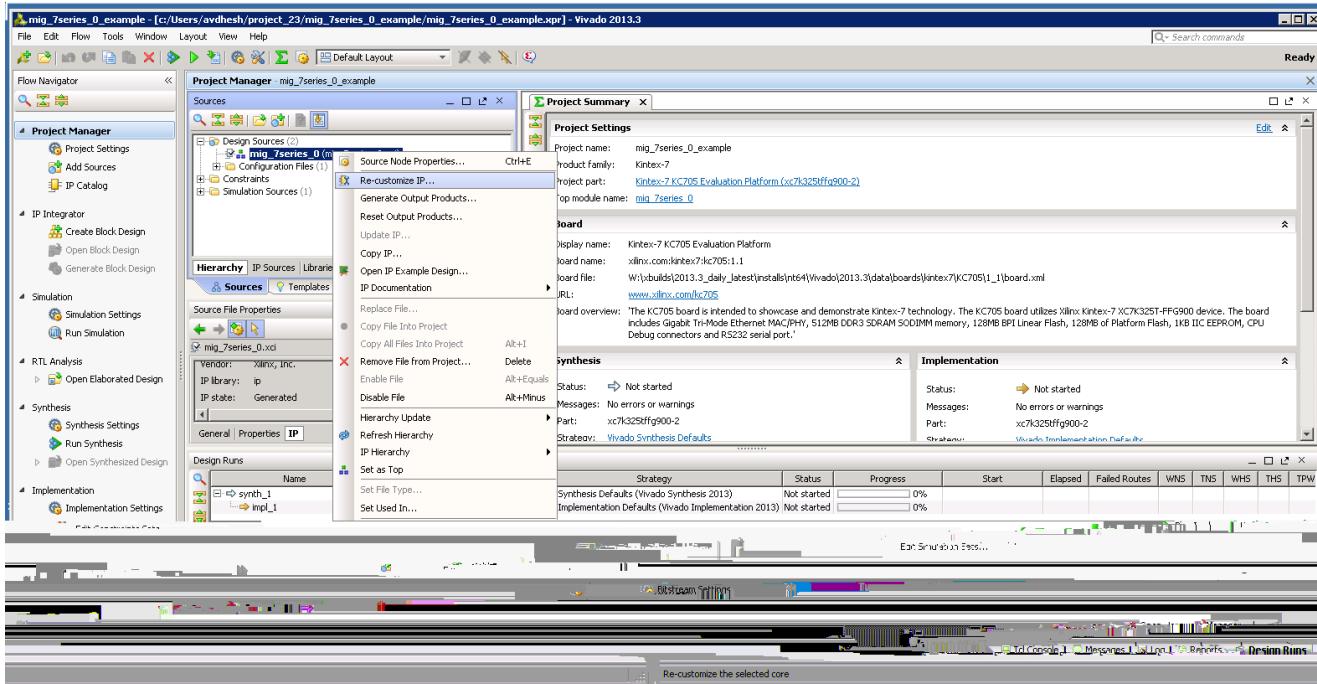


Figure 2-37:

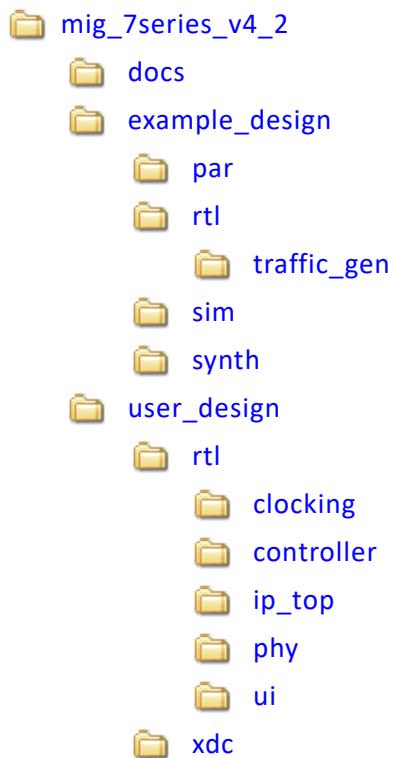
## Directory Structure and File Descriptions

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

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 285](#) of the MIG design creation flow.

The output directory structure of the selected Memory Controller (MC) design from the MIG tool is shown here. There are three folders created within the `<component name>` directory:

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



The 7 series FPGAs core directories and their associated files are listed in this section for Vivado implementations.

**<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 Vivado logic analyzer feature module is also included in this directory structure.

Table 2-1 lists the files in the **example\_design/rtl** directory.

*Table 2-1:*

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:

memc_traffic_gen.v	This is the top-level module of the traffic generator.
cmd_gen.v	This is the command generator. This module provides independent control of generating the types of commands, addresses, and burst lengths.
cmd_prbs_gen.v	This pseudo-random binary sequence (PRBS) generator generates PRBS commands, addresses, and burst lengths.
memc_flow_vcontrol.v	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	This is the top-level for the read datapath.
read_posted_fifo.v	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	This module generates timing control for reads and ready signals to memc_flow_vcontrol.v.
write_data_path.v	This is the top-level for the write datapath.
wr_data_gen.v	This module generates timing control for writes and ready signals to memc_flow_vcontrol.v.
s7ven_data_gen.v	This module generates different data patterns.
a_fifo.v	This is a synchronous FIFO using LUT RAMs.
data_prbs_gen.v	This 32-bit linear feedback shift register (LFSR) generates PRBS data patterns.
init_mem_pattern_ctr.v	This module generates flow control logic for the traffic generator.
traffic_gen_top.v	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	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	This module compares the memory read data against compare data generated from the <code>data_gen</code> module. The error signal is asserted if the comparison is not equal.
vio_init_pattern_bram.v	This module takes external defined data inputs as its block RAM init pattern. It allows users to change simple test data pattern without recompilation.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of memc\_traffic\_gen in generated output is now mig\_7series\_v4\_2\_memc\_traffic\_gen.

Send Feedback

[Table 2-3](#) lists the files in the `example_design/sim` directory.

*Table 2-3:*

<code>ies_run.sh</code> <sup>(1)</sup>	Linux Executable file for simulating the design using IES simulator.
<code>vcs_run.sh</code> <sup>(1)</sup>	Linux Executable file for simulating the design using VCS simulator.
<code>readme.txt</code> <sup>(1)</sup>	Contains the details and prerequisites for simulating the designs using Mentor Graphics Questa Advanced Simulator, IES, and VCS simulators.
<code>sim_tb_top.v</code>	This file is the simulation top-level file.

**Notes:**

1. The `ies_run.sh` and `vcs_run.sh` files are generated in the folder `mig_7series_ex/imports` when the example design is created using **Open IP Example Design** for the design generated with **Component Name** entered in Vivado IDE as `mig_7series_0`.

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

The `user_design` folder contains the following:

- `rtl` and `xdc` folders
- Top-level wrapper module `<component_name>.v/vhd`
- Top-level modules `<component_name>_mig.v/vhd` and `<component_name>_mig_sim.v/vhd`

The top-level wrapper file `<component_name>.v/vhd` has an instantiation of top-level file `<component_name>_mig.v/vhd`. Top-level wrapper file has no parameter declarations and all the port declarations are of fixed width.

Top-level files `<component_name>_mig.v/vhd` and `<component_name>_mig_sim.v/vhd` have the same module name as `<component_name>_mig`. These two files are same in all respects except that the file `<component_name>_mig_sim.v/vhd` has parameter values set for simulation where calibration is in fast mode `viz., SIM_BYPASS_INIT_CAL = "FAST"` etc.




---

**IMPORTANT:** The top-level file `<component_name>_mig.v/vhd` is used for design synthesis and implementation, whereas the top-level file `<component_name>_mig_sim.v/vhd` is used in simulations.

---

The top-level wrapper file serves as an example for connecting the `user_design` to the 7 series FPGA memory interface core.

### **user\_design/rtl/clocking**

[Table 2-4](#) lists the files in the `user_design/rtl/clocking` directory.

**Table 2-4:**

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.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of clk\_ibuf in generated output is now mig\_7series\_v4\_2\_clk\_ibuf.

**user\_design/rtl/phy**

Table 2-5 lists the files in the **user\_design/rtl/phy** directory:

**Table 2-5:**

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 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.

Table 2-5:

(Cont'd)

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.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of qdr\_phy\_top in generated output is now mig\_7series\_v4\_2\_qdr\_phy\_top.

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

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

Table 2-6:

<component name>.xdc	This file is the XDC for the core of the user design.

This feature verifies the input XDC for bank selections, byte selections, and pin allocation. It also generates errors and warnings in a separate dialog box when you click **Validate** on the page. This feature is useful to verify the XDC for any pinout changes made after the design is generated from the MIG tool. You must load the MIG generated **.prj** file, the original **.prj** file without any modifications. In the Vivado IP catalog, 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 XDC is not sufficient; it is mandatory to proceed with design generation to get the XDC with updated clock and phaser-related constraints and RTL top-level module for various updated Map parameters.

The Update Design feature is required in the following scenarios:

- A pinout is generated using an older version of MIG and the design is to be revised to the current version of MIG. In MIG the pinout allocation algorithms have been changed for certain MIG designs.
- A pinout is generated independent of MIG or is modified after the design is generated. When a design is generated from MIG, the XDC and HDL code are generated with the correct constraints.

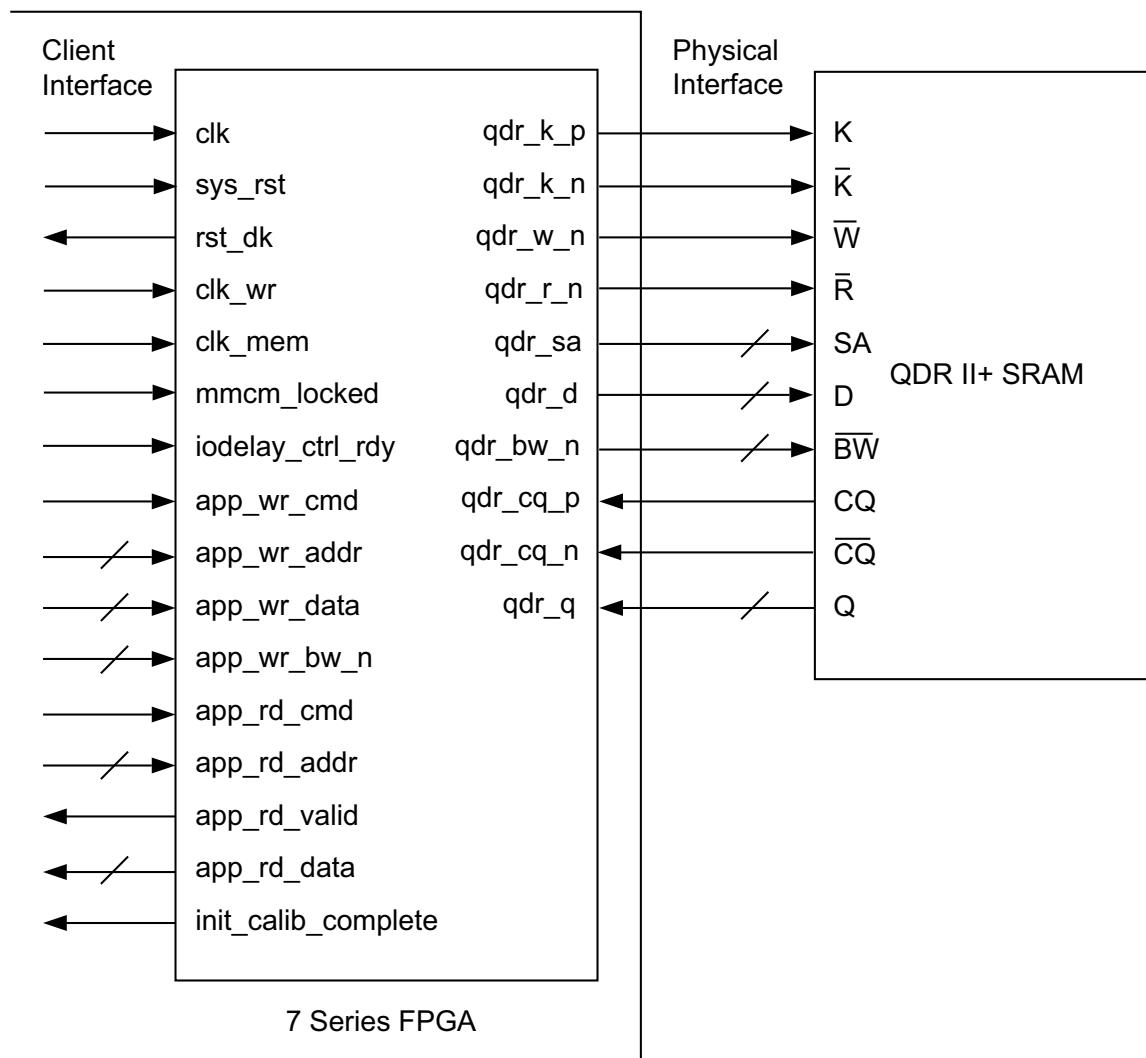
Here are the rules verified from the input XDC:

- If a pin is allocated to more than one signal, the tool reports an error. Further verification is not done if the XDC 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.
- $V_{REF}$  I/Os should be used as GPIOs when an internal  $V_{REF}$  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 two byte lanes.
  - A byte lane should contain pins of only one read byte, for example, Q[8:0] or Q[17:9].
  - 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.
  - $V_{REF}$  I/O can be used only when the internal  $V_{REF}$  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[8:0] or D[17:9].
  - Irrespective of internal  $V_{REF}$  usage,  $V_{REF}$  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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  is used.
  - Status signals:
    - The **sys\_rst** signal should be allocated in the bank where the  $V_{REF}$  I/O is unallocated or the internal  $V_{REF}$  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.

Figure 2-38 shows a high-level block diagram of the 7 series FPGA QDR II+ SRAM 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.

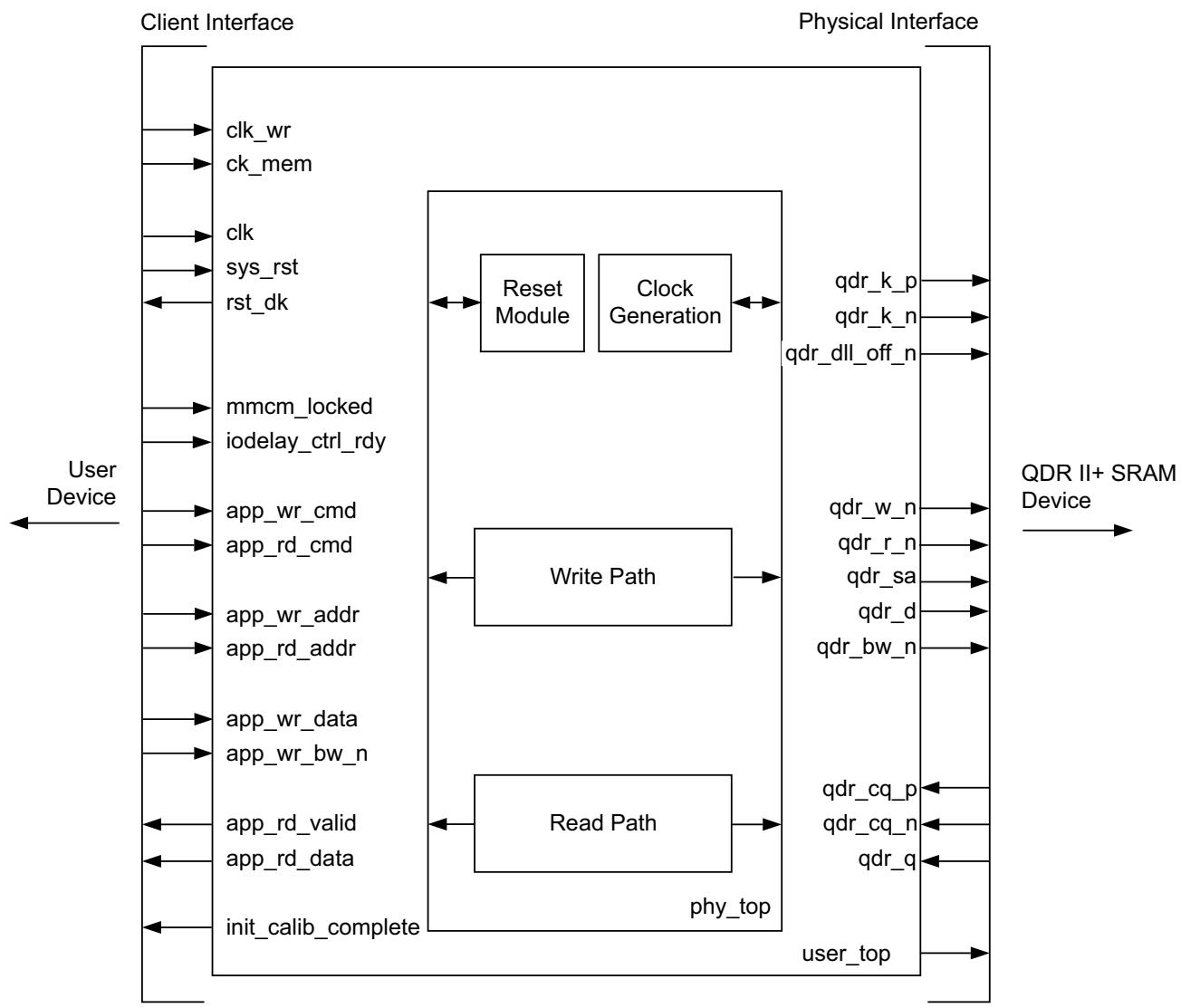


UG586\_c2\_34\_090911

Figure 2-38:

The PHY is composed of these elements, as shown in [Figure 2-39](#):

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



UG586\_c2\_35\_090911

*Figure 2-39:*

The client interface (also known as the user interface) uses a protocol based entirely on single data rate (SDR) signals to make read and write requests. For more details about this protocol, see the [User Interface](#) section. 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. For more information, see the [Physical Interface](#) section.

Within the PHY, logic is broken up into read and write paths. The read path is responsible for calibration and providing read responses back to you with a corresponding valid signal. For more details about this process, see the [Calibration](#) section. The write path generates the QDR II+ signaling for generating read and write requests. This includes control signals, address, data, and byte writes.

The client interface connects the 7 series FPGA user design to the QDR II+ SRAM solutions core to simplify interactions between you 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 2-7](#). To accommodate for burst length four devices, the client interface contains ports for two read and two write transactions. When using burst length four, only the ports ending in zero should be used.

*Table 2-7:*

init_calib_complete	Output	<b>Calibration Done.</b> This signal indicates to the user design that read calibration is complete and you 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.

Table 2-7:

(Cont'd)

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.
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 2-40](#).

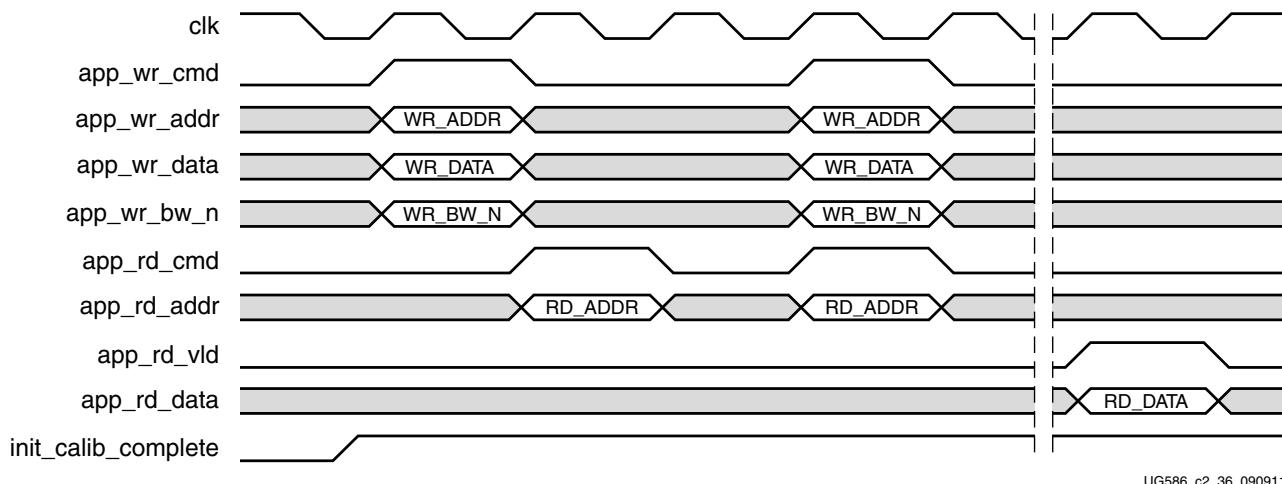


Figure 2-40:

Before any requests can be made, the `init_calib_complete` signal must be asserted High, as shown in [Figure 2-40](#), 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 2-40](#) 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. If desired, you can add this functionality. The data returned is not necessarily from the read commands shown in [Figure 2-40](#) and is solely to demonstrate protocol.

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 MMCM and one PLL are 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:

- **Memory Reference Clock** – The memory reference clock is required to be at the same frequency as that of the QDR II+ memory interface clock.
- **Frequency Reference Clock** – The frequency reference clock must be equal to the memory clock frequency for frequencies  $\geq$  400 MHz and 2x the memory clock frequency for frequencies  $<$  400 MHz such that it meets the reference range requirement of 400 MHz to 1,066 MHz.
- **Phase Reference Clock from the PLL** – 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.

Figure 2-41 shows the clocking architecture.

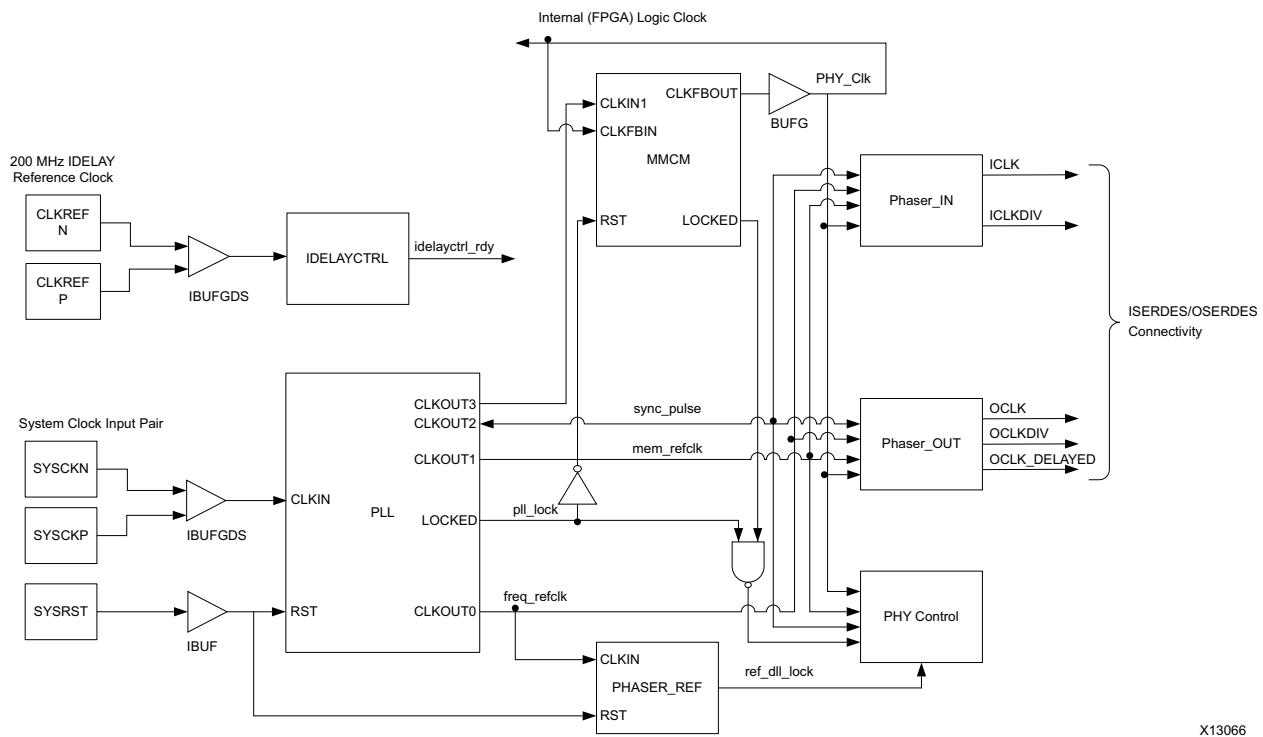


Figure 2-41:

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 1:1 ratio is not required. The PLL input divider (D) can be any value listed in the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] 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 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 343](#).

The internal FPGA logic clock generated by the PLL is clocked by a global clocking resource at half the frequency of the QDR 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 2-8](#) lists the signals used in the infrastructure module that provides the necessary clocks and reset signals required in the design.

*Table 2-8:*

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 FPGA logic 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 to 1,066 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.
rstdivo	Output	Reset output synchronized to internal FPGA logic half frequency clock.

The physical interface is the connection from the FPGA memory interface solution to an external QDR II+ SRAM device. The I/O signals for this interface are shown in [Table 2-9](#).

These signals can be directly connected to the corresponding signals on the QDR II+ SRAM device.

*Table 2-9:*

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 QDR II+ SRAM device.
qdr_k_n	InOut	<b>QDR Clock K#.</b> This is the inverted input clock to the memory device.
qdr_k_p	InOut	<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.
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 2-42 shows the physical interface protocol for a four-word memory device.

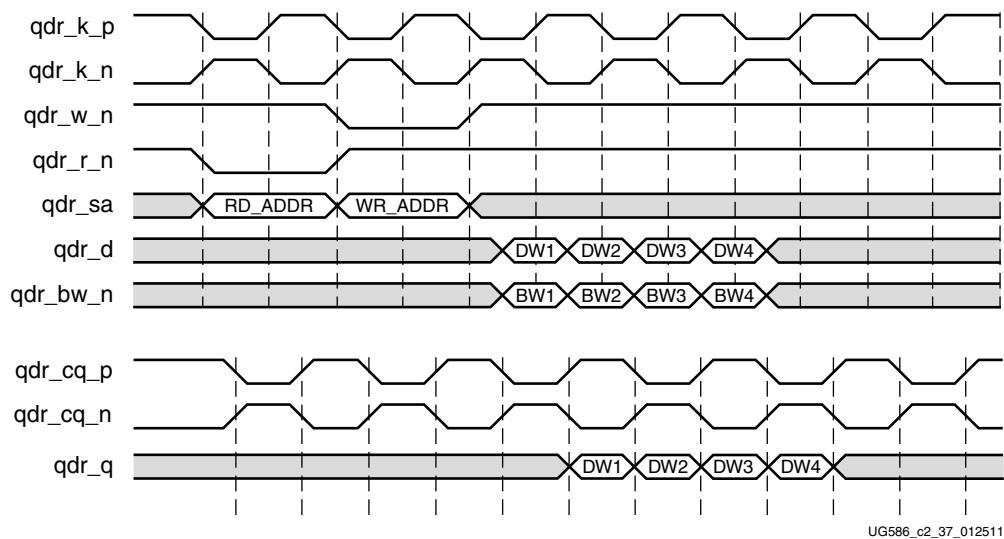


Figure 2-42:

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\_q** signal is edge aligned to **qdr\_cq\_p** and **qdr\_cq\_n**

## *PHY Architecture*

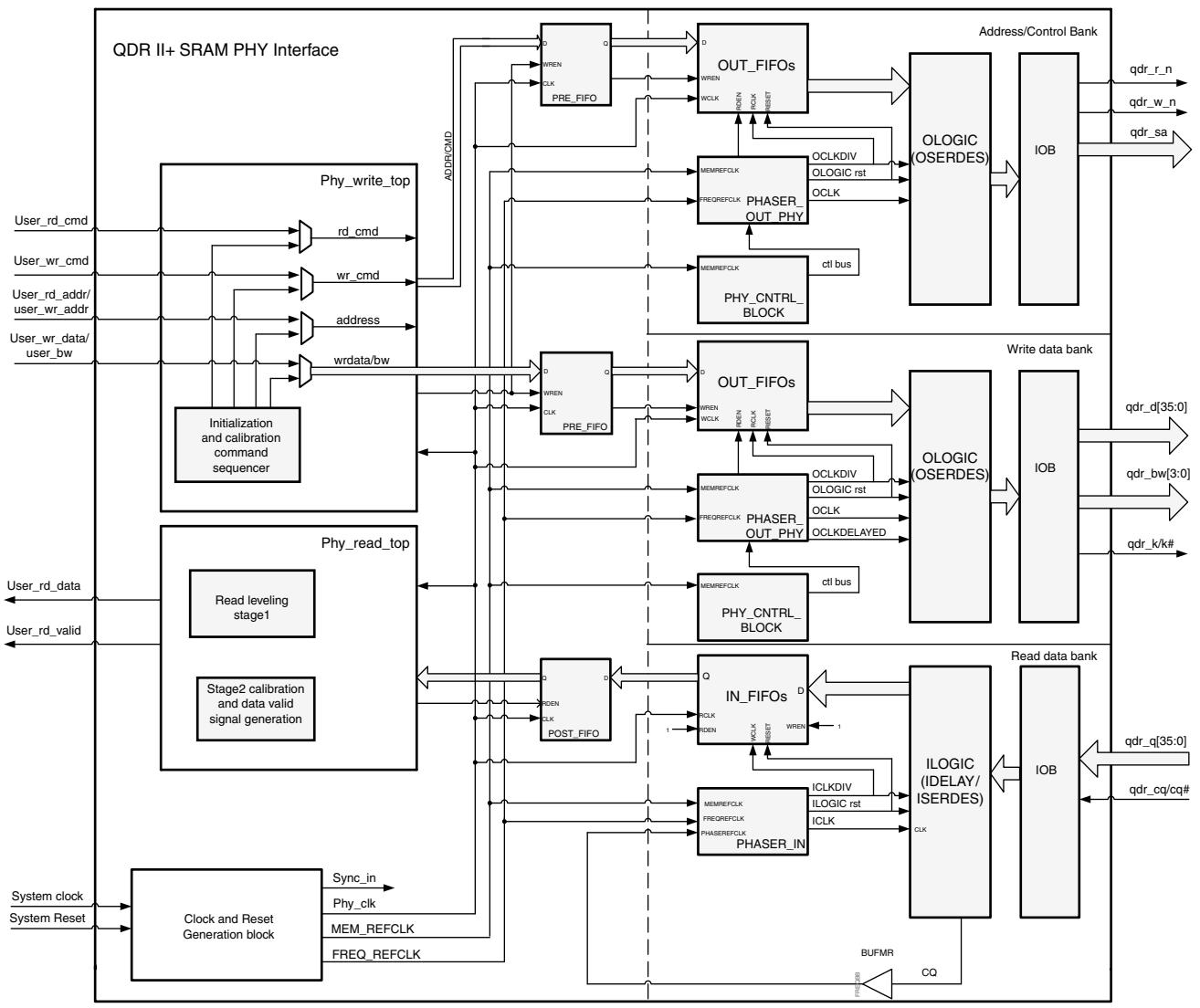
The 7 series FPGA PHY is composed of dedicated blocks and soft calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high-performance physical layers.

Some of the dedicated blocks that are used in the QDR II+ SRAM PHY and their features are described as follows:

- 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 FPGA logic 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 logic.

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 2-43:

The write path to the QDR II+ 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 use DDR formatting to achieve the required four-word burst within the given clock periods. [Figure 2-44](#) shows a high-level block diagram of the write path and its submodules.

## ***Output Architecture***

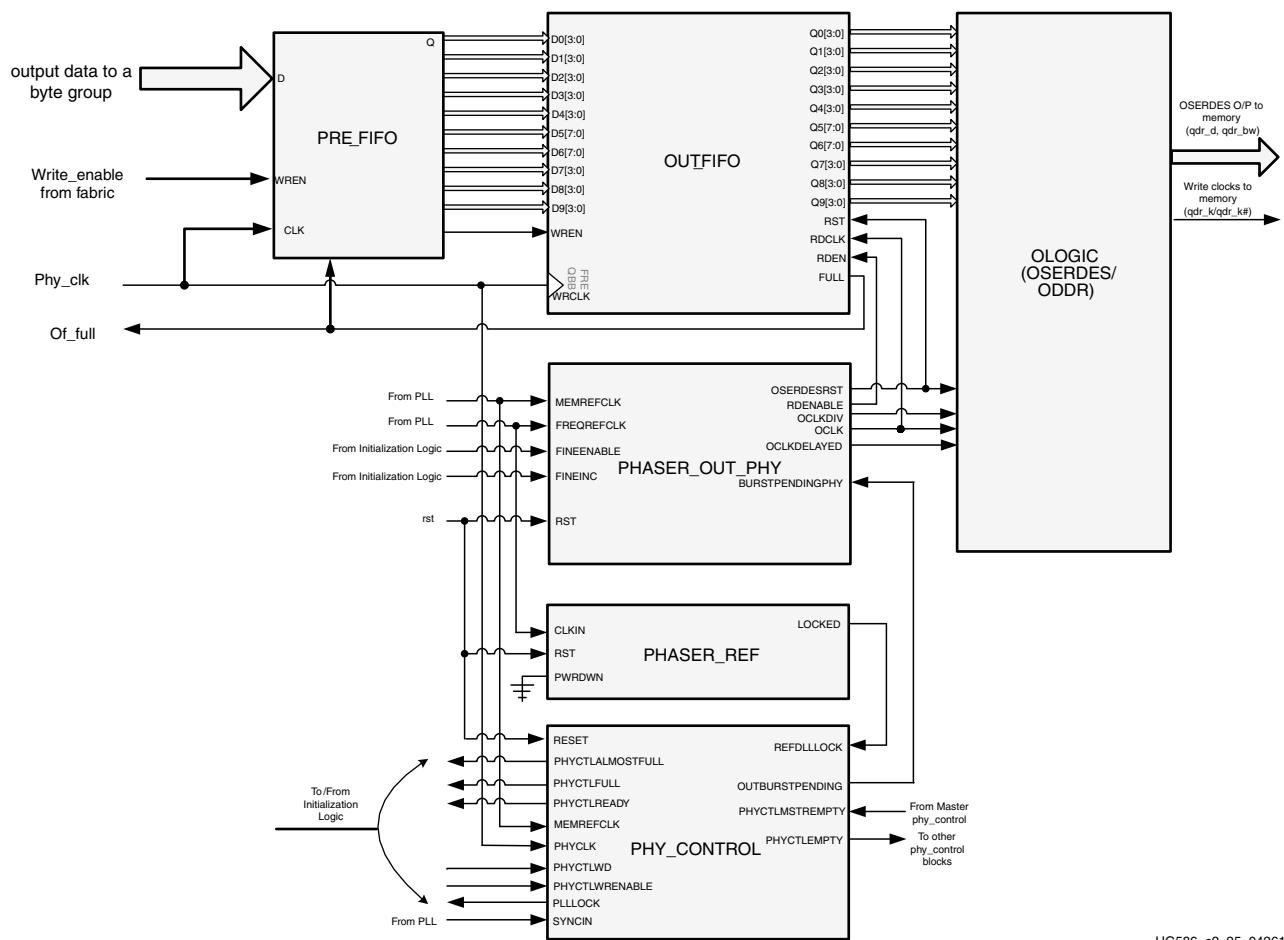
The output path of the QDR II+ interface solution uses OUT\_FIFOs, PHASER\_OUT\_PHY, and OSERDES primitives available in the 7 series FPGAs. These blocks are used for clocking all outputs of the PHY to the memory device.

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. OCLK\_DELAYED tap position is calibrated using a PHASER\_OUT stage 2 and stage 3 delay to determine the center position of a bit window. The detail of the K clock calibration flow is described in the [Write Calibration](#) section.

PO stage 2 fine delay elements are used for either decrement or increment. The direction of PO stage 2 taps adjustment is determined during the K clock left edge detection as described above. A positive skew has the PO taps decreased until the correct calibration pattern is obtained. A negative skew has the PO taps increased until the correct calibration pattern is lost. After all of the other bytes have been deskewed, the OCLK\_DELAY tap is moved to the calibrated position that has been obtained during the first part of write calibration.

The OUT\_FIFOs serve as a temporary buffer to convert the write data from the FPGA logic domain to the PHASER clock domain, which clocks out the output data from the I/O logic. The FPGA logic writes into the OUT\_FIFOs in the FPGA logic 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.

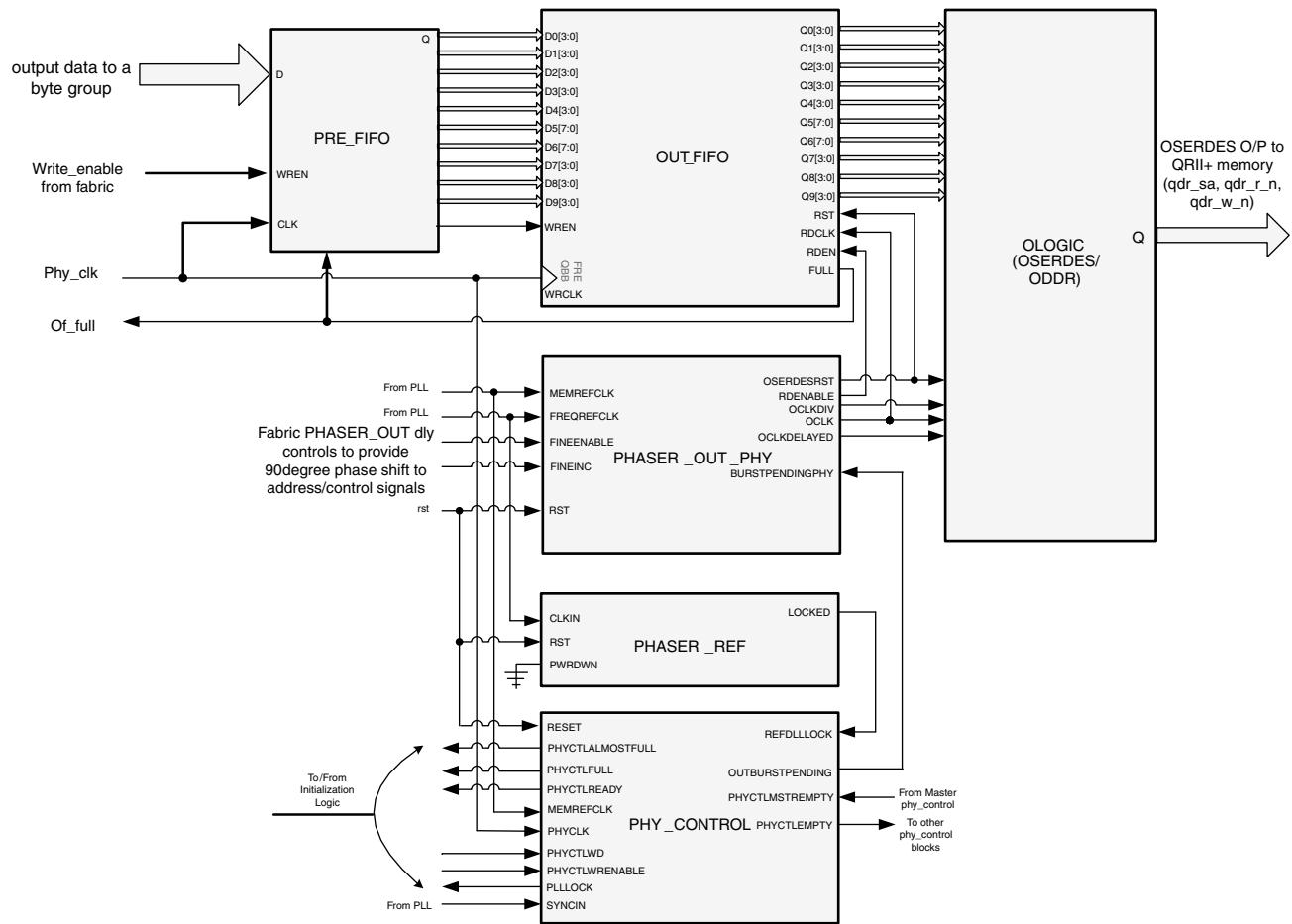
The clocking details of the write paths using PHASER\_OUT are shown in [Figure 2-44](#).



UG586\_c2\_25\_042611

*Figure 2-44:*

The clocking details of the address/control using PHASER\_OUT are shown in [Figure 2-45](#).


UG586\_c2\_26\_042611

*Figure 2-45:*

Because the address/command and write data are provided by the user backend, the QDR PHY transfers the signals from the FPGA logic 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:

1. 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 [Table 2-10](#) and [Table 2-11](#).

*Table 2-10:*

Field	AO1	Major OP	Minor OP	Event Delay	Seq	Data Offset	IndexHi (Rank)	IndexLo (Bank)	AO0	Command Offset	Non-Data	Read	Data
-------	-----	----------	----------	-------------	-----	-------------	----------------	----------------	-----	----------------	----------	------	------

*Table 2-11:*

O-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
					The STALL operation delays the issue of the Ready signal from pc_phy_counters to the sequencing state machines.
	1 – PRE	5'b000xx – STALL	DC	DC	
		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	

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. Now, 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 you 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.

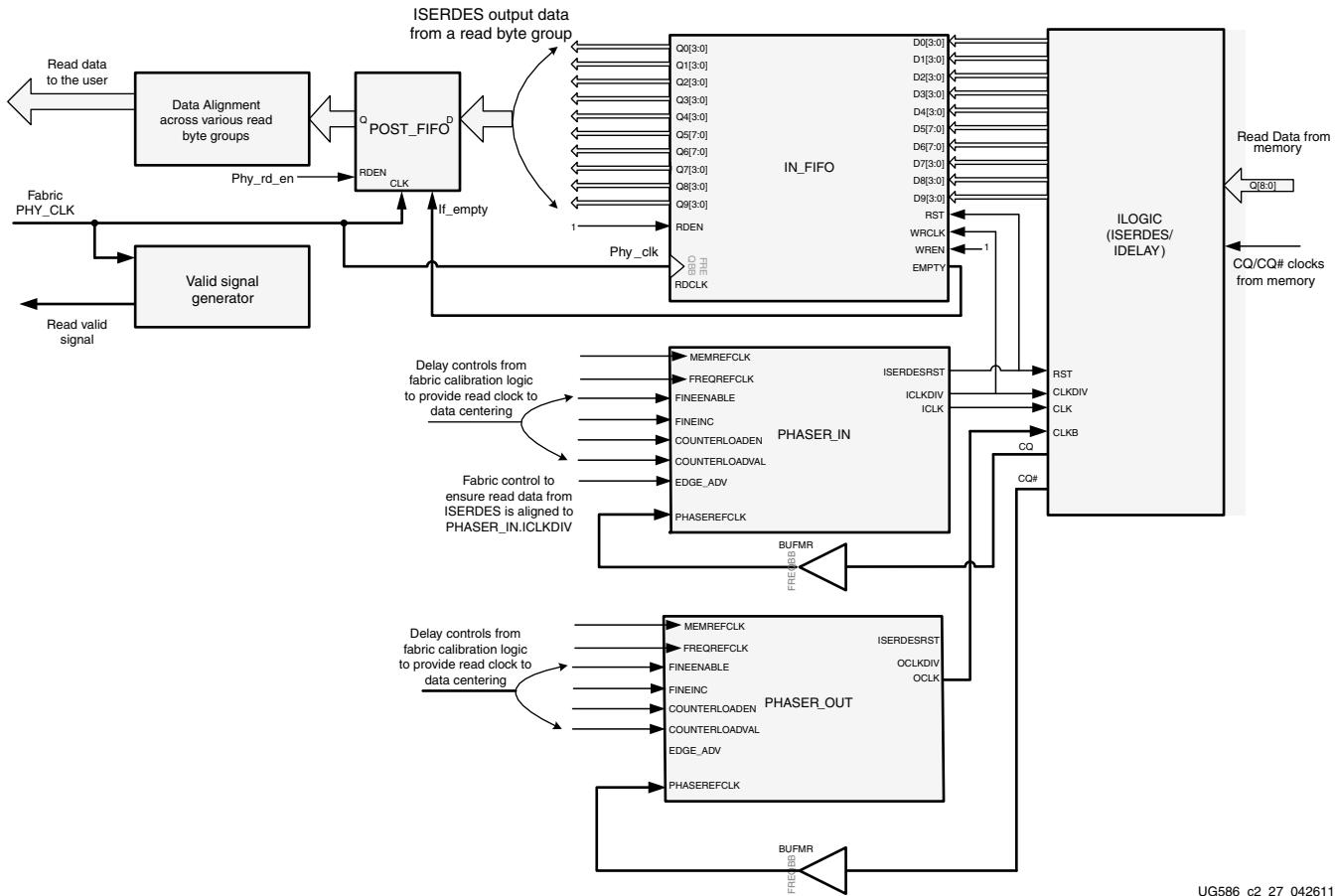
The read path includes data capture using the memory provided read clocks and also ensuring that the read clock is centered within the data window to ensure that good margin is available during data capture. Before any read can take place, calibration must occur. Calibration is the main function of the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

### ***Data Capture***

[Figure 2-46](#) shows a high-level block diagram of the path the read clock and the read data take from entering the FPGA until given to you. 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.

The IN\_FIFOs available in each byte group shown in [Figure 2-46](#) 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 FPGA logic half-frequency clock and that read data from all the byte groups have the same delay. For more details about the actual calibration and alignment logic, see the [Calibration](#) section.



UG586\_c2\_27\_042611

*Figure 2-46:*

The calibration logic provides 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.

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 QDR II+ 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  $\geq 400$  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  $< 400$  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  $< 400$  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.

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.

Complex pattern read calibration stage is added as the last stage of calibration to improve margin.

## ***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 QDR II+ 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 your set indicated value 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**.

When a write calibration is enabled for design that has memory frequency runs at 400 MHz or above, the results of read calibration data alignment are used to determine if a given setting is valid for correct write operation. After memory initialization, the read capture is first calibrated using this set pattern before moving on to calibrate the writes. There is no training register inside QDR II+ SRAM, the reads and writes cannot be independently verified.

At each step of write calibration, the alignment of the read clock (CQ/CQB) with Q is performed to ensure the correct capture of data. If the data alignment portion of read calibration is performed for a given byte lane and the expected result is not found, the write is assumed to have caused the failure. At each step of write calibration, the read calibration and associated logic are reset and restarted.

Bit window size of data byte lane with K clock (K-byte lane) is first determined by using PHASER\_OUT stage 3 delay. Stage 3 tap starts increment from tap 0 until the left-edge is found. If expected pattern return at tap 0, tap 0 is set as left edge tap position.

After the left edge of the K-byte lane is detected, the K clock is kept at this left tap position to perform non-K-byte lanes alignment. All non-K-byte lanes are aligned to the left edge of K-byte lane using the PHASER\_OUT stage 2 delay.

Then, K clock (K-byte lane) is moved to the right using the PHASER\_OUT stage 3 delay to determine the aggregate right edge of all byte lanes. After the right edge of the data window is determined, the centering process of K clock in the window is performed using the PHASER\_OUT stage3 delay.

The 7 series FPGAs QDR II+ SRAM interface solution is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top-level of the core. As per the OOC flow, none of the parameter values are passed down to the user design RTL file from the example design top RTL file. So, any design related parameter change is not reflected in the user design logic. The MIG tool should be used to regenerate a design when parameters need to be changed. These parameters are summarized in [Table 2-12](#).

*Table 2-12:*

MEM_TYPE	This is the memory address bus width	QDR2PLUS
CLK_PERIOD	This is the memory clock period (ps).	
BURST_LEN	This is the memory data burst length.	4
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.	
BW_WIDTH	This must be set to DATA_WIDTH/9	
NUM_DEVICES	This is the number of memory devices used.	
MEM_RD_LATENCY	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.	2.0 2.5
FIXED_LATENCY_MODE	This indicates whether or not to use a predefined latency for a read response from the memory to the client interface.	0, 1
CPT_CLK_CQ_ONLY	This indicates only one of the read clocks provided by the memory (rise clock) is used for the data capture.	TRUE
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	This is the number of cycles to wait until the echo clocks are stable.	(See memory vendor data sheet)
IODELAY_GRP <sup>(1)</sup>	This is a unique name for the IODELAY_CTRL that is provided when multiple IP cores are used in the design.	
REFCLK_FREQ	This is the reference clock frequency for IODELAYCTRLs. This parameter should <b>not</b> be changed.	200.0
RST_ACT_LOW	This is the active-Low or active-High reset. This is set to 1 when System Reset Polarity option is selected as active-Low and set to 0 when the option is selected as active-High.	0, 1
IBUF_LPWR_MODE	This enables or disables low power mode for the input buffers.	ON OFF
IODELAY_HP_MODE	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.	ON OFF

**Table 2-12:**
*(Cont'd)*

SYSCLK_TYPE	This parameter indicates whether the system uses single-ended system clocks, differential system clocks, or is driven from an internal clock (No Buffer). 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. For the No Buffer option, sys_clk_i, which appears in the port list, needs to be driven from the internal clock.	DIFFERENTIAL SINGLE_ENDED NO_BUFFER
REFCLK_TYPE	This parameter indicates whether the system uses single-ended reference clocks, differential reference clocks, is driven from an internal clock (No Buffer), or can connect system clock inputs only (Use System Clock). Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, clk_ref_p/clk_ref_n must be used. For single-ended clocks, clk_ref_i must be used. For the No Buffer option, clk_ref_i, which appears in port list, needs to be driven from an internal clock. For the Use System Clock option, clk_ref_i is connected to the system clock in the user design top module.	DIFFERENTIAL SINGLE_ENDED NO_BUFFER USE_SYSTEM_CLOCK
DIFF_TERM	This parameter indicates whether differential or non-differential termination is required for the system clock inputs.	TRUE FALSE
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.	-
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.	-
CLKOUT0_PHASE	Phase of PLL output CLKOUT0. This value is set by the MIG based on the banks selected for memory interface pins and the frequency of operation.	-
DIVCLK_DIVIDE	PLLE2 VCO divisor. This value is set by the MIG tool based on the frequency of operation.	-
SIM_BYPASS_INIT_CAL	This simulation only parameter is used to speed up simulations.	FAST OFF



Table 2-13:

(Cont'd)

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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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 CQ_P/CQ_N, K_P/K_N, and DLL_OFF_N pins, all Data Write, Data Read, and Address/Control pins are considered for this parameter generation. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	This parameter denotes for all byte groups of a selected bank. All 12 bits are denoted for a byte lane and are ordered from MSB:LSB as BA98_7654_3210. For example, this parameter is 48'hFFE_FFF_000_2FF 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.
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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	4'b0110: Middle two byte lanes contain INPUT pins, and the other byte lanes contain OUTPUT pins.
K_MAP	<p>Bank and byte lane position information for write clocks (K/K#). 8-bit parameter provided per pair of signals.</p> <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> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<p>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.</p> <p>Byte groups T0, T1, T2, and T3 are numbered in parameters as 3, 2, 1, and 0, respectively.</p> <p>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).</p> <p>8'h13: K/K# placed in bank 1, byte lane 3.</p> <p>8'h20: K/K# placed in bank 2, byte lane 0.</p>

Table 2-13:

(Cont'd)

CQ_MAP	Bank and byte lane position information for the read clocks (CQ/CQ#). See the <a href="#">K_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">K_MAP</a> 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>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<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 DQS I/Os. DQS_N and DQS_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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">ADD_MAP</a> example.

	Bank and byte lane position information for Address byte groups. Address requires three 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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">K_MAP</a> example.
ADDR_CTL_MAP		S e the M MAMv Ms# n
DO_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. This w M q	

---

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.

For general PCB routing guidelines, see [Appendix A, General Memory Routing Guidelines](#).

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. With the exception of the shared address and control in the dual controller supported in MIG.

The trace lengths described here are for high-speed operation and can be relaxed depending on the application target bandwidth requirements. 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 generate the package lengths using the Vivado Design Suite. The following commands generate a **csv** file that contains the package delay values for every pin of the device under consideration.

```
link_design -part <part_number>
write_csv <file_name>
```

For example, to obtain the package delay information for the 7 series FPGA XC7K160T-FF676, this command should be issued:

```
link_design -part xc7k160tfg676
write_csv flight_time
```

This generates a file named **flight\_time.csv** in the current directory with package trace delay information for each pin. While applying specific trace-matching guidelines for the QDR II+ SRAM 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 QDR II+ 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).

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the QDR 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 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 QDR II+ 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 DQS pairs inside the write bank. Within a read bank, the read data are placed on 9 of the 10 I/Os, 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, QDR II+ 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 pinout has been selected for the design. Manual changes through the XDC are not recommended. However, if the XDC needs to be altered, the following rules must be taken into consideration:

- The write data bus (D) of a memory interface must be placed within a single bank. It is required to arrange the write data bus byte wise (nine 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 DQS pin pair.
- The read data bus (Q) must be arranged byte wise (nine bits wide) among the FPGA byte groups. Xilinx recommends keeping 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. Xilinx recommends keeping the read data and read clocks in the same bank.
- All address/control signals must be placed within a single bank. The address bank should be placed adjacent to the data write (D) bank.
- The `d11_off_n` signal can be placed on any free I/O available in the banks used for the memory interface.

- Xilinx recommends keeping the system clock pins in the data write bank.



**RECOMMENDED:** *Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.*

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. However, exceptions are possible where pins might not be available for the clock input in the bank as that of the PLL. 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/Os.

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 323](#) for information on allowed PLL parameters.

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

```
create_clock -period 1.875 [get_ports sys_clk_p]
```

The **clk\_ref** constraint sets the frequency for the IDELAY reference clock, which is typically 200 MHz. For example:

```
create_clock -period 5 [get_ports clk_ref_p]
```

The I/O standards are set appropriately for the QDR II+ SRAM interface with LVCMOS15, HSTL15\_I, HSTL15\_I\_DCI, DIFF\_HSTL15\_I, or DIFF\_HSTL15\_I\_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 signal is the main system reset (asynchronous).
- **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 traffic generator if read data does not match the write data.

These signals are all set to LVCMS25 and can be altered as needed for the system design. They can be generated and used internally instead of being brought out to pins.

Some 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, the MIG tool puts an additional constraint in the XDC. For example:

```
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_nets sys_clk_p]
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hierarchical *pll*CLKIN1]
```

It results in the following warning 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_infrastructure/plle2_i> is placed at site <PLLE2_ADV_X1Y2>. The clock
I/O 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 through 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. For more information, see the *7 Series FPGAs Clocking Resources User Guide* (UG472) [\[Ref 10\]](#).

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

```
# PadFunction: IO_L13P_T2_MRCC_37
set_property VCCAUX_IO DONTCARE [get_ports {sys_clk_p}]
set_property IOSTANDARD DIFF_HSTL_I [get_ports {sys_clk_p}]
set_property PACKAGE_PIN K22 [get_ports {sys_clk_p}]

# PadFunction: IO_L13N_T2_MRCC_37
set_property VCCAUX_IO DONTCARE [get_ports {sys_clk_n}]
set_property IOSTANDARD DIFF_HSTL_I [get_ports {sys_clk_n}]
set_property PACKAGE_PIN J22 [get_ports {sys_clk_n}]
```

For more information, see the *Xilinx Timing Constraints Guide* (UG612) [\[Ref 15\]](#).

For QDR II+ SRAM interfaces that have the memory system input clock (**sys\_clk\_p/sys\_clk\_n**) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_HSTL\_I I/O standard (VCCO = 1.5V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_HSTL\_I and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_HSTL\_I CCIO pins. For more details on usage and required circuitry for LVDS and LVDS\_25 I/O Standards, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2].

These recommendations apply to termination for QDR II+ SRAM:

- Simulation (using IBIS or other) is highly recommended. The loading of command and address signals depends on various factors, such as speed requirements and termination topology. Loading can be a limiting factor in reaching a performance target.
- Command and Address signals should be terminated to  $V_{TT}$  through a  $50\Omega$  resistor.
- Write Clock (K\_P/N) does not require an external termination if ODT is available. If ODT is not available, each line should be terminated to  $V_{TT}$  through a  $50\Omega$  resistor.
- Write Data lines (D) do not require an external termination if ODT is available. If ODT is not available, each line should be terminated to  $V_{TT}$  through a  $50\Omega$  resistor.
- Read Clock (CQ) does not require an external termination and should use DCI. Set the DCI termination for each single-ended line to  $50\Omega$ .
- Read Data lines (Q, QVLD) do not require an external termination and should use DCI. Set the DCI termination to  $50\Omega$ .

The MIG tool generates the appropriate XDC 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 2-14](#) contains a list of the ports together with the I/O standard used.

*Table 2-14:*

qdr_bw_n	Output	HSTL_I
qdr_cq_p, qdr_cq_n	Input	HSTL_I_DC1
qdr_d	Output	HSTL_I
qdr_k_p, qdr_k_n	InOut	DIFF_HSTL_II
qdr_q	Input	HSTL_I_DC1
qdr_r_n	Output	HSTL_I

Table 2-14: *(Cont'd)*

qdr_sa	Output	HSTL_I
qdr_w_n	Output	HSTL_I

**Notes:**

1. All signals operate at 1.5V.

DCI (HP 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.

The 7 series FPGA MIG QDR II+ SRAM design has two clock inputs, the reference clock and the system clock. The reference clock drives the IODELAYCTRL components in the design, while the system clock input is used to create all MIG design clocks that are used to clock 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 more information on clocking architecture, see [Clocking Architecture, page 323](#).

The MIG tool allows you to input the Memory Clock Period and then lists available Input Clock Periods that follow the supported clocking guidelines. Based on these two clock periods selections, the generated MIG core appropriately sets the PLL parameters. The MIG tool enables automatic generation of all supported clocking structures. For information on how to use the MIG tool to set up the desired clocking structure including input clock placement, input clock frequency, and IDELAYCTRL `ref_clk` generation, see [Creating the 7 Series FPGA QDR II+ SRAM Design, page 287](#).

### ***Input Clock Guidelines***




---

**IMPORTANT:** *The input system clock cannot be generated internally.*

---

- PLL Guidelines
  - CLKFBOUT\_MULT\_F (M) must be between 1 and 16 inclusive.
  - DIVCLK\_DIVIDE (D, Input Divider) can be any value supported by the PLLE2 parameter.
  - CLKOUT\_DIVIDE (O, Output Divider) must be 2 for 400 MHz and up operation and 4 for below 400 MHz operation.

- The above settings must ensure the minimum PLL VCO frequency (FVCOMIN) is met. For specifications, see the appropriate DC and Switching Characteristics Data Sheet. The *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] includes the equation for calculating FVCO.
- The relationship between the input period and the memory period is  $\text{InputPeriod} = (\text{MemoryPeriod} \times M) / (D \times D1)$ .
- The clock input (**sys\_clk**) can be input on any CCIO in the column where the memory interface is located; this includes CCIO in banks that do not contain the memory interface, but must be in the same column as the memory interface. The PLL must be located in the bank containing the clock sent to the memory. To route the input clock to the memory interface PLL, the CMT backbone must be used. With the MIG implementation, one spare interconnect on the backbone is available that can be used for this purpose.
  - MIG versions 1.4 and later allow this input clocking setup and properly drive the CMT backbone.
  - **CLOCK\_DEDICATED\_ROUTE = BACKBONE** constraint is used to implement CMT backbone, following warning message is expected. It can be ignored safely.

**WARNING:** [Place 30-172] Sub-optimal placement for a clock-capable IO pin and PLL pair. The flow will continue as the **CLOCK\_DEDICATED\_ROUTE** constraint is set to **BACKBONE**.

```
u_mig_7series_0/c0_u_clk_ibuf/diff_input_clk.u_ibufg_sys_clk (IBUFDS.O) is locked  
to IOB_X0Y176  
u_mig_7series_0/c0_u_infrastructure/plle2_i (PLLE2_ADV.CLKIN1) is locked to  
PLLE2_ADV_X0Y1  
u_mig_7series_0/c1_u_infrastructure/plle2_i (PLLE2_ADV.CLKIN1) is locked to  
PLLE2_ADV_X0Y5  
.....
```

- For QDR II+ SRAM interfaces that have the memory system input clock (**sys\_clk**) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_HSTL\_I I/O standard (VCCO = 1.5V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_HSTL\_I and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_HSTL\_I CCIO pins.
- It is acceptable to have differential inputs such as LVDS and LVDS\_25 in I/O banks that are powered at voltage levels other than the nominal voltages required for the outputs of those standards (1.8V for LVDS outputs, and 2.5V for LVDS\_25 outputs). However, these criteria must be met:
  - a. The optional internal differential termination is not used (DIFF\_TERM = FALSE, which is the default value).  
**Note:** This might require manually changing DIFF\_TERM parameter located in the top-level module or setting this in the UCF or XDC.
  - b. The differential signals at the input pins meet the VIN requirements in the Recommended Operating Conditions table of the specific device family data sheet.

- c. The differential signals at the input pins meet the VIDIFF (min) requirements in the corresponding LVDS or LVDS\_25 DC specifications tables of the specific device family data sheet.

One way to accomplish the above criteria is to use an external circuit that both AC-couples and DC-biases the input signals. The figure shows an example circuit for providing an AC-coupled and DC-biased circuit for a differential clock input. RDIFF provides the  $100\Omega$  differential receiver termination because the internal DIFF\_TERM is set to FALSE. To maximize the input noise margin, all RBIAS resistors should be the same value, essentially creating a VCM level of  $VCCO/2$ . Resistors in the 10k to  $100\text{ k}\Omega$  range are recommended. The typical values for the AC coupling capacitors CAC are in the range of  $100\text{ nF}$ . All components should be placed physically close to the FPGA inputs.

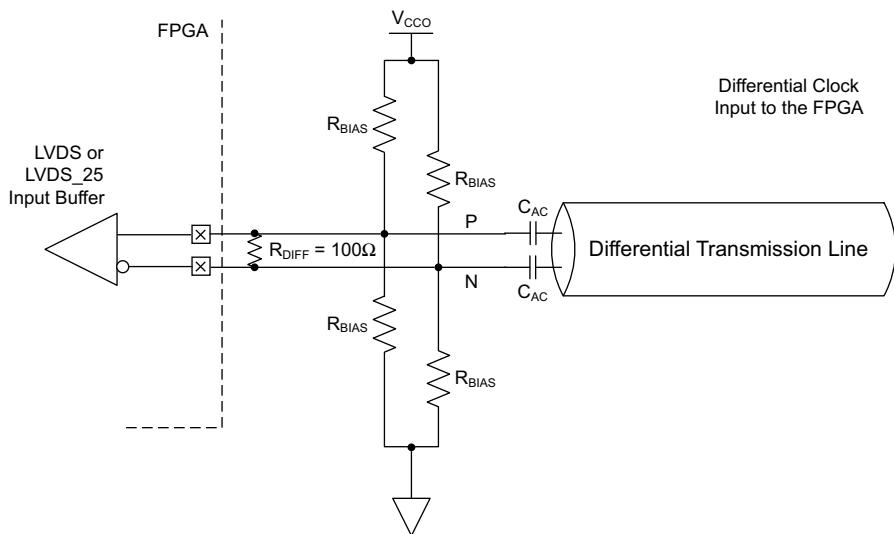


Figure 2-47:

**Note:** The last set of guidelines on differential LVDS inputs are added within the LVDS and LVDS\_25 (Low Voltage Differential Signaling) section of the *7 Series SelectIO Resources User Guide* (UG471) [Ref 2] in the next release of the document.

These guidelines are irrespective of Package, Column (HR/HP), or I/O Voltage.

### Sharing sys\_clk between Controllers

The MIG 7 series FPGA designs require **sys\_clk** to be in the same I/O bank column as the memory interface to minimize jitter.

- **Interfaces Spanning I/O Columns** – A single **sys\_clk** input cannot drive memory interfaces spanning multiple I/O columns. The input clock input must be in the same column as the memory interface to drive the PLL using the CMT Backbone, which minimizes jitter.
- **Interfaces in Single I/O Column** – If the memory interfaces are entirely contained within the same I/O column, a common **sys\_clk** can be shared among the interfaces.

The **sys\_clk** can be input on any CCIO in the column where the memory interfaces are located. This includes CCIO in banks that do not contain the memory interfaces, but must be in the same column as the memory interfaces.

### ***Information on Sharing BUFG Clock (phy\_clk)***

The MIG 7 series QDR II+ SRAM design includes an MMCM which outputs the **phy\_clk** on a BUFG route. It is not possible to share this clock amongst multiple controllers to synchronize the user interfaces. This is not allowed because the timing from the FPGA logic to the PHY Control block must be controlled. This is not possible when the clock is shared amongst multiple controllers. The only option for synchronizing user interfaces amongst multiple controllers is to create an asynchronous FIFO for clock domain transfer.

### ***Information on Sync\_Pulse***

The MIG 7 series QDR II+ SRAM design includes one PLL that generates the necessary design clocks. One of these outputs is the **sync\_pulse**. The sync pulse clock is 1/16 of the **mem\_refclk** frequency and must have a duty cycle distortion of 1/16 or 6.25%. This clock is distributed across the low skew clock backbone and keeps all PHASER\_IN/\_OUT and PHY\_Control blocks in sync with each other. The signal is sampled by the **mem\_refclk** in both the PHASER\_INs/\_OUTs and PHY\_Control blocks. The phase, frequency, and duty cycle of the **sync\_pulse** is chosen to provide the greatest setup and hold margin across PVT.

---

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.

**Note:** The overall read latency of the MIG 7 series QDR II+ core is dependent on how the Memory Controller is configured, but most critically on the target traffic/access pattern and the number of commands already in the pipeline before the read command is issued. Read latency is measured from the point wpp      e      a      colum      ch

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 QDR II+ 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 2-48 shows the overall flow for debugging problems associated with these two general types of issues.

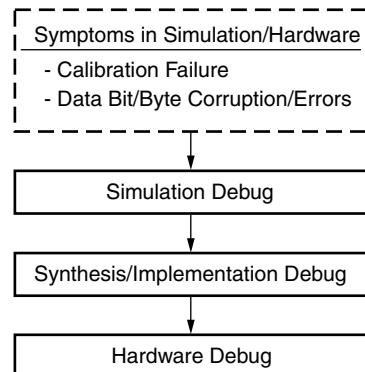


Figure 2-48:

Many tools are available to debug memory interface design issues. This section indicates which resources are useful for debugging a given situation.

### ***Example Design***

QDR II+ 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 Vivado logic analyzer feature. Selecting this option port maps the debug signals to VIO modules of the Vivado logic analyzer feature in the design top module.

Sample debug logic by connecting the debug ports to the Vivado Design Suite debug feature modules (that is, ILA and 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 a few buses and provided in the port list.

To confirm that all debug ports are connected to various modules, 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_ilao_data`," "`qdriip_ilao_trig`," "`qdriip_ilai1_data`," "`qdriip_ilai1_trig`," "`qdriip_vio2_async_in`," and "`qdriip_vio2_sync_out`."

## Vivado Design Suite Debug Feature

The Vivado Design Suite debug feature inserts logic analyzer, bus analyzer, and VIO software cores directly into the design. Supported versions of ILA and VIO are 3.0. The debug feature also allows you to set trigger conditions to capture application and MIG debug signals in hardware. Captured signals can be analyzed through the Vivado logic analyzer feature. For more information about the Vivado logic analyzer, software is available in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 16].



**IMPORTANT:** *The Integrated Logic Analyzer (ILA) operates on a synchronous clock and cannot be triggered during reset. Instead, set the trigger on an ILA signal to look for a rising edge ("R") or falling edge ("F") with the radix value of the signal set to "Binary." With this trigger setting, the trigger can be armed. When the reset is applied and released, the trigger captures the desired ILA results.*

Figure 2-49 shows the debug flow for simulation.

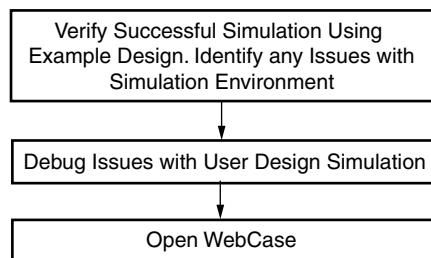


Figure 2-49:

## 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.

The Questa Advanced Simulator, Vivado Simulator, IES, and VCS simulation tools are used for verification of MIG IP core at each software release. Script files to run simulations with IES and VCS simulators are generated in MIG generated output. Simulations using Questa Advanced Simulator and Vivado simulators can be done through Vivado Tcl Console commands or in Vivado IDE.



**IMPORTANT:** *Other simulation tools can be used for MIG IP core simulation but are not specifically verified by Xilinx.*

To run the simulation, go to this directory:

```
<project_dir>/<Component_Name>_ex/imports
```

For a project created with the name set as **project\_1** and the Component Name entered in Vivado IDE as **mig\_7series\_0**, go to the directory as follows:

```
project_1/mig_7series_0_ex/imports
```

IES and VCS simulation scripts are meant to be executed only in Linux operating systems.

The **ies\_run.sh** and **vcs\_run.sh** files are the executable files for running simulations using IES and VCS simulators respectively. Library files should be added to the **ies\_run.sh** and **vcs\_run.sh** files respectively. See the **readme.txt** file for details regarding simulations using IES and VCS.

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings** ([Figure 2-50](#)).

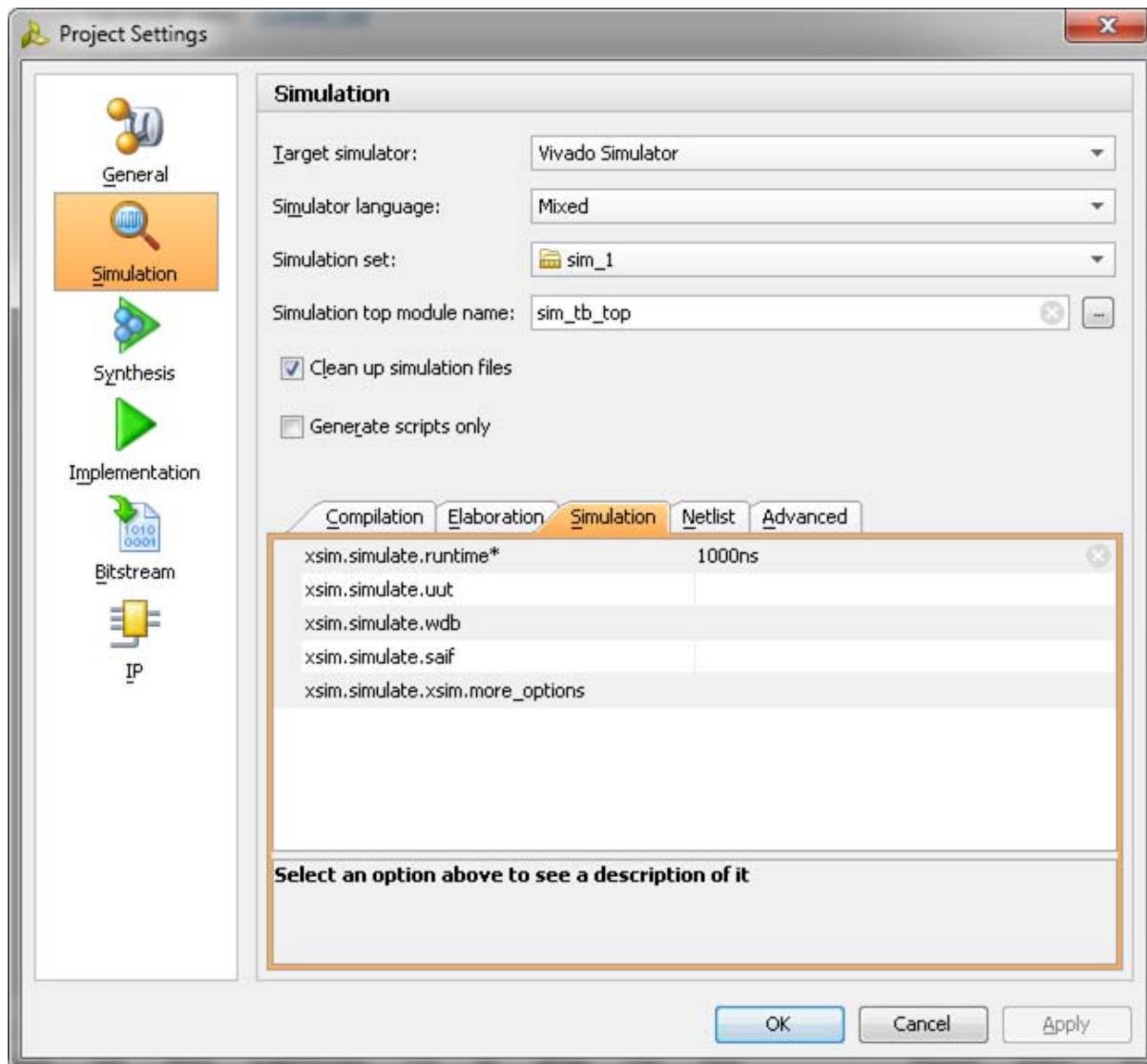
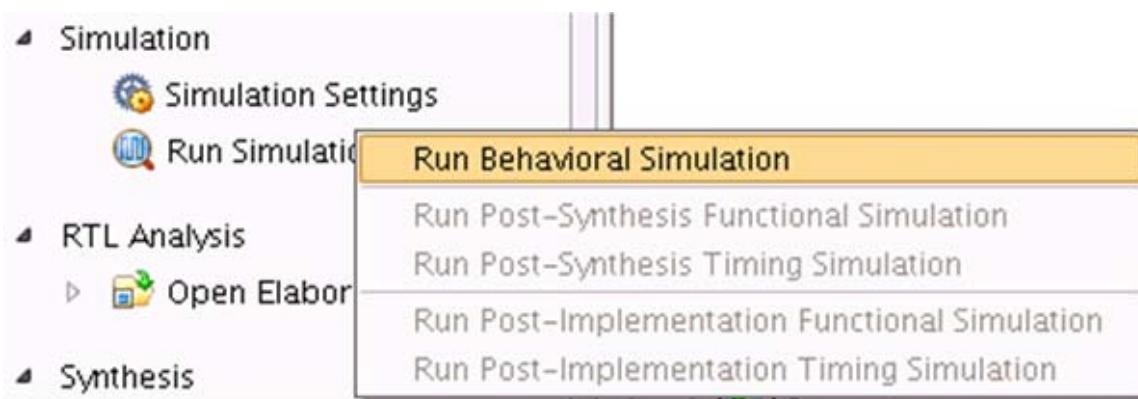


Figure 2-50:

- Under the **Simulation** tab as shown in [Figure 2-50](#), set the **xsim.simulate.runtime** as 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms). Apply the settings and select **OK**.

3. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 2-51](#).



*Figure 2-51:*

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Questa Advanced Simulator/ModelSim.
  - a. Browse to the **Compiled libraries location** and set the path on **Compiled libraries location** option.
  - b. Under the **Simulation** tab, set the **modelsim.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms), set **modelsim.simulate.vsim.more\_options** to **-novopt** as shown in [Figure 2-50](#).
3. Apply the settings and select **OK**.

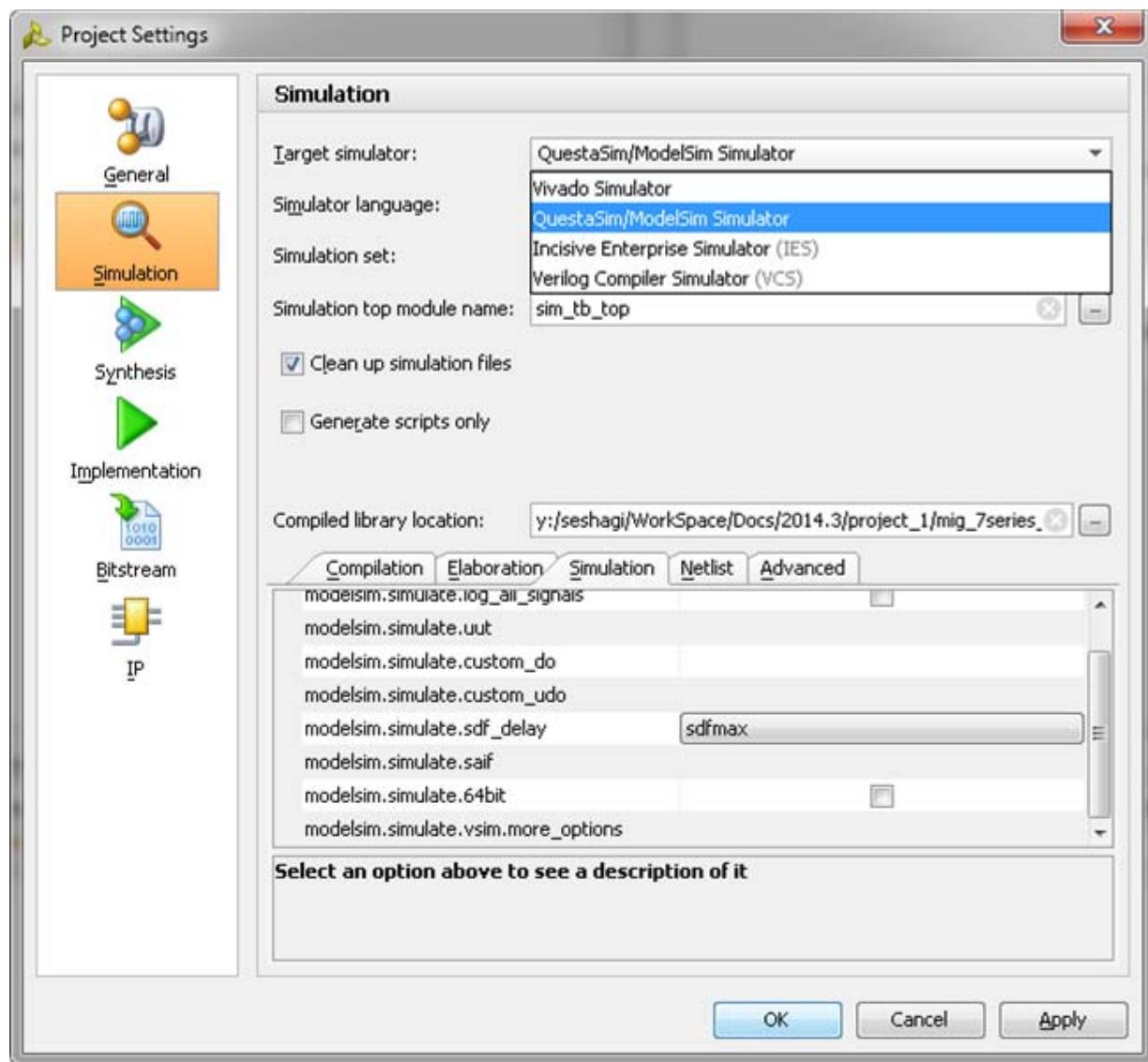


Figure 2-52:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 2-51](#).
5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG 900)* [\[Ref 8\]](#).
  
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Verilog Compiler Simulator (VCS).

- a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **vcs.compile.vlogan.more\_options** to **-sverilog**.
  - c. Under the **Simulation** tab, set the **vcs.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after a certain period of time which is less than 1 ms) as shown in [Figure 2-53](#).
3. Apply the settings and select **OK**.

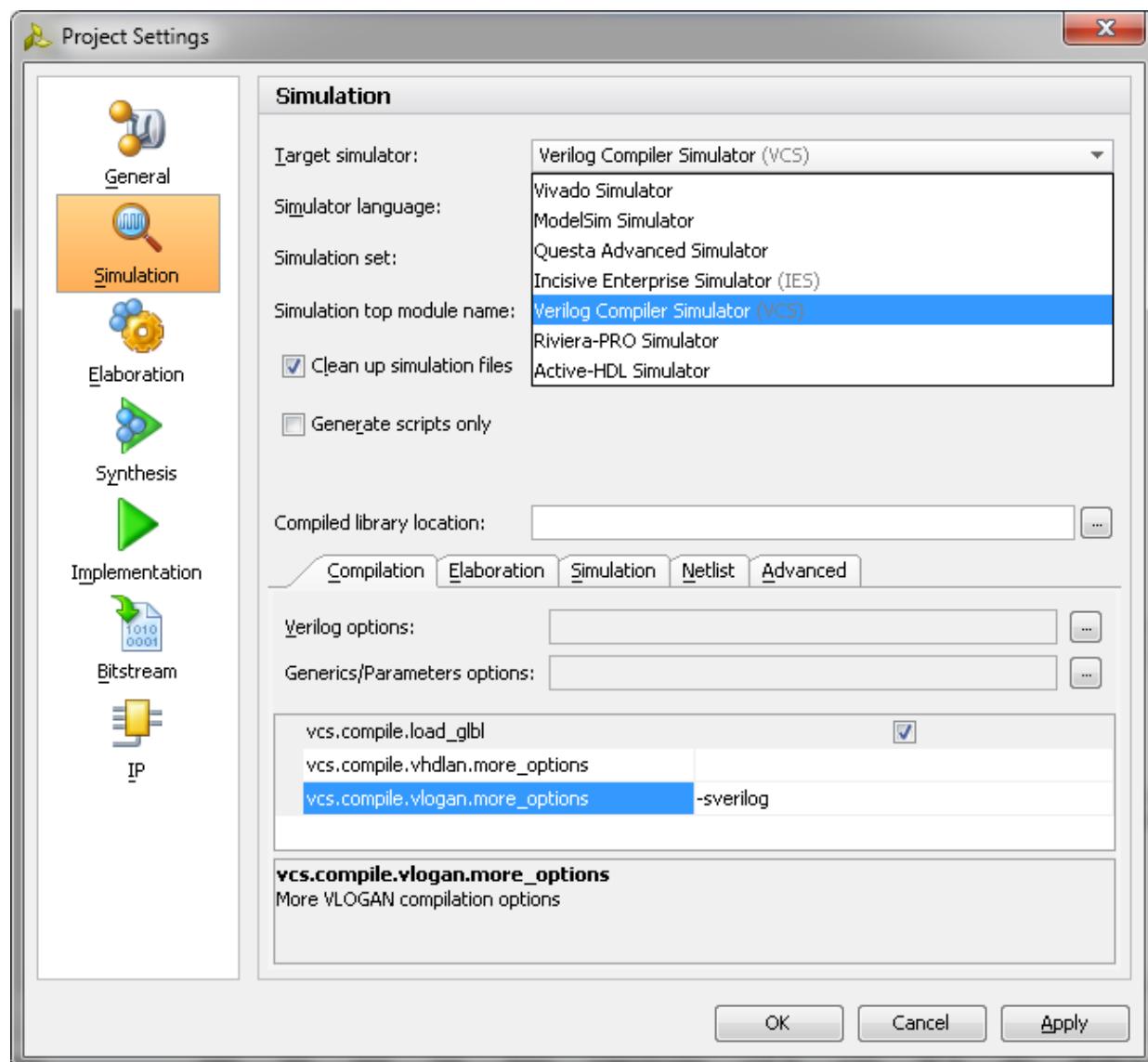


Figure 2-53:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 2-51](#).

5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [Ref 8].
  
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
  - a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **ies.compile.ncvlog.more\_options** to **-sv**.
  - c. Under the **Elaboration** tab, set the **ies.elaborate.ncelab.more\_options** to **-namemap\_mixgen**.
  - d. Under the **Simulation** tab, set the **ies.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time which is less than 1 ms) as shown in [Figure 2-54](#).

3. Apply the settings and select **OK**.

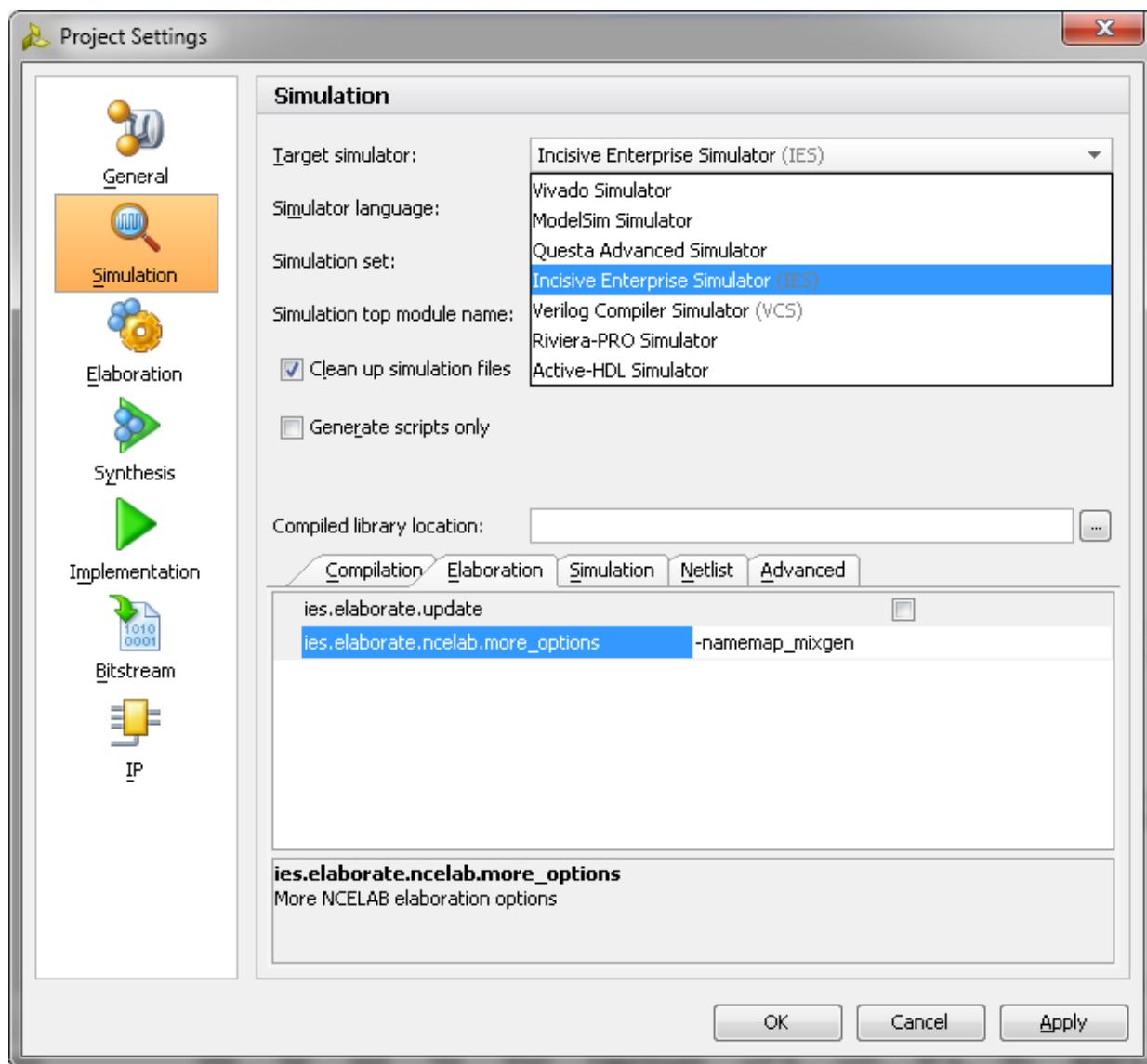


Figure 2-54:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 2-51](#).
5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [\[Ref 8\]](#).

**Note:** MIG does not generate memory model files for QDR II+ designs. Appropriate memory model should be added to the **Simulation Sources** under **Sources** window of the **Open IP Example Design** project.

For Samsung Memory models appropriate define values should be added to the memory model itself. Vivado settings does not allow applying define values explicitly on memory models.

For detailed information on setting up Xilinx libraries, see COMPXLIB in the *Command Line Tools User Guide* (UG628) [Ref 17] and the *Synthesis and Simulation Design Guide* (UG626) [Ref 18]. For simulator tool support, see the *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions Data Sheet* (DS176) [Ref 1].

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 2-15 shows the signals and parameters of interest, respectively, during simulation.

*Table 2-15:*

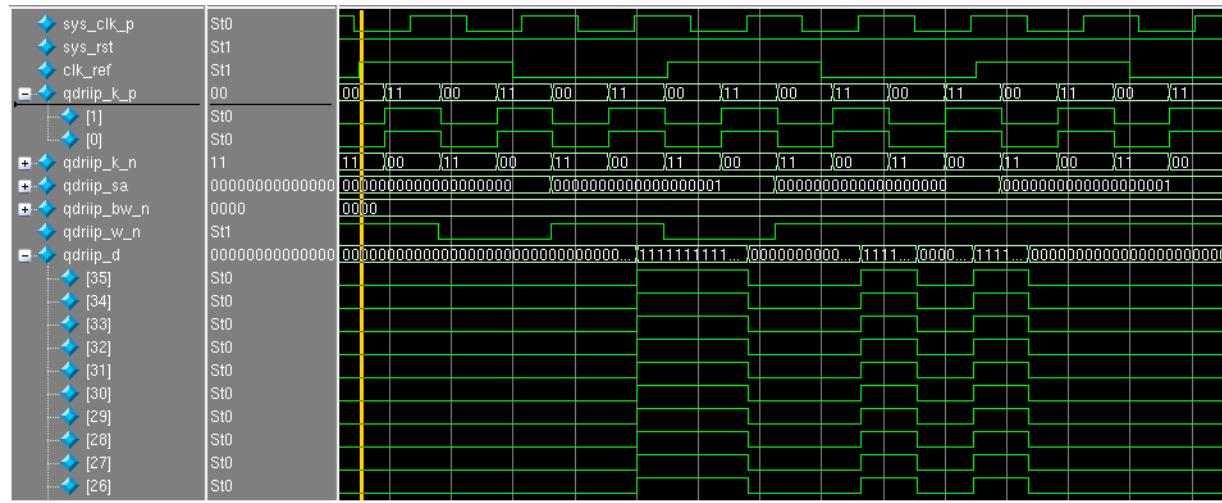
tg_compare_error	This signal indicates a mismatch between the data written from the UI and data received during a read on the UI. This signal is a part of the example design. A single error asserts this signal; it is held until the design is reset.
tg_cmp_error	This signal indicates a mismatch between 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

The QDR II+ memories do not require an elaborate initialization procedure. However, you must ensure that the **Doff\_n** signal is provided to the memory as required by the vendor. The QDR II+ 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 2,048) 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, you must perform the required termination procedure.

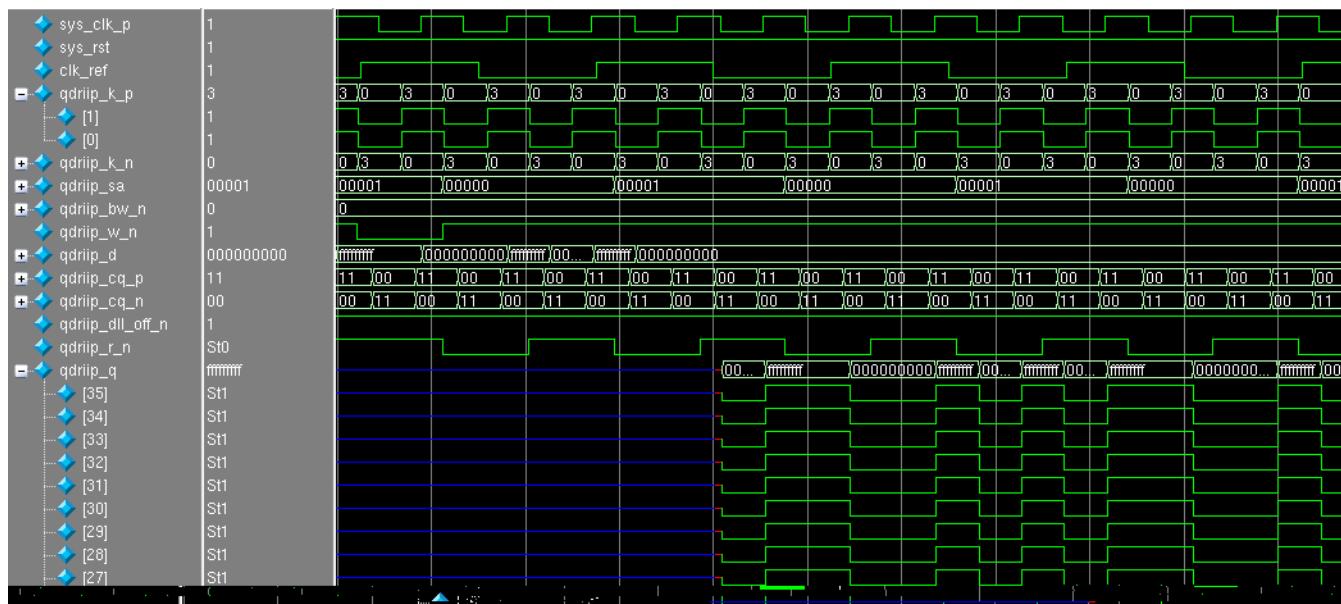
Calibration completes read leveling, write calibration, and read enable calibration. This is completed over two stages. This sequence successfully completes when the **cal\_done** signal is asserted. For more details, see [Physical Interface, page 325](#).

The first stage performs per-bit read leveling calibration. The data pattern used during this stage is OOFFOOFFFOOOFFFFFOO. The data pattern is first written to the memory, as shown in [Figure 2-55](#).



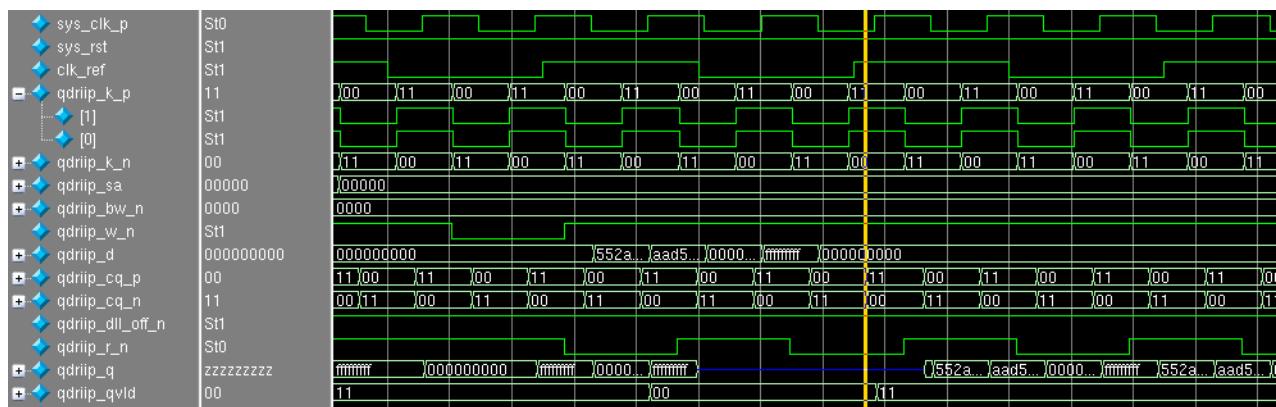
*Figure 2-55:*

This pattern is then continuously read back while the per-bit calibration is completed, as shown in [Figure 2-56](#).



*Figure 2-56:*

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 2-57](#).

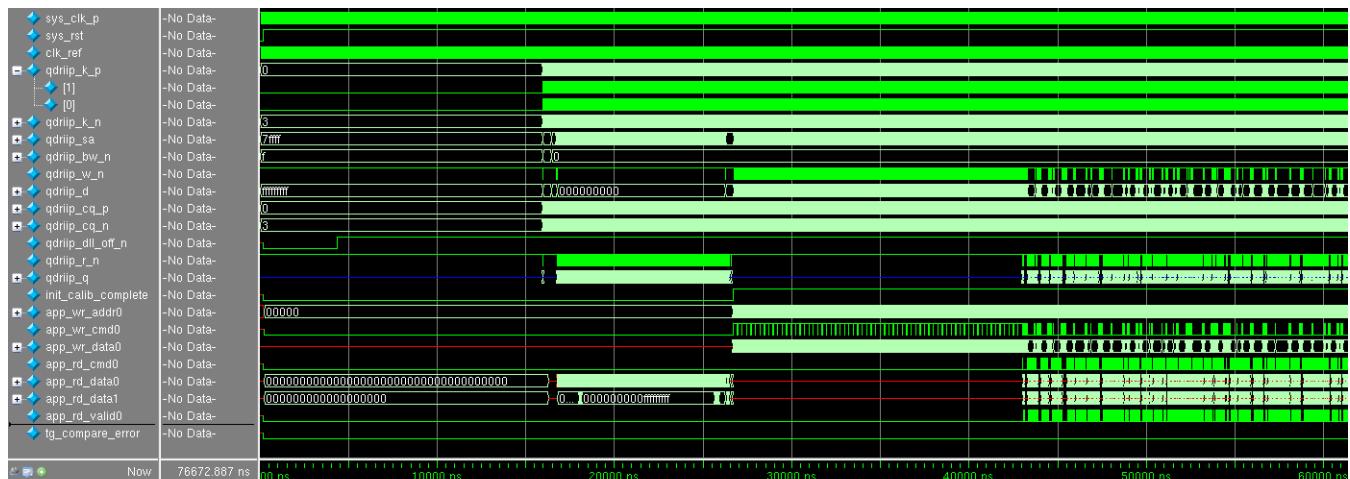


*Figure 2-57:*

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.

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 2-58 shows a successful implementation of the test bench with no assertions on error.

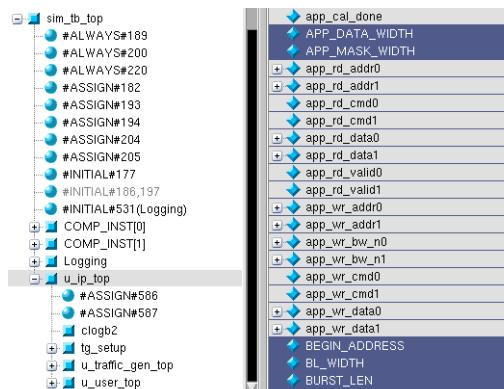


*Figure 2-58:*

When sending write and read commands, you must properly assert and deassert the corresponding UI inputs. See [User Interface, page 320](#) and [Interfacing with the Core through the Client Interface, page 322](#) 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.

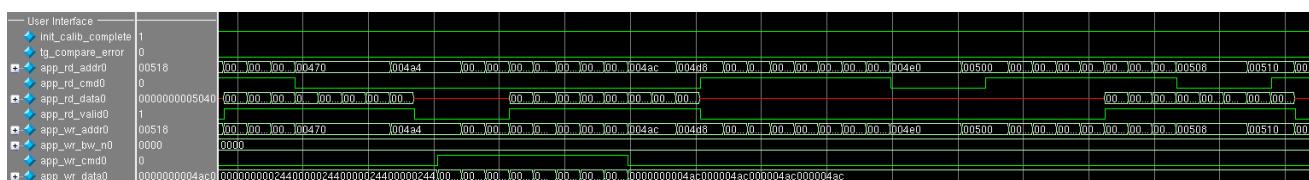
To debug data errors on the QDR II+ SRAM interface, it is necessary to pull the UI signals into the simulation waveform.

In the Questa Advanced Simulator Instance window, highlight **u\_ip\_top** to display the necessary UI signals in the **Objects** window, as shown in [Figure 2-59](#). Highlight the user interface signals noted in [Table 2-15, page 362](#), right-click, and select **Add > To Wave > Selected Signals**.

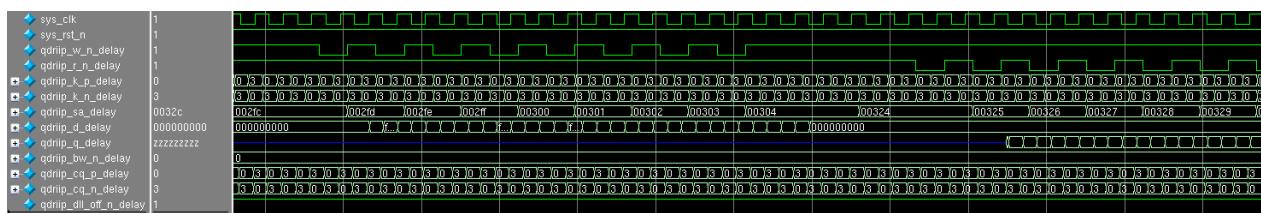


*Figure 2-59:*

[Figure 2-60](#) and [Figure 2-61](#) show example waveforms of a write and read on both the user interface and the QDR II+ interface.



*Figure 2-60:*



*Figure 2-61:*

Figure 2-62 shows the debug flow for synthesis and implementation.

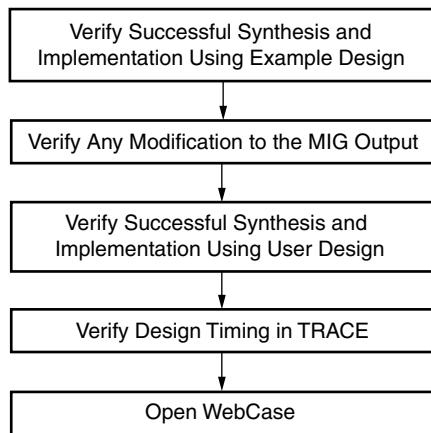


Figure 2-62:



**IMPORTANT:** *The standard synthesis flow for Synplify is not supported for the core.*

### ***Verify Successful Synthesis and Implementation***

The example design and user design generated by the MIG tool include synthesis/implementation script files and **.xdc** files. These files should be used to properly synthesize and implement the targeted design and generate a working bitstream.

### ***Verify Modifications to the MIG Tool Output***

The MIG tool allows you to select the FPGA banks for the memory interface signals. Based on the banks selected, the MIG tool outputs a XDC with all required location constraints. This file is located in both the **example\_design/par** and **user\_design/constraints** 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 QDR II+ 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 your specific application logic.

Any timing violations that are encountered must be isolated. The timing report output by TRACE (.twx/.twr) should be analyzed to determine if the failing paths exist in the MIG tool QDR II+ SRAM design or the UI (backend application) to the MIG tool design. If failures are encountered, you must ensure the build options (that is, XST, MAP, PAR) specified in the file are used.

If failures still exist, Xilinx has many resources available to aid in closing timing. The PlanAhead™ tool [Ref 19] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* (UG612) [Ref 15] provides valuable information on all available Xilinx constraints.

Figure 2-63 shows the debug flow for hardware.

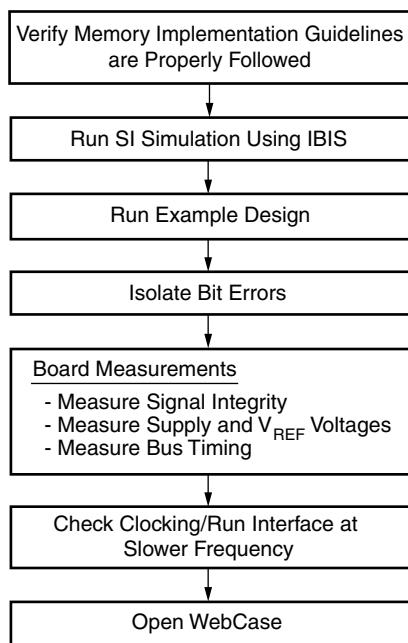


Figure 2-63:

## Clocking

The external clock source should be measured to ensure frequency, stability (jitter), and usage of the expected FPGA pin. You 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

You 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.

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* (UG199) [Ref 20] 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.

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 Vivado logic analyzer feature should be used to analyze the behavior of MIG tool core signals. For more information about the Vivado logic analyzer, software is available in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 16].

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 issue.
- 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.

The Debug port is a set of input and output signals that either provide status (outputs) or allow you to make adjustments as the design is operating (inputs). When generating the QDR II+ 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 2-16](#) 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 2-16:*

dbg_phy_status[0]	rst_wr_clk	FPGA logic 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	QDR II+ 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.
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.

## 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 the Vivado hardware session and program the FPGA under test with generated BIT and LTX files.
2. 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).
3. To measure margin with PRBS8 pattern, set VIO signals with the listed values in the **traffic\_gen\_top** instance in **example\_top**:

```
vio_modify_enable = 'd1  
vio_data_mode_value = 'd7  
vio_addr_mode_value = 'd3  
vio_instr_mode_value = 'd4  
vio_bl_mode_value = 'd2  
vio_fixed_bl_value = 'd128  
vio_fixed_instr_value = 'd1  
vio_data_mask_gen = 'd0
```

4. Assert **vio\_dbg\_clear\_error** or system reset.
5. Select a given byte lane using **dbg\_byte\_sel**.
6. Observe the tap values on PHASER\_IN for the selected byte lane using **dbg\_pi\_counter\_read\_val**.
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. Decrease the tap values on PHASER\_IN using **dbg\_pi\_f\_dec** back to the starting value.
9. Clear the error recorded previously by asserting **vio\_dbg\_clear\_error**.

10. Decrement the PHASER\_IN taps using `dbg_pi_f_dec` to find the other edge of the window until another error occurs (`tg_compare_error` should be asserted).
11. Record those results, return the PHASER\_IN taps to the starting location, and clear the error again (`vio_dbg_clear_error`).

This 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 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 issue bits or bytes, but it can be difficult to step through an entire interface looking for issues. For this reason, the QDR II+ SRAM Memory Interface Debug port contains automated window checking that can be used to step through the entire interface. A 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 windowsizes, so depending on the tap values after calibration, the left edge of the read data window might not be found properly.

To measure margin with PRBS8 traffic pattern, set VIO signals with the listed values in the `traffic_gen_top` instance in `example_top`:

```
vio_modify_enable = 'd1
vio_data_mode_value = 'd7
vio_addr_mode_value = 'd3
vio_instr_mode_value = 'd4
vio_bl_mode_value = 'd2
vio_fixed_bl_value = 'd128
vio_fixed_instr_value = 'd1
vio_data_mask_gen = 'd0
```

Next, assert `vio_dbg_clear_error` or assert system reset before proceeding with automated margin check. The steps to follow for automated margin check include:

1. Start automated window check by issuing a single pulse on the VIO signal `dbg_win_start`.
2. The VIO signal `dbg_win_active` indicates that the automated window check is in progress. The signals `dbg_pi_f_inc` and `dbg_pi_f_dec` must not be used when `dbg_win_active` is asserted.

3. The current bit and byte being measured are indicated by the VIO signals `dbg_win_current_bit` and `dbg_win_current_byte`, respectively.
4. To get the left and right tap counts for a completed bit, select the desired bit using VIO signal `dbg_win_bit_select` and observe the results on `dbg_win_left_ram_out` and `dbg_win_right_ram_out`, respectively.

[Table 2-17](#) lists the signals associated with this automated window checking functionality.

*Table 2-17:*

<code>dbg_win_start</code>	Single pulse that starts the <code>chk_win</code> state machine. Use the Vivado logic debug VIO module to control this.
<code>dbg_win_bit_select[6:0]</code>	Manual bit selection for reporting of results. The results are provided on <code>dbg_win_left_ram_out</code> and <code>dbg_win_right_ram_out</code> for the bit indicated.
<code>dbg_win_active</code>	Flag to indicate <code>chk_win</code> is active and measuring read window margins. While active, the state machine has control over the debug port signals.
<code>vio_dbg_clear_error</code>	Clear error control signal controlled by <code>chk_win</code> .
<code>dbg_win_current_bit[6:0]</code>	Feedback to indicate which bit is currently being monitored during automatic window checking.
<code>dbg_win_current_byte[3:0]</code>	Feedback to indicate which byte is currently being monitored (and used to select the byte lane controls with <code>dbg_byte_sel</code> ).
<code>dbg_win_left_ram_out [WIN_SIZE - 1:0]</code>	<code>PHASER_IN</code> tap count to reach the left edge of the read window for a given bit.
<code>dbg_win_right_ram_out [WIN_SIZE - 1:0]</code>	<code>PHASER_IN</code> tap count to reach the right edge of the read window for a given bit.
<code>dbg_pi_f_inc</code>	<code>chk_win</code> control signal to increment <code>PHASER_IN</code> . This signal should be used only when <code>dbg_win_active</code> is deasserted.
<code>dbg_pi_f_dec</code>	<code>chk_win</code> control signal to decrease <code>PHASER_IN</code> . This signal should be used only when <code>dbg_win_active</code> is deasserted.

### ***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:

dbg_phy_wr_cmd_n[1:0]	Output	This active-Low signal is the internal wr_cmd used for debug with the Vivado logic analyzer feature.
dbg_phy_rd_cmd_n[1:0]	Output	This active-Low signal is the internal rd_cmd used for debug with the Vivado logic analyzer feature.
dbg_phy_addr[ADDR_WIDTH × 4 – 1:0]	Output	Control address bus used for debug with the Vivado logic analyzer feature.
dbg_phy_wr_data[DATA_WIDTH × 4 – 1:0]	Output	Data being written that is used for debug with the Vivado logic analyzer feature.
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 QDR II+ memory. This verifies calibration write timing. This signal must be Low for normal operation.
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 QDR II+ 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 and you set the phaser/IDELAY tap controls
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.
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.

Table 2-18:

(Cont'd)

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
dbg_cmplx_rd_loop	Input	When High, complex read level continues forever.
dbg_cmplx_rd_lane[2:0]	Input	Selects the lane to hang on when dbg_cmplx_rd_loop == 'b1.
dbg_K_left_shift_right	Input	Shifts the location of the left edge sent to the POC right.
dbg_K_right_shift_left	Input	Shifts the location of the right edge sent to the POC left.
dbg_cmplx_wr_loop	Input	When High, complex write pattern is written indefinitely.

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:

dbg_wr_init[14:0]	phy_init_r	One hot state machine.
dbg_wr_init[18:15]	phase_valid	Per byte lane comparison results.
dbg_wr_init[22:19]	lanes_solid_r	Comparison success post threshold per lane.
dbg_wr_init[23]	po_delay_done	Phaser out adjustment complete.
dbg_wr_init[24]	rdlvl_stg1_done	Read level cycle complete.
dbg_wr_init[25]	rdlvl_stg1_start	Start read level calibration.
dbg_wr_init[26]	edge_adv_cal_start	Start cycle (edge) alignment.
dbg_wr_init[27]	edge_adv_cal_done	Edge alignment complete.
dbg_wr_init[28]	cal_stage2_start	Start latency calibration.
dbg_wr_init[29]	read_cal_done	Latency calibration complete.
dbg_wr_init[30]	rst_stg1_r	Reset read level block.
dbg_wr_init[31]	rst_stg2_r	Reset edge and latency calibration logic.
dbg_wr_init[32]	suppress_stg1	All bytes successfully read leveled. Suppress further read levels.

Table 2-19:

(Cont'd)

dbg_wr_jinit[36:33]	seen_valid_r	Successful read level recorded per lane.
dbg_wr_init[37]	qdr_edge_adv_err	Edge advance timeout.
dbg_wr_init[38]	qdr_stg2_err	Latency calibration timeout.
dbg_wr_init[39]	rst_samp_cnt	Reset sample counters.

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_rdlvl` module and are all valid in the `clk` domain.

Table 2-20:

dbg_rd_stage1_cal[2:0]	sm_r	Read level main state machine.
dbg_rd_stage1_cal[7:6]	seq_sm_r	Read level sequence state bits.
dbg_rd_stage1_cal[14:12]	rdlvl_work_lane_r	Lane currently undergoing read level calibration.
dbg_rd_stage1_cal[15]	rdlvl_stg1_start	Write side signal causing read level block to start.
dbg_rd_stage1_cal[16]	rdlvl_stg1_done	Read level block signals completion.
dbg_rd_stage1_cal[17]	rdlvl_stg1_start	Write side signal causing read level to copy first lane result across all lanes.
dbg_rd_stage1_cal[25:18]	rdlvl_stg1_cal_bytes_r	Lanes for which write side is requesting calibration.
dbg_rd_stage1_cal[31]	cmplx_rdcal_start	Write side signal causing read level to do complex cal.
dbg_rd_stage1_cal[32]	cmplx_rd_data_valid	Write side signal informing read level that complex read data is valid.
dbg_rd_stage1_cal[48:41]	rd_data_comp_r	Per byte comparison results for complex calibration.
dbg_rd_stage1_cal[56:49]	iserdes_comp_r	Per byte comparison results for simple calibration.
dbg_rd_stage1_cal[57]	rdlvl_lane_match	Overall comparison result for both simple and complex.
dbg_rd_stage1_cal[66:61]	largest_left_edge	Phaser in taps when the right most left edge was found.
dbg_rd_stage1_cal[72:67]	smallest_right_edge	Phaser in taps when the left most right edge was found.
dbg_rd_stage1_cal[78:73]	mem_out_dec	Output of static compensation ROM.
dbg_rd_stage1_cal[81]	rdlvl_pi_stg2_f_incdec	Controls directing of phaser in stepping.
dbg_rd_stage1_cal[82]	rdlvl_pi_en_stg2_f	Phaser in step command.
dbg_rd_stage1_cal[85:83]	pi_lane_r	Lane to which phaser in commands apply.
dbg_rd_stage1_cal[91]	prev_match_r	Previous sample matched.
dbg_rd_stage1_cal[96:92]	match_out_r	idelay of last detected invalid to valid match transition.
dbg_rd_stage1_cal[102:97]	samp_cnt_r	Sample counter.
dbg_rd_stage1_cal[108:103]	samps_match_r	Cumulative sample match count.

Table 2-20:

(Cont'd)

dbg_rd_stage1_cal[109]	samp_result_held_r	Result from previous sample cycle.
dbg_rd_stage1_cal[154+:40]	simp_dlyval_r	Five bits per lane dlyval results for simple pattern.
dbg_rd_stage1_cal[194+:48]	simp_left_r	Six bits per lane left results for simple pattern.
dbg_rd_stage1_cal[194+:48]	simp_right_r	Six bits per lane right results for simple pattern.
dbg_rd_stage1_cal[194+:48]	simp_center_r	Six bits per lane center results for simple pattern.
dbg_rd_stage1_cal[378+:48]	cmplx_left_r	Six bits per lane left results for complex pattern.
dbg_rd_stage1_cal[426+:48]	cmplx_right_r	Six bits per lane right results for complex pattern.
dbg_rd_stage1_cal[474+:48]	cmplx_center_r	Six bits per lane center results for complex pattern.
dbg_rd_stage1_cal[682+:48]	simp_left_63	Left edge result is 63 for simple pattern, one bit per lane.
dbg_rd_stage1_cal[690+:48]	cmplx_left_63	Left edge result is 63 for complex pattern, one bit per lane.
dbg_rd_stage1_cal[698+:48]	simp_right_63	Right edge result is 63 for simple pattern, one bit per lane.
dbg_rd_stage1_cal[706+:48]	cmplx_right_63	Right edge result is 63 for complex pattern, one bit per lane.
dbg_rd_stage1_cal[522+:72]	rd_data_lane_r	Aligned PHY data for lane currently undergoing calibration.
dbg_rd_stage1_cal[594+:72]	iserdes_lane_r	Raw PHY data for lane currently undergoing calibration.
dbg_rd_stage1_cal[714+:72]	cmplx_rd_burst_bytes	Complex data to compare against memory read data.
dbg_rd_stage1_cal[786+:9]	bit_comp	Cumulative compare per bit.
dbg_rd_stage1_cal[795+:8]	simp_min_eye_r	Minimum eye detected per lane simple pattern.
dbg_rd_stage1_cal[803+:8]	cmplx_min_eye_r	Minimum eye detected per lane complex pattern.

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:

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

Table 2-21:

(Cont'd)

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 for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[12]	fd0_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[13]	rd1_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[14]	fd1_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[15]	phase_vld	Valid data is seen for the particular byte for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[16]	rd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[17]	fd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[18]	rd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[19]	fd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[20]	phase_bslip_vld	Valid data is seen when bitslip applied to read data for the byte being calibrated (indicated by byte_cnt)
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. Only used for nCK_PER_CLK == 2.
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[30]	bitslip	FPGA logic bitslip control signal, indicates when the logic shifts the data alignment. Only used for nCK_PER_CLK == 4.
dbg_stage2_cal[31]	rd2_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[32]	fd2_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[33]	rd3_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.

Table 2-21:

(Cont'd)

dbg_stage2_cal[34]	fd3_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[35]	latency_measured[1]	Indicates latency has been measured for byte lane 1
dbg_stage2_cal[36]	latency_measured[2]	Indicates latency has been measured for byte lane 2
dbg_stage2_cal[37]	latency_measured[3]	Indicates latency has been measured for byte lane 3
dbg_stage2_cal[38]	error_adj_latency	Indicates error when target PHY_LATENCY cannot be achieved
dbg_stage2_cal[127:39]	Reserved	Reserved

## System Clock

If the SRCC/MRCC I/O pin and PLL are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint must be set to BACKBONE. QDR II+ SRAM manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on the SRCC/MRCC I/O and PLL allocation needs to be handled manually for the IP flow. QDR II+ SRAM does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

## Reference Clock

If the SRCC/MRCC I/O pin and MMCM are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint is set to FALSE. Reference clock is a 200 MHz clock source used to drive IODELAY CTRL logic (through an additional MMCM). This clock is not utilized, CLOCK\_DEDICATED\_ROUTE (as they are limited in number), hence the FALSE value is set. QDR II+ SRAM manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on SRCC/MRCC I/O and MMCM allocation needs to be handled manually for the IP flow. QDR II+ SRAM does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

# RLDRAM II and RLDARAM 3 Memory Interface Solutions

---

The RLDARAM II and RLDARAM 3 Memory Interface Solutions (MIS) are a Memory Controller and physical layer for interfacing Xilinx® 7 series FPGAs user designs to RLDARAM II and RLDARAM 3 devices. An RLDARAM II/RLDARAM 3 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/RLDARAM 3 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/RLDARAM 3 protocol before sending them to the memory. Unique capabilities of the 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/RLDARAM 3 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 design. For detailed board design guidelines, see [Design Guidelines, page 467](#).



**IMPORTANT:** *RLDRAM II and RLDARAM 3 designs currently do not support memory-mapped AXI4 interfaces.*

---

For detailed information and updates about the 7 series FPGAs RLDARAM II and RLDARAM 3 interface cores, see the appropriate 7 series FPGAs data sheet [\[Ref 13\]](#) and the *Zynq-7000 SoC and 7 Series FPGAs Memory Interface Solutions Data Sheet* (DS176) [\[Ref 1\]](#).

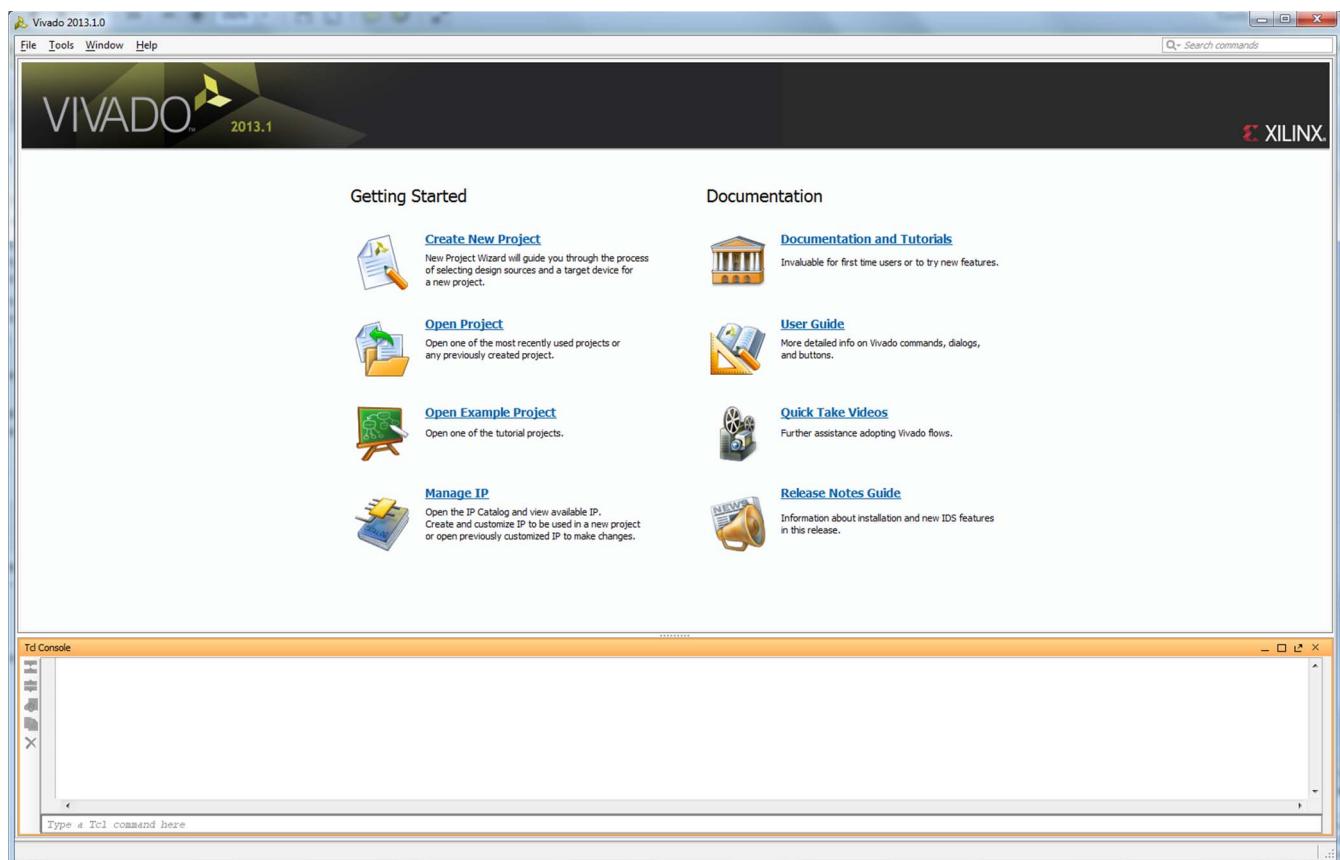


**IMPORTANT:** *Memory Interface Solutions v4.2 only supports the Vivado® Design Suite. The ISE® Design Suite is not supported in this version.*

---

This section provides the steps to generate the Memory Interface Generator (MIG) IP core using the Vivado Design Suite and run implementation.

1. Start the Vivado Design Suite (see [Figure 3-1](#)).



*Figure 3-1:*

2. To create a new project, click the **Create New Project** option shown in [Figure 3-1](#) to open the page as shown in [Figure 3-2](#).

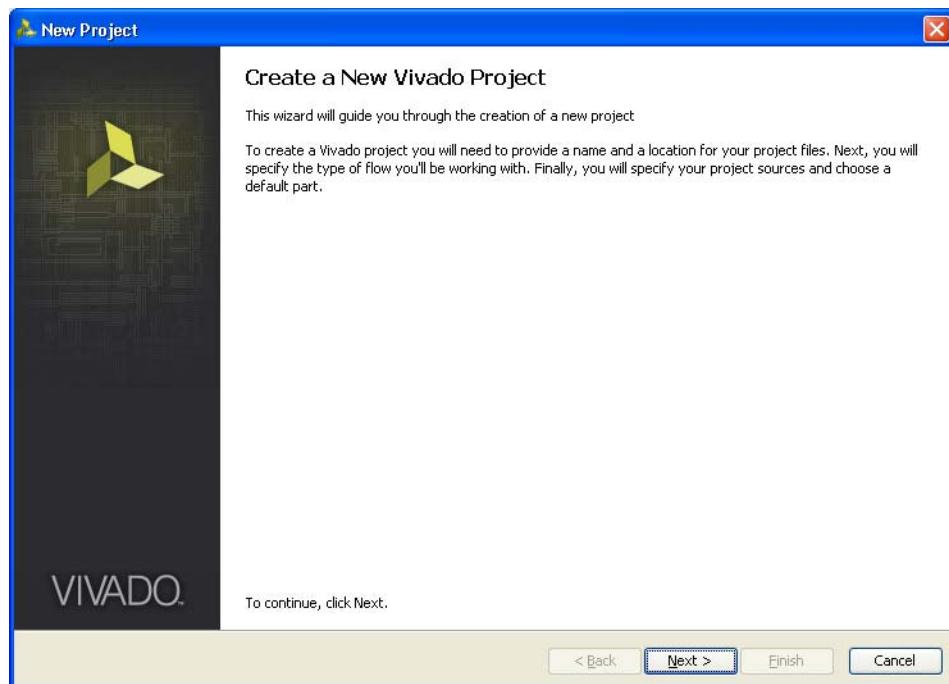


Figure 3-2:

3. Click **Next** to proceed to the **Project Name** page (Figure 3-3). Enter the **Project Name** and **Project Location**. Based on the details provided, the project is saved in the directory.

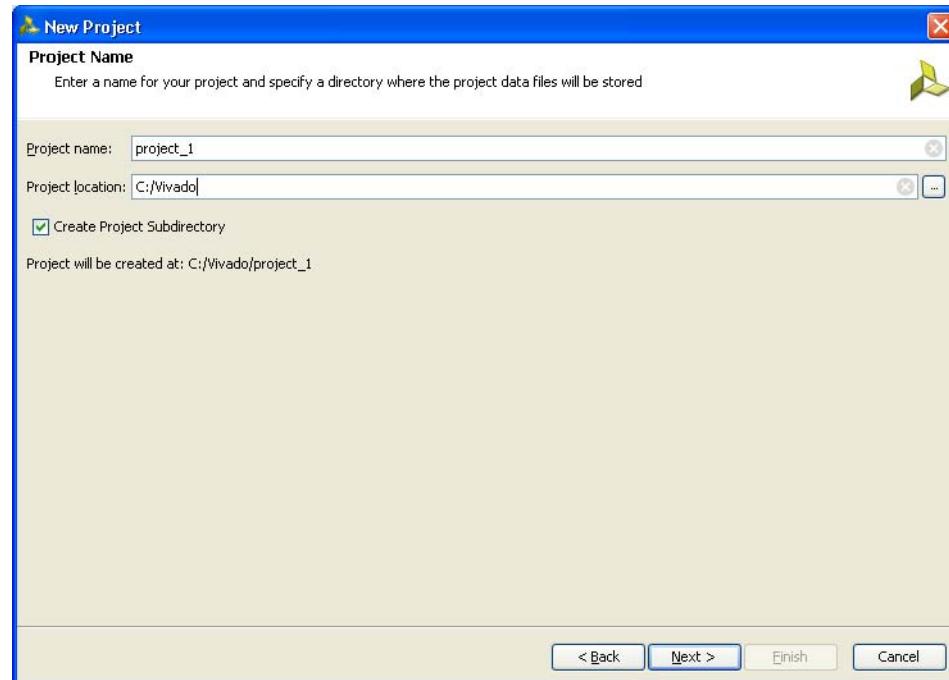
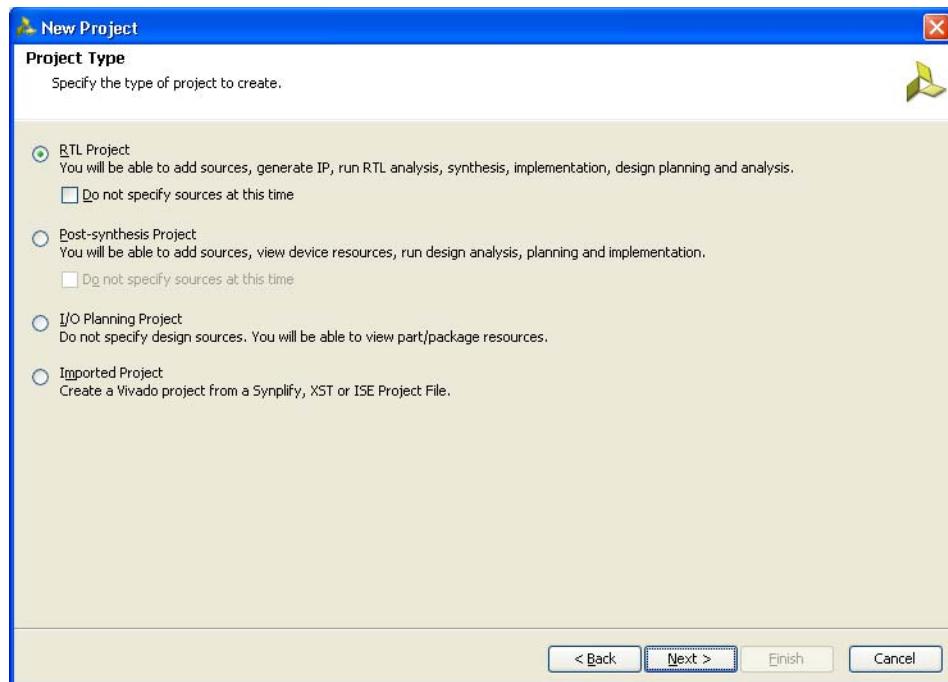


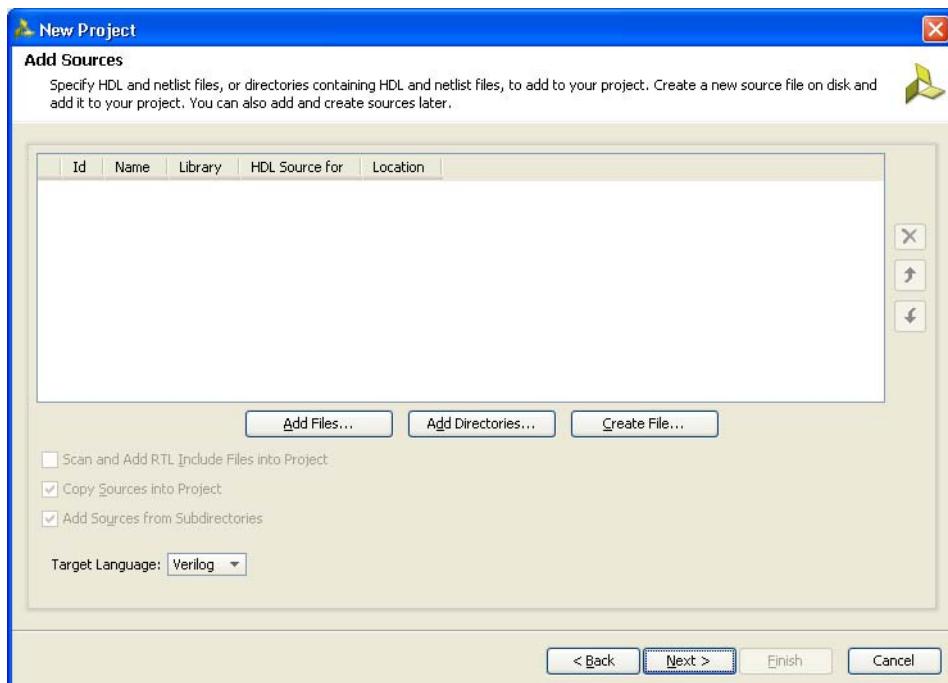
Figure 3-3:

4. Click **Next** to proceed to the **Project Type** page (Figure 3-4). Select the **Project Type** as **RTL Project** because MIG deliverables are RTL files.



*Figure 3-4:*

5. Click **Next** to proceed to the **Add Sources** page (Figure 3-5). RTL files can be added to the project in this page. If the project was not created earlier, proceed to the next page.



*Figure 3-5:*

6. Click **Next** to open the **Add Existing IP (Optional)** page (Figure 3-6). If the IP is already created, the XCI file generated by the IP can be added to the project and the previous created IP files are automatically added to the project. If the IP was not created earlier, proceed to the next page.

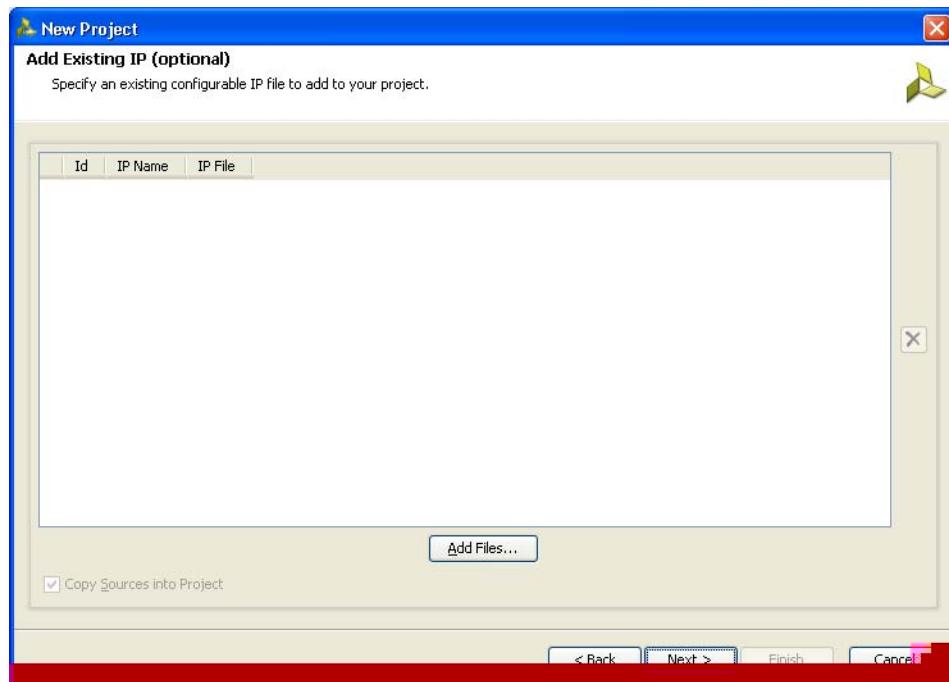


Figure 3-6:

7. Click **Next** to open the **Add Constraints (Optional)** page (Figure 3-7). If the constraints file exists in the repository, it can be added to the project. Proceed to the next page if the constraints file does not exist.

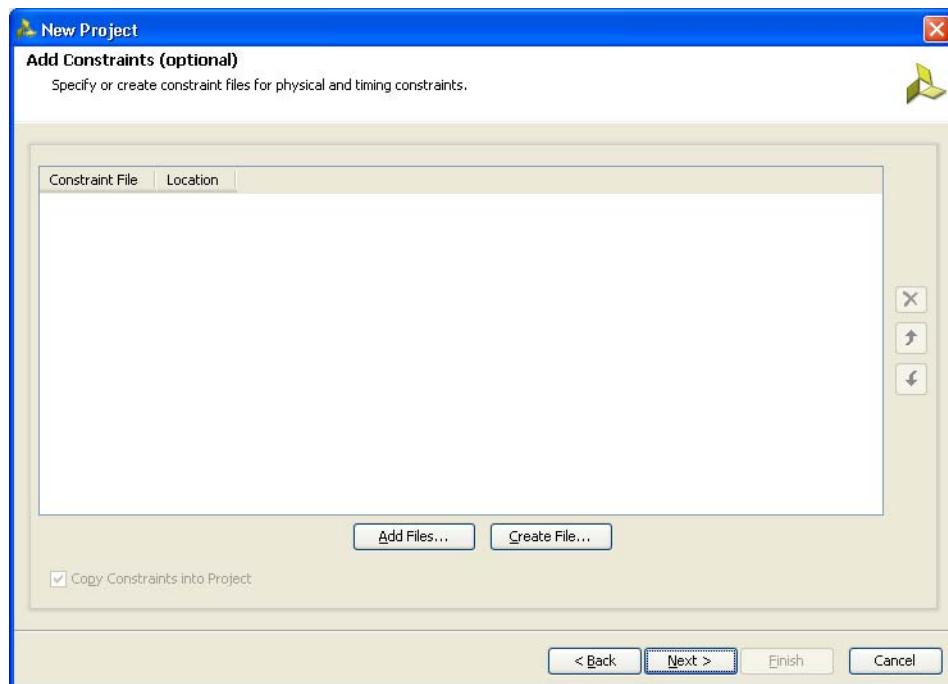


Figure 3-7:

- Click **Next** to proceed to the **Default Part** page (Figure 3-8) where the device that needs to be targeted can be selected. The **Default Part** page appears as shown in Figure 3-8.

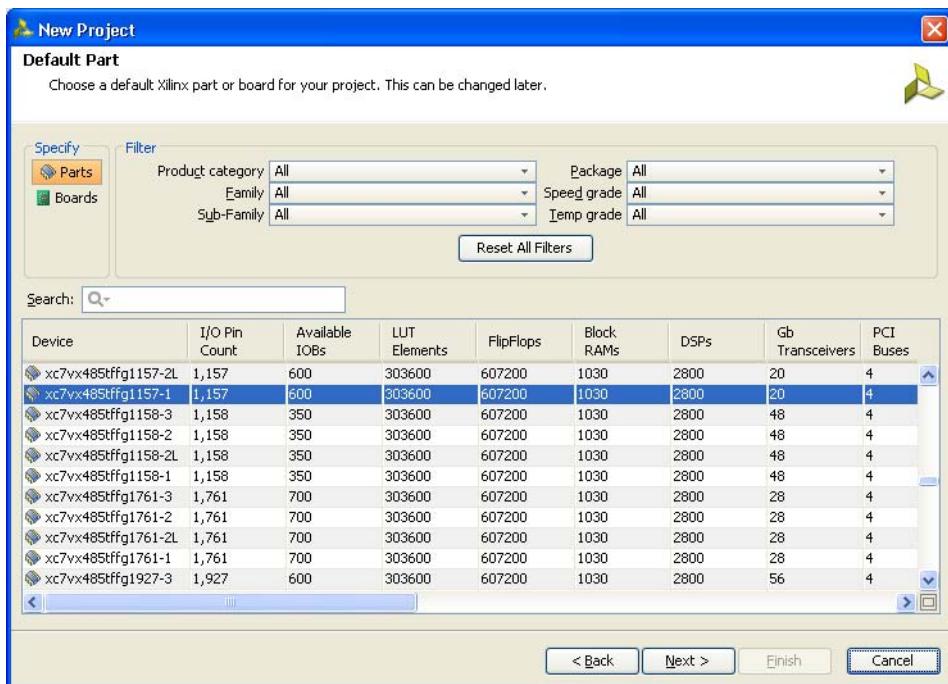
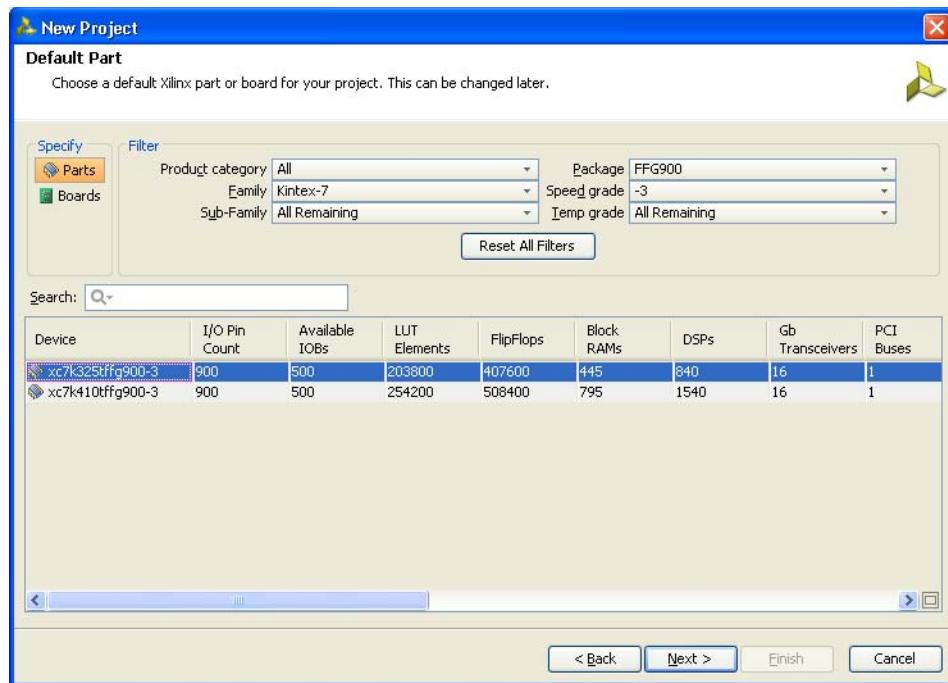


Figure 3-8:

Select the target **Family**, **Package**, and **Speed Grade**. The valid devices are displayed in the same page, and the device can be selected based on the targeted device ([Figure 3-9](#)).



*Figure 3-9:*

Apart from selecting the parts by using **Parts** option, parts can be selected by choosing the **Boards** option, which brings up the evaluation boards supported by Xilinx ([Figure 3-10](#)). With this option, design can be targeted for the various evaluation boards. If the XCI file of an existing IP was selected in an earlier step, the same part should be selected here.

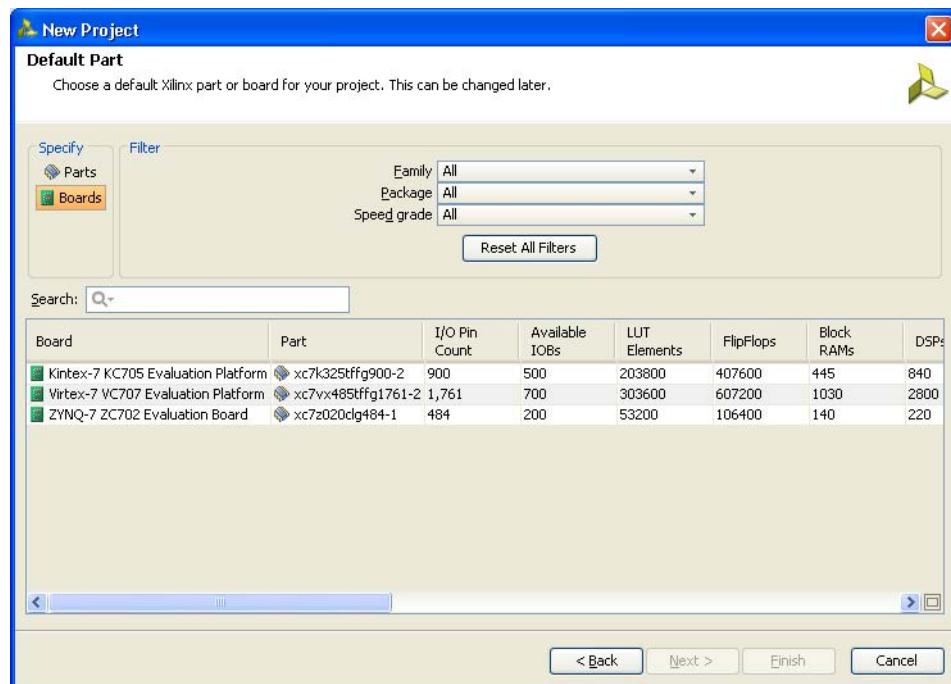


Figure 3-10:

- Click **Next** to open the **New Project Summary** page (Figure 3-11). This includes the summary of selected project details.

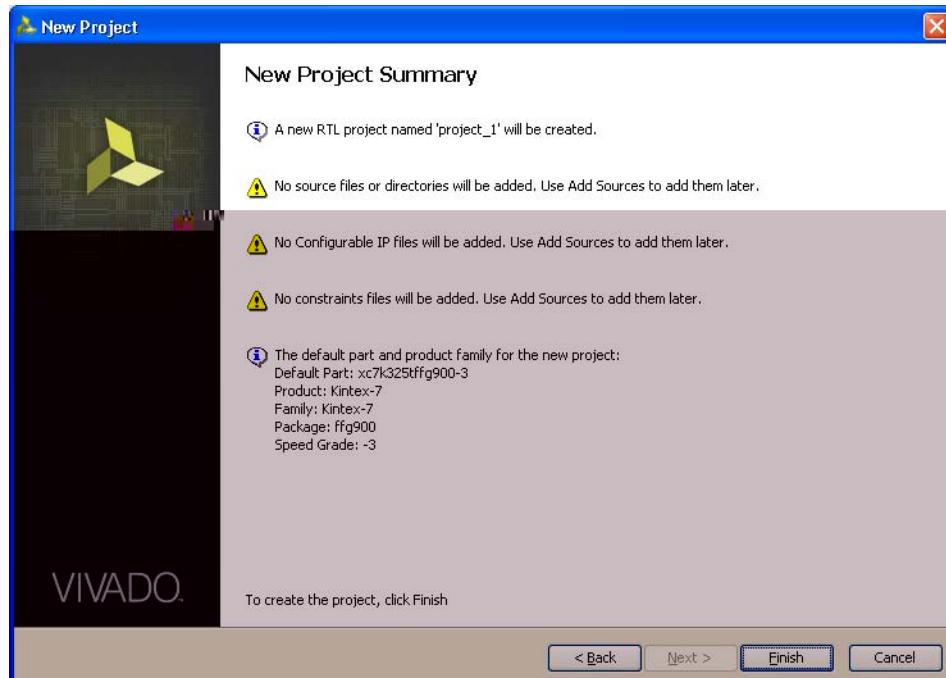
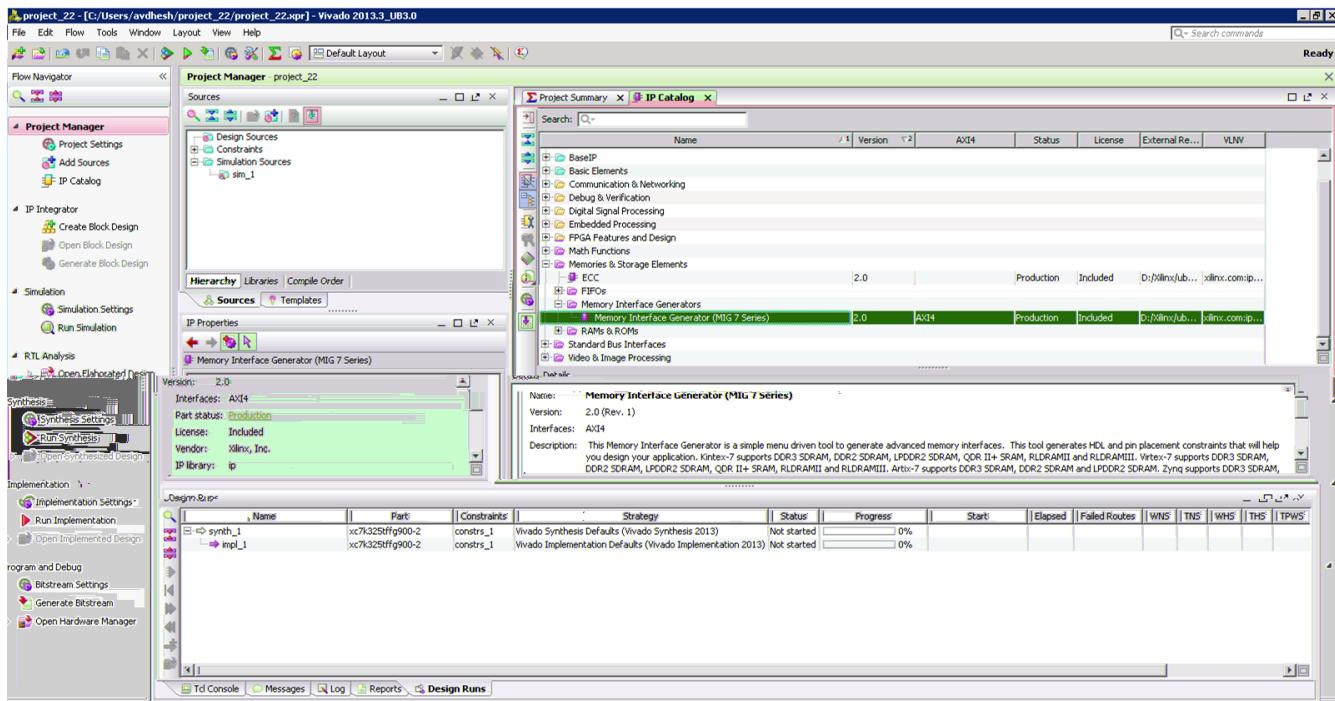


Figure 3-11:

- Click **Finish** to complete the project creation.

11. Click **IP Catalog** on the **Project Manager** window to open the Vivado IP catalog window. The IP catalog window appears on the right side panel (see [Figure 3-12](#), highlighted in a red circle).
12. The MIG tool exists in the **Memories & Storage Elements > Memory Interface Generators** section of the IP catalog window ([Figure 3-12](#)) or you can search from the Search tool bar for the string “MIG.”



*Figure 3-12:*

13. Select **MIG 7 Series** to open the MIG tool (Figure 3-13).

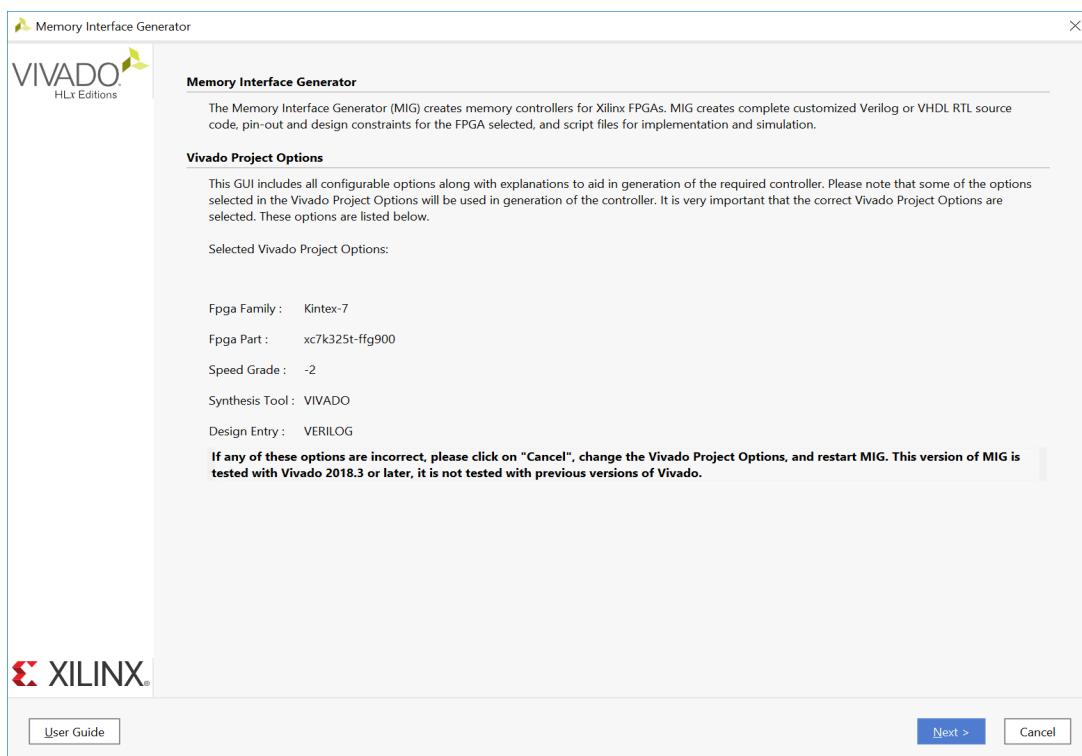


Figure 3-13:

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



**CAUTION!** *The Windows operating system has a 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs.*

## MIG Output Options

1. Select **Create Design** to create a new Memory Controller design. Enter a component name in the Component Name field (Figure 3-14).
2. Choose the number of controllers to be generated. This option determines the replication of further pages.

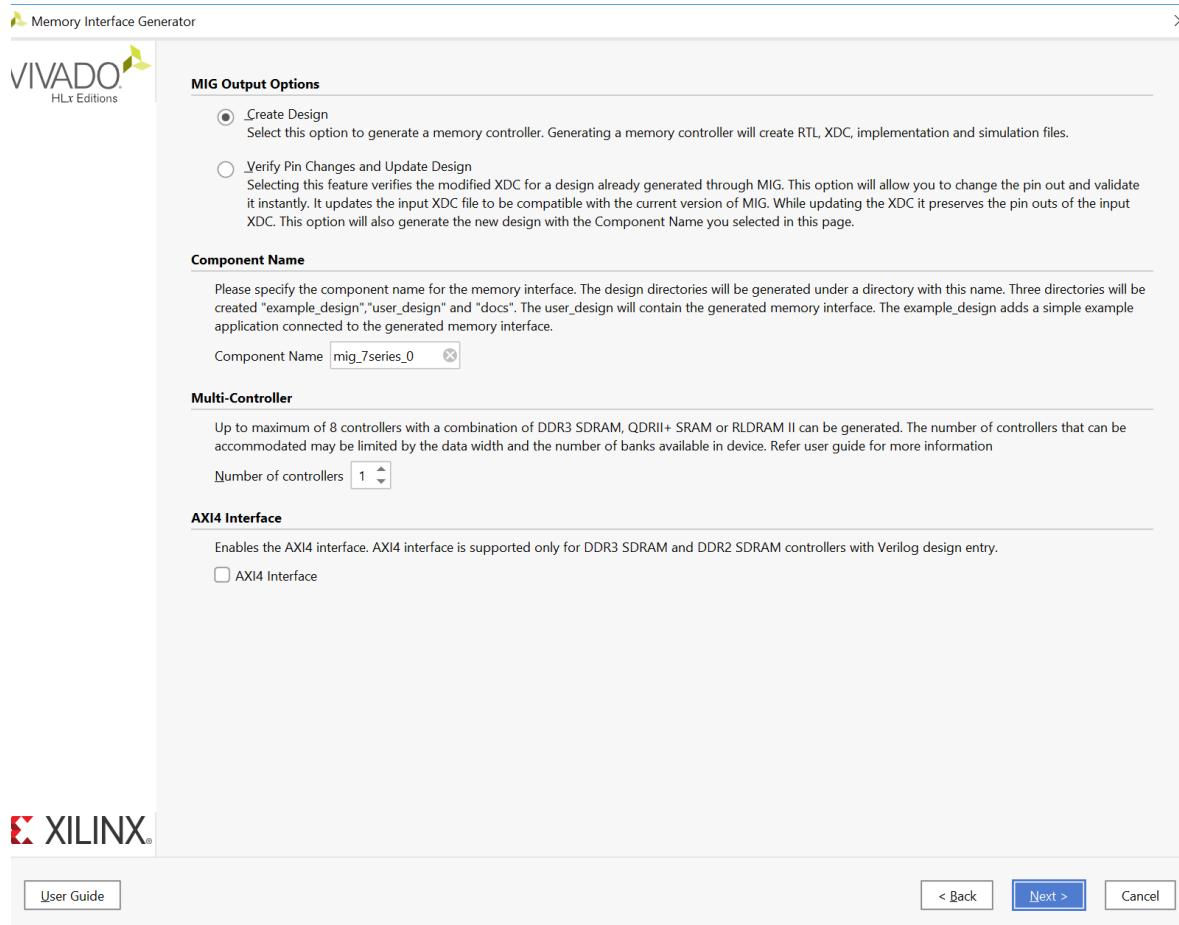


Figure 3-14:

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



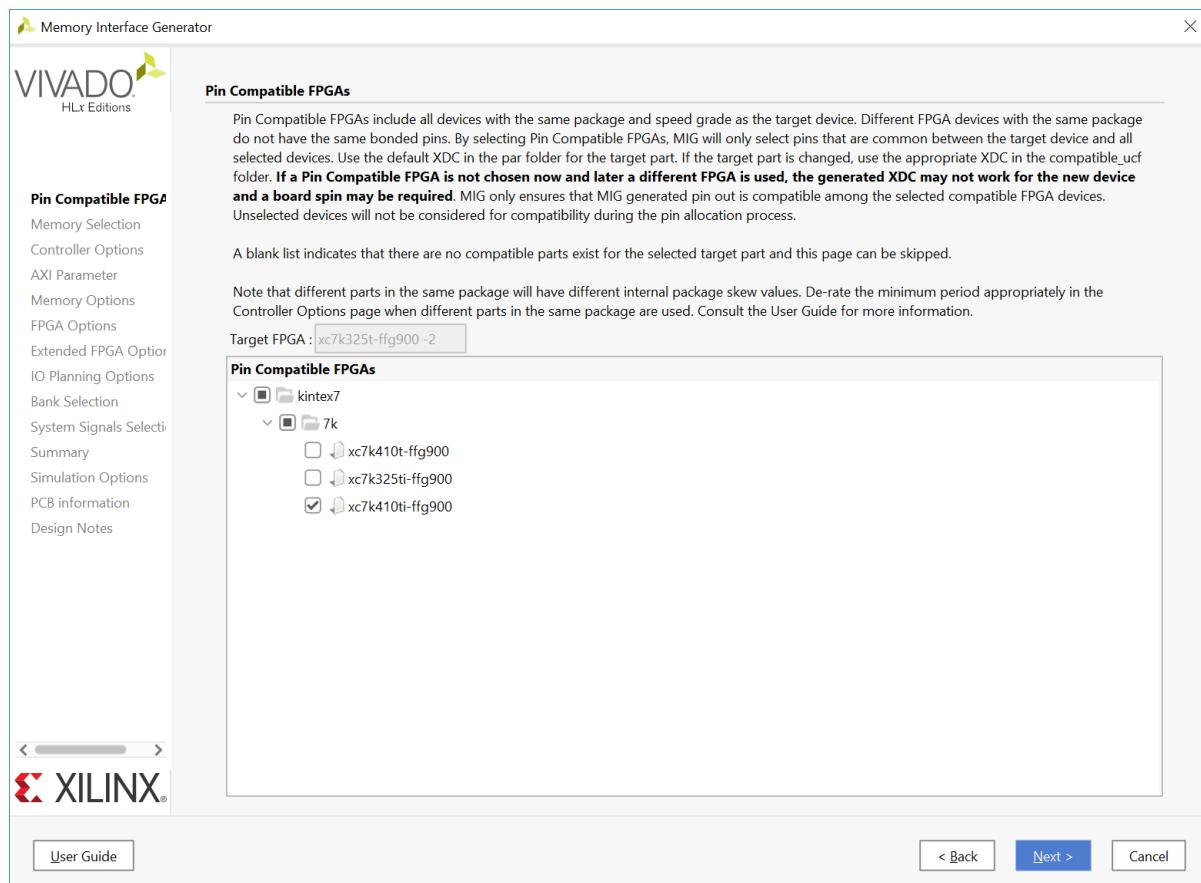
**IMPORTANT:** Only alphanumeric characters can be used for <component name>. Special characters cannot be used. This name should always start with an alphabetical character and can end with an alphanumeric character.

When invoked from Xilinx Platform Studio (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 3-15).



*Figure 3-15:*

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/RLDRAM 3 Memory Design***

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

1. Select the RLDRAM II or RLDRAM 3 controller type.
2. Click **Next** to display the **Controller Options** page.

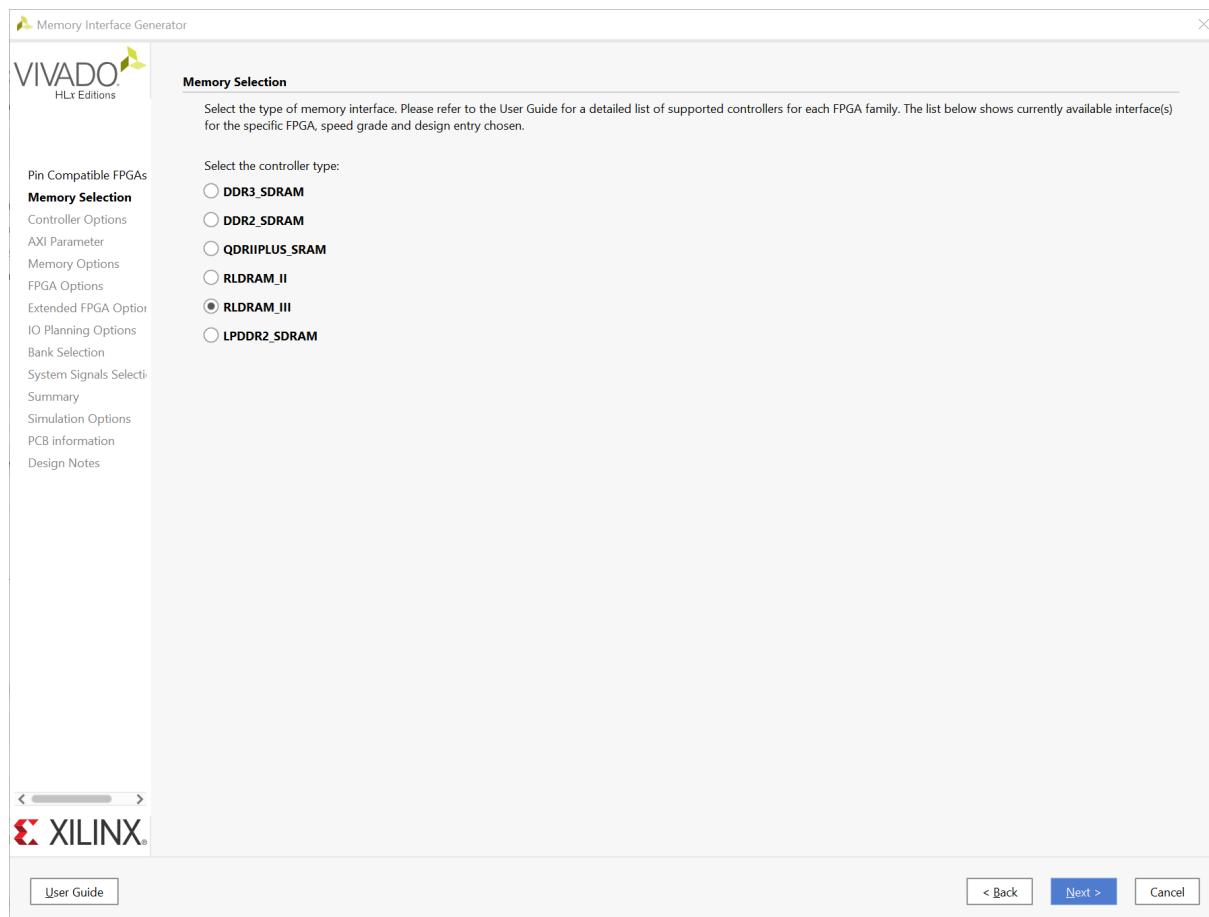


Figure 3-16:

RLDRAM II and RLDAM 3 designs currently do not support memory-mapped AXI4 interfaces.

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

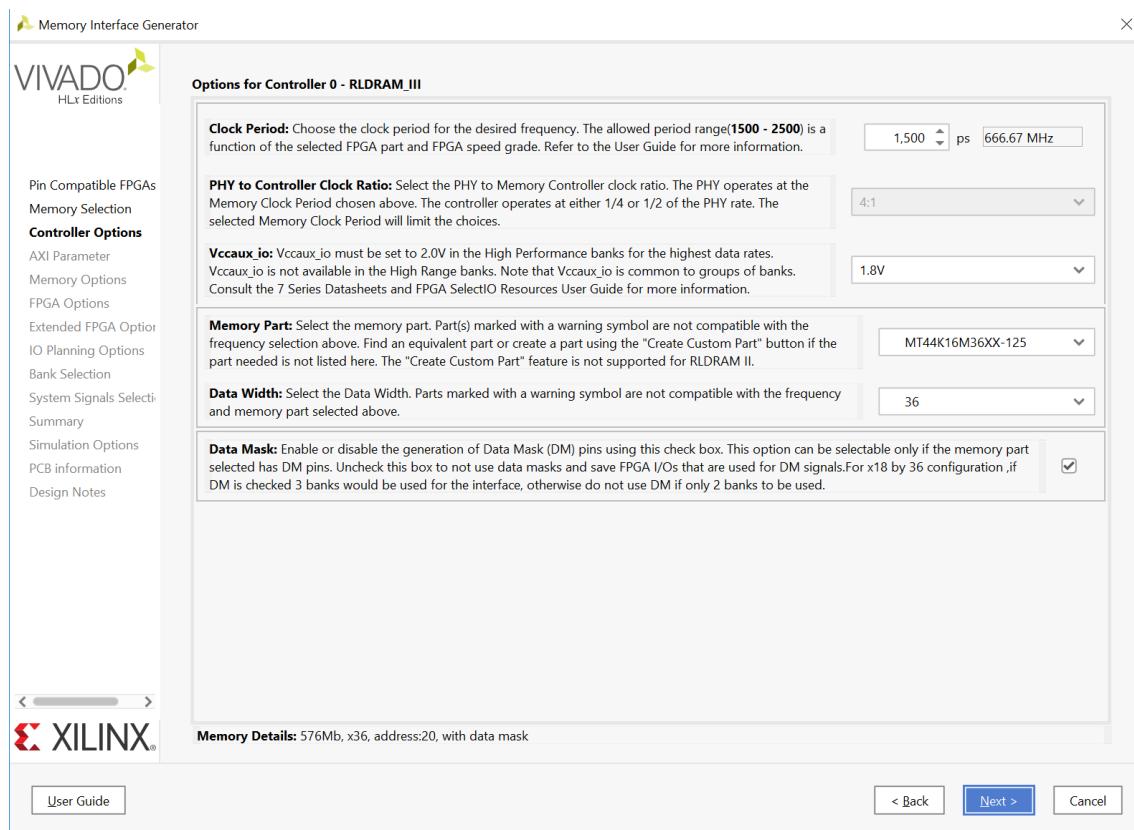


Figure 3-17:

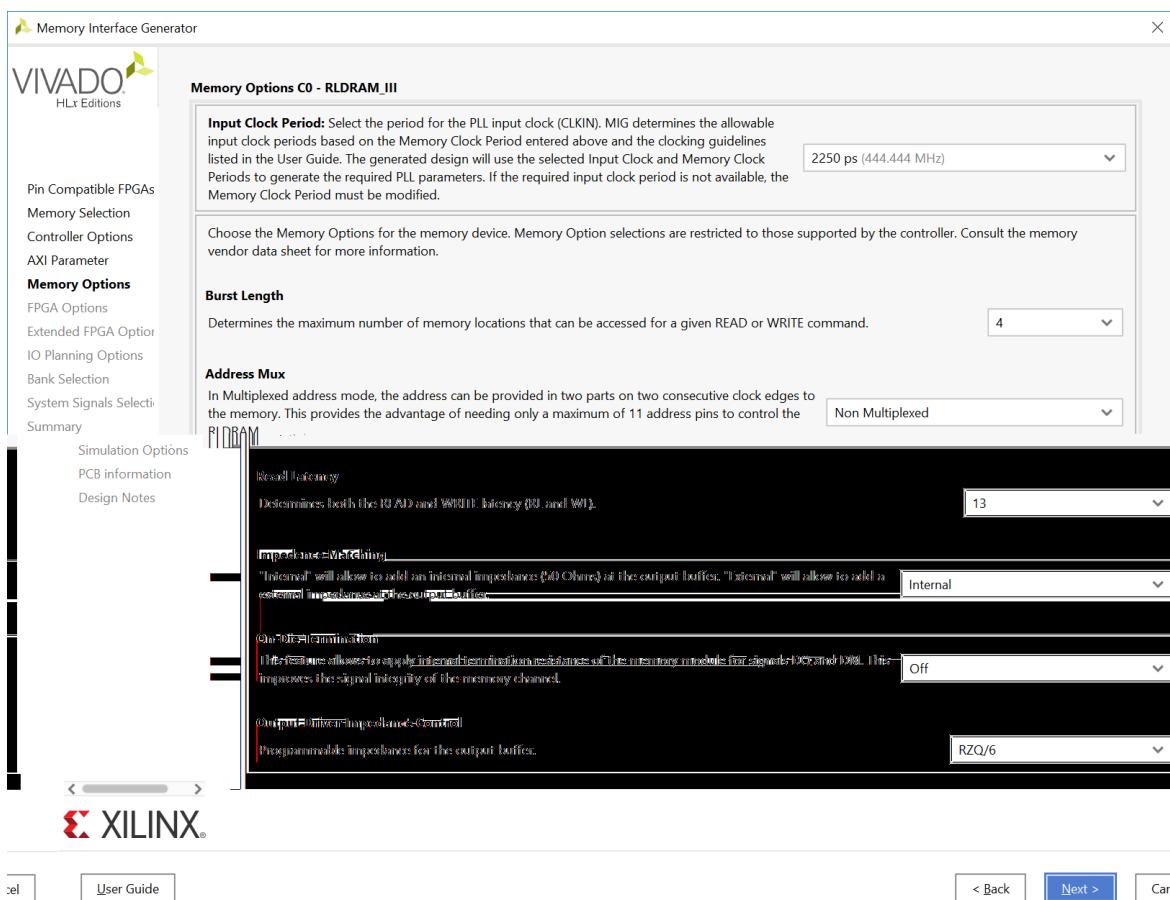
- **Frequency** – This feature indicates the operating frequency for all the controllers. The frequency block is limited by factors such as the selected FPGA and device speed grade.
- **PHY to Controller Clock Ratio** – This feature determines the ratio of the physical layer (memory) clock frequency to the controller and user interface clock frequency. The user interface data bus width of the 2:1 ratio is four times the width of the physical memory interface width, while the bus width of the 4:1 ratio is eight times the physical memory interface width. RLDRAM II must use 2:1 while RLDRAM 3 must use 4:1.
- **VCCAUX\_IO** – 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. For more information, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2].
- **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**). If a desired part is not available in the list, you 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.
- **Data Mask** – This option allocates data mask pins when selected. This option should be deselected to deallocate data mask pins and increase pin efficiency.
- **Memory Details** – The bottom of the **Controller Options** page. [Figure 3-18](#) displays the details for the selected memory configuration.

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

*Figure 3-18:*

This feature allows the selection of various memory mode register values, as supported by the controller specification ([Figure 3-19](#)).



*Figure 3-19:*

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 435](#) for more information on the PLL parameter limits.
- **Configuration** – (RLDRAM II only). 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 for RLDRAM II, and 2, 4, and 8 are allowed for RLDRAM 3.
- **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 two.
- **Impedance Matching** – This option determines how the memory device tunes its outputs, either by an internal setting or using an external reference resistor connected to the ZQ input of the memory device.
- **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. For RLDRAM II this can only be off or on, but for RLDRAM 3 when a value must be selected, either RZQ/6, RZQ/4, or RZQ/2.
- **Output Driver Impedance Control** – Not available for RLDRAM II. MRS setting in the DRAM that selects the impedance of the output buffers during reads.

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

Figure 3-20 shows the **FPGA Options** page.

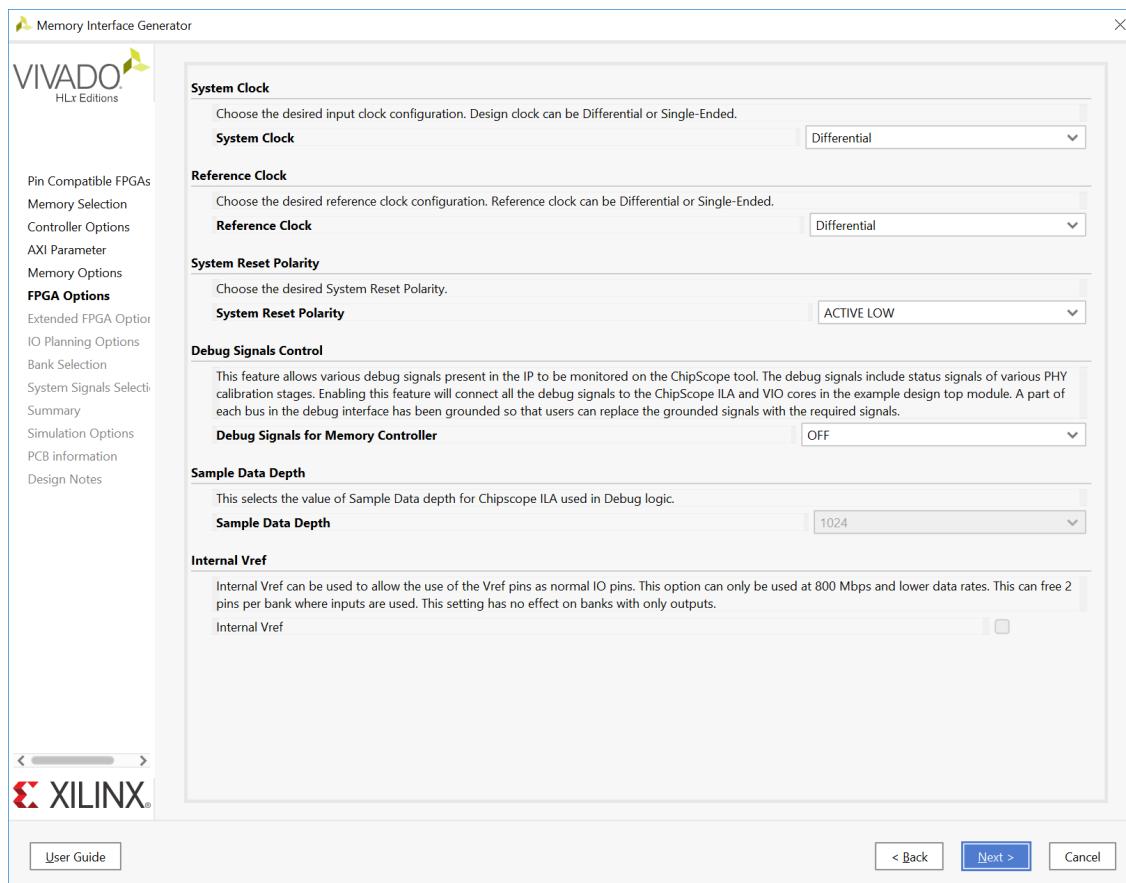


Figure 3-20:

- **System Clock** – This option selects the clock type (Single-Ended, Differential, or No Buffer) for the **sys\_clk** signal pair. When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the system clock.

If the designs generated from MIG for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the **sys\_clk\_i** signal. So for **No Buffer** scenarios, **sys\_clk\_i** signal needs to be connected to an internal clock.

- **Reference Clock** – This option selects the clock type (Single-Ended, Differential, No Buffer, or Use System Clock) for the **clk\_ref** signal pair. The **Use System Clock** option appears when the input frequency is between 199 and 201 MHz (that is, the Input Clock Period is between 5,025 ps (199 MHz) and 4,975 ps (201 MHz)). When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the reference clock.

If the designs generated from MIG for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the **ref\_clk\_i** signal. So for **No Buffer** scenarios, **ref\_clk\_i** signal needs to be connected to an internal clock.

- **System Reset Polarity** – The polarity for system reset (**sys\_rst**) can be selected. If the option is selected as active-Low, the parameter **RST\_ACT\_LOW** is set to 1 and if set to active-High the parameter **RST\_ACT\_LOW** is set to 0.
- **Debug Signals Control** – Selecting this option enables calibration status and user port signals to be port mapped to the ILA and VIO in the **example\_top** module. This helps in monitoring traffic on the user interface port with the Vivado Design Suite debug feature. Deselecting the **Debug Signals Control** option leaves the debug signals unconnected in the **example\_top** module and no ILA/VIO modules are generated by the IP catalog. Additionally, the debug port is always disabled for functional simulations.
- **Sample Data Depth** – This option selects the Sample Data depth for the ILA module used in the Vivado debug logic. This option can be selected when the **Debug Signals for Memory Controller** option is ON.
- **Internal V<sub>REF</sub> Selection** – Internal V<sub>REF</sub> can be used for data group bytes to allow the use of the V<sub>REF</sub> pins for normal I/O usage. Internal V<sub>REF</sub> should only be used for data rates of 800 Mb/s or below.

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

Figure 3-21 shows the Extended FPGA Options page.

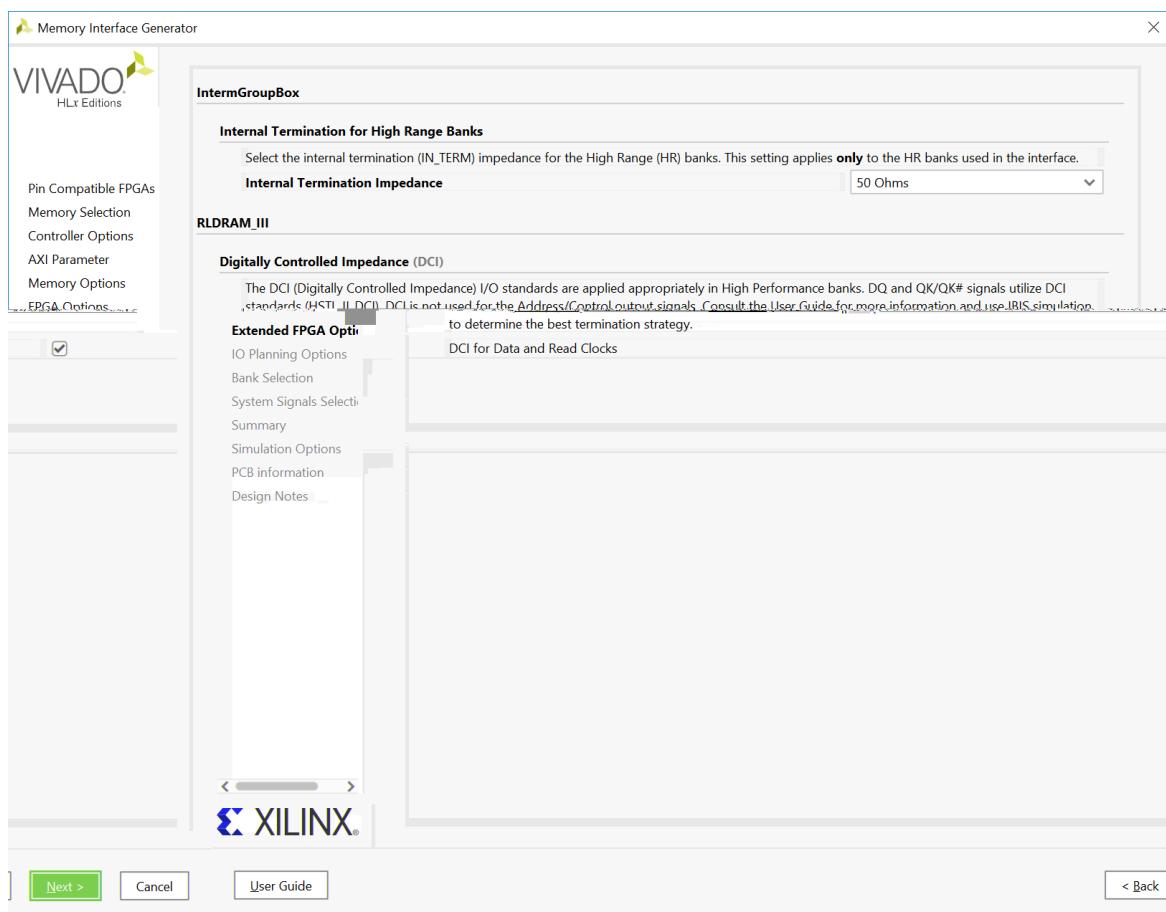


Figure 3-21:

- **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 and RLDRAm 3 read path. This selection is only for High Range banks.
- **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.

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 **Deselect Banks**. To restore the defaults, click **Restore Defaults**. 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.

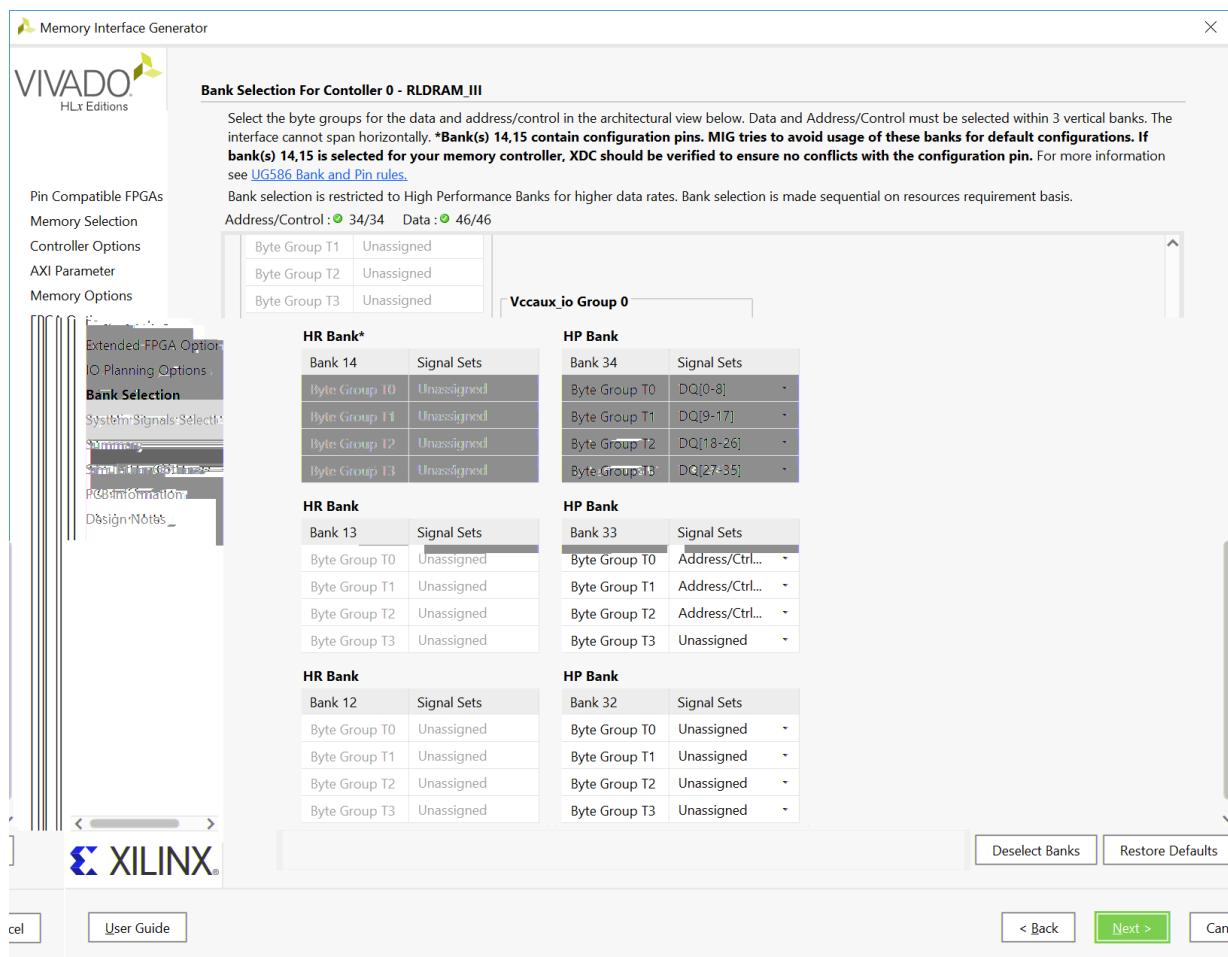


Figure 3-22:

Figure 3-23 shows the **System Pins Selection** page.

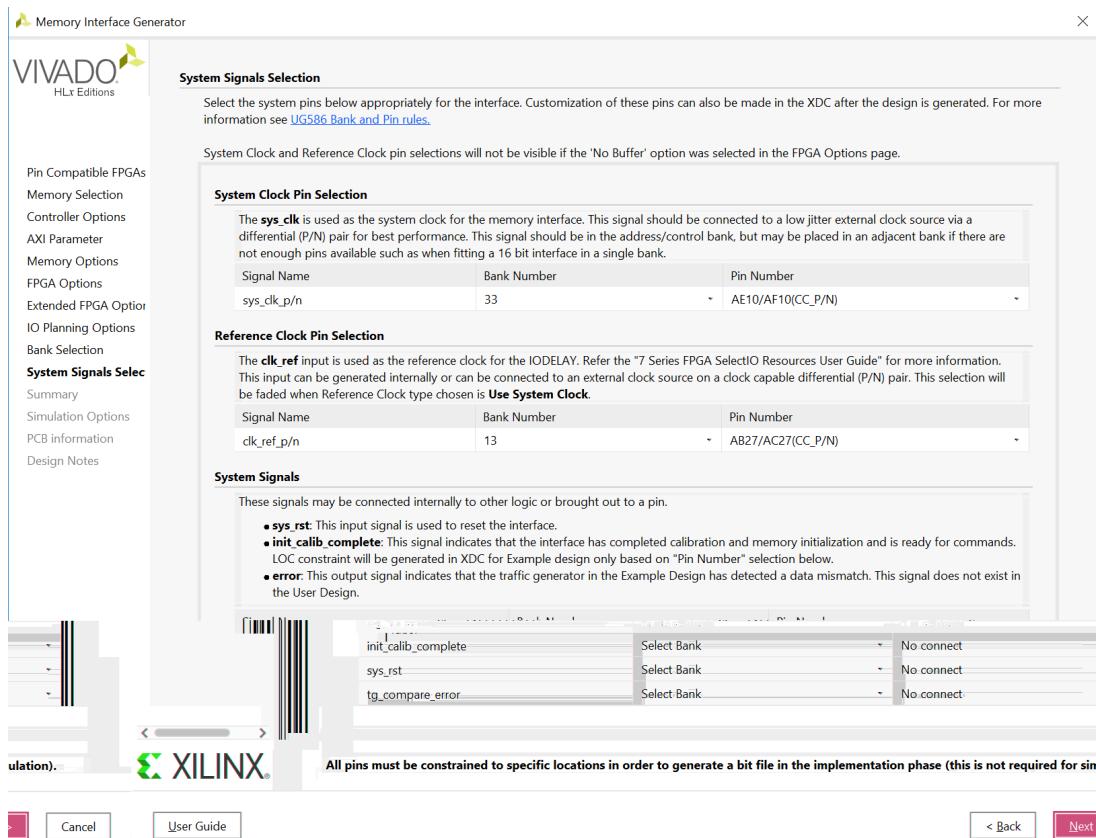


Figure 3-23:

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 3-20). 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\_HSTL\_I or HSTL\_I. If **sys\_clk** is not connected in a memory interface bank, the MIG tool selects an appropriate standard such as LVCMOS18 or LVDS. The XDC 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 3-20). The I/O standard is selected in a similar way as **sys\_clk** above.

- **sys\_rst** – This is the asynchronous 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 LVCMOS18 and LVCMOS25 for HP and HR banks, respectively. The default polarity of **sys\_rst** pin is active-Low. The polarity of **sys\_rst** pin varies based on the **System Reset Polarity** option chosen in **FPGA Options** page ([Figure 3-20](#)).
- **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.

This page ([Figure 3-24](#)) provides the complete details about the memory core selection, interface parameters, Vivado IP catalog options, and FPGA options of the active project.

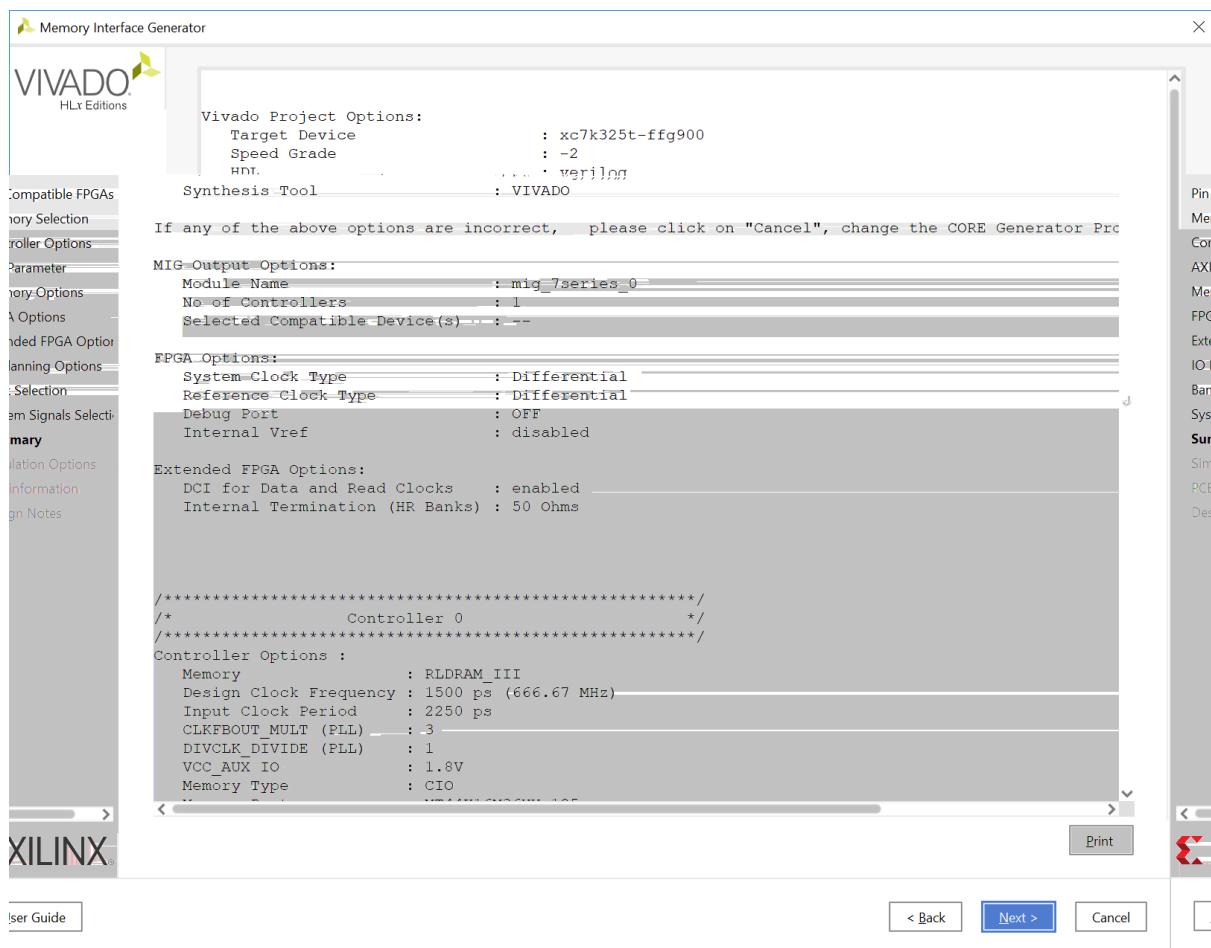


Figure 3-24:

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

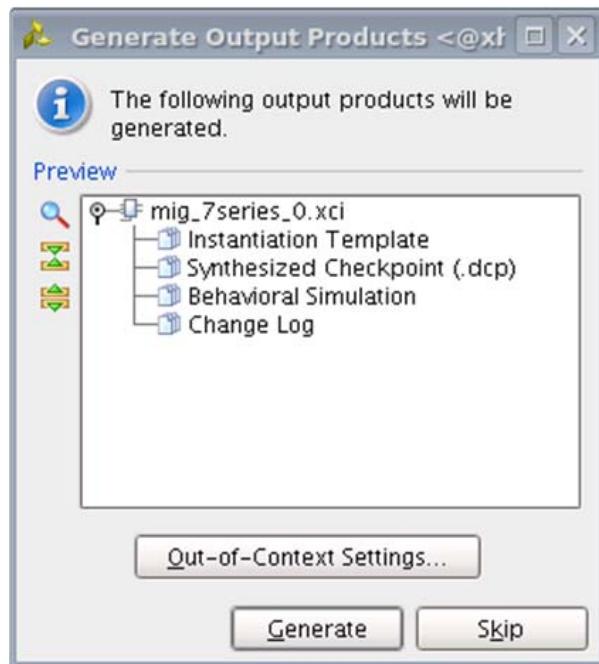
This page displays the PCB-related information to be considered while designing the board that uses the MIG tool generated designs. Click **Next** to move to the **Design Notes** page.

Click **Generate** 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.

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

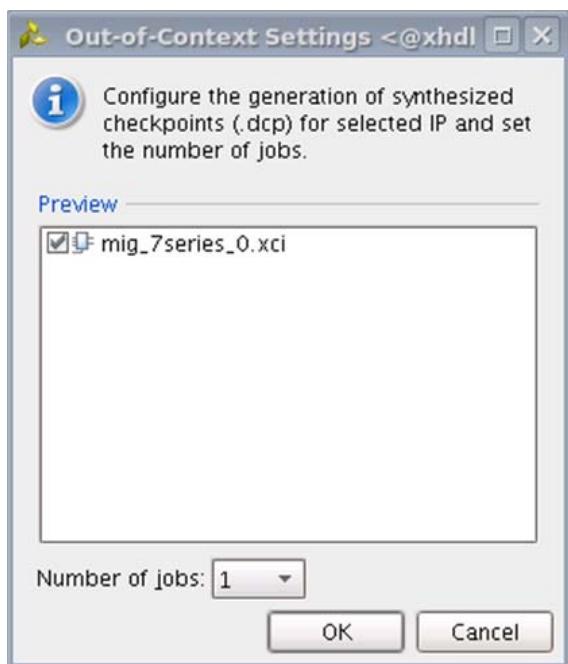
Click **Close** to complete the MIG tool flow.

1. After clicking **Generate**, the **Generate Output Products** window appears. This window has the **Out-of-Context Settings** as shown in [Figure 3-25](#).



*Figure 3-25:*

2. Click **Out-of-Context Settings** to configure generation of synthesized checkpoints. To enable the **Out-of-Context** flow, enable the check box. To disable the **Out-of-Context** flow, disable the check box. The default option is "enable" as shown in [Figure 3-26](#).



*Figure 3-26:*

3. MIG designs comply with "Hierarchical Design" flow in Vivado. For more information, see the *Vivado Design Suite User Guide: Hierarchical Design* (UG905) [\[Ref 5\]](#) and the *Vivado Design Suite Tutorial: Hierarchical Design* (UG946) [\[Ref 6\]](#).

4. After generating the MIG design, the project window appears as shown in [Figure 3-27](#).

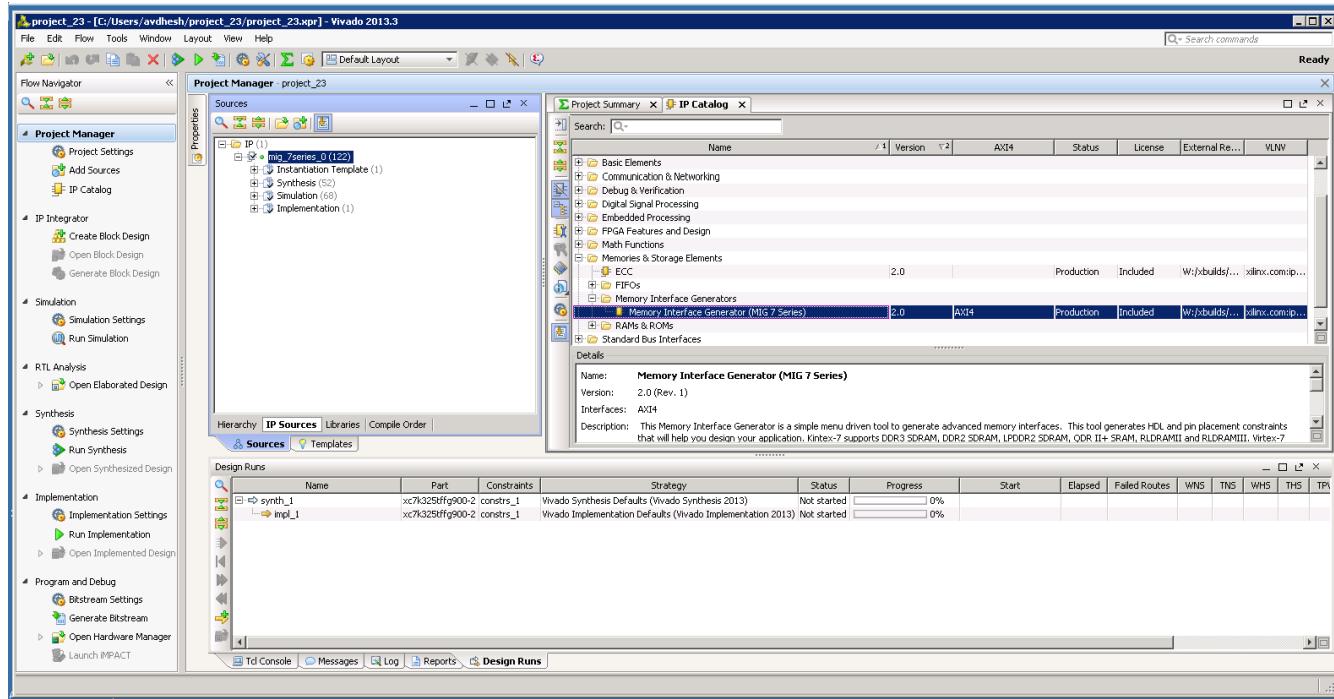


Figure 3-27:

5. After project creation, the XCI file is added to the Project Hierarchy. The same view also displays the module hierarchies of the user design. The list of HDL and XDC files is available in the **IP Sources** view in the **Sources** window. Double-clicking on any module or file opens the file in the Vivado Editor. These files are read only.

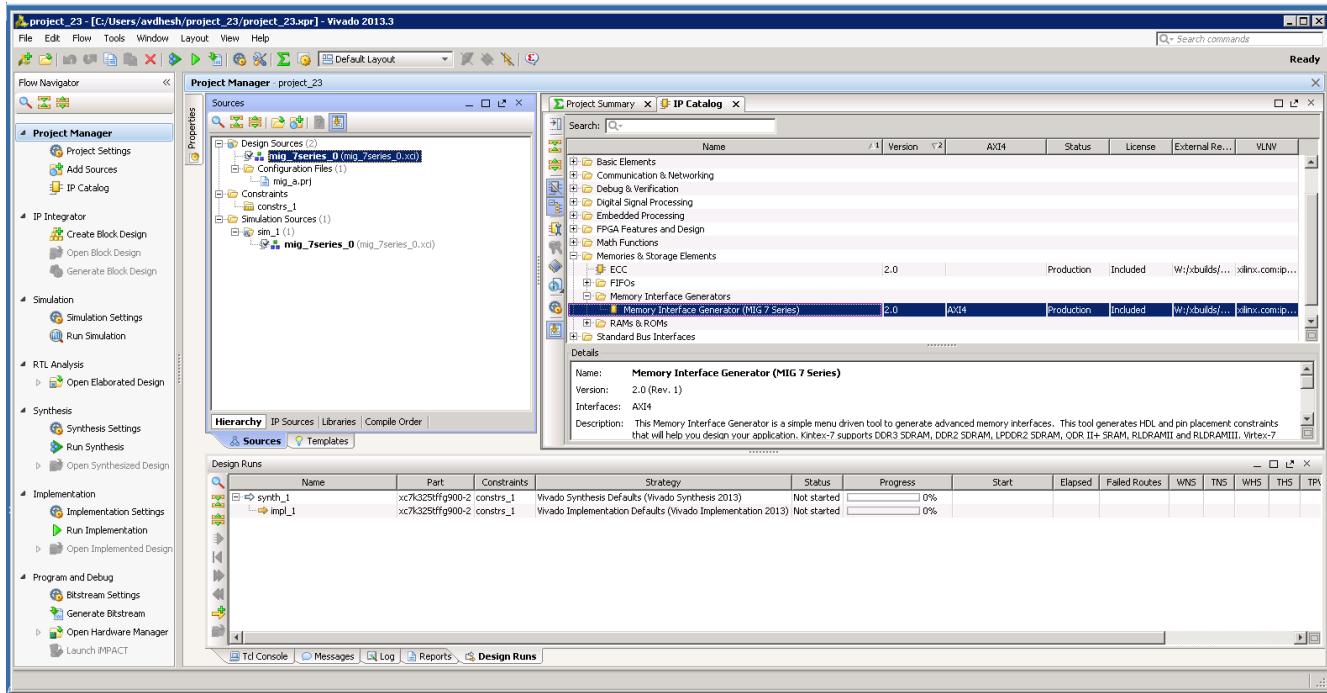
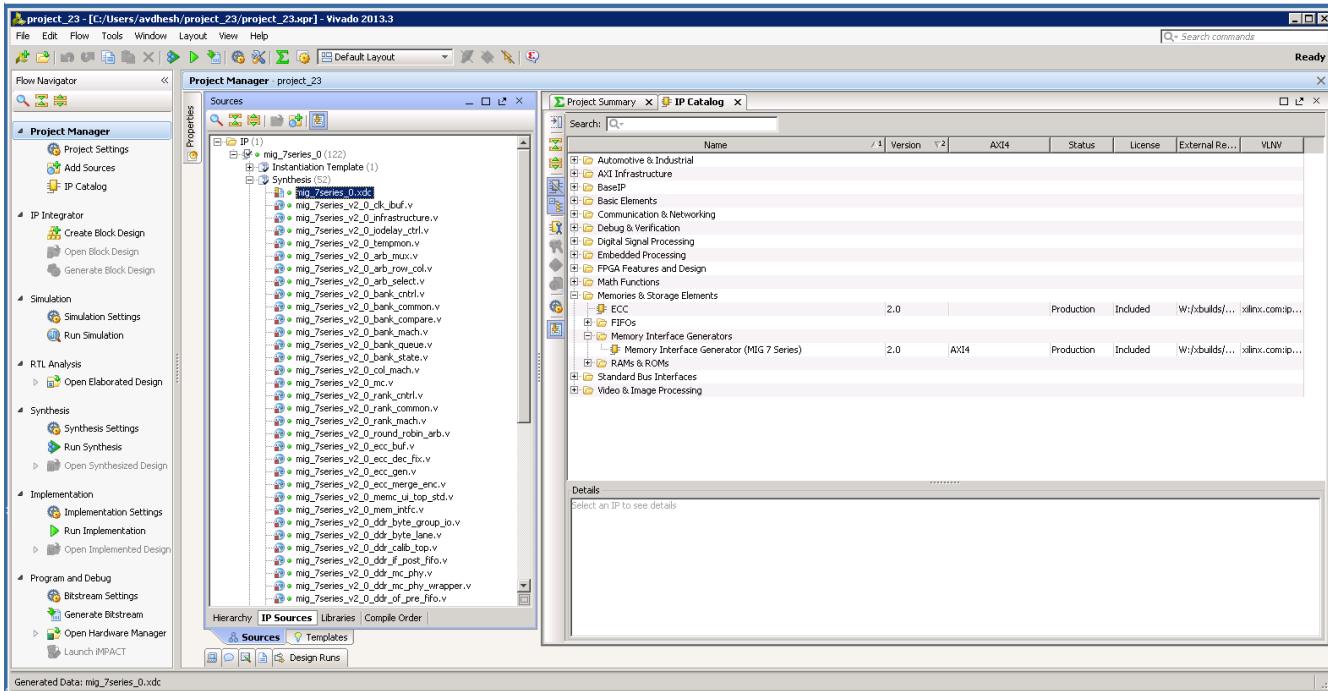


Figure 3-28:

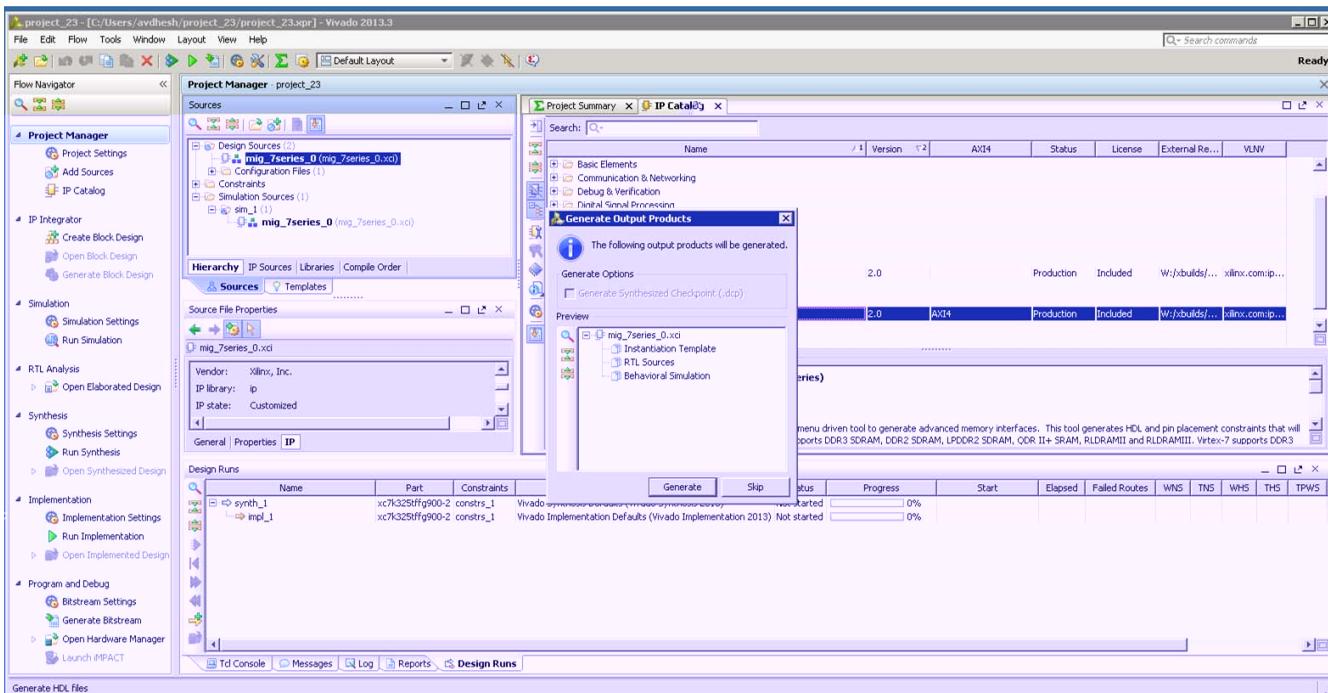
Design generation from MIG can be generated using the **Create Design** flow or the **Verify Pin Changes** and **Update Design** flows. There is no difference between the flow when generating the design from the MIG tool. Irrespective of the flow by which designs are generated from the MIG tool, the XCI file is added to the Vivado tool project. The implementation flow is the same for all scenarios because the flow depends on the XCI file added to the project.

6. All MIG generated user design RTL and XDC files are automatically added to the project. If files are modified and you wish to regenerate them, right-click the XCI file and select **Generate Output Products** (Figure 3-29).



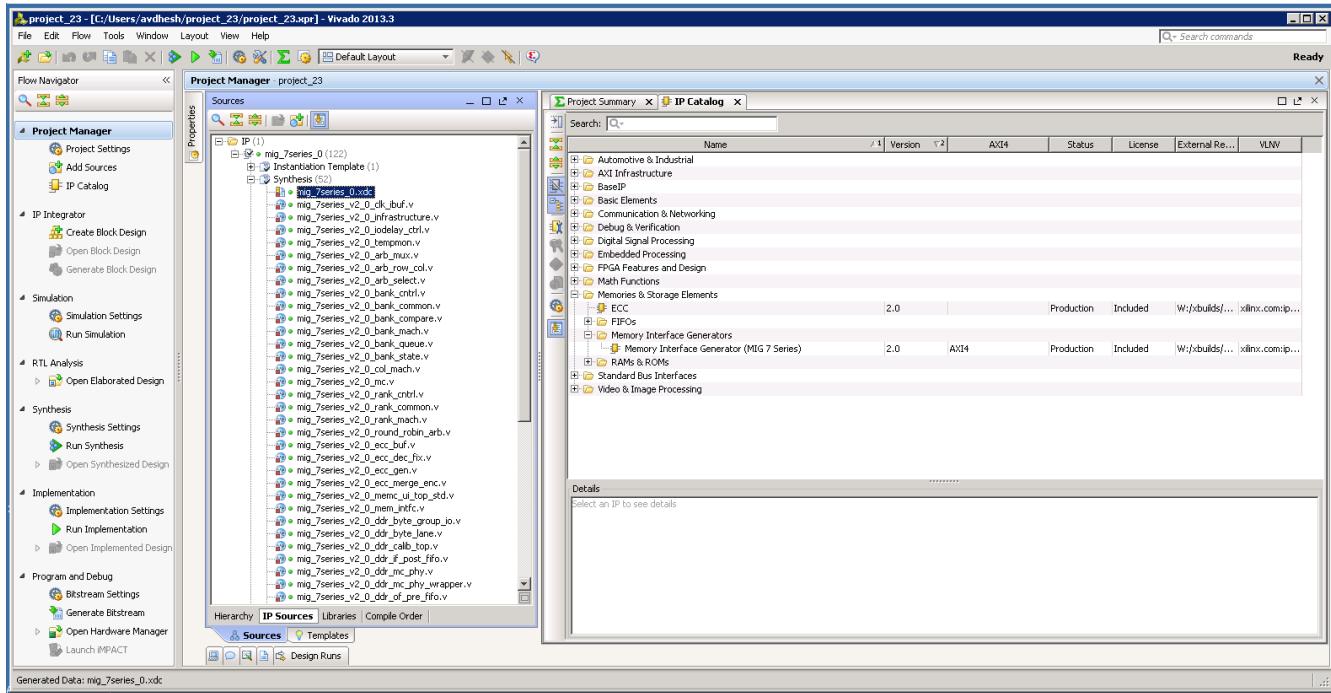
*Figure 3-29:*

7. Clicking Generate Output Products option brings up the Manage Outputs window (Figure 3-30).



*Figure 3-30:*

8. All user-design RTL files and constraints files (XDC files) can be viewed in the **Sources > Libraries** tab ([Figure 3-31](#)).



*Figure 3-31:*

9. The Vivado Design Suite supports **Open IP Example Design** flow. To create the example design using this flow right-click the IP in the **Source Window**, as shown in [Figure 3-32](#) and select.



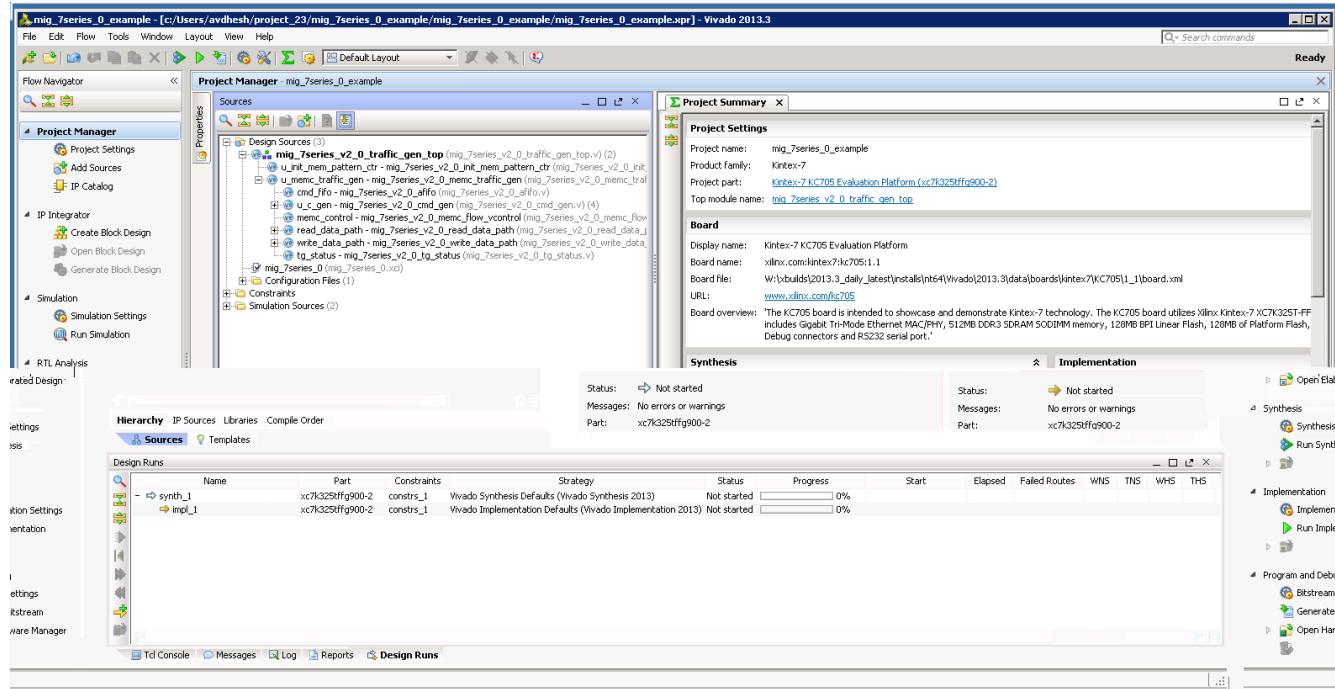


Figure 3-33:

- Click **Generate Bitstream** under **Project Manager > Program and Debug** to generate the BIT file for the generated design.

The `<project_directory>/<project_directory>.runs/ impl_1` directory includes all report files generated for the project after running the implementation. It is also possible to run the simulation in this project.

- Recustomization of the MIG IP core can be done by using the **Recustomize IP** option. It is not recommended to recustomize the IP in the `example_design` project. The correct solution is to close the `example_design` project, go back to original project and customize there. Right-click the XCI file and click **Recustomize IP** (Figure 3-34) to open the MIG GUI and regenerate the design with the preferred options.

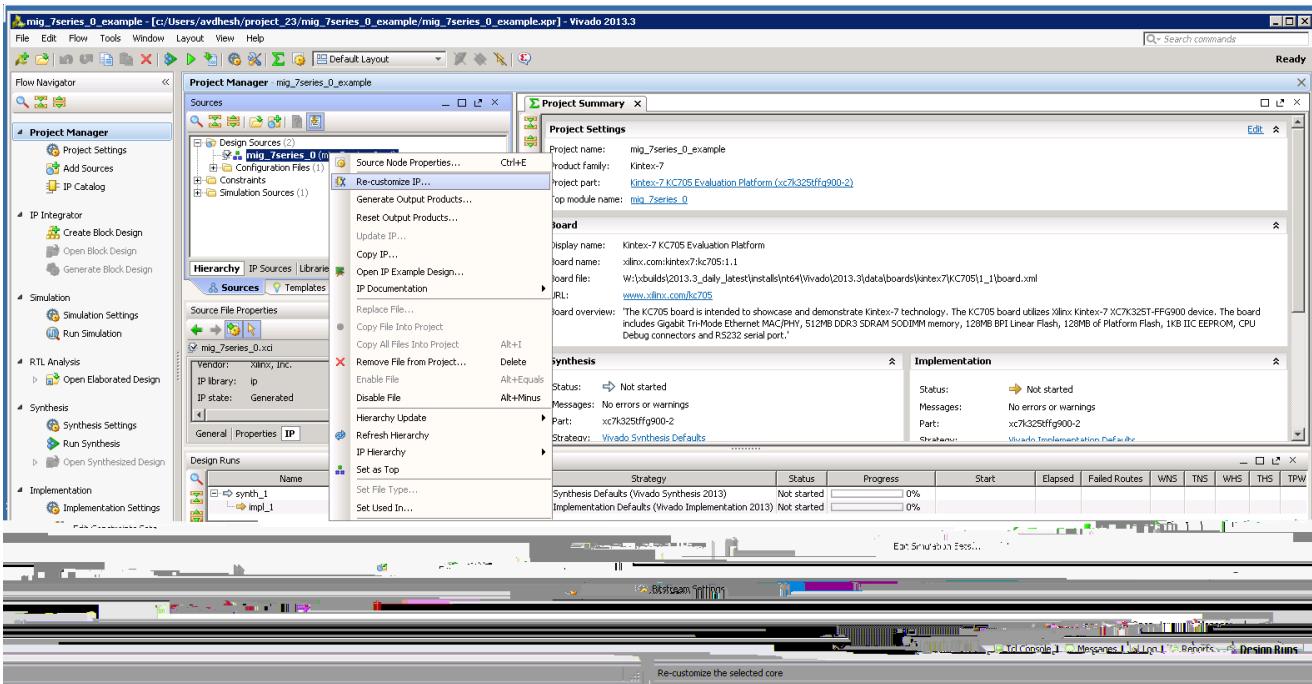


Figure 3-34:

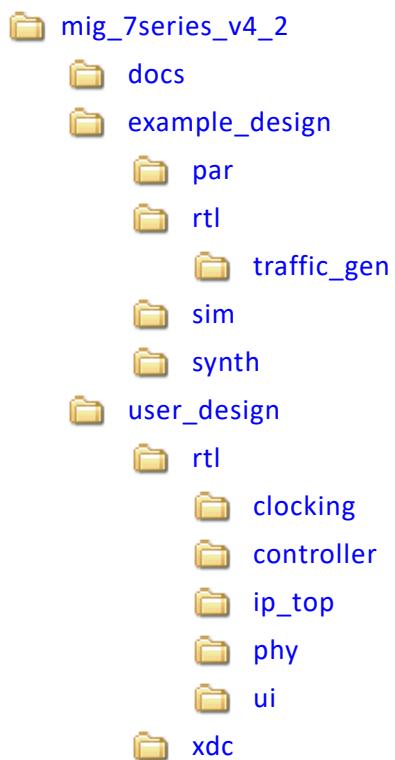
## Directory Structure and File Descriptions

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

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 389](#) of the MIG design creation flow.

The output directory structure of the selected Memory Controller (MC) design from the MIG tool is shown here. There are three folders created within the <component name> directory:

- **docs**
- **example\_design**
- **user\_design**



The 7 series FPGAs core directories and their associated files are listed in this section for Vivado implementations.

**<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.

Table 3-1 lists the files in the **example\_design/rtl** directory.

*Table 3-1:*

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:**

memc_traffic_gen.v	This is the top-level of the traffic generator.
cmd_gen.v	This is the command generator. This module provides independent control of generating the types of commands, addresses, and burst lengths.
cmd_prbs_gen.v	This is a pseudo-random binary sequence (PRBS) generator for generating PRBS commands, addresses, and burst lengths.
memc_flow_vcontrol.v	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	This is the top-level for the read datapath.
read_posted_fifo.v	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	This module generates timing control for reads and ready signals to mem_flow_vcontrol.v.
write_data_path.v	This is the top-level for the write datapath.
wr_data_g.v	This module generates timing control for writes and ready signals to mem_flow_vcontrol.v.
s7ven_data_gen.v	This module generates different data patterns.
a_fifo.v	This is a synchronous FIFO using LUT RAMs.
data_prbs_gen.v	This is a 32-bit linear feedback shift register (LFSR) for generating PRBS data patterns.
init_mem_pattern_ctr.v	This module generates flow control logic for the traffic generator.
traffic_gen_top.v	This module is the top-level of the traffic generator and comprises the memc_traffic_gen and init_mem_pattern_ctr modules.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of cmd\_gen in generated output is now mig\_7series\_v4\_2\_cmd\_gen.

**Table 3-3** lists the files in the `example_design/sim` directory.

**Table 3-3:**

ies_run.sh <sup>(1)</sup>	Linux Executable file for simulating the design using IES simulator.
vcs_run.sh <sup>(1)</sup>	Linux Executable file for simulating the design using VCS simulator.
readme.txt <sup>(1)</sup>	Contains the details and prerequisites for simulating the designs using Mentor Graphics Questa Advanced Simulator, IES, and VCS simulators.

Table 3-3:

(Cont'd)

sim_tb_top.v	This file is the simulation top-level file.

**Notes:**

1. The ies\_run.sh and vcs\_run.sh files are generated in the folder mig\_7series\_0\_ex/imports when the example design is created using **Open IP Example Design** for the design generated with **Component Name** entered in Vivado IDE as mig\_7series\_0.

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

The **user\_design** folder contains the following:

- **rtl** and **xdc** folders
- Top-level wrapper module **<component\_name>.v/vhd**
- Top-level modules **<component\_name>\_mig.v/vhd** and **<component\_name>\_mig\_sim.v/vhd**

The top-level wrapper file **<component\_name>.v/vhd** has an instantiation of top-level file **<component\_name>\_mig.v/vhd**. Top-level wrapper file has no parameter declarations and all the port declarations are of fixed width.

Top-level files **<component\_name>\_mig.v/vhd** and **<component\_name>\_mig\_sim.v/vhd** have the same module name as **<component\_name>\_mig**. These two files are same in all respects except that the file **<component\_name>\_mig\_sim.v/vhd** has parameter values set for simulation where calibration is in fast mode **viz., SIM\_BYPASS\_INIT\_CAL = "FAST"** etc.




---

**IMPORTANT:** The top-level file **<component\_name>\_mig.v/vhd** is used for design synthesis and implementation, whereas the top-level file **<component\_name>\_mig\_sim.v/vhd** is used in simulations.

---

The top-level wrapper file serves as an example for connecting the **user\_design** to the 7 series FPGA memory interface core.

**user\_design/rtl/controller**

Table 3-4 lists the files in the **user\_design/rtl/controller** directory.

Table 3-4:

rld_mc.v	This module implements the Memory Controller.

**Notes:**

1. All file names are prefixed with MIG version number. For example, for the MIG 4.2 release module name of rld\_mc in generated output is now mig\_7series\_v4\_2\_rld\_mc.

**user\_design/rtl/ui**

[Table 3-5](#) lists the files in the **user\_design/rtl/ui** directory.

*Table 3-5:*

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.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of rld\_ui\_top in generated output is now mig\_7series\_v4\_2\_rld\_ui\_top.

**user\_design/rtl/phy**

[Table 3-6](#) lists the files in the **user\_design/rtl/phy** directory.

*Table 3-6:*

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 memory ports.

Table 3-6:

(Cont'd)

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.
rld_phy_write_cal.v	This module contains the logic for performing write calibration.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of rld\_phy\_top in generated output is now mig\_7series\_v4\_2\_rld\_phy\_top.

**user\_design/rtl/xdc**

Table 3-7 lists the files in the **user\_design/rtl/xdc** directory.

Table 3-7:

<component name>.xdc	This file is the XDC for the core of the user design.

This feature verifies the input XDC for bank selections, byte selections, and pin allocation. It also generates errors and warnings in a separate dialog box when you click **Validate** on the page. This feature is useful to verify the XDC for any pinout changes made after the design is generated from the MIG tool. You must load the MIG generated **.prj** file, the original **.prj** file without any modifications, and the XDC that needs to be verified. In the Vivado IP catalog, 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 XDC is not sufficient; it is mandatory to proceed with design generation to get the XDC with updated clock and phaser related constraints and RTL top-level module for various updated Map parameters.

The Update Design feature is required in the following scenarios:

- A pinout is generated using an older version of MIG and the design is to be revised to the current version of MIG. In MIG the pinout allocation algorithms have been changed for certain MIG designs.
- A pinout is generated independent of MIG or is modified after the design is generated. When a design is generated from MIG, the XDC and HDL code are generated with the correct constraints.

Here are the rules verified from the input XDC:

- If a pin is allocated to more than one signal, the tool reports an error. Further verification is not done if the XDC 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.
  - $V_{REF}$  I/Os should be used as GPIOs when an internal  $V_{REF}$  is used or if there are no input and output 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.
  - Write clocks (DK/DK#) should be allocated to the DQS I/O pair.
  - Read clocks (QK/QK#) should be allocated to the MRCC pins for RLDRAM II and should be allocated to DQS I/O pair for RLDRAM 3.
  - Data (DQ) pins should not be allocated to DQS N pin.
  - An FPGA byte lane should not contain pins related to two different strobe sets.
  - $V_{REF}$  I/O can be used only when the internal  $V_{REF}$  is chosen.
- Verified address pin rules:
  - Address signals cannot mix with data bytes.
  - 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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  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, you need to check whether the reference voltage pins are unallocated in the bank or the internal V<sub>REF</sub> is used.
- Status signals:
  - The **sys\_rst** signal should be allocated in the bank where the V<sub>REF</sub> I/O is unallocated or the internal V<sub>REF</sub> 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.

## Overview

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

## Implementing the Example Design

For more information on using an IP example design, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 7\]](#).

## 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 3-35](#).

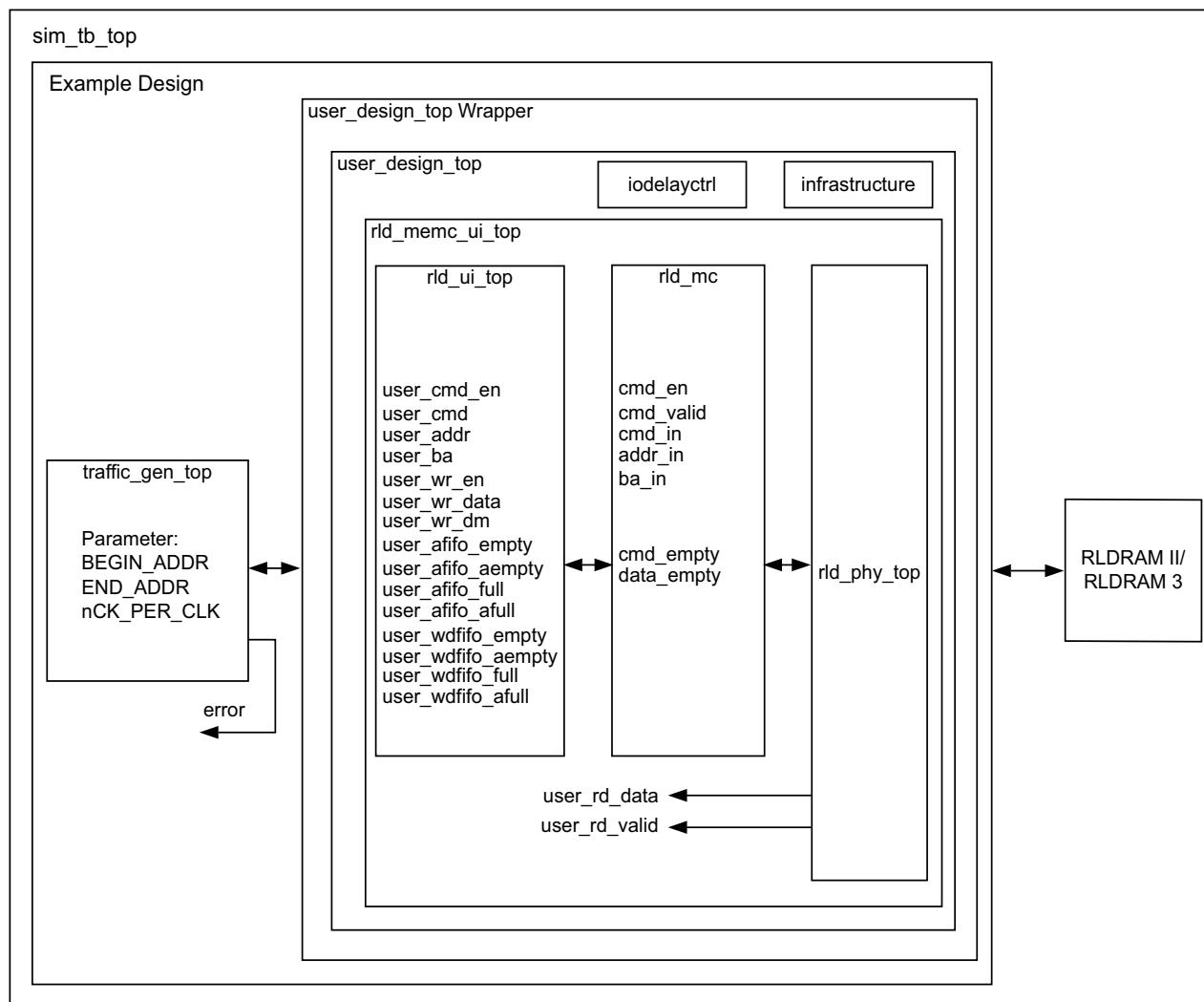


Figure 3-35:

Figure 3-36 shows the simulation result of a simple read and write transaction between the **tb\_top** and **memc\_intf** modules.

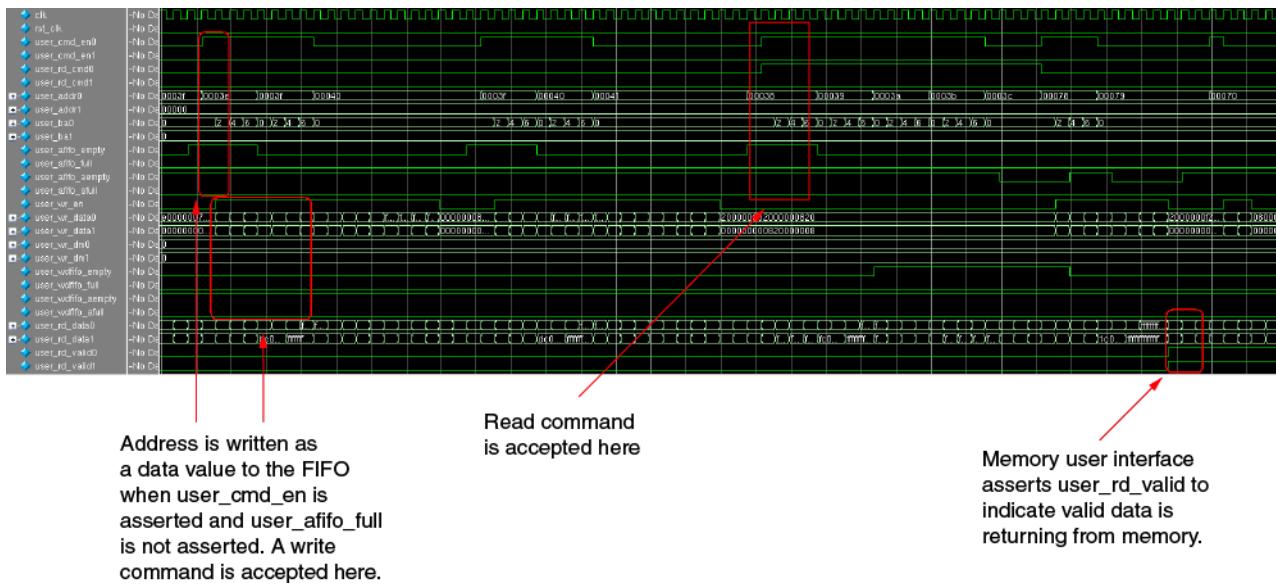


Figure 3-36:

### 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.

You 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 Vivado logic analyzer feature.

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) and addresses are determined by PRBS generator logic in the traffic generator module.

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

The 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 3-8](#) describes these parameters.

*Table 3-8:*

FAMILY	Indicates the family type.	"VIRTEX7"
MEMORY_TYPE	Indicate the Memory Controller type.	Current support is DDR2 SDRAM, DDR3 SDRAM, QDR II+ SRAM, and RLDRAM II.
nCK_PER_CLK	This is the Memory Controller clock to DRAM clock ratio. This parameter should <b>not</b> be changed.	RLDRAM II: 2 RLDRAM 3: 4
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.	$2 \times nCK\_PER\_CLK \times NUM\_DQ\_PINS$
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.	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.

Table 3-8:

(Cont'd)

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.
PRBS_SADDR_MASK_POS	Sets the 32-bit OR MASK position.	This parameter is used with the PRBS address generator to shift random addresses up into the port address space. The START_ADDRESS value is ORed with the PRBS address for bit positions that have a 1 in this mask
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 increased 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 3-8:

(Cont'd)

		Valid settings for this parameter are: <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>Ox1: FIXED – 32 bits of fixed_data.</li> <li>Ox2: ADDRESS – 32 bits address as data.</li> <li>Ox3: HAMMER</li> <li>Ox4: SIMPLE8 – Simple 8 data pattern that repeats every 8 words.</li> <li>Ox5: WALKING1s – Walking 1s are on the DQ pins.</li> <li>Ox6: WALKING0s – Walking 0s are on the DQ pins.</li> <li>Ox7: PRBS – A 32-stage LFSR generates random data. This mode only works with either a PRBS address or a SEQUENTIAL address pattern.</li> <li>Ox9: SLOW HAMMER – This is the slow MHz hammer data pattern.</li> <li>OxA: PHY_CALIB pattern – OxFF, 00, AA, 55, 55, AA, 99, 66. This mode only generates READ commands at address zero.</li> </ul> </li> </ul>
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.	Valid values: 0 to 32.
CMDS_GAP_DELAY	This parameter allows pause delay between each user burst command.	This parameter only applies to the Hammer pattern. Valid settings for this parameter are 0 to NUM_DQ_PINS. When value = NUM_DQ_PINS, all DQ pins have the same Hammer pattern.
SEL_VICTIM_LINE	Select a victim DQ line whose state is always at logic High.	

Table 3-8:

(Cont'd)

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.
----------	---	--

**Notes:**

1. 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 you to modify the command pattern in real-time using the Vivado logic analyzer feature virtual I/O (VIO) core.

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.

---

[Figure 3-37](#) shows a high-level block diagram of the RLDRAM II and RLDRAM 3 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.

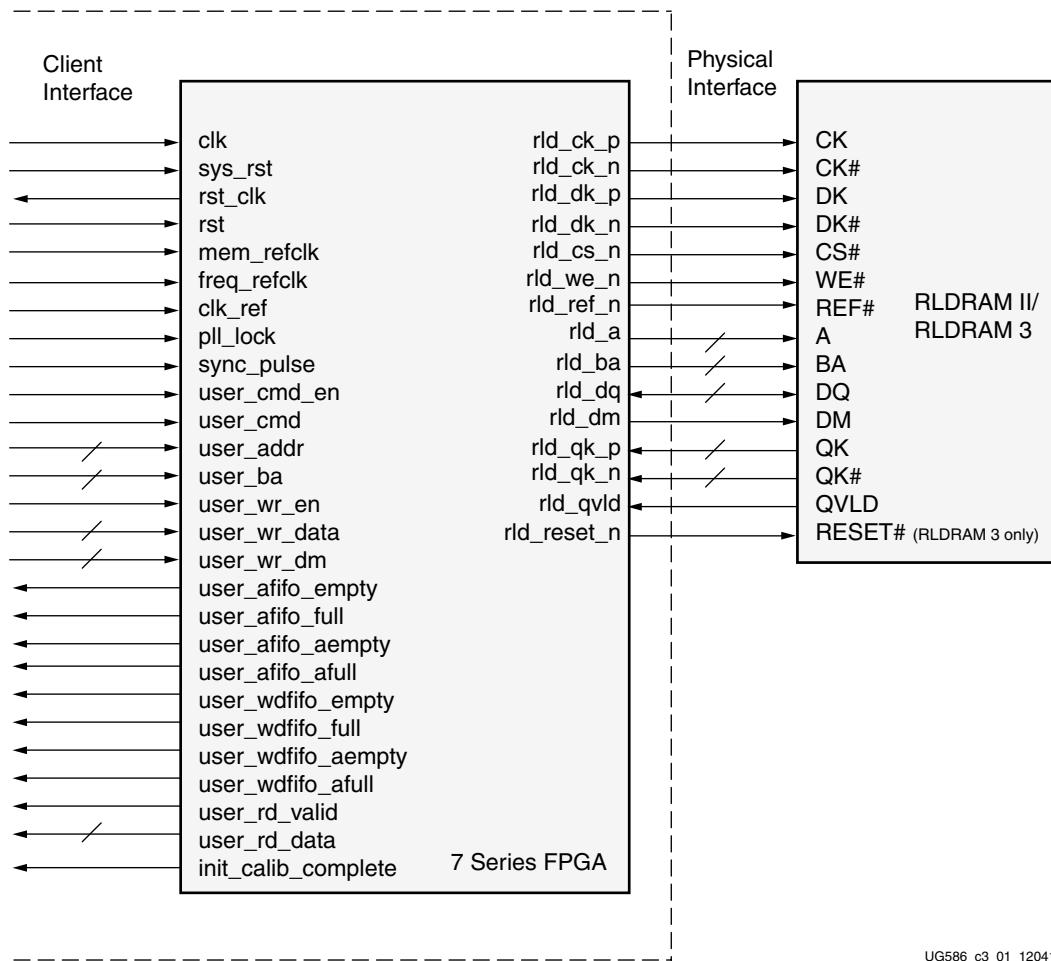
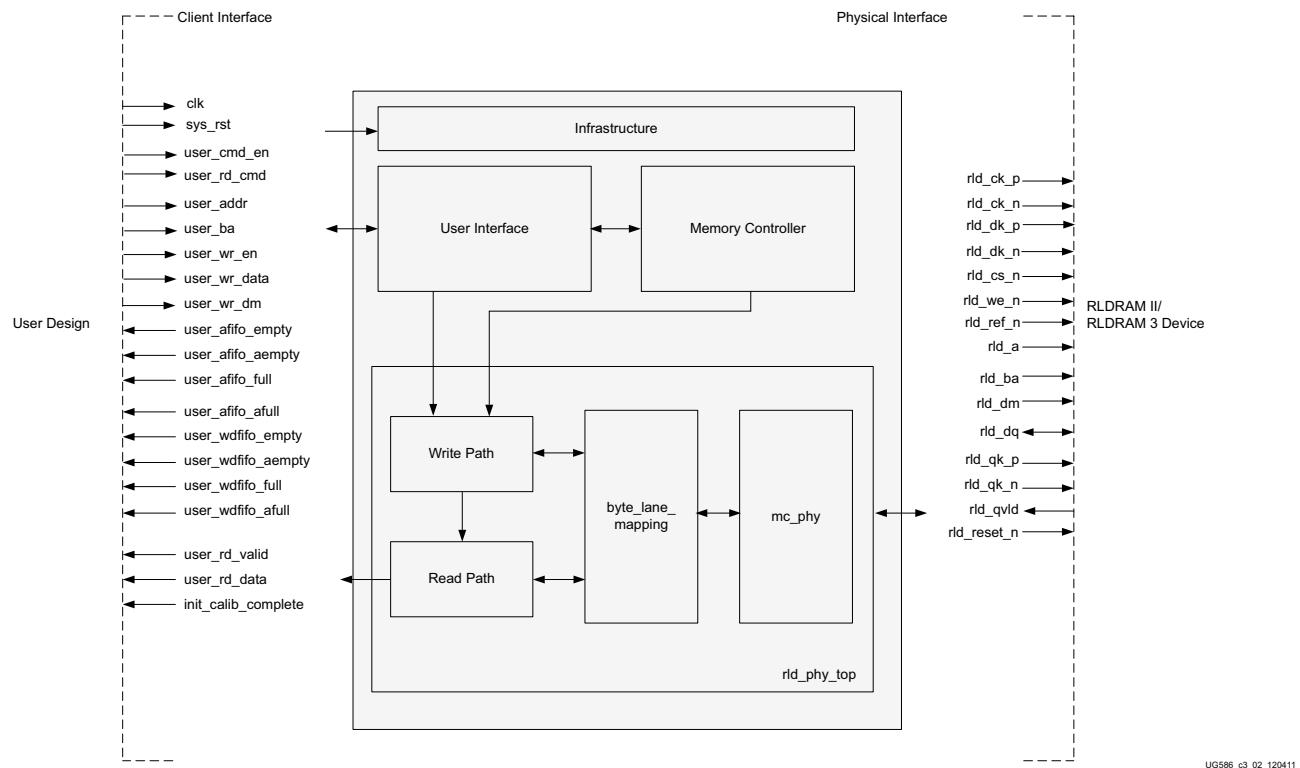


Figure 3-37:

The core is composed of these elements, as shown in [Figure 3-38](#):

- Client Interface
- Memory Controller
- Physical Interface
- Read Path
- Write Path



*Figure 3-38:*

The client interface (also known as the user interface) uses a simple protocol based entirely on SDR signals to make read and write requests. For more details describing this protocol, see the [Client Interface](#) section.

The Memory Controller takes commands from the user interface and adheres to the protocol requirements of the RLDRAm II/RLDRAM 3 device. For more information, see the [Memory Controller](#) section.

The physical interface generates the proper timing relationships and DDR signaling to communicate with the external memory device, while conforming to the RLDRAm II/RLDRAM 3 protocol and timing requirements. For more details, see the [Physical Interface](#) section.

Within the PHY, logic is broken up into read and write paths. The write path generates the RLDRAM II/RLDRAM 3 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 you with a corresponding valid signal. For more details describing this process, see the [Calibration](#) section.

The client interface connects the 7 series FPGA user design to the RLDRAM II/RLDRAM 3 memory solutions core to simplify interactions between you and the external memory device.

### *Command Request Signals*

The client interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in [Table 3-9](#).

*Table 3-9:*

user_cmd_en	Input	Command Enable. This signal issues a read or write request and indicates that the corresponding command signals are valid.
user_cmd[ $2 \times \text{CMD\_PER\_CLK} - 1:0$ ]	Input	Command. This signal issues a read, write, or NOP request. When user_cmd_en is asserted: 2'b00 = Write Command 2'b01 = Read Command 2'b10 = NOP 2'b11 = NOP  The NOP command is useful when more than one command per clock cycle must be provided to the Memory Controller yet not all command slots are required in a given clock cycle. The Memory Controller acts on the other commands provided and ignore the NOP command. NOP is not supported when CMD_PER_CLK == 1. CMD_PER_CLK is a top-level parameter used to determine how many memory commands are provided to the controller per FPGA logic clock cycle, it depends on nCK_PER_CLK and the burst length (see <a href="#">Figure 3-39</a> )
user_addr[ $\text{CMD\_PER\_CLK} \times \text{ADDR\_WIDTH} - 1:0$ ]	Input	Command Address. This is the address to use for a command request. It is valid when user_cmd_en is asserted.
user_ba[ $\text{CMD\_PER\_CLK} \times \text{BANK\_WIDTH} - 1:0$ ]	Input	Command Bank Address. This is the address to use for a write request. It is valid when user_cmd_en is asserted.

Table 3-9:

(Cont'd)

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_data[ $2 \times nCK\_PER\_CLK \times DATA\_WIDTH - 1:0$ ]	Input	Write Data. 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_dm[ $2 \times nCK\_PER\_CLK \times DM\_WIDTH - 1:0$ ]	Input	Write Data Mask. When active-High, the write data for a given selected device is masked and not written to the memory. It is valid when user_wr_en is asserted.
user_afifo_empty	Output	Address FIFO empty. If asserted, the command buffer is empty.
user_wdfifo_empty	Output	Write Data FIFO empty. If asserted, the write data buffer is empty.
user_afifo_full	Output	Address FIFO 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_valid[nCK_PER_CLK - 1:0]	Output	Read Valid. This signal indicates that data read back from memory is available on user_rd_data and should be sampled.
user_rd_data[ $2 \times nCK\_PER\_CLK \times DATA\_WIDTH - 1:0$ ]	Output	Read Data. 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.
mem_ck_lock_complete	Output	Memory CK Lock Done. The system should be kept in a quiet state until assertion of mem_ck_lock_complete to ensure minimal noise on the CK being driven to the memory.

## *Interfacing with the Core through the Client Interface*

The width of certain client interface signals is dependent on the system clock frequency and the burst length. This allows the client to send multiple commands per FPGA logic clock cycle as might be required for certain configurations.

Figure 3-39 shows the **user\_cmd** signal and how it is made up of multiple commands depending on the configuration.

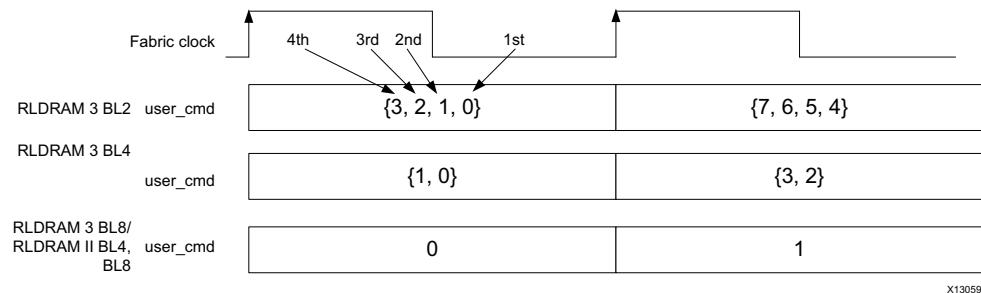


Figure 3-39:

As shown in Figure 3-39, four command slots are present in a single user interface clock cycle for BL2. Similarly, two command slots are present in a single user interface clock cycle for BL4. These command slots are serviced sequentially and the return data for read commands are presented at the user interface in the same sequence. Note that the read data might not be available in the same slot as that of its read command. The slot of a read data is determined by the timing requirements of the controller and its command slot. One such example is mentioned in the following BL2 design configuration.

Assume that the following set of commands is presented at the user interface for a given user interface cycle.

Table 3-10:

0	RDO
1	NOP
2	RD1
3	NOP

It is not guaranteed that the read data appears in {DATA0, NOP, DATA1, NOP} order. It might also appear in {NOP, DATA0, NOP, DATA1} or {NOP, NOP, DATA0, DATA1} etc. orders. In any case, the sequence of the commands are maintained.

The client interface protocol is shown in Figure 3-40 for the RLDRAm II four-word burst architecture.

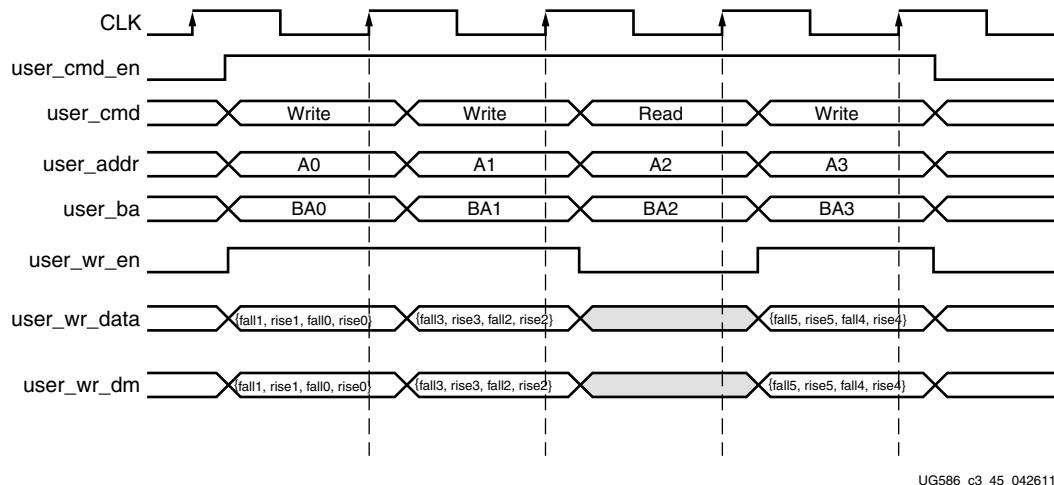


Figure 3-40:

The client interface protocol for the RLDRAM 3 four-word burst architecture is shown in Figure 3-41.

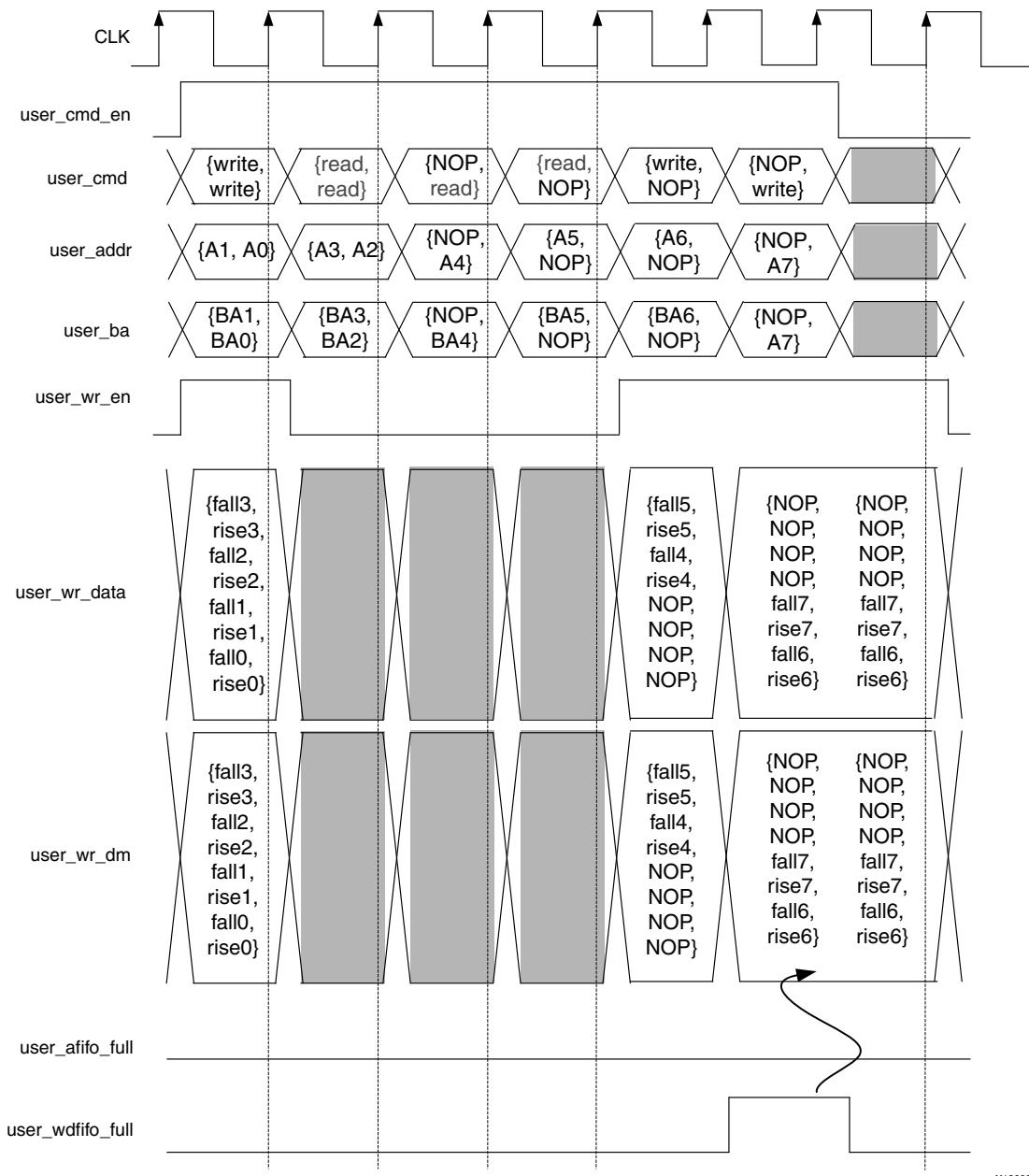


Figure 3-41:

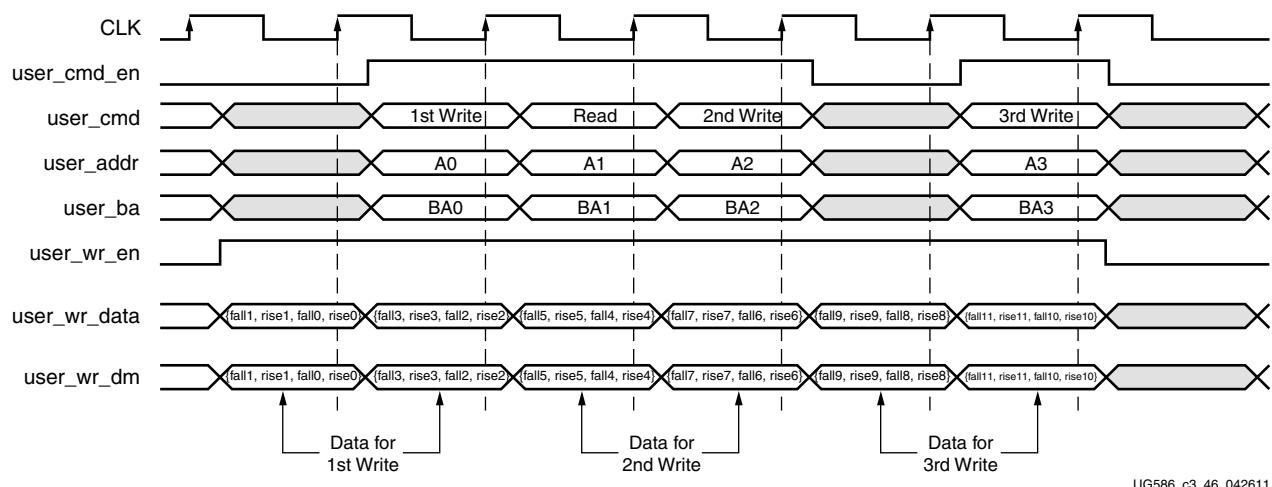
Before any requests can be accepted, the **ui\_clk\_sync\_rst** signal must be deasserted Low. After the **ui\_clk\_sync\_rst** signal is deasserted, the user interface FIFOs can accept commands and data for storage. The **init\_calib\_complete** signal is asserted after the memory initialization procedure and PHY calibration are complete, and the core can begin to service client requests.

A command request is issued by asserting **user\_cmd\_en** as a single cycle pulse. At this time, the **user\_cmd**, **user\_addr**, and **user\_ba** signals must be valid. To issue a read request, **user\_cmd** is set to 2'b01, while for a write request, **user\_cmd** is set to 2'b00. 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\_data** and **user\_wr\_dm**.



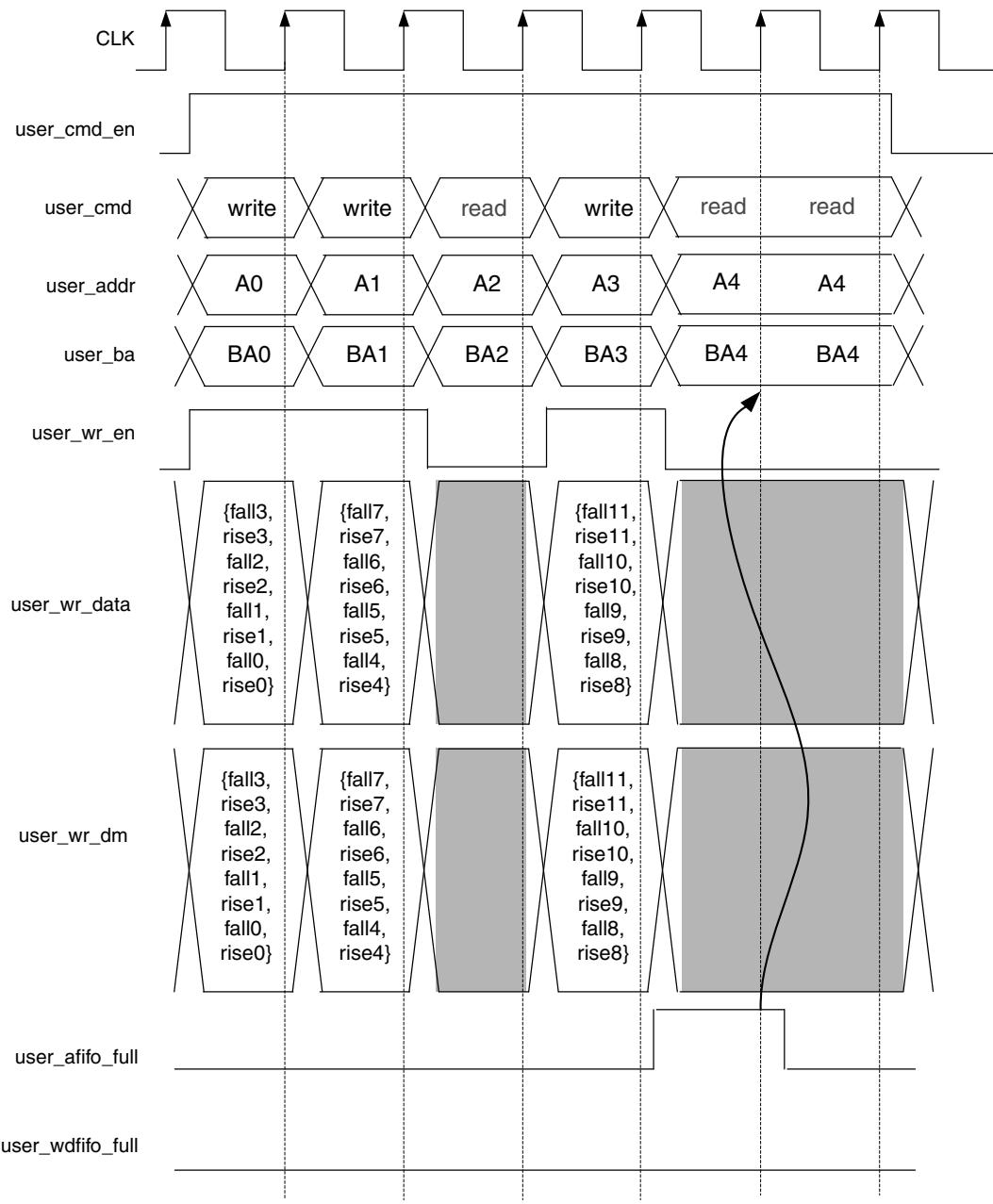
**IMPORTANT:** Both write and read commands in the same **user\_cmd** cycle is not allowed.

For RLDRAM II and eight-word burst architecture, an extra cycle of data is required for a given write command, as shown in [Figure 3-42](#). Any gaps in the command flow required can be filled with read commands, if desired.



*Figure 3-42:*

The client interface protocol for the RLDRAM 3 eight-word burst architecture is shown in [Figure 3-43](#).

*Figure 3-43:*

When a read command is issued some time later (based on the configuration and latency of the system), the **user\_rd\_valid[0]** signal is asserted, indicating that **user\_rd\_data** is now valid, while **user\_rd\_valid[1]** is asserted indicating that **user\_rd\_data** is valid, as shown in [Figure 3-44](#). The read data should be sampled on the same cycle that **user\_rd\_valid[0]** and **user\_rd\_valid[1]** are asserted because the core does not buffer returning data. If desired, you can add this functionality.

The Memory Controller only puts commands on certain slots to the PHY such that the **user\_rd\_valid** signals are all asserted together and return the full width of data, but the extra **user\_rd\_valid** signals are provided in case of controller modifications.

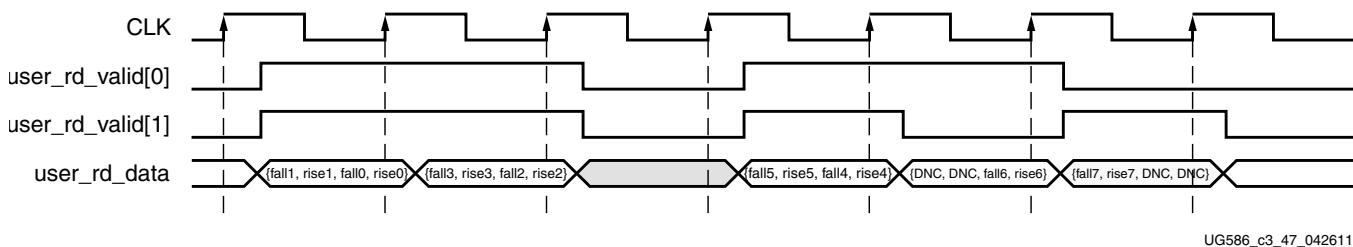


Figure 3-44:

The PHY design requires that an MMCM and 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 MMCM and one PLL is required for the PHY. The MMCM and PLL 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:

- **Memory Reference Clock** – The memory reference clock is required to be at the same frequency as that of the RLDRAM II/RLDRAM 3 memory interface clock.
- **Frequency Reference Clock** – The frequency reference clock must be equal to the memory clock frequency for frequencies  $\geq$  400 MHz and 2x the memory clock frequency for frequencies  $<$  400 MHz such that it meets the reference range requirement of 400 MHz to 1,066 MHz.
- **Phase Reference Clock from the PLL** – The phase reference clock is used in the read banks, and is generated using the memory read clock (QK/QK#) routed internally and provided to the Phaser logic to assist with data capture.

Figure 3-45 shows the clocking architecture.

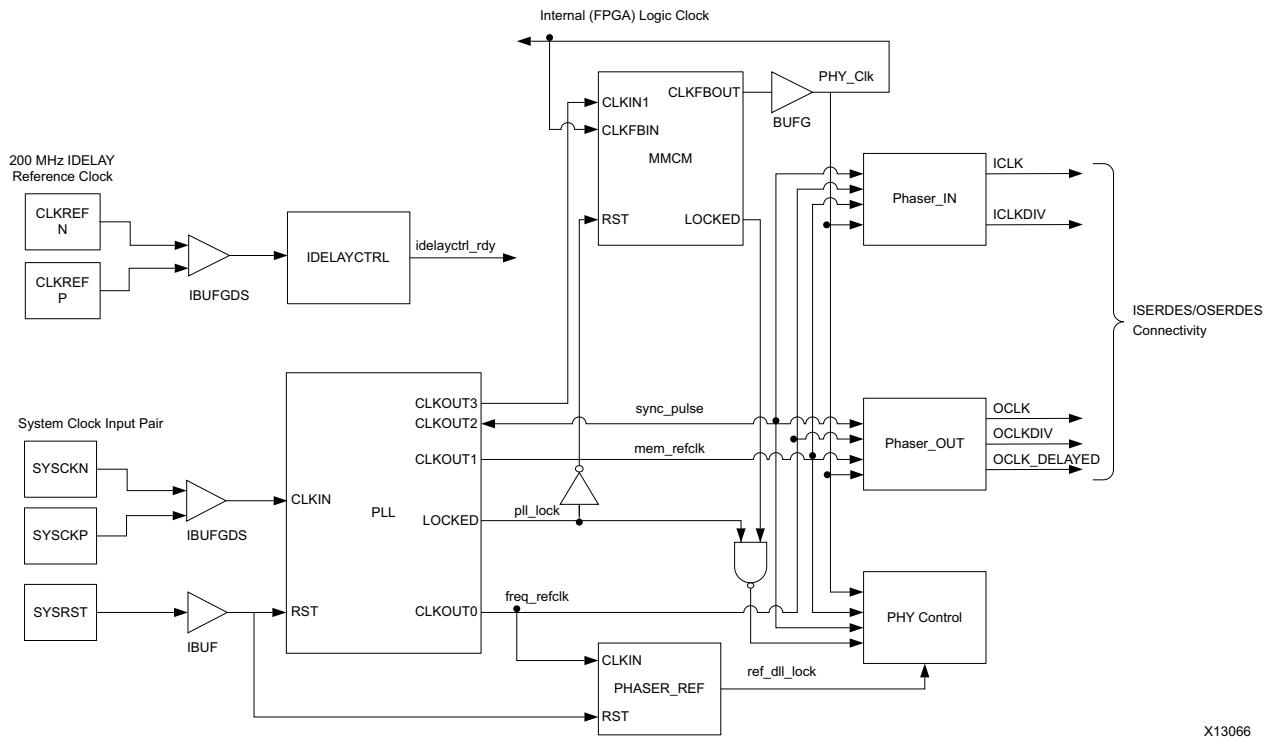


Figure 3-45:

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 1:1 ratio is not required. The PLL input divider (D) can be any value listed in the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] 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 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 467](#).

The internal FPGA logic clock generated by the PLL is clocked by a global clocking resource at half the frequency of the RDRAM II memory frequency and a quarter of the frequency of the RLDRAM 3 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 3-11](#) lists the signals used in the infrastructure module that provides the necessary clocks and reset signals required in the design.

*Table 3-11:*

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 FPGA logic 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–1,066 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
rstdivo	Output	Reset output synchronized to internal FPGA logic half-frequency clock.
rst_phaser_ref	Output	Reset for the Phaser in the Physical Layer.

The physical interface is the connection from the FPGA memory interface solution to an external RLDRAM II/RLDRAM 3 device. The I/O signals for this interface are defined in [Table 3-12](#). These signals can be directly connected to the corresponding signals on the RLDRAM II/RLDRAM 3 device.

*Table 3-12:*

rld_ck_p	Output	System Clock CK. This is the address/command clock to the memory device.
rld_ck_n	Output	System Clock CK#. This is the inverted system clock to the memory device.
rld_dk_p	InOut	Write Clock DK. This is the write clock to the memory device.
rld_dk_n	InOut	Write Clock DK#. This is the inverted write clock to the memory device.
rld_a	Output	Address. This is the address supplied for memory operations.
rld_ba	Output	Bank Address. This is the bank address supplied for memory operations.
rld_cs_n	Output	Chip Select CS#. This is the active-Low chip select control signal for the memory.
rld_we_n	Output	Write Enable WE#. This is the active-Low write enable control signal for the memory.
rld_ref_n	Output	Refresh REF#. This is the active-Low refresh control signal for the memory.
rld_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.

Figure 3-46 shows the timing diagram for a typical RLDRAM II 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\_dmo/1** signals are latched for the given write command. The timing diagram shows the sequence of events: **cal\_done** → **cs0** → **we0** → **addr0** → **ba0** → **ref0** → **wr\_en0** → **wr\_data0** → **wr\_dmo**.

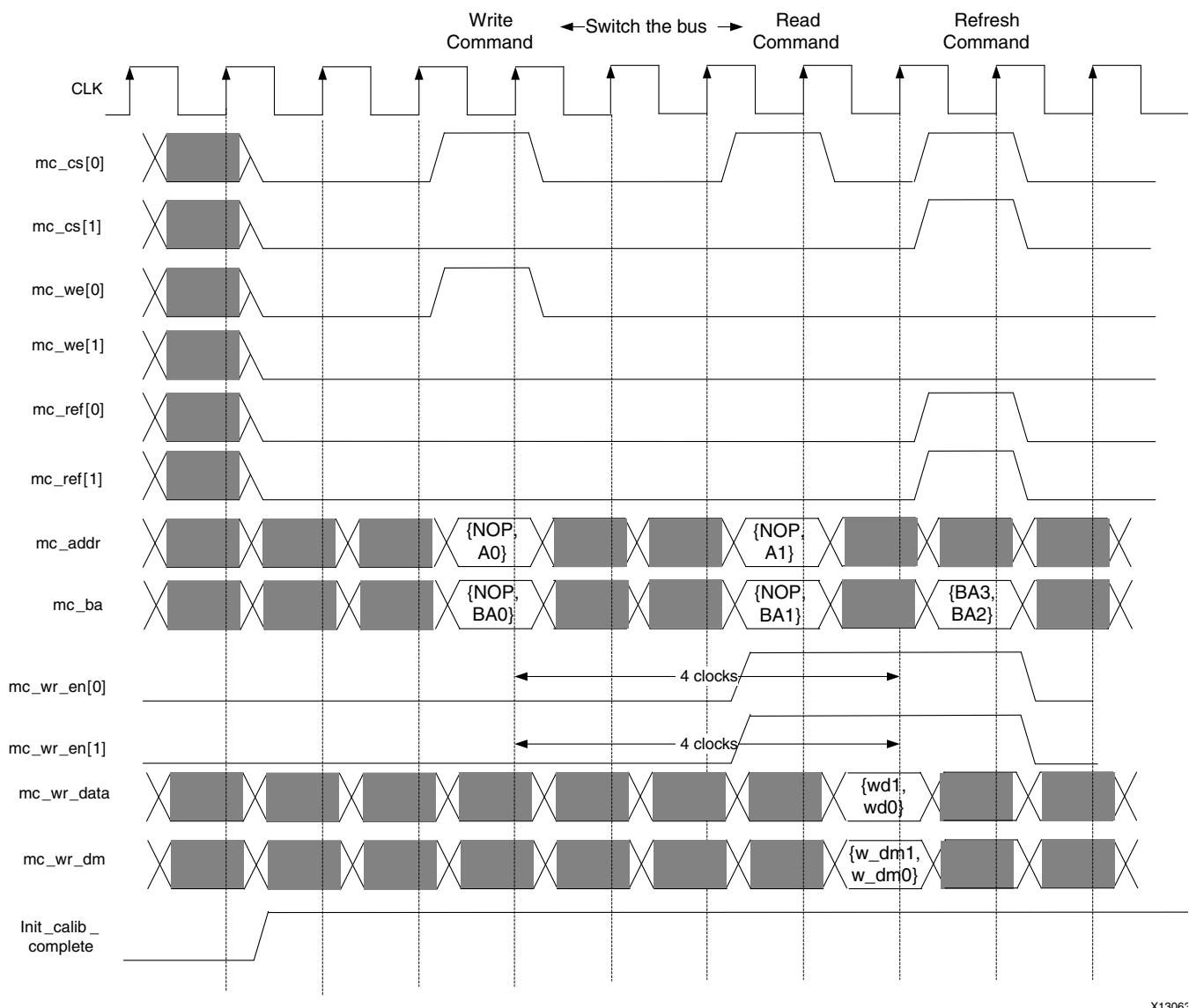


Figure 3-46:

The controller sends the **wr\_en** signals and data at the necessary time based on the configuration setting. This time changes depending on the configuration. [Table 3-13](#) details when the **wr\_en** 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 3-13:

ON	1	3
	2	4

Table 3-13:

(Cont'd)

	3	5
OFF	1	2
	2	3
	3	4 (1)

**Notes:**

1. Shown in [Figure 3-44](#).

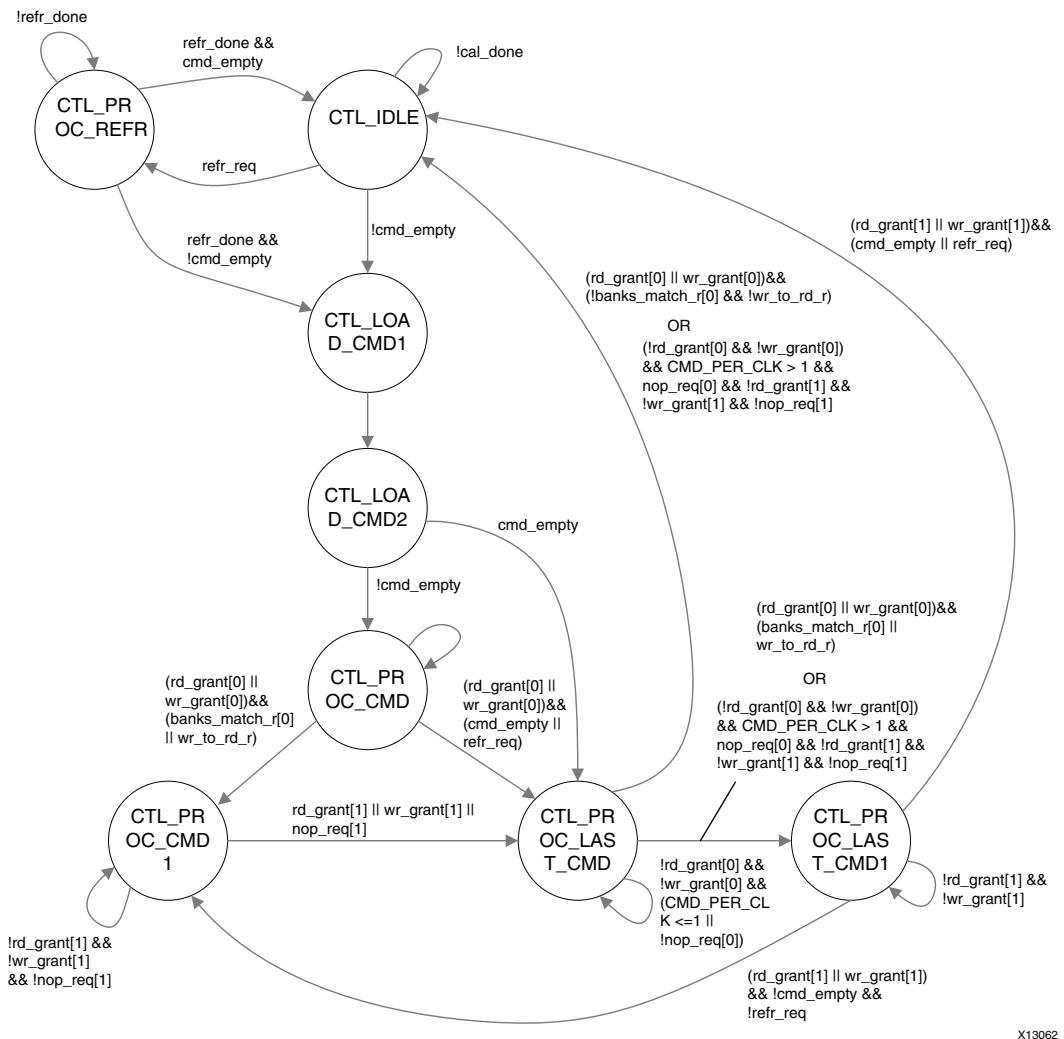
The **wr\_en** signals are required to be asserted an extra clock cycle before the first **wr\_en** signal is asserted, and held for an extra clock cycle after deassertion. This ensures that the shared bus has time to change from read to write and from write to read. The physical layer has a requirement 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.

The Memory Controller enforces the RLDRAM II/RLDRAM 3 access requirements and interfaces with the PHY. The controller processes commands in order, so the rank 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.

Figure 3-47 shows the state machine logic for the controller.



*Figure 3-47:*

## *PHY Architecture*

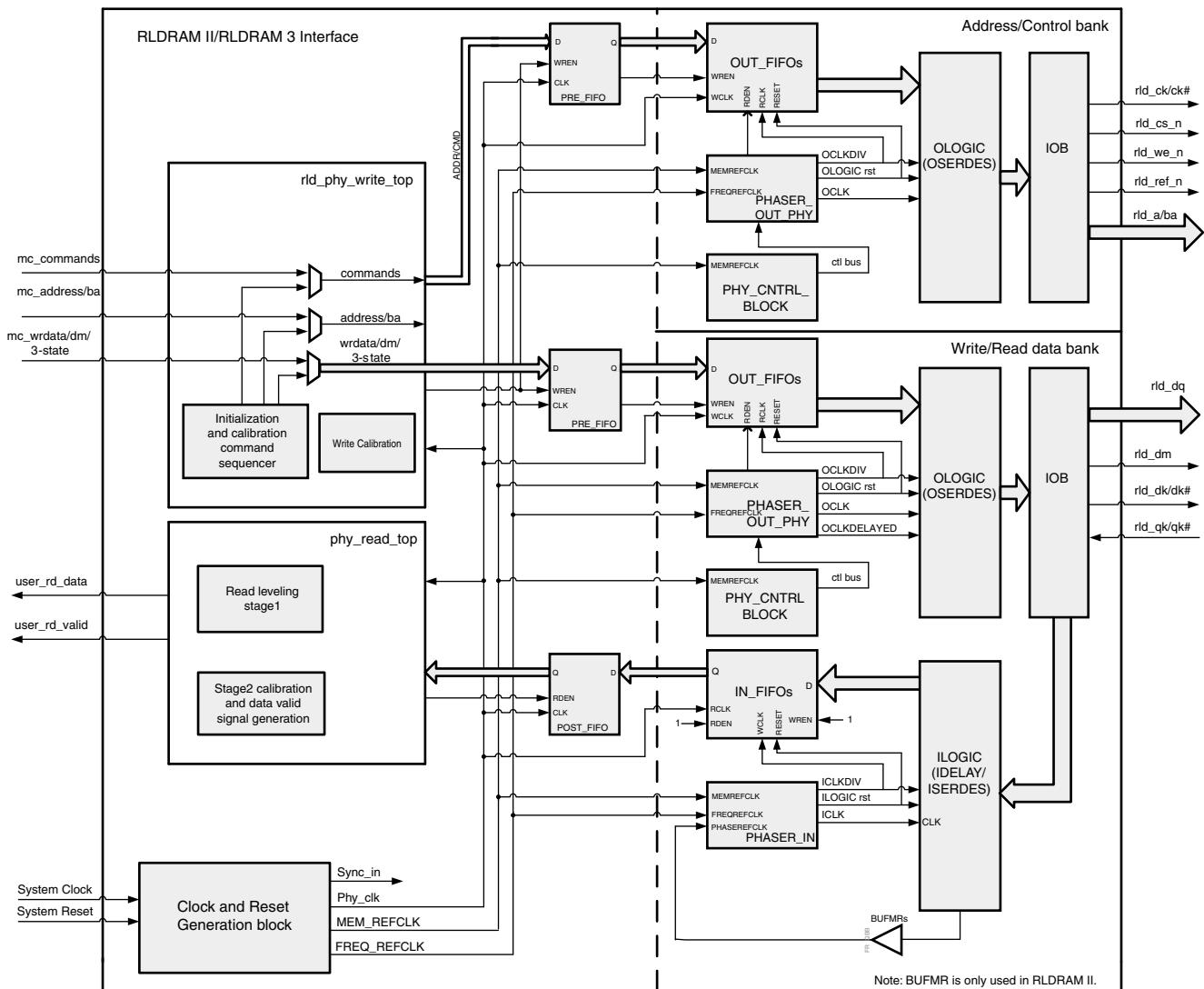
The PHY consists of dedicated blocks and soft calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high-performance physical layers.

Some of the dedicated blocks that are used in the RLDRAM II/RLDRAM 3 PHY and their features are described as follows:

- 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 multi-stage 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 or four-deep FIFOs available in each byte group and serve to transfer data from the FPGA logic 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 FPGA logic.

The [Pinout Requirements, page 468](#) explains the rules that need to be followed when placing the memory interface signals inside the byte groups.



UG586\_c3\_04\_051811

Figure 3-48:

**Note:** The overall read latency of the MIG 7 series RLDRAm II/RLDRAm 3 core is dependent on how the Memory Controller is configured, but most critically on the target traffic/access pattern and the number of commands already in the pipeline before the read command is issued. Read latency is measured from the point where the read command is accepted by the user or native interface. Simulation should be run to analyze read latency.

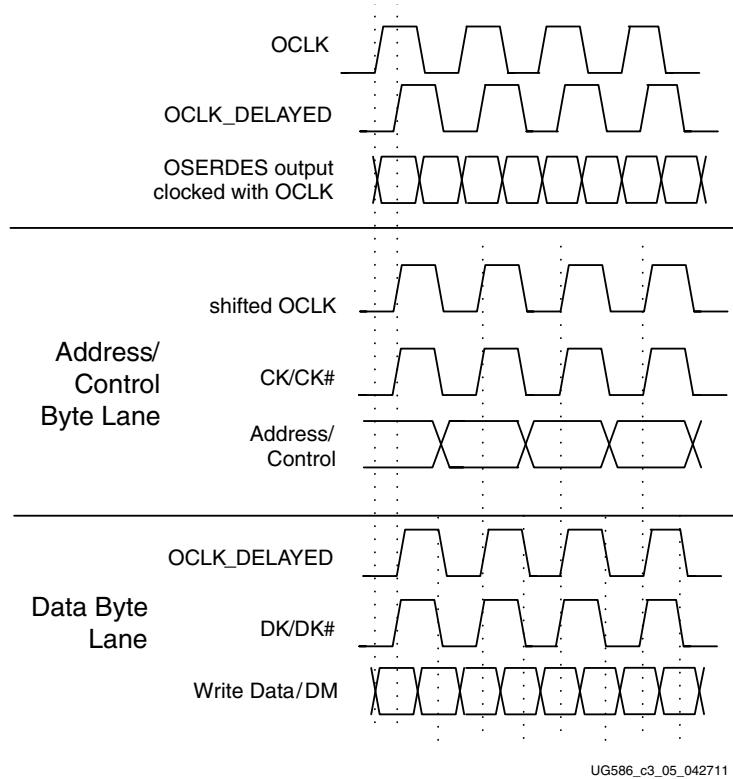
The write path to the RLDRAM II/RLDRAM 3 includes the address, data, and control signals necessary to execute any memory operation. The control strobes `rld_cs_n`, `rld_we_n`, `rld_ref_n`, and `rld_reset_n` (RLDRAM 3 only), including addresses `rld_a` and `rld_ba` to the memory all use SDR formatting. The write data values `rld_dq` and `rld_dm` also use DDR formatting to achieve the required two/four/eight-word burst within the given clock periods.

## ***Output Architecture***

The output path of the RLDRAM II/RLDRAM 3 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 an adjustable phase-shifted output with respect to the byte clock (OCLK) and is used to generate the write clock (DK/DK#) to the memory.

Figure 3-49 shows the alignment of the various clocks and how they are used to generate the necessary signal alignment.



UG586\_c3\_05\_042711

*Figure 3-49:*

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 depending on if calibration must be done on the write datapath either the address/control byte lanes are shifted or data byte lanes are shifted, to properly align the memory clock CK and the write clock DK. For certain frequencies a one-time calibration is performed for OCLK\_DELAYED to ensure reliable write operations. For details, see the [Calibration](#) section.

The OUT\_FIFO serves as a temporary buffer to convert the write data from the FPGA logic 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 FPGA logic 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 3-50](#). The PHY Control block is used to ensure proper start-up of all PHASER\_OUT\_PHY blocks used in the interface as well as to control the 3-state timing for RLDRAM 3 operation.

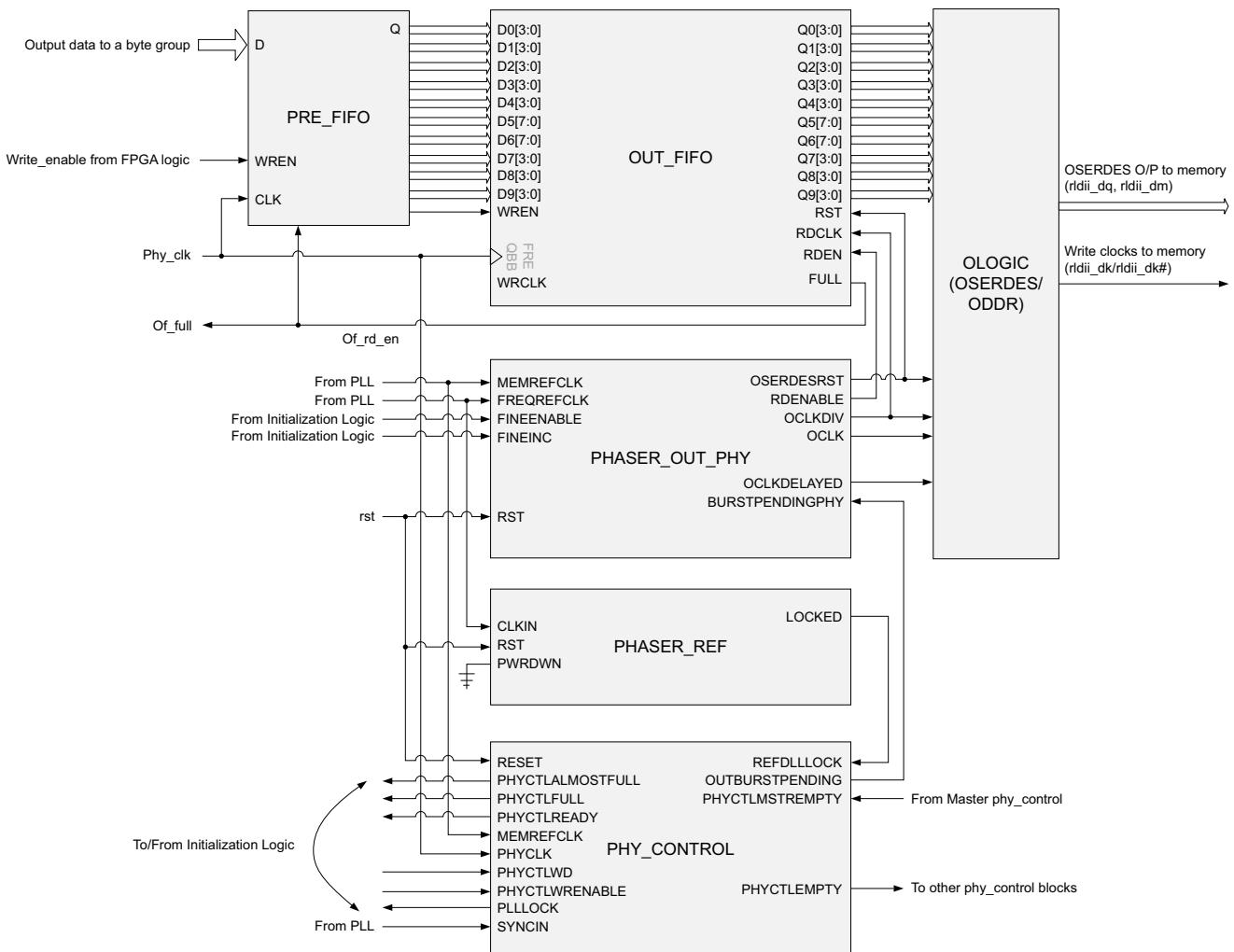


Figure 3-50:

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.

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 3-51](#) shows a high-level block diagram of the path the read clock and the read data captures from entering the FPGA until it is given to you. The read clock bypasses the ILOGIC and is routed through PHASERs within each byte group. For RLDRAM II, the multiregion BUFMR is used to get the read capture clock to the necessary PHASERS used in read data capture. 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 BUFMR is needed for RLDRAM II because there can potentially be a single capture clock for two bytes of data, and only the BUFMR can allocate the clock to the multiple PHASERs as required.

Because RLDRAM 3 includes a capture clock per byte of data, the multiregion BUFMR is not required. 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 3-51](#)) 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.

For RLDRAM 3, the IN\_FIFO also transfers the data from the ICLKDIV domain (which runs at half the memory clock frequency) to the FPGA logic clock domain (which runs at a quarter the memory clock frequency). 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, 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 FPGA logic 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 the [Calibration](#) section.

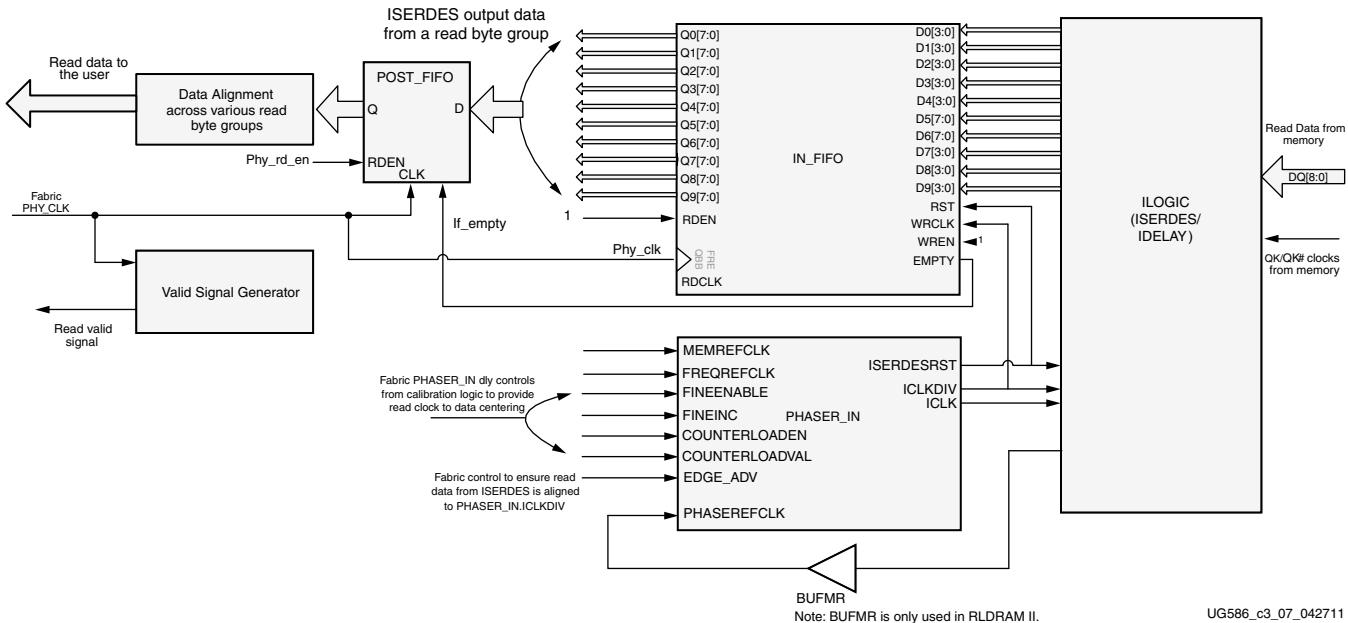


Figure 3-51:

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 64 taps possible.

For designs running at or above 400 MHz, the calibration logic also performs a one-time write calibration to ensure the write clock is center aligned properly with the write data. Calibration begins after memory initialization. Prior to this point, all read path logic is held in reset.

The calibration procedure is different depending on memory type. While similar, RLDRAm II and RLDRAm 3 require different FPGA pin rules that must be accounted for in the calibration algorithm (see Pin Rules in [Verify Pin Changes and Update Design, page 416](#)). RLDRAm 3 also runs at higher frequencies which requires using a quarter rate FPGA logic clock versus the half-rate FPGA logic clock used for RLDRAm II.

[Figure 3-52](#) shows the calibration simulation flow for the RLDRAm II and RLDRAm 3. In simulation some of the steps are skipped to speed up the time required before processing user commands.

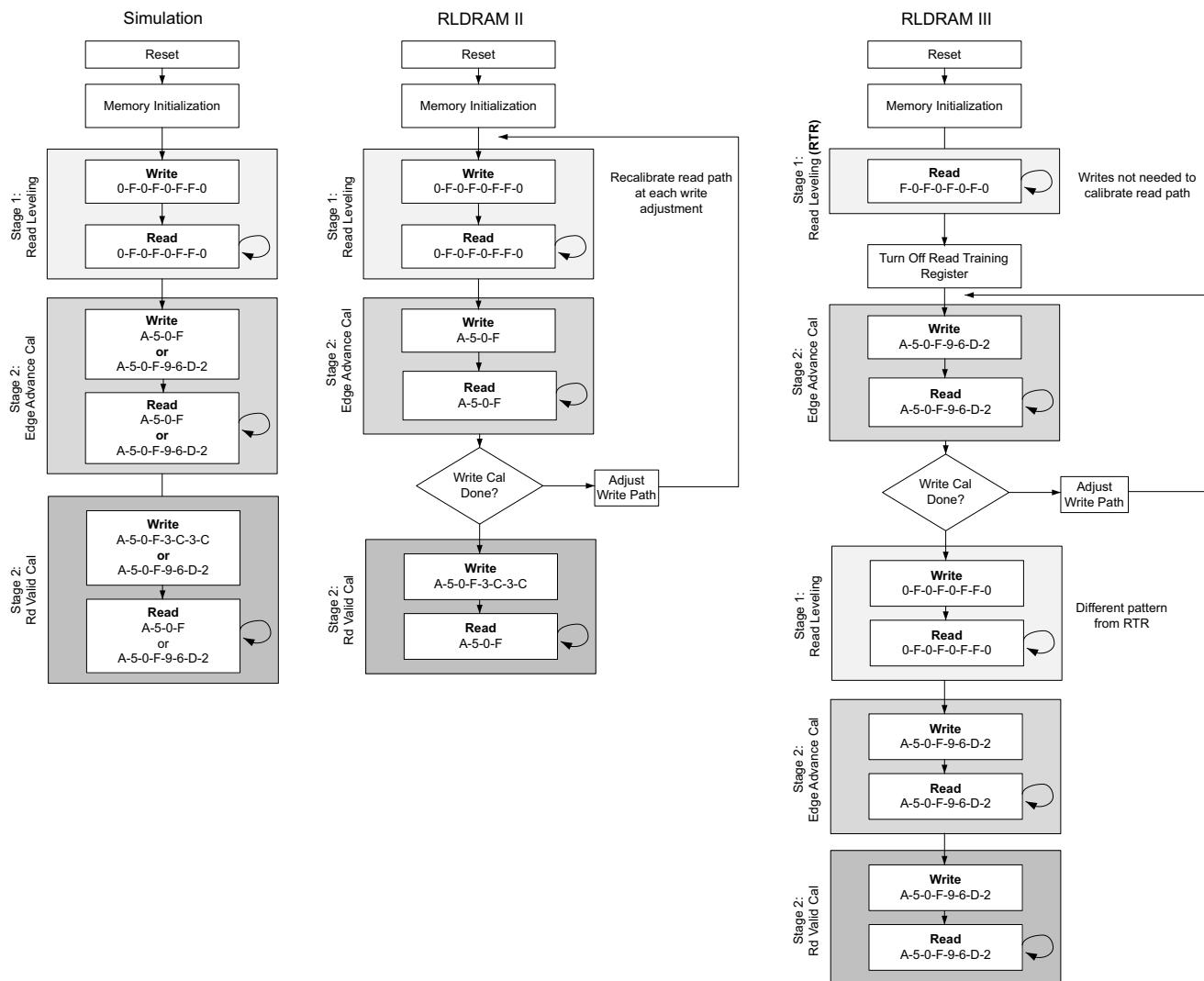


Figure 3-52:

### *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.

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 single write command is issued to address location 0 in each bank of the memory device (eight banks for RLDRAM II and 16 for RLDRAM 3). All banks are used to ensure no matter which burst length is selected, the read commands can be issued to ensure read data is returned back-to-back without any gaps in the data stream.

If performing write calibration for RLDRAM 3, you can calibrate reads first by using the Read-Training Register (RTR) of the DRAM. This provides a clock-like pattern from the DRAM that does not require writing in a pattern first. All other times a pattern of "OFOF\_OFFO" is used to calibrate read clock and data capture.

If performing write calibration for RLDRAM II, this stage of calibration is continually restarted based on the requirements on the write calibration algorithm.

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  $\geq 400$  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. [Figure 3-53](#) shows this example.

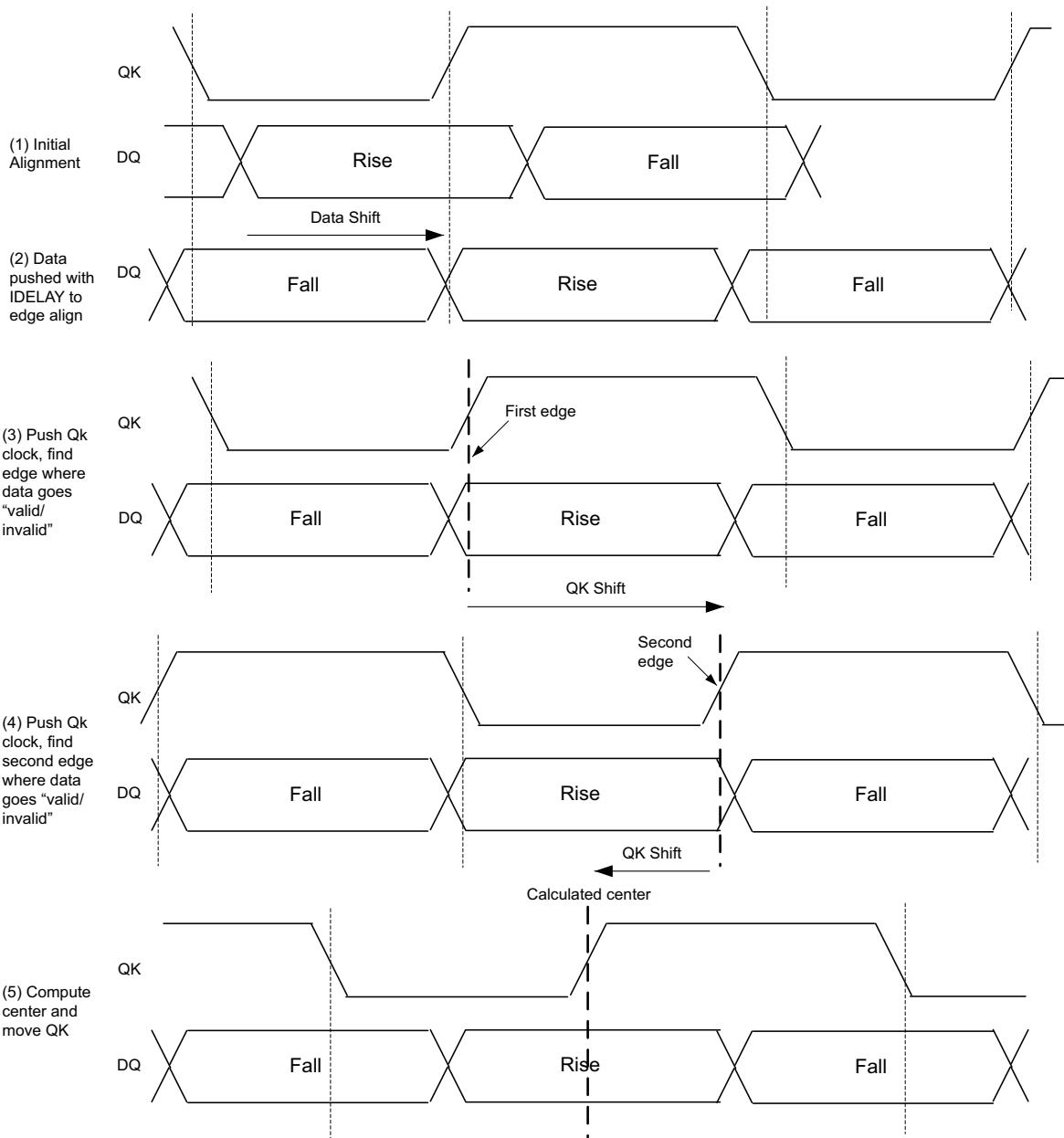


Figure 3-53:

For frequencies < 400 MHz, because FREO\_REF\_CLK has twice the frequency of 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 < 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.

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.

Complex pattern read calibration stage is added as the last stage of calibration to improve margin.

### ***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
- 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 memory 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. This stage of read calibration acts as feedback to the write calibration state machine (if enabled) to determine the results of a write for a given byte lane.

For RLDRAM II, 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.

For RLDRAM 3, because the FPGA logic is running at a quarter the rate of the memory clock frequency, the data is bitslipped in the FPGA logic by a memory clock cycle each time the pulse is issued to ensure proper alignment of all captured data in the expected order.

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 data equal to a single FPGA logic clock cycle 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 either be your set indicated value 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 you.
4. Assert `init_calib_complete`.

### ***Write Calibration***

When write calibration is enabled, the results of read calibration data alignment are used to determine if a given setting is valid for correct write operation. RLDRAM 3 contains an MRS read training register that can be used for reading out a set pattern from the memory without having to write a pattern to the memory first. After memory initialization, the read capture is first calibrated using this set pattern before moving on to calibrate the writes.

Because RLDRAM II lacks this read training register, the reads and writes cannot be independently verified. At each step of write calibration, the alignment of the read clock with DQ is performed to ensure the correct capture of data. If the data alignment portion of read calibration is performed for a given byte lane and the expected result is not found, the write is assumed to have caused the failure. For RLDRAM II, at each step of write calibration, the read calibration and associated logic are reset and restarted.

See [Figure 3-52, page 449](#) for write calibration flow.

PHASER\_OUT provides all of the clocking resources for the output path and is adjusted on a byte lane basis by the calibration algorithm. Each byte lane is independently checked against the write clock being sent to the DRAM to ensure proper write timing. Depending on the pinout, either OCLK\_DELAYED is used to adjust the DK clock in relation to the data DQ, or OCLK for a given byte lane is adjusted in relation to the DK clock in another byte lane. Due to the length of time required to independently calibrate each byte lane, write calibration is usually skipped for simulation.

The steps taken for write calibration is dependent on the pinout. [Figure 3-54](#) shows the RLDRAM II pinout block diagram with two data byte lanes and the overview for the steps taken for write calibration.

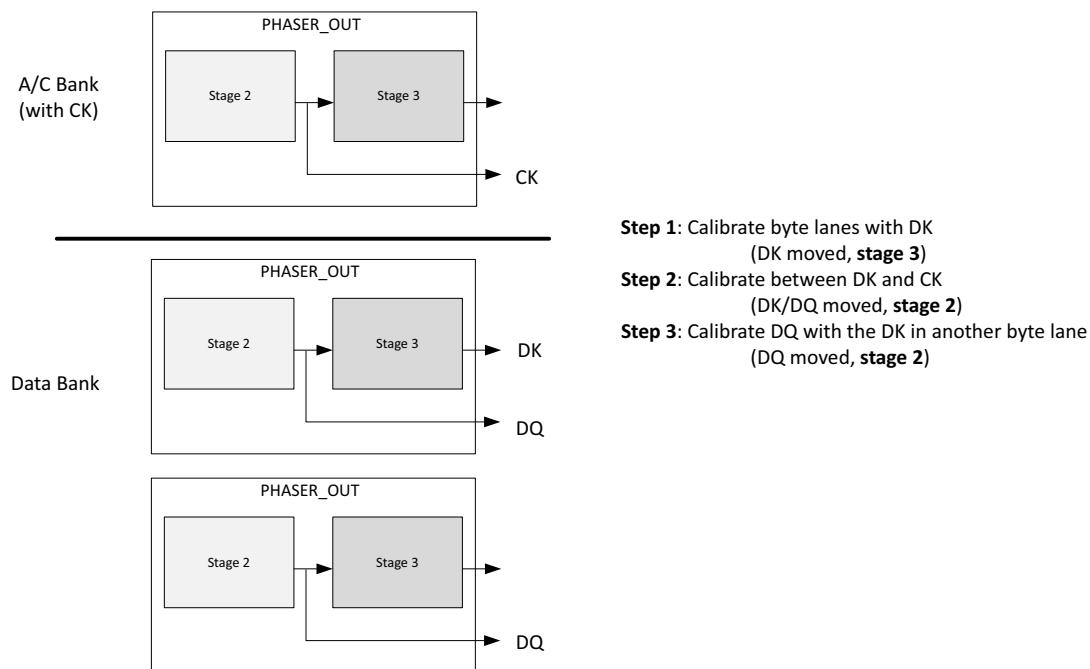
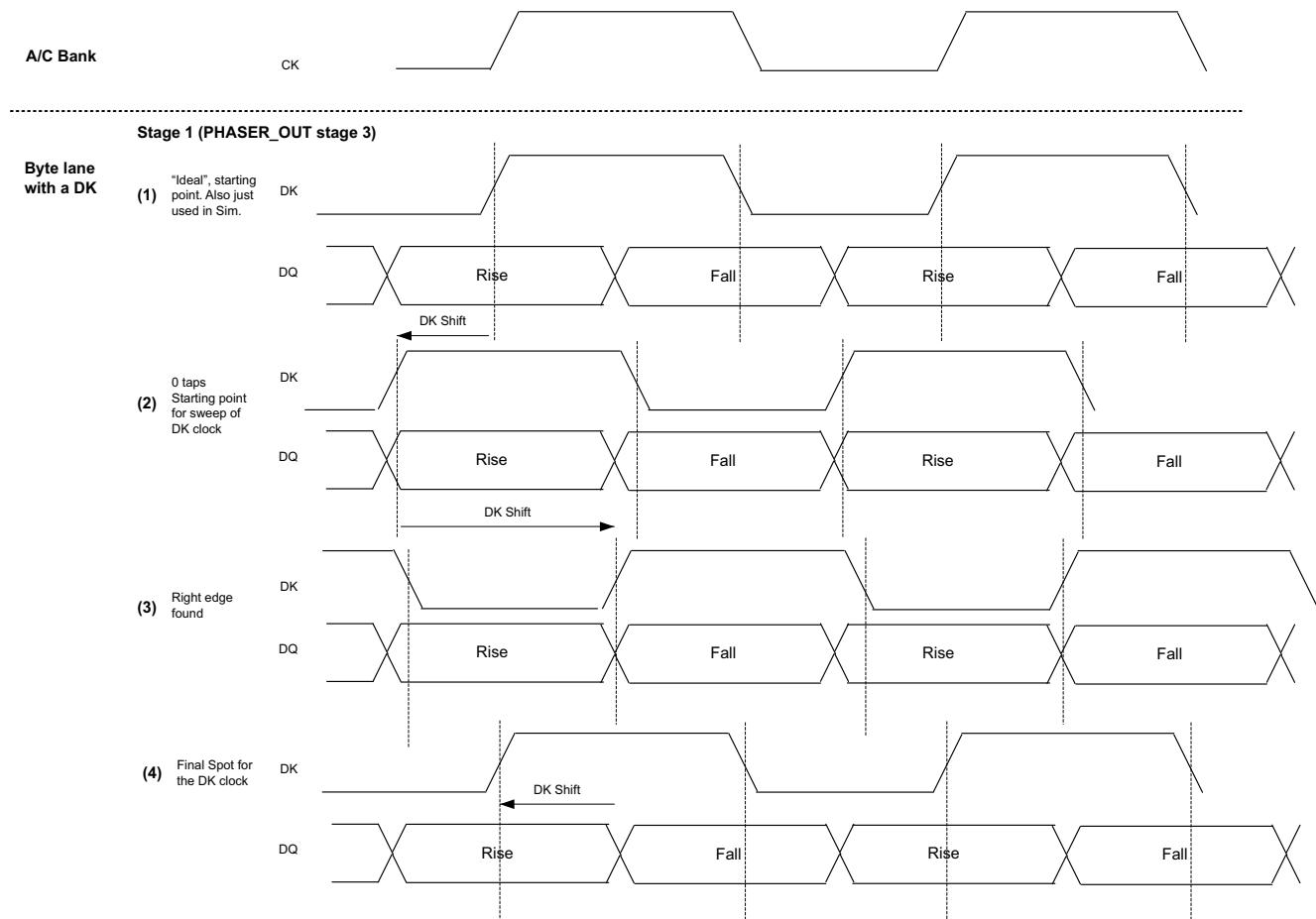


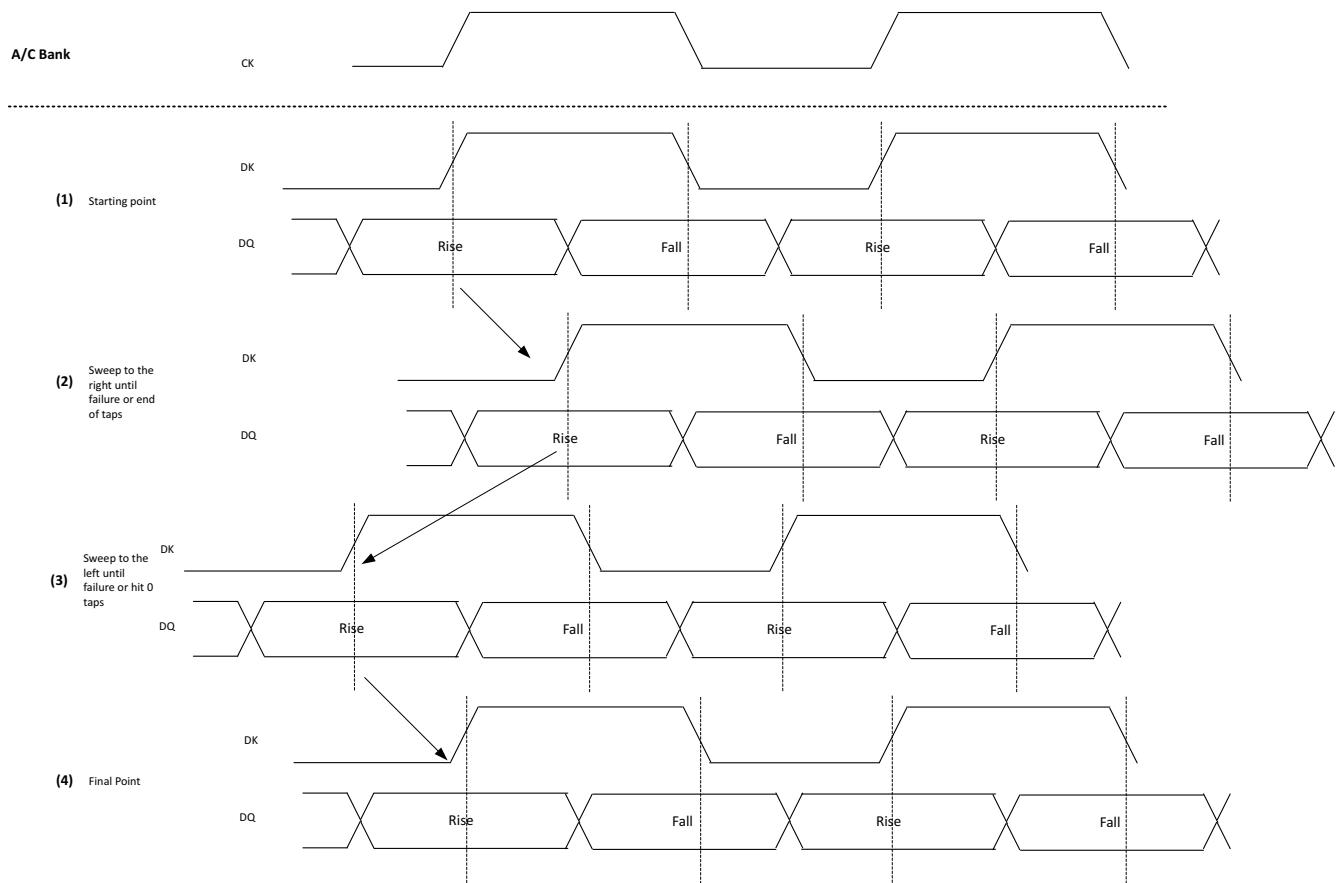
Figure 3-54:

The first stage of RLDRAm II write calibration is to calibrate DK clock with respect to DQ in the same byte lane. The write clock DK is adjusted in relation to the DQ to find the data valid window and center in that window as shown in [Figure 3-55](#).



*Figure 3-55:*

Figure 3-56 shows the second stage of RLDRAm II write calibration for in which the entire byte lane is shifted in relation to the CK to sweep and find where the write data transfer breaks for the DK-to-CK alignment.

*Figure 3-56:*

[Figure 3-57](#) shows the last step of RLDRAm II write calibration, where the byte lanes that do not share a DK clock as part of their PHASER\_OUT output, are calibrated with respect to the DK clock in another byte lane.

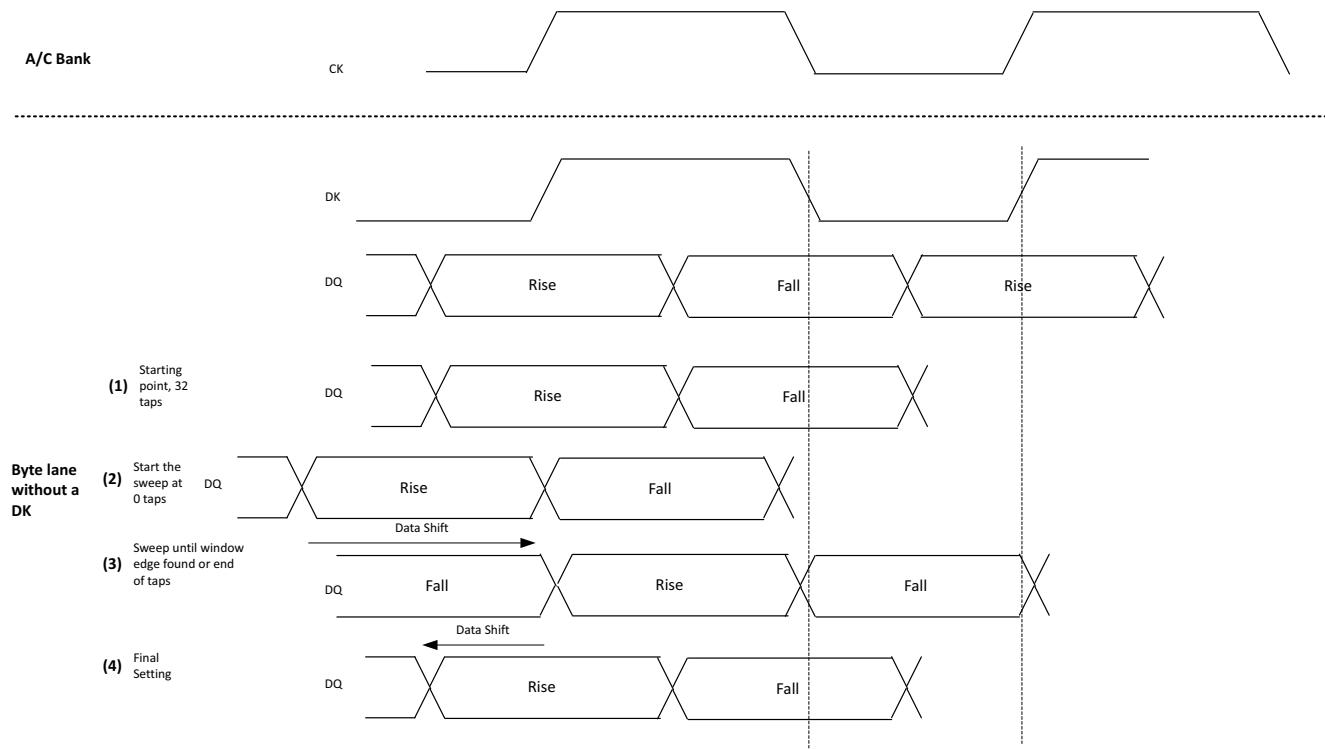


Figure 3-57:

Figure 3-58 shows the RLDRAm 3 pinout with two data byte lanes and the overview for the steps taken for write calibration.

Figure 3-59 shows the steps taken for a byte lane for RLDRAm 3. The data is adjusted with respect to the DK clock coming from another bank. This is the same as RLDRAm 3 stage 3, just the first two stages of calibration are skipped for RLDRAm 3.

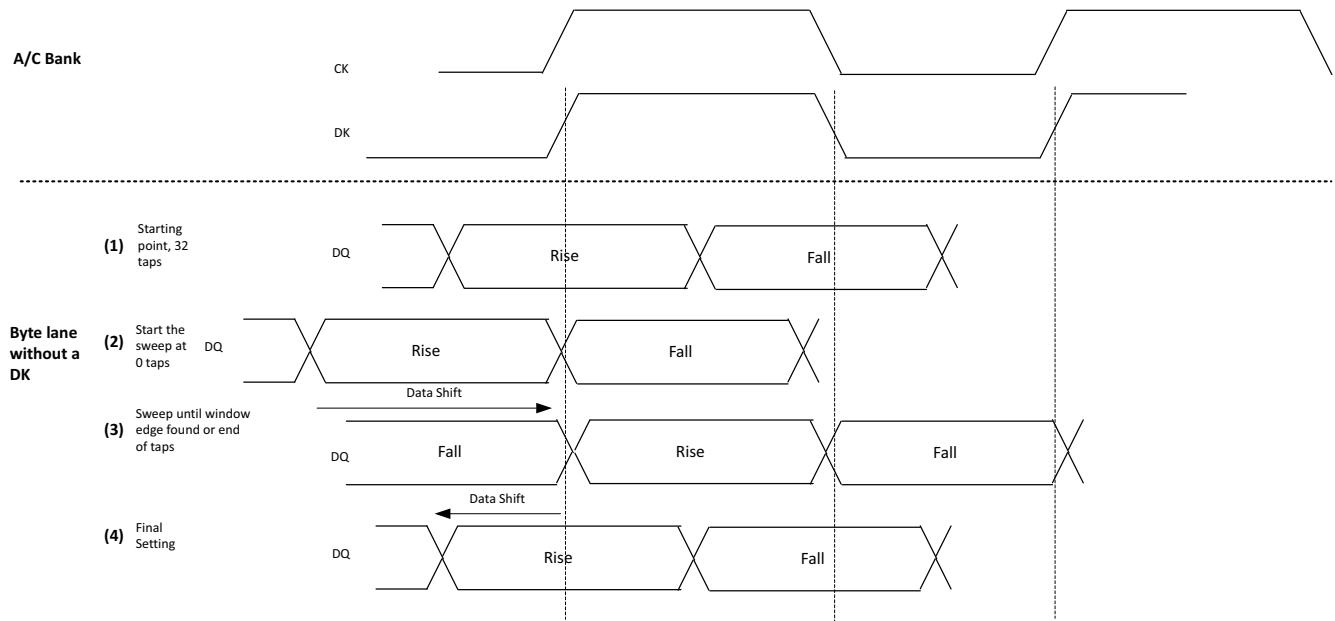


Figure 3-59:

When write calibration completes, the read calibration is restarted one last time to run with the proper write settings and allowed to complete through read valid generation.

The simulation waveforms for write calibration of a 36-bit RLDRAm II design is shown in Figure 3-60. The state machine steps through the calibration one byte at a time, selecting the PHASERs for a given byte lane, making adjustments, and recording the results to optimize the write timing. Adjustments are only made within the limits of the PHASER\_OUT fine tap delay. To debug any problems, it is important to check the margin found during both read and write calibration, and to check the cmd-to-data write latency seen by the DRAM matches what is programmed in the MRS register. For more details, see [Debugging RLDRAm II and RLDRAm 3 Designs, page 482](#).

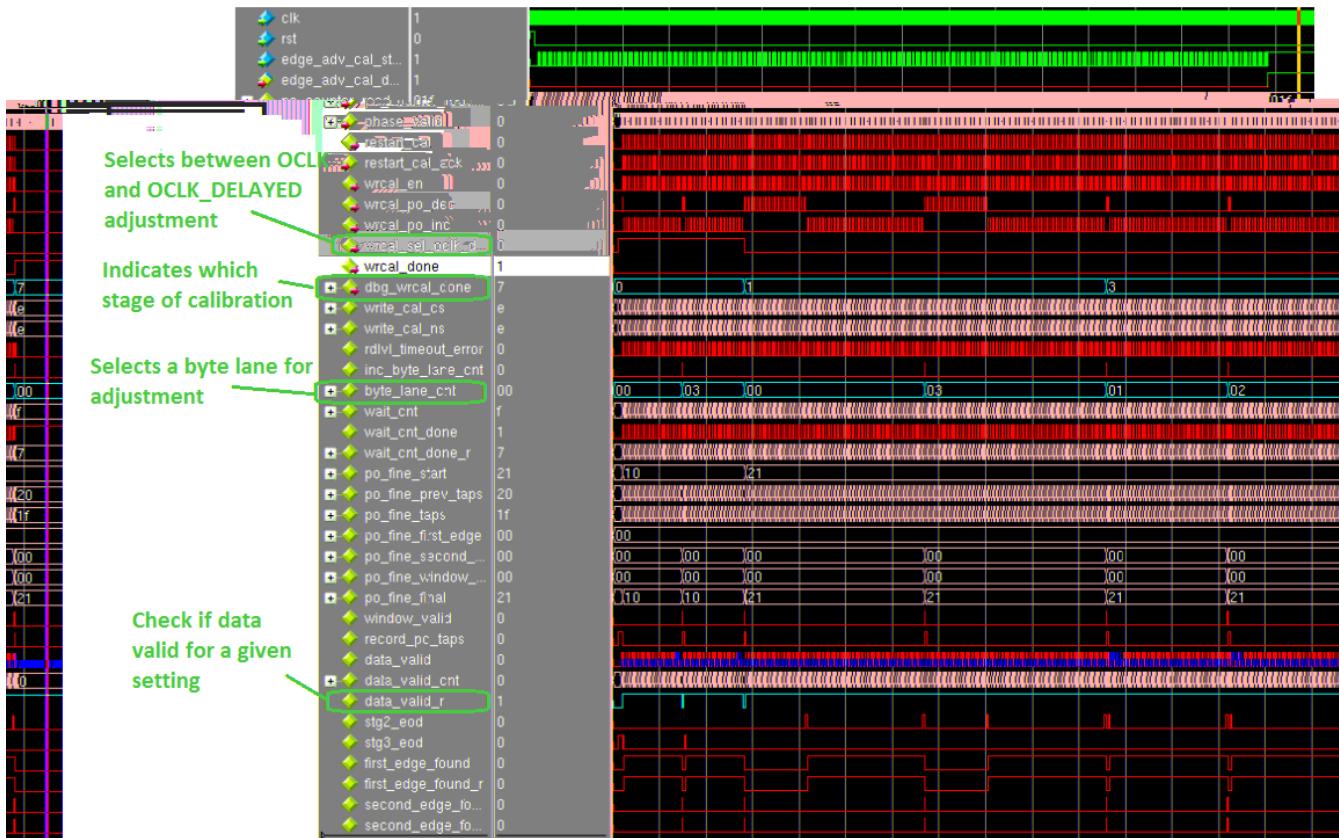


Figure 3-60:

The RLDRAM II/RLDRAM 3 memory interface solution is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top-level of the core. As per the OOC flow, none of the parameter values are passed down to the user design RTL file from the example design top RTL file. So, any design related parameter change is not reflected in the user design logic. The MIG tool should be used to regenerate a design when parameters need to be changed. The parameters are summarized in Table 3-14.

Table 3-14:

CLK_PERIOD	Memory clock period (ps).	-
ADDR_WIDTH	Memory address bus width.	18-22
RLD_ADDR_WIDTH	Physical Memory address bus width when using Address Multiplexing mode.	11, 18-22
BANK_WIDTH	Memory bank address bus width.	RLDRAM II: 3 RLDRAM 3: 4

Table 3-14:

(Cont'd)

DATA_WIDTH	Memory data bus width and can be set through the MIG tool. A maximum DATA_WIDTH of 72 is supported.	-
QK_WIDTH	Memory read clock bus width.	RLDRAM II: 2 per x18/x36 device RLDRAM 3: DATA_WIDTH/9
DK_WIDTH	Memory write clock bus width.	RLDRAM II: 2 per x36 device, 1 per x18 device RLDRAM 3: 2 per device
BURST_LEN	Memory data burst length.	RLDRAM II: 4, 8 RLDRAM 3: 2, 4, 8
DM_PORT	This parameter enables and disables the generation of the data mask ports.	ON, OFF
NUM_DEVICES	Number of memory devices used.	1-4
MRS_CONFIG	This parameter sets the configuration setting in the RLDRAm II/RLDRAM 3 memory register.	RLDRAM II: 1, 2, 3 RLDRAM 3: 3, 4, 5, 6, 7, 8, 9, 10, 11
MRS_ADDR_MUX	This parameter sets the address multiplexing setting in the RLDRAm II/RLDRAM 3 memory register.	ON, OFF
MRS_DLL_RESET	This parameter sets the DLL setting in the RLDRAm II/RLDRAM 3 memory register.	DLL_ON
MRS_IMP_MATCH	This parameter sets the impedance setting in the memory register.	INTERNAL, EXTERNAL
MRS_ODT	This parameter sets the ODT setting in the memory register.	ON, OFF
MRS_RD_LATENCY	This parameter sets the Read latency and write latency setting in the RLDRAm 3 memory register, and is dependent on memory device and frequency of operation.	8-16
MRS_RTT_WR	This parameter sets the output drive impedance setting in the MRS register for RLDRAm 3.	40, 60, 120
MRS_RTT_RD	This parameter sets the ODT setting in the MRS register for RLDRAm 3. If ODT is not used this parameter becomes a "Do not care."	40, 60
MEM_TRC	This parameter sets the RLDRAm 3 TRC setting, and is dependent on the memory device and read latency selected.	4-11
MEM_TYPE	This parameter specifies the memory type.	RLD2_CIO, RLD3
IODELAY_GRP <sup>(1)</sup>	This is a unique name for the IODELAY_CTRL provided when multiple IP cores are used in the design.	-
REFCLK_FREQ	Reference clock frequency for IDELAYCTRLs. This parameter should <b>not</b> be changed.	200.0

Table 3-14:

(Cont'd)

BUFMR_DELAY	Simulation-only parameter used to model buffer delays (RLDRAM II only).	-
RST_ACT_LOW	Active-Low or active-High reset. This is set to 1 when System Reset Polarity option is selected as active-Low and set to 0 when the option is selected as active-High.	0, 1
IBUF_LPWR_MODE	Enables or disables low power mode for the input buffers.	ON, OFF
IODELAY_HP_MODE	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.	ON, OFF
SYSCLK_TYPE	This parameter indicates whether the system uses single-ended system clocks, differential system clocks, or is driven from an internal clock (No Buffer). 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. For the No Buffer option, sys_clk_i, which appears in the port list, needs to be driven from an internal clock.	DIFFERENTIAL, SINGLE_ENDED, NO_BUFFER
REFCLK_TYPE	This parameter indicates whether the system uses single-ended reference clocks, differential reference clocks, is driven from an internal clock (No Buffer), or can connect to the system clock input only (Use System Clock). Based on the selected CLK_TYPE, the clocks must be placed on the correct input ports. For differential clocks, clk_ref_p/clk_ref_n must be used. For single-ended clocks, clk_ref_i must be used. For the No Buffer option, clk_ref_i, which appears in the port list, needs to be driven from an internal clock. For the Use System Clock option, clk_ref_i is connected to the system clock in the user design top module.	DIFFERENTIAL, SINGLE_ENDED, NO_BUFFER, USE_SYSTEM_CLOCK
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.	-
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.	-
CLKOUT0_PHASE	Phase of PLL output CLKOUT0. This value is set by the MIG based on the banks selected for memory interface pins and the frequency of operation.	-
DIVCLK_DIVIDE	PLL VCO divisor. This value is set by the MIG tool based on the frequency of operation.	-

Table 3-14:

(Cont'd)

SIM_BYPASS_INIT_CAL	This simulation-only parameter is used to speed up simulations, by skipping the initialization wait time and speeding up calibration. SKIP_AND_WRCAL and FAST_AND_WRCAL are options to SKIP or perform FAST read calibration, but to simulate write calibration.	FAST, NONE, SKIP_AND_WRCAL, FAST_AND_WRCAL
SIMULATION	Set to "TRUE" for simulation; set to "FALSE" for implementation.	"TRUE," "FALSE"
DEBUG_PORT	Turning on the debug port allows for use with the VIO of the Vivado logic analyzer feature. This allows you 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.	ON, OFF
N_DATA_LANES	Calculated number of data byte lanes, used to set up signal widths for using the debug port. This parameter should <b>not</b> be changed.	DATA_WIDTH/9
DIFF_TERM_SYSCLK	Differential Termination for System clock input pins	"TRUE," "FALSE"
DIFF_TERM_REFCLK	Differential Termination for IDELAY reference clock input pins	"TRUE," "FALSE"
nCK_PER_CLK	Number of memory clocks per FPGA logic clocks. This parameter should <b>not</b> be changed.	RDRAM II: 2 RDRAM 3: 4
TCQ	Register delay for simulation.	100

**Notes:**

1. This parameter is prefixed with the module name entered in MIG during design generation. If the design is generated with the module name as mig\_7series\_0, then IODELAY\_GRP parameter name is mig\_7series\_0\_IODELAY\_MIG.

Table 3-15 contains parameters set up by the MIG tool based on the pinout selected. When making pinout changes, Xilinx recommends rerunning the MIG tool so the parameters are set up properly; otherwise see [Pinout Requirements, page 468](#). Mistakes to the pinout parameters can result in non-functional simulation, an unroutable 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 3-15:

MASTER_PHY_CTL	0, 1, 2. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	The bank where the master PHY_CONTROL resides (usually corresponds to MMCM/PLL bank location).
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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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	RDRAM II Only. 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> This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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).

*Table 3-15:*

*(Cont'd)*

Table 3-15:

(Cont'd)

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>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<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>9'&lt;code&gt;h00_00_00_00_00_00_00_00_00_10_13&lt;/code&gt; = 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, DK[0]/DK#[0] corresponds to the 8 LSBs of the parameter).</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>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<p>See the <a href="#">DK_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>
CS_MAP	<p>Bank and byte lane position information for the chip select. 12-bit parameter provided per pin.</p> <ul style="list-style-type: none"> <li>[3:0] – Bit position within a byte lane. Values of [0, 1, 2, . . . , A, B] are supported.</li> <li>[7:4] – Byte lane position within a bank. Values of 0, 1, 2, or 3 are supported.</li> <li>[11:8] – Bank position. Values of 0, 1, or 2 are supported</li> </ul> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<p>See the <a href="#">CK_MAP</a> example.</p>
WE_MAP	<p>Bank and byte lane position information for the write enable. See <a href="#">CS_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<p>See the <a href="#">CK_MAP</a> example.</p>

Table 3-15:

(Cont'd)

REF_MAP	Bank and byte lane position information for the refresh signal. See <a href="#">CS_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">CK_MAP</a> example.
ADDR_MAP	Bank and byte lane position information for the address. See <a href="#">CS_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">CK_MAP</a> example.
BANK_MAP	Bank and byte lane position information for the bank address. See <a href="#">CS_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">CK_MAP</a> example.
DQTS_MAP	Bank and byte lane position information for the 3-state control. See <a href="#">CS_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">CK_MAP</a> example.
DM_MAP	Bank and byte lane position information for the data mask. See <a href="#">CS_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">CK_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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">CK_MAP</a> example.



This generates a file named **flight\_time.csv** in the current directory with package trace delay information for each pin. While applying specific trace-matching guidelines for the RLDRAM II/RLDRAM 3 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 RLDRAM II/RLDRAM 3 signals:

- RLDRAM II
  - For x36 data width, the maximum skew between DQ[17:0] and DK/DK#[0] should be  $\pm 15$  ps.
  - For x36 data width, the maximum skew between DQ[35:18] and DM and DK/DK#[1] should be  $\pm 15$  ps.
  - For x18 data width, the maximum skew between any DQ/DM and DK/DK# should be  $\pm 15$  ps.
- RLDRAM 3
  - The maximum skew between DQ[8:0] and DQ[26:18] and DM[0] to DK/DK#[0] should be  $\pm 15$  ps.
  - The maximum skew between DQ[17:9] and DQ[35:27] and DM[1] to DK/DK#[1] should be  $\pm 15$  ps.
- The maximum skew between any DQ and its associated QK/QK# should be:
  - RLDRAM II:  $\pm 15$  ps
  - RLDRAM 3:  $\pm 10$  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.

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the RLDRAM II/RLDRAM 3 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 two pairs of multiregion clock capable I/O (MRCC) pins and four byte groups that contain 1 DQS clock-capable I/O pair and 10 associated I/Os.

## RLDRAM II

In a typical RLDARAM II data bank configuration, 9 of these 10I/Os are used for the data (DQ) and one can be used for the data mask (DM). DM must be placed in the same byte lane as the corresponding data; if two bytes share the same DM then it should be placed with one of those bytes. The write clocks (DK/DK#) use one of the DQS 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. Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, RLDARAM 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 XDC are not recommended. However, if the XDC 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 DQS 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 DQS 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 1-byte (nine bits) are placed inside a byte group. DQ must not be placed on the DQS N location in a byte lane, because this location is used for the 3-state control.
- Data Mask (DM) must be placed with one of the corresponding data byte lanes it is associated with.

**Note:** If DM pins are not used, they should be tied to ground. For more information, consult the memory vendor data sheet.

- Xilinx recommends keeping all of 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. The address/control must be in the middle I/O bank of the interfaces that span three I/O banks. Also, all address/control signals must be in the same I/O bank. Address/control cannot be split between banks.
- For a given byte lane, the DQS\_N location is used to generate the 3-state control signal. The 3-state can share the location with 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.



**RECOMMENDED:** *Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.*

- 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.

## RLDRAM 3

In a typical RLDAM 3 data bank configuration, 9 of these 10I/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 DQS pairs inside the Address/Control bank, or the DQS pairs in a free byte lane in a data bank. QK/QK# clocks must be placed on DQS pins in a given data bank lane associated with this same clock. Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, RLDAM 3 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 XDC are not recommended. However, if the XDC needs to be altered, these rules must be taken into consideration:

- The CK/CK# clocks must be placed in an address/control byte lane. CK must be placed on the P location, and CK# must be placed on the N location, of an I/O pin pair in that byte lane.
- The DK/DK# clocks must be placed on a DQS 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 1-byte (nine bits) are placed inside a byte group. DQ must not be placed on the DQS N location in a byte lane, because this location is used for the 3-state control.
- Data Mask (DM) must be placed with one of the corresponding data byte lanes it is associated with. For the x18 device DM[0] corresponds to DQ[8:0] and DM[1] to DQ[17:9], while for the x36 device DM[0] corresponds to DQ[8:0]/DQ[26:18] and DM[1] to DQ[17:9]/DQ[35:27].



**RECOMMENDED:** *If DM pins are not used, they should be tied to ground. For more information, see the memory vendor data sheet.*

- Xilinx recommends keeping all of the data generated from a single memory component within a bank.

- Read clocks (QK and QK#) need to be placed on the DQS pins that are available in a data byte lane, respectively. Data must be in the same byte lane as the associated QK/QK#.
- Address/control signals can be placed in byte groups that are not used for data and all should be placed in the same bank. The address/control must be in the middle I/O bank of the interfaces that span three I/O banks. Also, all address/control signals must be in the same I/O bank. Address/control cannot be split between banks.
- For a given byte lane, the DQS\_N location is used to generate the 3-state control signal. The 3-state can share the location with DK# only
- 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.



**RECOMMENDED:** *Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.*

- 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.

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. However, exceptions are possible where pins might not be available for the clock input in the bank as that of the PLL. 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/Os.

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 435](#) for information on allowed PLL parameters.

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

```
create_clock -period 1.875 [get_ports sys_clk_p]
```

The **clk\_ref** constraint sets the frequency for the IDELAY reference clock, which is typically 200 MHz. For example:

```
create_clock -period 5 [get_ports clk_ref_p]
```

The I/O standards are set appropriately for the RLDRAM II interface with LVCMOS15, HSTL15\_I, HSTL15\_I\_DCI, DIFF\_HSTL15\_I, or DIFF\_HSTL15\_I\_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:

the° she atersn '

Do not drive user clocks through 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. For more information, see the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10].

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

```
# PadFunction: IO_L13P_T2_MRCC_37
set_property VCCAUX_IO DONTCARE [get_ports {sys_clk_p}]
set_property IOSTANDARD DIFF_HSTL_I [get_ports {sys_clk_p}]
set_property PACKAGE_PIN K22 [get_ports {sys_clk_p}]

# PadFunction: IO_L13N_T2_MRCC_37
set_property VCCAUX_IO DONTCARE [get_ports {sys_clk_n}]
set_property IOSTANDARD DIFF_HSTL_I [get_ports {sys_clk_n}]
set_property PACKAGE_PIN J22 [get_ports {sys_clk_n}]
```

For more information, see the *Xilinx Timing Constraints Guide* (UG612) [Ref 15].

For RLDRAM II interfaces that have the memory system input clock (sys\_clk\_p/sys\_clk\_n) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_HSTL\_I I/O standard (VCCO = 1.5V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_HSTL\_I and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_HSTL\_I CCIO pins. For more details on usage and required circuitry for LVDS and LVDS\_25 I/O Standards, see the *7 Series FPGAs SelectIO™ Resources User Guide* (UG471) [Ref 2].

These recommendations apply to termination for RLDRAM II and RLDRAM 3 memory interface solution:

- Simulation (using IBIS or other) is highly recommended. The loading of command and address signals depends on various factors, such as speed requirements and termination topology. Loading can be a limiting factor in reaching a performance target.
- Command and Address signals should be terminated to  $V_{TT}$  through a  $40\Omega$  resistor if operating at and above 1,333 Mb/s, or  $50\Omega$  if operating below 1,333 Mb/s.
- Data signals (DQ) do not require an external termination, and should use DCI. Set DCI termination to  $40\Omega$  for operation at and above 1,333 Mb/s, or  $50\Omega$  if operating below 1,333 Mb/s.
- Data Mask (DM) does not require an external termination, as On-Die Termination (ODT) is sufficient.
- QVLD (RLDRAM 3 only) does not require an external termination and should use DCI. Set DCI termination to  $40\Omega$  for operation at and above 1,333 Mb/s, or  $50\Omega$  if operating below 1,333 Mb/s.

- Write Data clock (DK\_P/N) does not require an external termination, as On-Die Termination is sufficient.
- Input Clock (CK\_P/N) should be differentially terminated with an  $80\Omega$  resistor.
- Read Data clock (QK\_P/N) does not require an external termination and should use DCI. Set DCI termination to  $40\Omega$  for operation at and above 1,333 Mb/s, or  $50\Omega$  if operating below 1,333 Mb/s.

For manually manipulating the parameters described in [Table 3-15](#), the following examples show how to allocate parameters for a given byte lane. [Table 3-16](#) shows a typical RLDRAM II data byte lane, indicating the bank, byte lane, and bit position for each signal.

*Table 3-16:*

0	0	9	VREF	A_11	P	12	VREF	
		8	DQ8	A_10	N	11		
		7	DQ7	A_09	P	10		
		6	DQ6	A_08	N	9		
		B	DKO_P	A_07	P	8	DQS-P	
		A	DKO_N	A_06	N	7	DQS-N	
		5	DQ5	A_05	P	6		
		4	DQ4	A_04	N	5		
		3	DQ3	A_03	P	4		
		2	DQ2	A_02	N	3		
		1	DQ1	A_01	P	2		
		0	DQ0	A_00	N	1		
		VRN	N/A	SE	0			

0	1	
1	0	0001
1	0	1
1	1	111
1	F	1
1	1	111
1	1	1
1	F	FF
<b>1FF</b>		

The byte lane parameters for [Table 3-16](#) are shown in [Table 3-17](#).

*Table 3-17:*

DK_MAP	8'h00
DQTS_MAP	12'h00A
PHY_O_BITLANES	12'h1FF
DATA0_MAP	108'h008_007_006_005_004_003_002_001_000

[Table 3-18](#) shows the byte lane with the Data Mask (DM) placed on the 3-state location. 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 3-18](#). 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 3-18:*

XDC									
0	1	9		QKO_P	B_11	P	24	CCIO-P	
		8	DQ17	QKO_N	B_10	N	23	CCIO-N	0
		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	DQS-P	0
		A	DM	DM	B_06	N	19	DQS-N	1
		5	DQ14	DQ14	B_05	P	18		0101
		4	DQ13	DQ13	B_04	N	17		5
		3	DQ12	DQ12	B_03	P	16		111
		2	DQ11	DQ11	B_02	N	15		1
		1	DQ10	DQ10	B_01	P	14		111
		0	DQ9	DQ9	B_00	N	13		1

The byte lane parameters for [Table 3-18](#) are shown in [Table 3-19](#).

*Table 3-19:*

DM_MAP	12'h01A
DOTS_MAP	12'h01B
PHY_O_BITLANES	12'h5FF
DATA1_MAP	108'h018_017_016_015_014_013_012_011_010
QK_MAP	8'h01

Table 3-20 shows another RLDRAM II byte lane with the 3-state control location unused.

*Table 3-20:*

XDC											
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	DQS-P		1	1011
		A			C_06	N	7	DQS-N		0	B
		5	DQ21	DQ21	C_05	P	6			1	111
		4	DQ20	DQ20	C_04	N	5			1	1
		3	DQ19	DQ19	C_03	P	4	CCIO-P	1		
		2	DQ18	DQ18	C_02	N	3	CCIO-N	1		
		1		QK1_P	C_01	P	2	CCIO-P	0	110	
		0		QK1_N	C_00	N	1	CCIO-N	0	0	C

The byte lane parameters for Table 3-20 are shown in Table 3-21.

*Table 3-21:*

DQTS_MAP	12'h02A
PHY_O_BITLANES	12'hBFC
DATA1_MAP	108'h029_028_027_026_02B_025_024_023_022
QK_MAP	8'h02

[Table 3-22](#) shows the RLDRAM II byte lane with the 3-state pin location used for DM. In this situation the signals are shifted down in the **OUT\_FIFO**.

*Table 3-22:*

XDC									
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		
		6	DQ23	DQ23	C_08	N	9		
		B	DQ22	DQ22	C_07	P	8	DQS-P	
		A	<b>DM</b>	<b>DM</b>	C_06	N	7	DQS-N	
		5	3-state	DQ21	C_05	P	6		0
		4	DQ21	DQ20	C_04	N	5		1
		3	DQ20	DQ19	C_03	P	4	CCIO-P	1
		2	DQ19	DQ18	C_02	N	3	CCIO-N	1
		1	DQ18	<b>QK1_P</b>	C_01	P	2	CCIO-P	1
		0		<b>QK1_N</b>	C_00	N	1	CCIO-N	0

1111  
**F**  
 110  
**D**  
 111  
**E**  
**FDE**

The byte lane parameters for [Table 3-22](#) are shown in [Table 3-23](#).

*Table 3-23:*

DM_MAP	12'h02A
DQTS_MAP	12'h025
PHY_O_BITLANES	12'hFDE
DATA1_MAP	108'h029_028_027_026_02B_024_023_022_021
QK_MAP	8'h02

The MIG tool generates the appropriate XDC 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 3-24](#) and [Table 3-25](#) contain a list of the ports with the I/O standard used.

**Table 3-24:**

rld_ck_p, rld_ck_n	Output	DIFF_HSTL_I
rld_dk_p, rld_dk_n	InOut	DIFF_HSTL_II
rld_cs_n	Output	HSTL_I
rld_we_n	Output	HSTL_I
rld_ref_n	Output	HSTL_I
rld_a	Output	HSTL_I
rld_ba	Output	HSTL_I
rld_dm	Output	HSTL_I
rld_dq	Input/Output	HSTL_II_T_DCI, HSTL_II
rld_qk_p, rld_qk_n	Input	DIFF_HSTL_II_DCI, DIFF_HSTL_II

**Table 3-25:**

rld_ck_p, rld_ck_n	Output	DIFF_SSTL12
rld_dk_p, rld_dk_n	InOut	DIFF_SSTL12
rld_cs_n	Output	SSTL12
rld_we_n	Output	SSTL12
rld_ref_n	Output	SSTL12
rld_a	Output	SSTL12
rld_ba	Output	SSTL12
rld_dm	Output	SSTL12
rld_dq	Input/Output	SSTL12_T_DCI, SSTL12
rld_qk_p, rld_qk_n	Input	DIFF_SSTL12_DCI, DIFF_SSTL12

DCI (HP 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 for RLDRAM II, the MIG tool uses the HSTL\_II and DIFF\_HSTL\_II standards with the internal termination (IN\_TERM) attribute chosen in the GUI.

The 7 series FPGA MIG RLDRAM II/RLDRAM 3 design has two clock inputs, the reference clock and the system clock. The reference clock drives the IODELAYCTRL components in the design, while the system clock input is used to create all MIG design clocks that are used to clock 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 more information on clocking architecture, see [Clocking Architecture, page 435](#).

The MIG tool allows you to input the Memory Clock Period and then lists available Input Clock Periods that follow the supported clocking guidelines. Based on these two clock periods selections, the generated MIG core appropriately sets the PLL parameters. The MIG tool enables automatic generation of all supported clocking structures. For information on how to use the MIG tool to set up the desired clocking structure including input clock placement, input clock frequency, and IDELAYCTRL `ref_clk` generation, see [Creating the 7 Series FPGAs RLDRAM II/RLDRAM 3 Memory Design, page 391](#).

## ***Input Clock Guidelines***



---

**IMPORTANT:** *The input system clock cannot be generated internally.*

---

- PLL Guidelines
  - CLKFBOUT\_MULT\_F (M) must be between 1 and 16 inclusive.
  - DIVCLK\_DIVIDE (D, Input Divider) can be any value supported by the PLLE2 parameter.
  - CLKOUT\_DIVIDE (O, Output Divider) must be 2 for 400 MHz and up operation and 4 for below 400 MHz operation.
  - The above settings must ensure the minimum PLL VCO frequency (FVCOMIN) is met. For specifications, see the appropriate DC and Switching Characteristics Data Sheet. The *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] includes the equation for calculating FVCO.
  - The relationship between the input period and the memory period is  $\text{InputPeriod} = (\text{MemoryPeriod} \times M) / (D \times D1)$ .
- The clock input (`sys_clk`) can be input on any CCIO in the column where the memory interface is located; this includes CCIO in banks that do not contain the memory interface, but must be in the same column as the memory interface. The PLL must be located in the bank containing the clock sent to the memory. To route the input clock to the memory interface PLL, the CMT backbone must be used. With the MIG implementation, one spare interconnect on the backbone is available that can be used for this purpose.

- MIG versions 1.4 and later allow this input clocking setup and properly drive the CMT backbone.
- CLOCK\_DEDICATED\_ROUTE = BACKBONE constraint is used to implement CMT backbone, following warning message is expected. It can be ignored safely.

**WARNING: [Place 30-172] Sub-optimal placement for a clock-capable IO pin and PLL pair. The flow will continue as the CLOCK\_DEDICATED\_ROUTE constraint is set to BACKBONE.**

```
u_mig_7series_0/c0_u_clk_ibuf/diff_input_clk.u_ibufg_sys_clk (IBUFDS.O) is locked  
to IOB_X0Y176  
u_mig_7series_0/c0_u_infrastructure/plle2_i (PLLE2_ADV.CLKIN1) is locked to  
PLLE2_ADV_X0Y1  
u_mig_7series_0/c1_u_infrastructure/plle2_i (PLLE2_ADV.CLKIN1) is locked to  
PLLE2_ADV_X0Y5  
.....
```

- For RLDRAM II interfaces that have the memory system input clock (**sys\_clk**) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_HSTL\_I I/O standard (VCCO = 1.5V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_HSTL\_I and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_HSTL\_I CCIO pins.
- For RLDRAM 3 interfaces that have the memory system input clock (**sys\_clk**) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_SSTL12 I/O standard (VCCO = 1.2V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_SSTL12 and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_SSTL12 CCIO pins.
- It is acceptable to have differential inputs such as LVDS and LVDS\_25 in I/O banks that are powered at voltage levels other than the nominal voltages required for the outputs of those standards (1.8V for LVDS outputs, and 2.5V for LVDS\_25 outputs). However, these criteria must be met:
  - a. The optional internal differential termination is not used (DIFF\_TERM = FALSE, which is the default value).
  - b. The differential signals at the input pins meet the VIN requirements in the Recommended Operating Conditions table of the specific device family data sheet.
  - c. The differential signals at the input pins meet the VIDIFF (min) requirements in the corresponding LVDS or LVDS\_25 DC specifications tables of the specific device family data sheet.

**Note:** This might require manually changing DIFF\_TERM parameter located in the top-level module or setting this in the UCF or XDC.

- a. The optional internal differential termination is not used (DIFF\_TERM = FALSE, which is the default value).
- b. The differential signals at the input pins meet the VIN requirements in the Recommended Operating Conditions table of the specific device family data sheet.
- c. The differential signals at the input pins meet the VIDIFF (min) requirements in the corresponding LVDS or LVDS\_25 DC specifications tables of the specific device family data sheet.

One way to accomplish the above criteria is to use an external circuit that both AC-couples and DC-biases the input signals. The figure shows an example circuit for providing an AC-coupled and DC-biased circuit for a differential clock input. RDIFF provides the  $100\Omega$  differential receiver termination because the internal DIFF\_TERM is set to FALSE. To maximize the input noise margin, all RBIAS resistors should be the same value, essentially creating a VCM level of  $VCCO/2$ . Resistors in the 10k to  $100\text{ k}\Omega$  range are recommended. The typical values for the AC coupling capacitors CAC are in the range of 100 nF. All components should be placed physically close to the FPGA inputs.

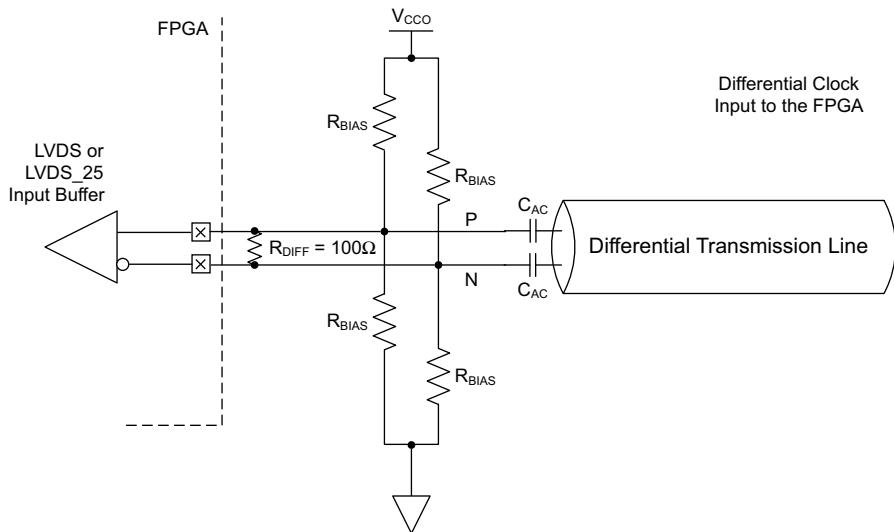


Figure 3-61:

**Note:** The last set of guidelines on differential LVDS inputs are added within the LVDS and LVDS\_25 (Low Voltage Differential Signaling) section of the *7 Series SelectIO Resources User Guide* (UG471) [Ref 2] in the next release of the document. These guidelines are irrespective of Package, Column (HR/HP), or I/O Voltage.

### Sharing sys\_clk between Controllers

MIG 7 series FPGA designs require **sys\_clk** to be in the same I/O bank column as the memory interface to minimize jitter.

- **Interfaces Spanning I/O Columns** – A single **sys\_clk** input cannot drive memory interfaces spanning multiple I/O columns. The input clock input must be in the same column as the memory interface to drive the PLL using the CMT Backbone, which minimizes jitter.
- **Interfaces in Single I/O Column** – If the memory interfaces are entirely contained within the same I/O column, a common **sys\_clk** can be shared among the interfaces. The **sys\_clk** can be input on any CCIO in the column where the memory interfaces are located. This includes CCIO in banks that do not contain the memory interfaces, but must be in the same column as the memory interfaces.

## *Information on Sharing BUFG Clock (phy\_clk)*

The MIG 7 series RLDRAM II/RLDRAM 3 design includes an MMCM which outputs the **phy\_clk** on a BUFG route. It is not possible to share this clock amongst multiple controllers to synchronize the user interfaces. This is not allowed because the timing from the FPGA logic to the PHY Control block must be controlled. This is not possible when the clock is shared amongst multiple controllers. The only option for synchronizing user interfaces amongst multiple controllers is to create an asynchronous FIFO for clock domain transfer.

## *Information on Sync\_Pulse*

The MIG 7 series RLDRAM II/RLDRAM 3 design includes one PLL that generates the necessary design clocks. One of these outputs is the **sync\_pulse**. The sync pulse clock is 1/16 of the **mem\_refclk** frequency and must have a duty cycle distortion of 1/16 or 6.25%. This clock is distributed across the low skew clock backbone and keeps all PHASER\_IN/\_OUT and PHY\_Control blocks in sync with each other. The signal is sampled by the **mem\_refclk** in both the PHASER\_INs/\_OUTs and PHY\_Control blocks. The phase, frequency, and duty cycle of the **sync\_pulse** is chosen to provide the greatest setup and hold margin across PVT.

---

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.

The RLDRAM II and RLDRAM 3 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

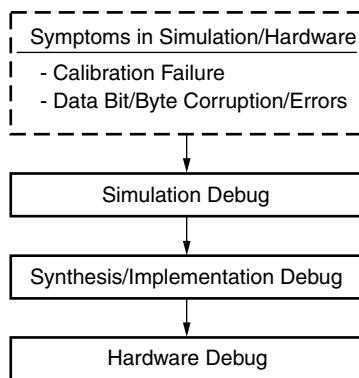
- Using the RLDRAM II/RLDRAM 3 physical layer to debug board-level issues
- General board-level debug techniques

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 3-62](#) shows the overall flow for debugging problems associated with these two general types of issues.



*Figure 3-62:*

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/RLDRAM 3 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 Vivado logic analyzer feature. Selecting this option port maps the debug signals to VIO modules of the Vivado logic analyzer feature in the design top module.

## Vivado Design Suite Debug Feature

The Vivado Design Suite debug feature inserts logic analyzer, bus analyzer, and VIO software cores directly into the design. Supported versions of ILA and VIO are 3.0. The debug feature also allows you to set trigger conditions to capture application and MIG debug signals in hardware. Captured signals can be analyzed through the Vivado logic analyzer feature. For more information about the Vivado logic analyzer, software is available in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 16].



**IMPORTANT:** *The Integrated Logic Analyzer (ILA) operates on a synchronous clock and cannot be triggered during reset. Instead, set the trigger on an ILA signal to look for a rising edge ("R") or falling edge ("F") with the radix value of the signal set to "Binary." With this trigger setting, the trigger can be armed. When the reset is applied and released, the trigger captures the desired ILA results.*

Figure 3-63 shows the debug flow for simulation.

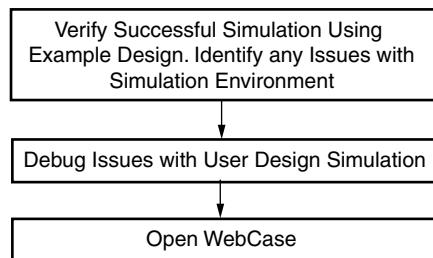


Figure 3-63:

### 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. Successful completion of this example design simulation verifies a proper simulation environment.

The Questa Advanced Simulator, Vivado Simulator, IES, and VCS simulation tools are used for verification of MIG IP core at each software release. Script files to run simulations with IES and VCS simulators are generated in MIG generated output. Simulations using Questa Advanced Simulator and Vivado simulators can be done through Vivado Tcl Console commands or in Vivado IDE.



**IMPORTANT:** *Other simulation tools can be used for MIG IP core simulation but are not specifically verified by Xilinx.*

To run the simulation, go to this directory:

```
<project_dir>/<Component_Name>_ex/imports
```

For a project created with the name set as **project\_1** and the Component Name entered in Vivado IDE as **mig\_7series\_0**, go to the directory as follows:

```
project_1/mig_7series_0_ex/imports
```

IES and VCS simulation scripts are meant to be executed only in Linux operating systems.

The **ies\_run.sh** and **vcs\_run.sh** files are the executable files for running simulations using IES and VCS simulators respectively. Library files should be added to the **ies\_run.sh** and **vcs\_run.sh** files respectively. See the **readme.txt** file for details regarding simulations using IES and VCS.

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings** ([Figure 3-64](#)).

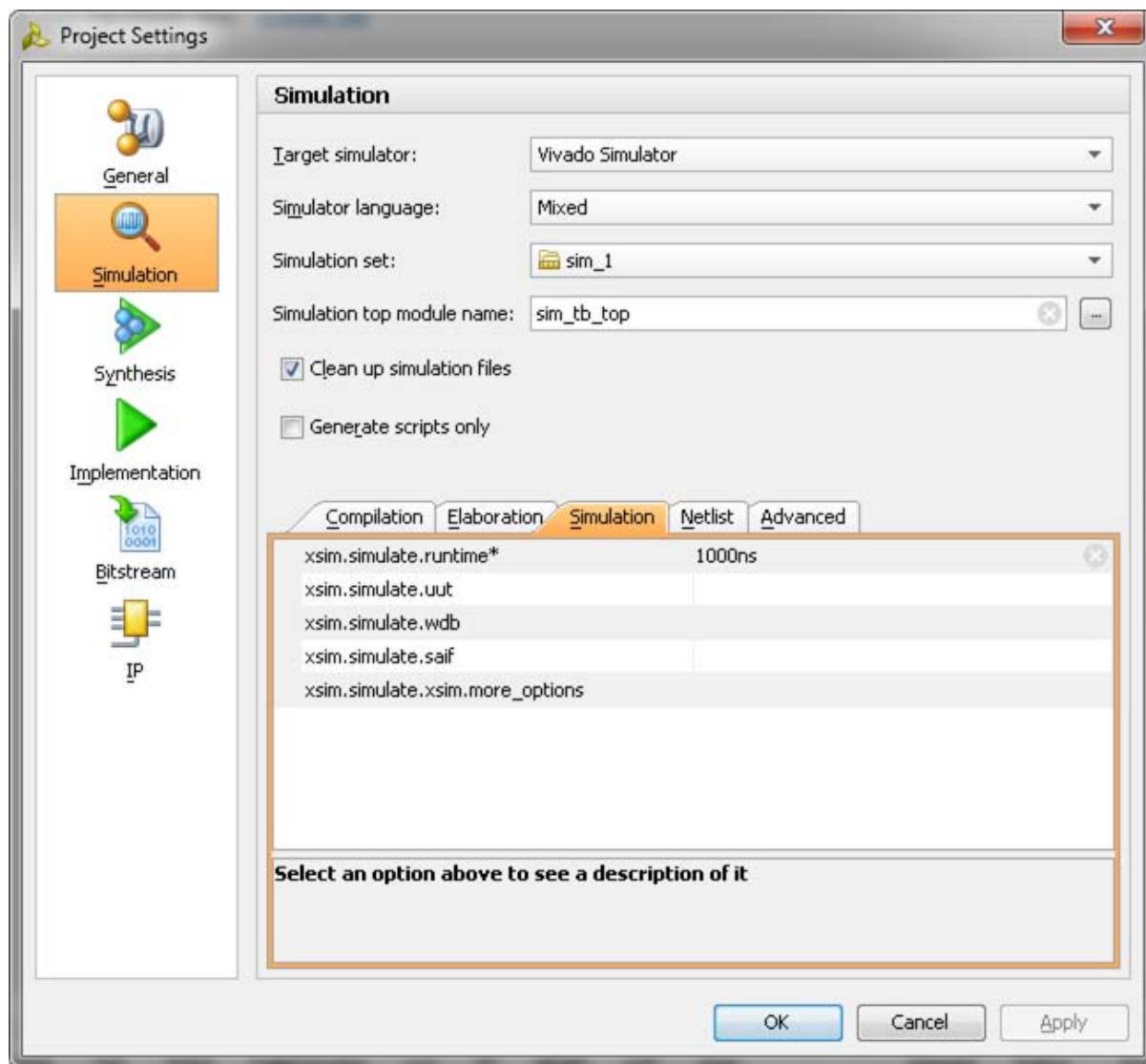
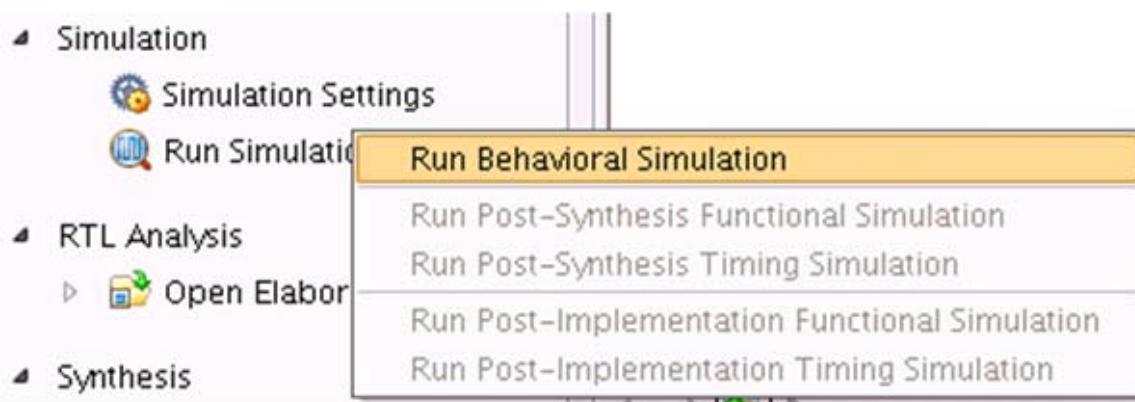


Figure 3-64:

- Under the **Simulation** tab as shown in [Figure 3-64](#), set the **xsim.simulate.runtime** as 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms). Apply the settings and select **OK**.

3. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 3-65](#).



*Figure 3-65:*

**Note:** RLDRAM 3 memory model has System Verilog constructs, which are not supported by Vivado Simulator.

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Questa Advanced Simulator/ModelSim.
  - a. Browse to the **Compiled libraries location** and set the path on **Compiled libraries location** option.
  - b. Under the **Simulation** tab, set the **modelsim.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms), set **modelsim.simulate.vsim.more\_options** to **-novopt** as shown in [Figure 3-64](#).
  - c. Under **Compilation** tab, set **modelsim.compile.vlog.more\_options** to **-sv** (only for RLDRAM 3 designs).
3. Apply the settings and select **OK**.

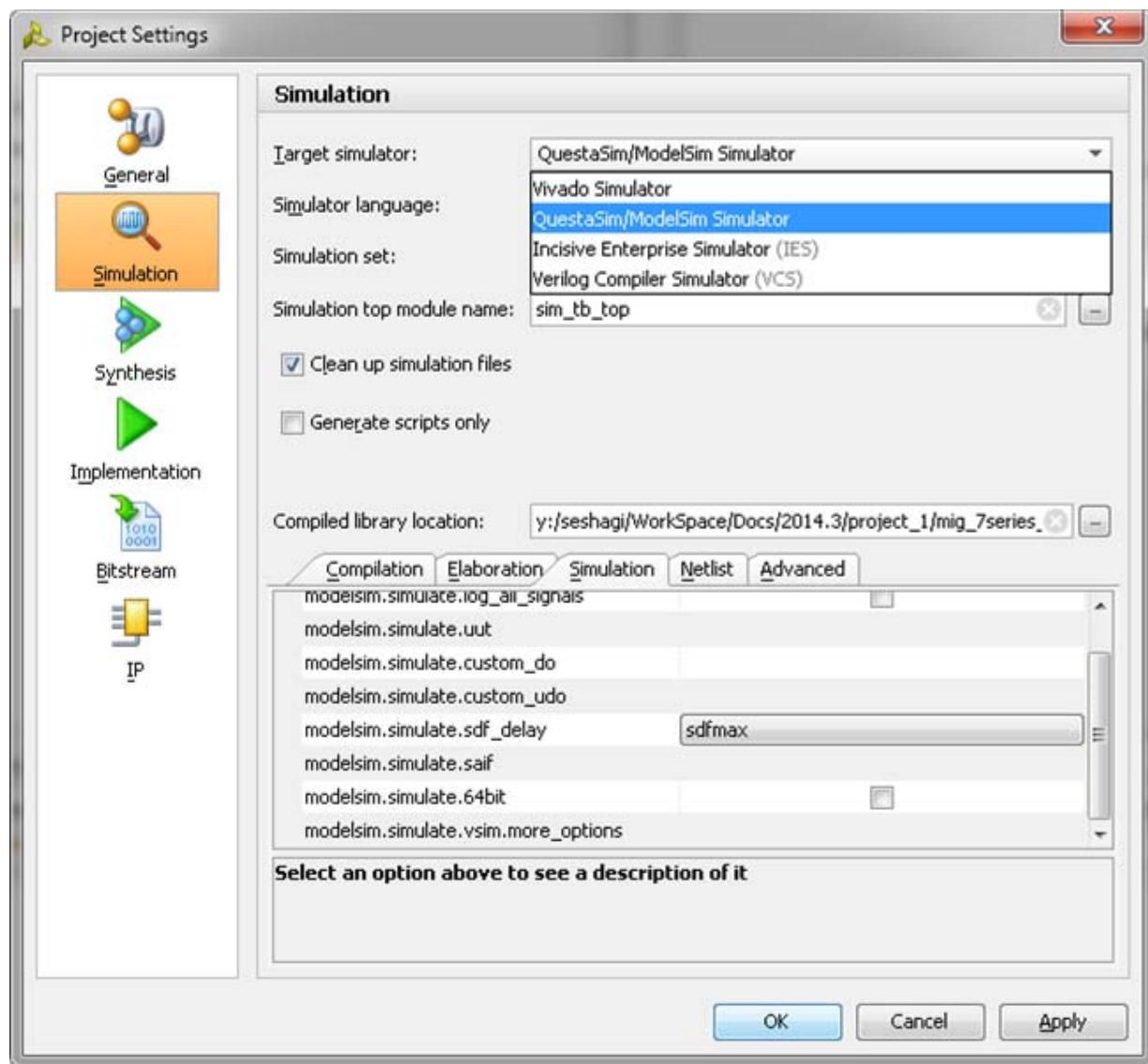


Figure 3-66:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 3-65](#).
  5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG 900)* [\[Ref 8\]](#).
- 
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
  2. Select **Target simulator** as Verilog Compiler Simulator (VCS).

- a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **vcs.compile.vlogan.more\_options** to **-sverilog**.
  - c. Under the **Simulation** tab, set the **vcs.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after a certain period of time which is less than 1 ms) as shown in [Figure 3-67](#).
3. Apply the settings and select **OK**.

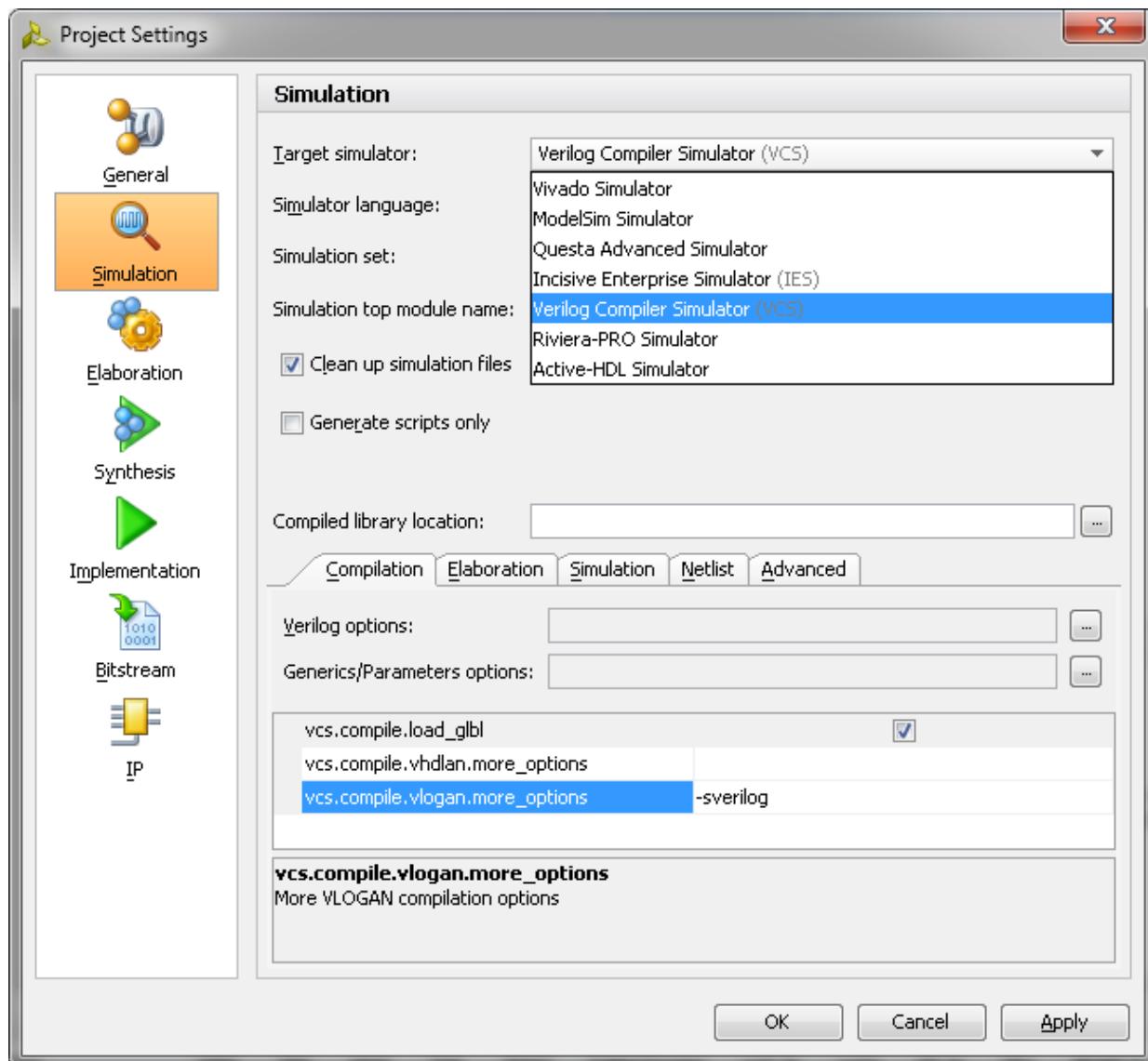


Figure 3-67:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 3-65](#).

5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [Ref 8].
  
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
  - a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **ies.compile.ncvlog.more\_options** to **-sv**.
  - c. Under the **Elaboration** tab, set the **ies.elaborate.ncelab.more\_options** to **-namemap\_mixgen**.
  - d. Under the **Simulation** tab, set the **ies.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time which is less than 1 ms) as shown in [Figure 3-68](#).

3. Apply the settings and select **OK**.

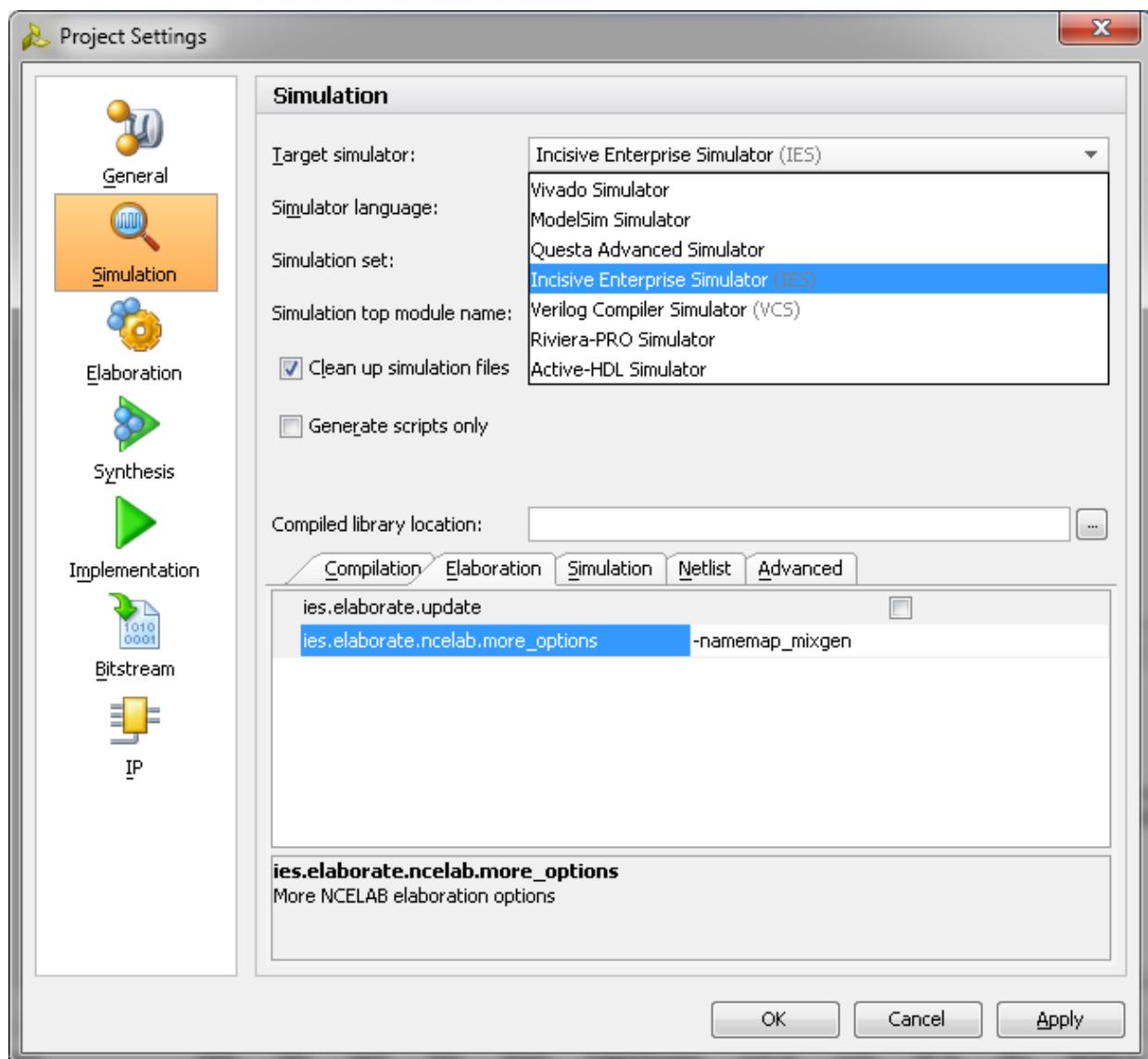


Figure 3-68:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 3-65](#).
5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [\[Ref 8\]](#).

For detailed information on setting up Xilinx libraries, see COMPXLIB in the *Command Line Tools User Guide (UG628)* [\[Ref 17\]](#) and the *Synthesis and Simulation Design Guide (UG626)* [\[Ref 18\]](#). For simulator tool support, see the *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions Data Sheet (DS176)* [\[Ref 1\]](#).

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 3-26](#) shows the signals and parameters of interest, respectively, during simulation.

*Table 3-26:*

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_cmd	This signal indicates if you requests a write or a read command. 2'b00 = Write Command 2'b01 = Read Command 2'b10 = NOP 2'b11 = NOP
user_addr	This is the address location for the current command.
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.

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 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, see [Physical Interface, page 437](#).

The first stage performs per-bit read leveling calibration. The data pattern used during this stage is `0_F_0_F_0_F_F_0`. The data pattern is first written to the memory, as shown in Figure 3-69.

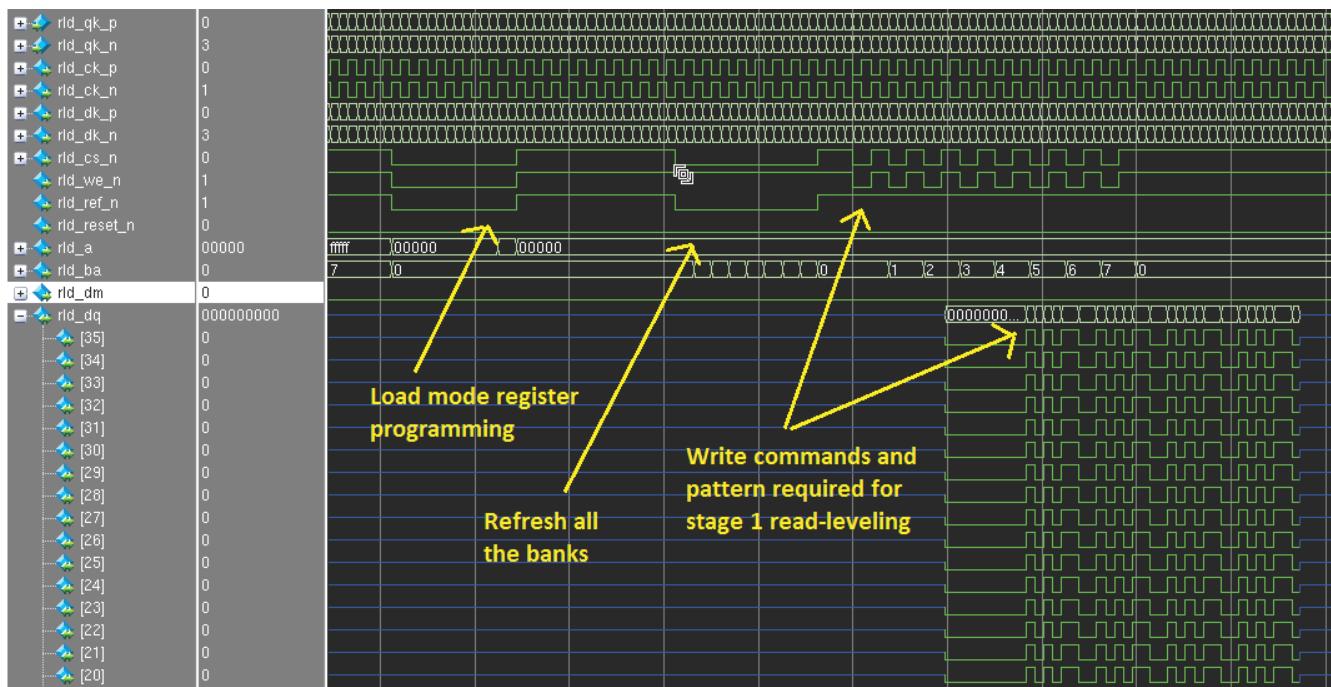


Figure 3-69:

This pattern is then continuously read back while the calibration is completed, as shown in Figure 3-70.

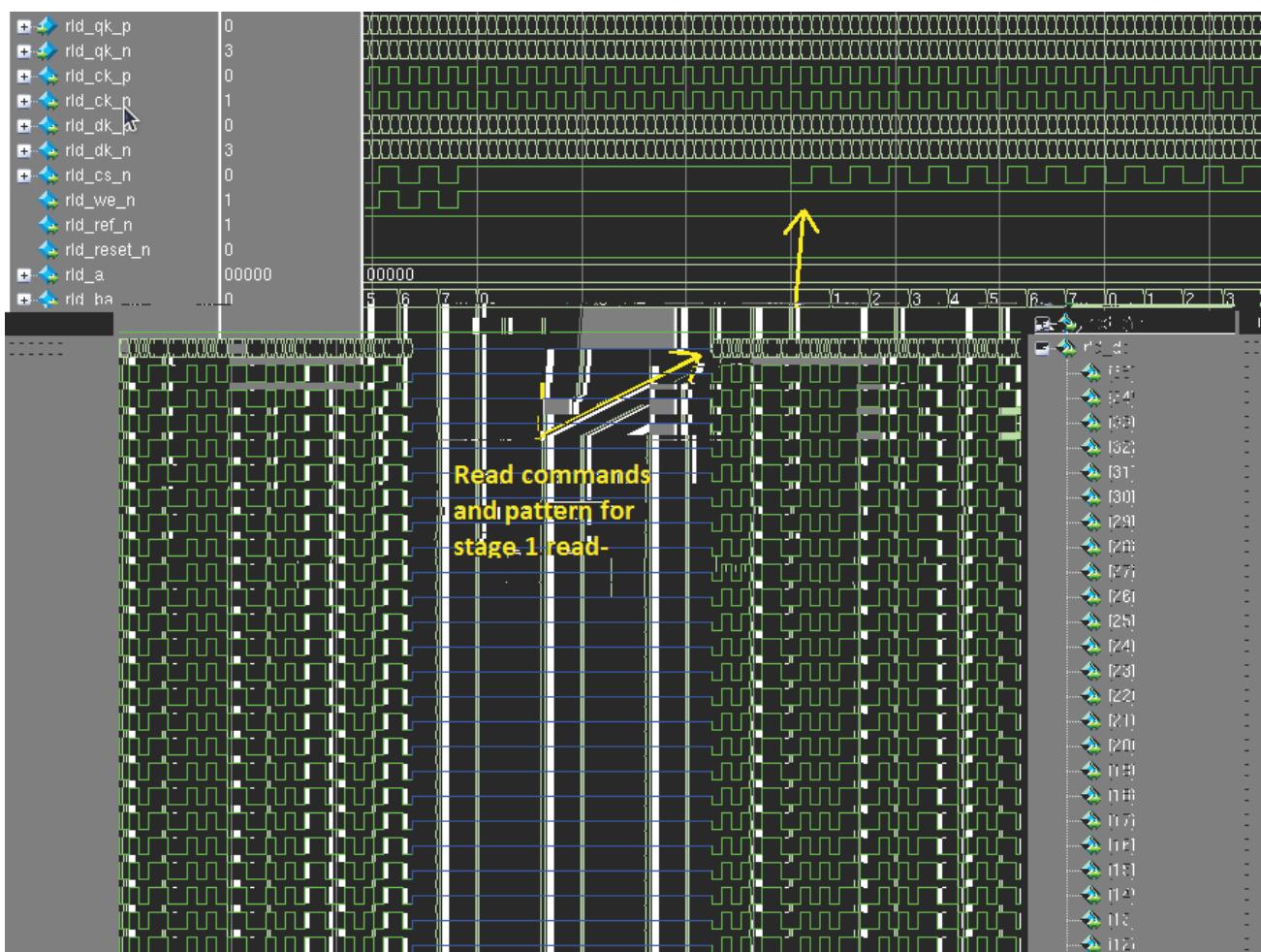


Figure 3-70:

The second stage performs an alignment to ensure data is returned in the correct order. The length of the data pattern depends on the ratio of the memory clock to the FPGA logic clock. For RLDRAM II, the data pattern of A\_5\_O\_F is first written to the memory and continuously read back and adjusted internally if required.

For RLDRAM 3, the data pattern of A\_5\_O\_F\_9\_6\_D\_2 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 3-71.

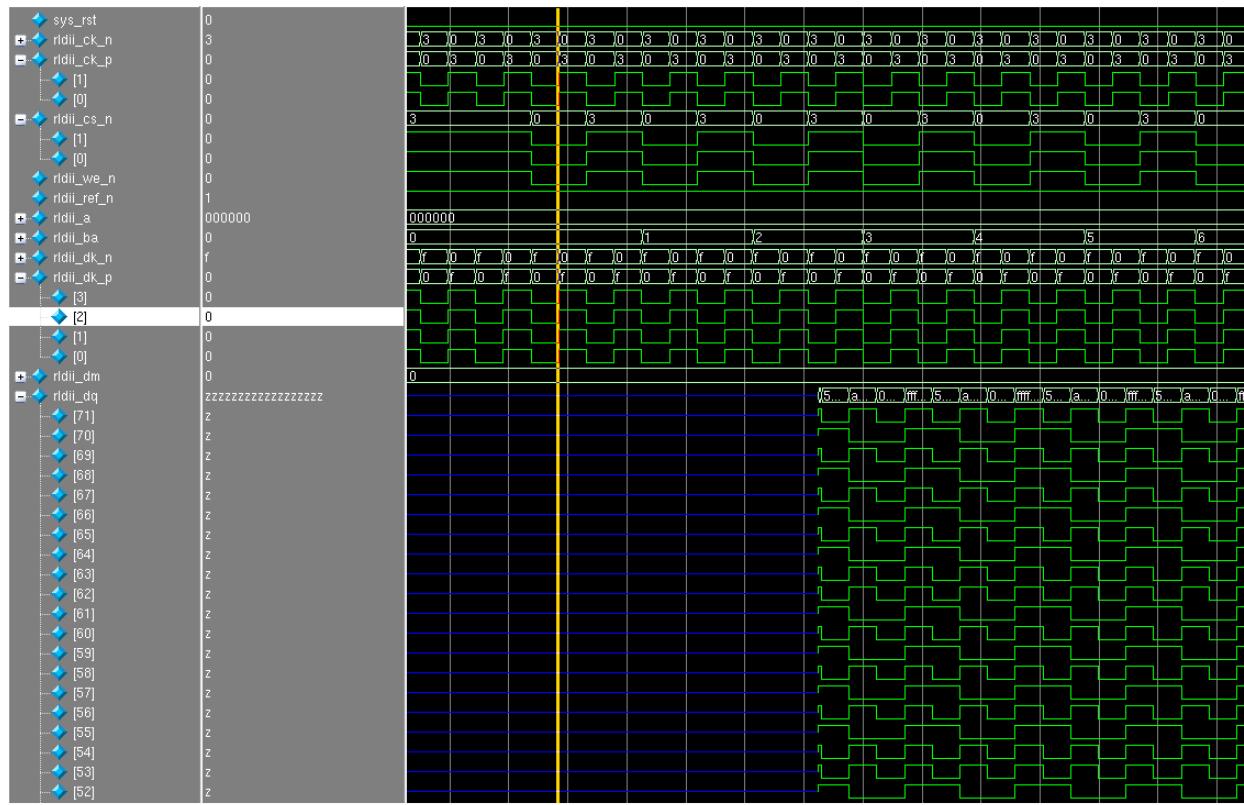


Figure 3-71:

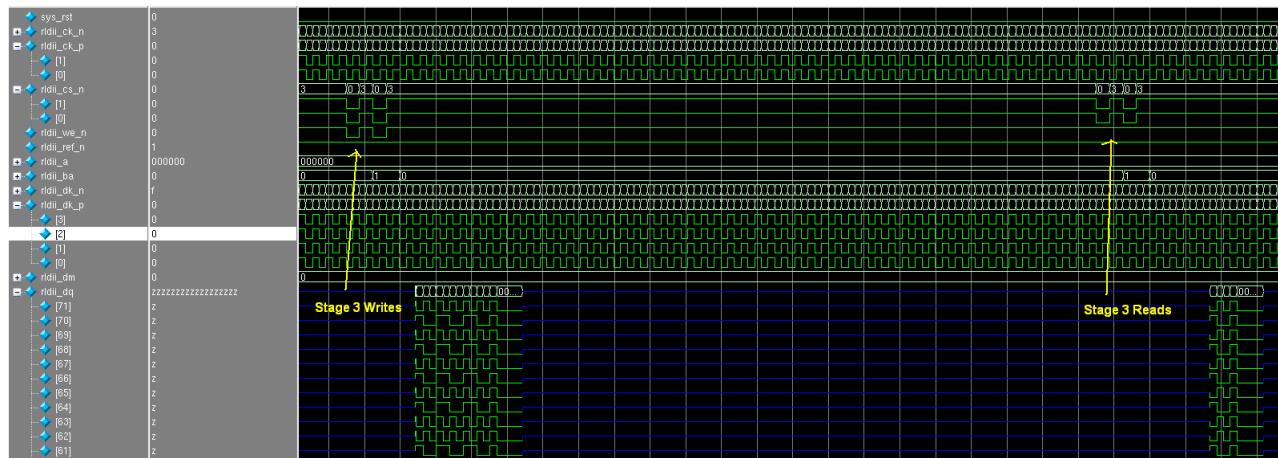
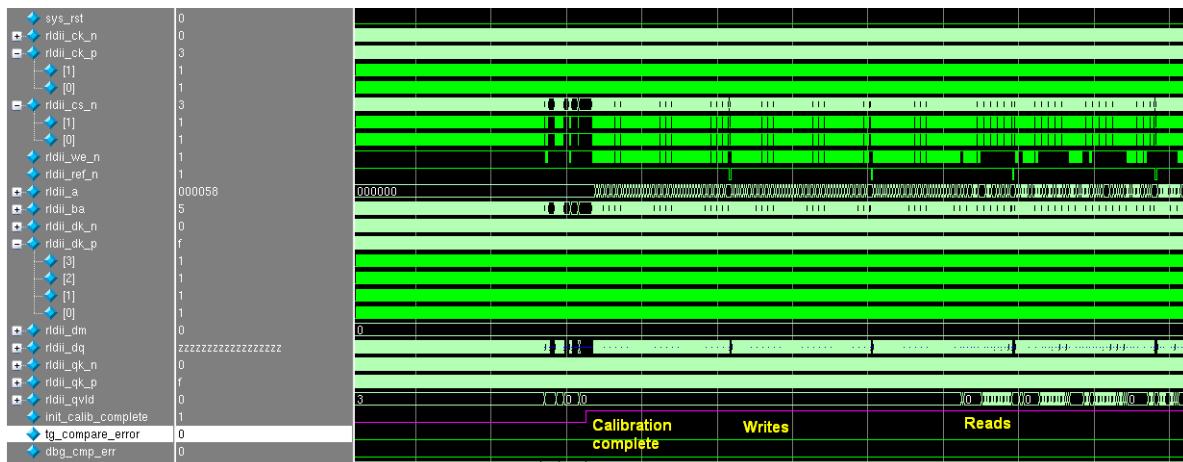


Figure 3-72:

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 the third stage calibration completes, `init_calib_complete` is asserted, signifying successful completion of the calibration process.

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 3-73](#) shows a successful implementation of the test bench with no assertions on error.



*Figure 3-73:*

When sending write and read commands, you must properly assert and deassert the corresponding UI inputs. See [Client Interface, page 428](#) and [Interfacing with the Core through the Client Interface, page 430](#) 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 RLDI interface, it is necessary to pull the UI signals into the simulation waveform.

In the Questa Advanced Simulator Instance window, highlight `u_ip_top` to display the necessary UI signals in the Objects window, as shown in [Figure 3-74](#). Highlight the user interface signals noted in [Table 3-26, page 492](#), right-click, and select **Add > To Wave > Selected Signals**.

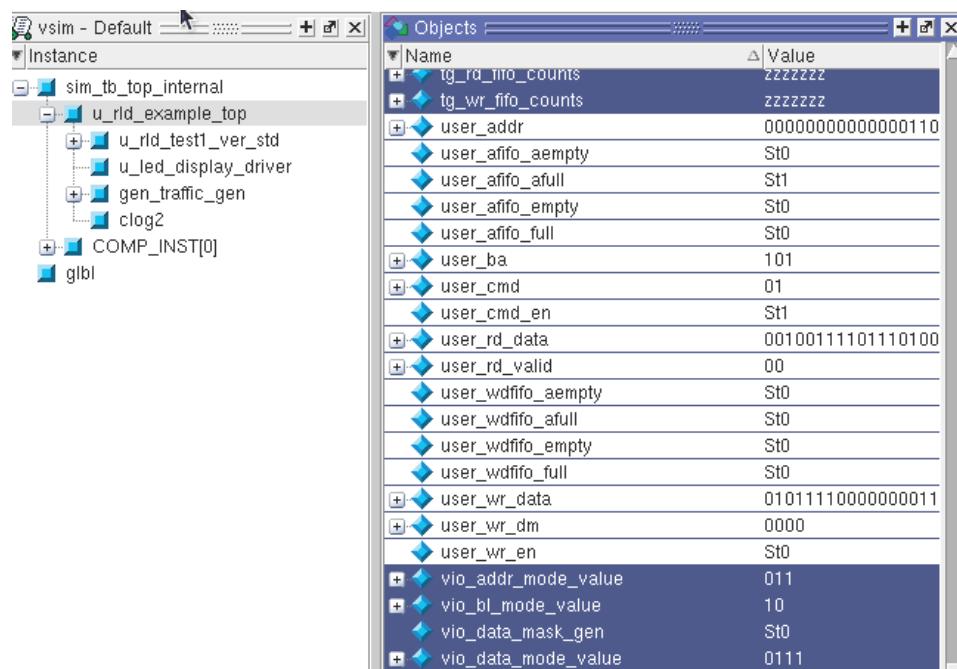


Figure 3-74:

[Figure 3-75](#) and [Figure 3-76](#) show example waveforms of a write and read on both the user interface.

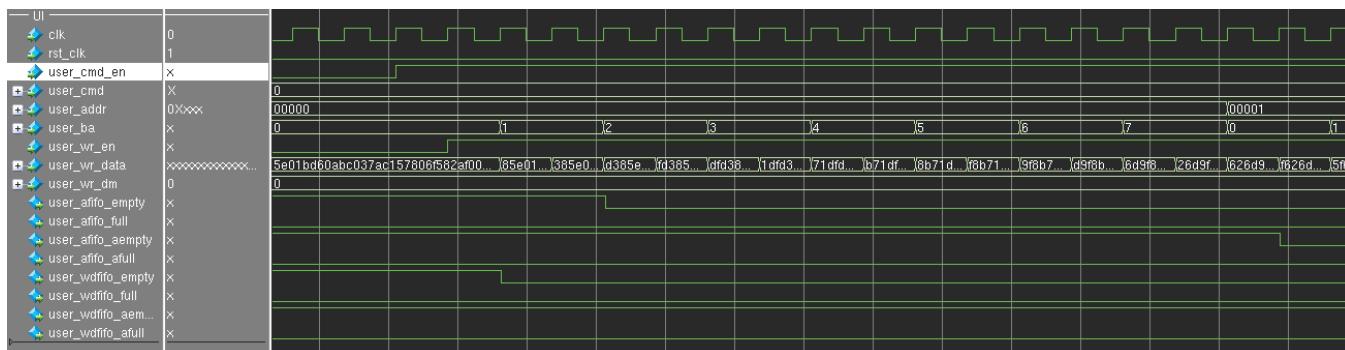


Figure 3-75:

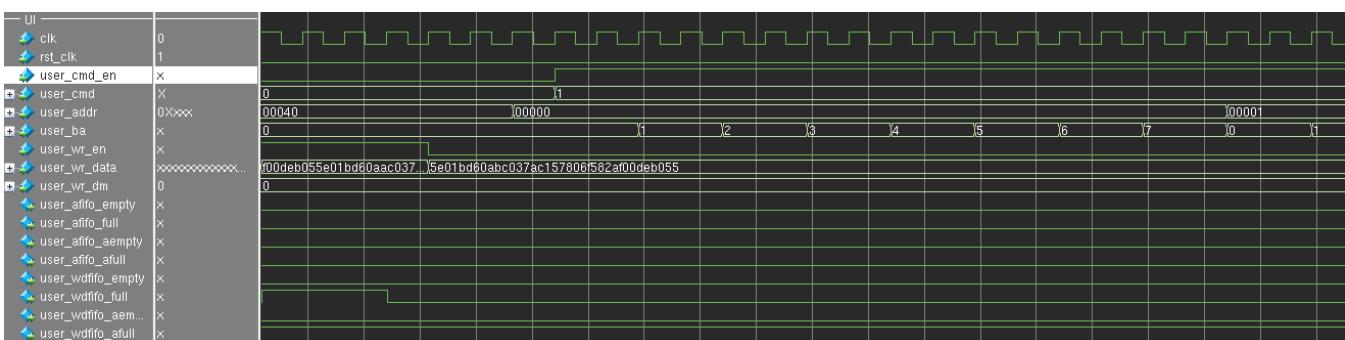


Figure 3-76:

Figure 3-77 shows the debug flow for synthesis and implementation.

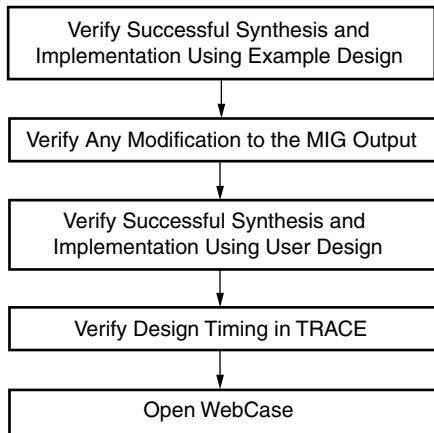


Figure 3-77:

---

**IMPORTANT:** *The standard synthesis flow for Synplify is not supported for the core.*

---



### ***Verify Successful Synthesis and Implementation***

The example design and user design generated by the MIG tool include synthesis/implementation script files and **.xdc** files. These files should be used to properly synthesize and implement the targeted design and generate a working bitstream.

### ***Verify Modifications to the MIG Tool Output***

The MIG tool allows you to select the FPGA banks for the memory interface signals. Based on the banks selected, the MIG tool outputs a XDC 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/RLDRAM 3 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 your specific application logic.

Any timing violations that are encountered must be isolated. The timing report output by TRACE (.twx/.twr) should be analyzed to determine if the failing paths exist in the MIG tool RLDRAM II design or the UI (backend application) to the MIG tool design. If failures are encountered, you must ensure the build options (that is, XST, MAP, PAR) specified in the file are used.

If failures still exist, Xilinx has many resources available to aid in closing timing. The PlanAhead™ tool [Ref 19] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* (UG612) [Ref 15] provides valuable information on all available Xilinx constraints.

Figure 3-78 shows the debug flow for hardware.

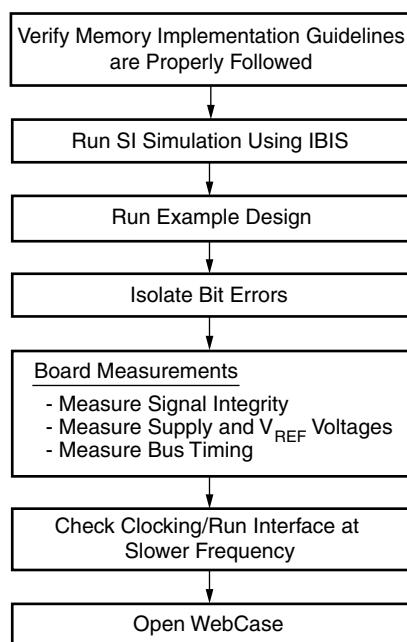


Figure 3-78:

## Clocking

The external clock source should be measured to ensure frequency, stability (jitter), and usage of the expected FPGA pin. You 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

You 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.

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* (UG199) [Ref 20] 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.

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 Vivado logic analyzer feature should be used to analyze the behavior of MIG tool core signals. For more information about the Vivado logic analyzer, software is available in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 16].

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 issue.
- 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. Another option is to adjust the PHASER\_OUT timing using the fine\_adjust feature of the PHASER to adjust output timing.
  - 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.

The Debug port is a set of input and output signals that either provide status (outputs) or allow you to make adjustments as the design is operating (inputs). When generating the RLDRAM II/RLDRAM 3 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 3-27](#) 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 3-27:*

dbg_phy_status[0]	~rst_wr_clk	FPGA logic reset based on PLL lock and system input reset	If this signal is deasserted, 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/RLDRAM 3 initialization sequence is complete	N/A <sup>(1)</sup>
dbg_phy_status[3]	rtr_cal_start	(RLDRAM 3 only) Read training register stage 1 read calibration start signal	N/A
dbg_phy_status[4]	rtr_cal_done	(RLDRAM 3 only) Read training register stage 1 read calibration is complete	N/A
dbg_phy_status[5]	wrcal_start	Write Calibration start signal	N/A
dbg_phy_status[8:6]	dbg_wrcal_done	Write Calibration stage complete	Check the results of the read training register stage 1 calibration. See <a href="#">Write Calibration, page 453</a> .
dbg_phy_status[9]	rdlvl_stg1_start	Stage 1 read calibration start signal	Check results of write calibration
dbg_phy_status[10]	rdlvl_cal_done	Stage 1 read calibration is complete	N/A

(Cont'd)

Table 3-27:

dbg_phy_status[11]	edge_adv_cal_start	Edge Advance calibration start signal	N/A
dbg_phy_status[12]	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. Check the results of write calibration (if enabled).
dbg_phy_status[13]	init_cal_done	Latency calibration completed	Check which byte lane failed and check the margin found during read training stage 1 and write calibration for the byte lane that fails.
dbg_phy_status[14]	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 3-28](#).

Table 3-28:

dbg_phy_cmd_n[nCK_PER_CLK × 3 – 1:0]	Output	This active-Low signal is the internal command that is sent to the memory used for debug with the Vivado logic analyzer feature.
dbg_phy_addr[nCK_PER_CLK × RLD_ADDR_WIDTH × 2 – 1:0]	Output	Address being accessed for the commands given with <b>dbg_phy_cmd_n</b> .
dbg_phy_ba[nCK_PER_CLK × BANK_WIDTH × 2 – 1:0]	Output	Control bank address bus used for debug with the Vivado logic analyzer feature.

Table 3-28:

(Cont'd)

dbg_phy_wr_data[nCK_PER_CLK × 2 × DATA_WIDTH – 1:0]	Output	Data being written that is used for debug with the Vivado logic analyzer feature.
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.
dbg_byte_sel[CQ_BITS – 1:0]	Input	This input selects the corresponding byte lane and you set the phaser/IDELAY tap controls.
dbg_bit_sel[Q_BITS – 1:0]	Input	This input selects the corresponding bit lane and you set the IDELAY tap controls. It also controls which read data signals are presented to you for debug (dbg_rd_data_rd and dbg_rd_data_fd).
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.

Table 3-28:

(Cont'd)

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.
dbg_rd_stage1_rtr_error[N_DATA_LANES - 1:0]	Output	(RLDRAM 3 only) Per byte lane error signal indicating valid window not found during read training register stage 1 read calibration.
dbg_rd_stage1_error[N_DATA_LANES - 1:0]	Output	Per byte lane error signal indicating valid window not found during stage 1 read calibration.
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.
dbg_rd_data_rd[nCK_PER_CLK × 9 - 1:0]	Output	This bus shows the captured output of the rising data for a single byte lane, selected using dbg_byte_sel.
dbg_rd_data_fd[nCK_PER_CLK × 9 - 1:0]	Output	This bus shows the captured output of the falling data for a single byte lane, selected using dbg_byte_sel.
dbg_rd_valid	Output	Read data valid signal that aligns with the dbg_rd_data_rd and dbg_rd_data_fd.
dbg_wrcal_sel_stg[1:0]	Input	Selects which stage of write calibration to output: dbg_wrcal_po_first_edge, dbg_wrcal_po_second_edge, or dbg_wrcal_po_final.
dbg_wrcal[63:0]	Output	General Debug port for write calibration

Table 3-28:

(Cont'd)

dbg_wrcal_done[2:0]	Output	Indicates stage of write calibration completed.
dbg_wrcal_po_first_edge[5:0]	Output	First edge of write calibration window found for the selected byte lane (using dbg_byte_sel). To select a given stage of calibration, use dbg_wrcal_sel_stg, 2'b01 is for byte lanes with a DK clock, and 2'b10 is for byte lanes without a DK clock.
dbg_wrcal_po_second_edge[5:0]	Output	Second edge of write calibration window found for the selected byte lane (using dbg_byte_sel). To select a given stage of calibration, use dbg_wrcal_sel_stg, 2'b01 is for byte lanes with a DK clock, and 2'b10 is for byte lanes without a DK clock.
dbg_wrcal_po_final[5:0]	Output	Final tap setting for write calibration for the selected byte lane (using dbg_byte_sel). To select a given stage of calibration, use dbg_wrcal_sel_stg, 2'b01 is for byte lanes with a DK clock, and 2'b10 is for byte lanes without a DK clock.

Table 3-29 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 3-29:

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	Reserved

**Table 3-30** 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 3-30:*

<code>dbg_rd_stage1_cal[2:0]</code>	<code>sm_r</code>	Read level main state machine.
<code>dbg_rd_stage1_cal[7:6]</code>	<code>seq_sm_r</code>	Read level sequence state bits.
<code>dbg_rd_stage1_cal[14:12]</code>	<code>rdlvl_work_lane_r</code>	Lane currently undergoing read level calibration.
<code>dbg_rd_stage1_cal[15]</code>	<code>rdlvl_stg1_start</code>	Write side signal causing read level block to start.
<code>dbg_rd_stage1_cal[16]</code>	<code>rdlvl_stg1_done</code>	Read level block signals completion.
<code>dbg_rd_stage1_cal[17]</code>	<code>rdlvl_stg1_start</code>	Write side signal causing read level to copy first lane result across all lanes.
<code>dbg_rd_stage1_cal[25:18]</code>	<code>rdlvl_stg1_cal_bytes_r</code>	Lanes for which write side is requesting calibration.
<code>dbg_rd_stage1_cal[31]</code>	<code>cmplx_rdcal_start</code>	Write side signal causing read level to do complex cal.
<code>dbg_rd_stage1_cal[32]</code>	<code>cmplx_rd_data_valid</code>	Write side signal informing read level that complex read data is valid.
<code>dbg_rd_stage1_cal[48:41]</code>	<code>rd_data_comp_r</code>	Per byte comparison results for complex calibration.
<code>dbg_rd_stage1_cal[56:49]</code>	<code>iserdes_comp_r</code>	Per byte comparison results for simple calibration.
<code>dbg_rd_stage1_cal[57]</code>	<code>rdlvl_lane_match</code>	Overall comparison result for both simple and complex.
<code>dbg_rd_stage1_cal[66:61]</code>	<code>largest_left_edge</code>	Phaser in taps when the right most left edge was found.
<code>dbg_rd_stage1_cal[72:67]</code>	<code>smallest_right_edge</code>	Phaser in taps when the left most right edge was found.
<code>dbg_rd_stage1_cal[78:73]</code>	<code>mem_out_dec</code>	Output of static compensation ROM.
<code>dbg_rd_stage1_cal[81]</code>	<code>rdlvl_pi_stg2_f_incdec</code>	Controls directing of phaser in stepping.
<code>dbg_rd_stage1_cal[82]</code>	<code>rdlvl_pi_en_stg2_f</code>	Phaser in step command.
<code>dbg_rd_stage1_cal[85:83]</code>	<code>pi_lane_r</code>	Lane to which phaser in commands apply.
<code>dbg_rd_stage1_cal[91]</code>	<code>prev_match_r</code>	Previous sample matched.
<code>dbg_rd_stage1_cal[96:92]</code>	<code>match_out_r</code>	idelay of last detected invalid to valid match transition.
<code>dbg_rd_stage1_cal[102:97]</code>	<code>samp_cnt_r</code>	Sample counter.
<code>dbg_rd_stage1_cal[108:103]</code>	<code>samps_match_r</code>	Cumulative sample match count.
<code>dbg_rd_stage1_cal[109]</code>	<code>samp_result_held_r</code>	Result from previous sample cycle.
<code>dbg_rd_stage1_cal[154+:40]</code>	<code>simp_dlyval_r</code>	Five bits per lane dlyval results for simple pattern.
<code>dbg_rd_stage1_cal[194+:48]</code>	<code>simp_left_r</code>	Six bits per lane left results for simple pattern.
<code>dbg_rd_stage1_cal[194+:48]</code>	<code>simp_right_r</code>	Six bits per lane right results for simple pattern.
<code>dbg_rd_stage1_cal[194+:48]</code>	<code>simp_center_r</code>	Six bits per lane center results for simple pattern.
<code>dbg_rd_stage1_cal[378+:48]</code>	<code>cmplx_left_r</code>	Six bits per lane left results for complex pattern.
<code>dbg_rd_stage1_cal[426+:48]</code>	<code>cmplx_right_r</code>	Six bits per lane right results for complex pattern.

Table 3-30:

(Cont'd)

dbg_rd_stage1_cal[474+:48]	cmplx_center_r	Six bits per lane center results for complex pattern.
dbg_rd_stage1_cal[682+:48]	simp_left_63	Left edge result is 63 for simple pattern, one bit per lane.
dbg_rd_stage1_cal[690+:48]	cmplx_left_63	Left edge result is 63 for complex pattern, one bit per lane.
dbg_rd_stage1_cal[698+:48]	simp_right_63	Right edge result is 63 for simple pattern, one bit per lane.
dbg_rd_stage1_cal[706+:48]	cmplx_right_63	Right edge result is 63 for complex pattern, one bit per lane.
dbg_rd_stage1_cal[522+:72]	rd_data_lane_r	Aligned PHY data for lane currently undergoing calibration.
dbg_rd_stage1_cal[594+:72]	iserdes_lane_r	Raw PHY data for lane currently undergoing calibration.
dbg_rd_stage1_cal[714+:72]	cmplx_rd_burst_bytes	Complex data to compare against memory read data.
dbg_rd_stage1_cal[786+:9]	bit_comp	Cumulative compare per bit.
dbg_rd_stage1_cal[795+:8]	simp_min_eye_r	Minimum eye detected per lane simple pattern.
dbg_rd_stage1_cal[803+:8]	cmplx_min_eye_r	Minimum eye detected per lane complex pattern.

Table 3-31 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 3-31:

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 for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[12]	fd0_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[13]	rd1_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt)

Table 3-31:

(Cont'd)

dbg_stage2_cal[14]	fd1_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[15]	phase_vld	Valid data is seen for the particular byte for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[16]	rd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[17]	fd0_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[18]	rd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[19]	fd1_bslip_vld	Indicates valid ISERDES read data requiring bitslip for the byte being calibrated (indicated by byte_cnt)
dbg_stage2_cal[20]	phase_bslip_vld	Valid data is seen when bitslip applied to read data for the byte being calibrated (indicated by byte_cnt)
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. Only used for nCK_PER_CLK == 2.
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[30]	bitslip	FPGA logic bitslip control signal, indicates when the logic shifts the data alignment. Only used for nCK_PER_CLK == 4.
dbg_stage2_cal[31]	rd2_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[32]	fd2_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[33]	rd3_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[34]	fd3_vld	Indicates valid ISERDES read data for the byte being calibrated (indicated by byte_cnt). Only valid for nCK_PER_CLK == 4.
dbg_stage2_cal[35]	latency_measured[1]	Indicates latency has been measured for byte lane 1
dbg_stage2_cal[36]	latency_measured[2]	Indicates latency has been measured for byte lane 2
dbg_stage2_cal[37]	latency_measured[3]	Indicates latency has been measured for byte lane 3

Table 3-31:

(Cont'd)

dbg_stage2_cal[38]	error_adj_latency	Indicates error when target PHY_LATENCY cannot be achieved
dbg_stage2_cal[127:39]	Reserved	Reserved

Table 3-32 indicates the mapping between bits within the **dbg\_wr\_cal** bus and debug signals in the PHY.

Table 3-32:

dbg_wrcal[3:0]	write_cal_cs	State machine current state register
dbg_wrcal[4]	data_valid_r	Data is valid across data_valid_cnt FPGA logic clock cycles
dbg_wrcal[5]	first_edge_found	Flag to indicate first edge is found
dbg_wrcal[6]	second_edge_found	Flag to indicate second edge is found
dbg_wrcal[7]	rdlvl_timeout_error	Flag to indicate timeout error to ensure enough time given to stage 2 edge advanced calibration so you can sample the results of a given byte lane.
dbg_wrcal[8]	inc_byte_lane_cnt	Flag to increment byte lane counter
dbg_wrcal[14:9]	po_fine_taps	PHASER_OUT current tap setting
dbg_wrcal[20:15]	po_fine_first_edge	PHASER_OUT first edge tap
dbg_wrcal[26:21]	po_fine_second_edge	PHASER_OUT second edge tap
dbg_wrcal[27]	stg2_eod	PHASER_OUT stage 2 end of delay
dbg_wrcal[28]	stg3_eod	PHASER_OUT stage 3 end of delay
dbg_wrcal[37:29]	po_counter_read_val	PHASER_OUT counter value from the PHY
dbg_wrcal[40:38]	wrcal_stg	Flag to indicate which stage of write calibration is currently running
dbg_wrcal[41]	record_po_taps	Flag to record a given PHASER_OUT value
dbg_wrcal[42]	data_valid	Instantaneous data valid check for a given byte lane
dbg_wrcal[48:43]	wrcal_byte_sel	Byte lane counter
dbg_wrcal[49]	window_valid	When first edge and second edge are found, a check is done to ensure the window is larger than a set size. If too small and first/second edges cleared, this bit keeps going.
dbg_wrcal[54:50]	data_valid_cnt	Counter used to check multiple read samples to ensure data is valid
dbg_wrcal[60:55]	po_fine_prev_taps	"Previous" counter to record direction so you know which direction to move to when an edge found.
dbg_wrcal[61]	first_edge_eod	First edge not a true edge, hit the limit of the PHASER_OUT taps

Table 3-32:

(Cont'd)

dbg_wrcal[62]	second_edge_eod	Second edge not a true edge, hit the limit of the PHASER_OUT taps
dbg_wrcal[63]	po_fine_underflow_check, po_fine_overflow_check	Check for underflow or overflow on the internal PHASER_OUT tap counter.

### 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 the Vivado hardware session and program the FPGA under test with generated BIT and LTX files.
2. 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).
3. To measure margin with PRBS8 pattern, set VIO signals with the listed values in the **traffic\_gen\_top** instance in example\_top:

```

vio_modify_enable = 'd1

vio_data_mode_value = 'd7

vio_addr_mode_value = 'd3

vio_instr_mode_value = 'd4

vio_bl_mode_value = 'd2

vio_fixed_bl_value = 'd128

vio_fixed_instr_value = 'd1

vio_data_mask_gen = 'd0

```

4. Assert **vio\_dbg\_clear\_error** or system reset.
5. Select a given byte lane using **dbg\_byte\_sel**.
6. Observe the tap values on PHASER\_IN for the selected byte lane using **dbg\_pi\_counter\_read\_val**.

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 count to reach one side of the window for the entire byte lane.
8. Decrease the tap values on PHASER\_IN using **dbg\_pi\_f\_dec** back to the starting value.
9. Clear the error recorded previously by asserting **vio\_dbg\_clear\_error**.
10. Decrement the PHASER\_IN taps using **dbg\_pi\_f\_dec** to find the other edge of the window until another error occurs (**tg\_compare\_error** should be asserted).
11. Record those results, return the PHASER\_IN taps to the starting location and clear the error again (**vio\_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 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 issue bits or bytes, but it can be difficult to step through an entire interface looking for issues. For this reason, the RLDRAM II/RLDRAM 3 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.

To measure margin with PRBS8 traffic pattern, set the VIO signals with the listed values in the `traffic_gen_top` instance in `example_top`:

```

vio_modify_enable = 'd1
vio_data_mode_value = 'd7
vio_addr_mode_value = 'd3
vio_instr_mode_value = 'd4
vio_bl_mode_value = 'd2
vio_fixed_bl_value = 'd128
vio_fixed_instr_value = 'd1
vio_data_mask_gen = 'd0

```

Next, assert `vio_dbg_clear_error` or assert system reset before proceeding with automated margin check. The steps to follow for automated margin check include:

1. Start automated window check by issuing a single pulse on the VIO signal, `dbg_win_start`.
2. The VIO signal, `dbg_win_active` indicates that the automated window check is in progress. The signals `dbg_pi_f_inc` and `dbg_pi_f_dec` must not be used when `dbg_win_active` is asserted.
3. The current bit and byte being measured are indicated by the VIO signals, `dbg_win_current_bit` and `dbg_win_current_byte`, respectively.
4. To obtain the left and right tap counts for a completed bit, select the desired bit using VIO signal, `dbg_win_bit_select` and observe the results on `dbg_win_left_ram_out` and `dbg_win_right_ram_out`, respectively.

[Table 3-33](#) lists the signals associated with this automated window checking functionality.

Table 3-33:

<code>dbg_win_start</code>	Single pulse that starts the <code>chk_win</code> state machine. Use the Vivado logic debug VIO to control this.
<code>dbg_win_bit_select[6:0]</code>	Manual bit selection for reporting of results. The results are provided on <code>dbg_win_left_ram_out</code> and <code>dbg_win_right_ram_out</code> for the bit indicated.
<code>dbg_win_active</code>	Flag to indicate <code>chk_win</code> is active and measuring read window margins. While active, the state machine has control over the debug port signals.
<code>vio_dbg_clear_error</code>	Clear error control signal controlled by <code>chk_win</code> .

Table 3-33:

(Cont'd)

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_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_pi_f_inc	chk_win control signal to increment PHASER_IN. This signal should be used only when dbg_win_active is deasserted.
dbg_pi_f_dec	chk_win control signal to decrease PHASER_IN. This signal should be used only when dbg_win_active is deasserted.

### Debugging Write Calibration

Due to the length of time required for completing write calibration for RLDRAM II/RLDRAM 3, it is useful to use the N-sample feature of the Vivado logic analyzer feature to selectively trigger and display a small window after a given trigger point. This allows you to capture signals across a larger period of time than would be allowed if you just captured a single window when the trigger condition first occurs. A good trigger condition is the `wrcal_byte_sel` signal as well as the state machine indicator, `write_cal_cs` (see [Table 3-32](#)).

This allows you to focus in on a given byte lane and capture each time an adjustment is made to the PHASER\_OUT. An example of what to look for is shown in [Figure 3-79](#).

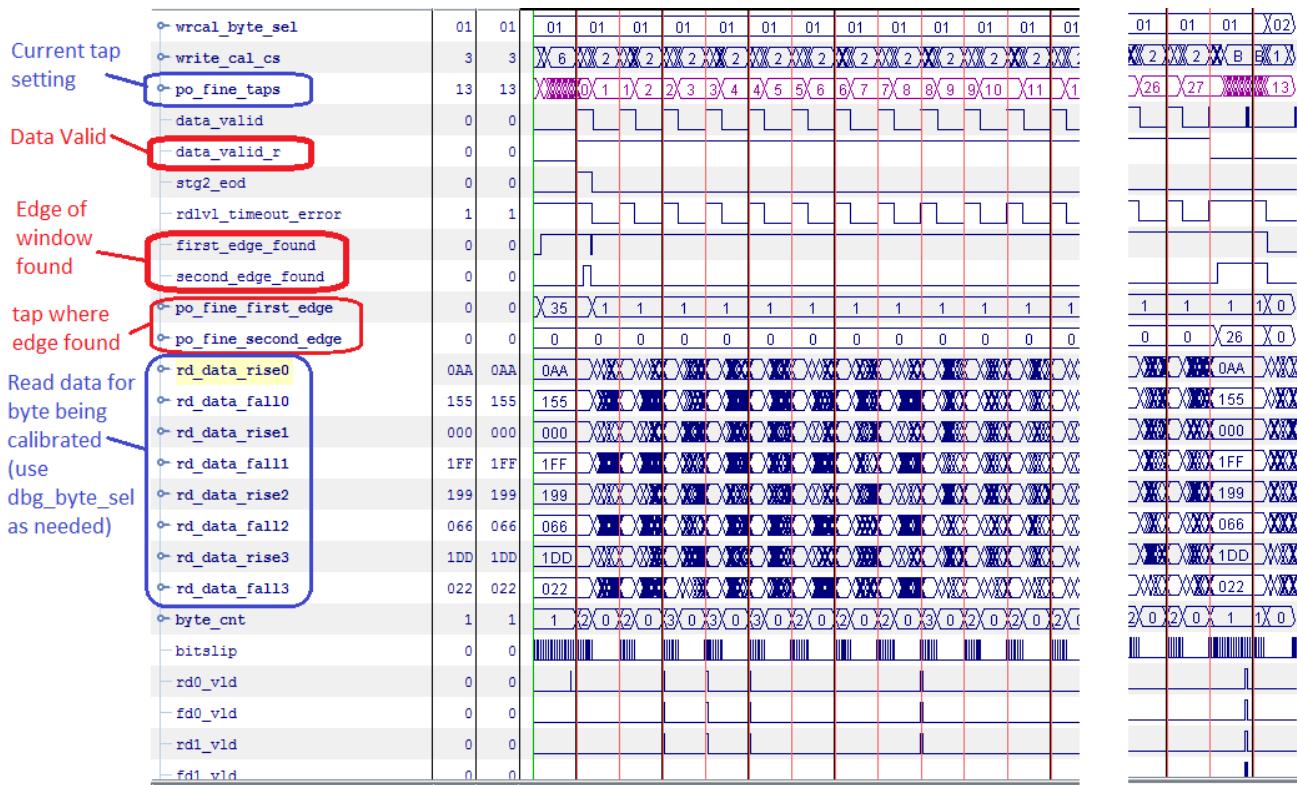


Figure 3-79:

When looking for issues, check to see if the read data being returned for a given byte lane is correct, as indicated by the **data\_valid\_r** signal. This signal checks the data across multiple clock cycles to ensure all data written during the burst is properly received. If you see gaps in this signal, this might indicate most of the data was written properly but not all of it was correct. Check to make sure the latency between the command and the data at the DRAM is correct for the given settings selected.

Next, check where the algorithm finds the edges of the window and compare with the data being received. If the data being received is always wrong, this can indicate an issue with the read leveling performed in an earlier step of calibration or indicate an issue on the PCB that requires additional debug.

## System Clock

If the SRCC/MRCC I/O pin and PLL are not allocated in the same bank, the **CLOCK\_DEDICATED\_ROUTE** constraint must be set to BACKBONE. RLDRAM II/RLDRAM 3 manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on the SRCC/MRCC I/O and PLL allocation needs to be handled manually for the IP flow. RLDARAM II/RLDARAM 3 does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

### *Reference Clock*

If the SRCC/MRCC I/O pin and MMCM are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint is set to FALSE. Reference clock is a 200 MHz clock source used to drive IODELAY CTRL logic (through an additional MMCM). This clock is not utilized, CLOCK\_DEDICATED\_ROUTE (as they are limited in number), hence the FALSE value is set. RLDARAM II/RLDARAM 3 manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on SRCC/MRCC I/O and MMCM allocation needs to be handled manually for the IP flow. RLDARAM II/RLDARAM 3 does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

# LPDDR2 SDRAM Memory Interface Solution

---

The Xilinx® 7 series FPGAs Memory Interface Solutions (MIS) core is a combined pre-engineered controller and physical layer (PHY) for interfacing 7 series FPGA user designs to LPDDR2 SDRAM devices. This user guide provides information about using, customizing, and simulating a LPDDR2 SDRAM interface core for 7 series FPGAs.

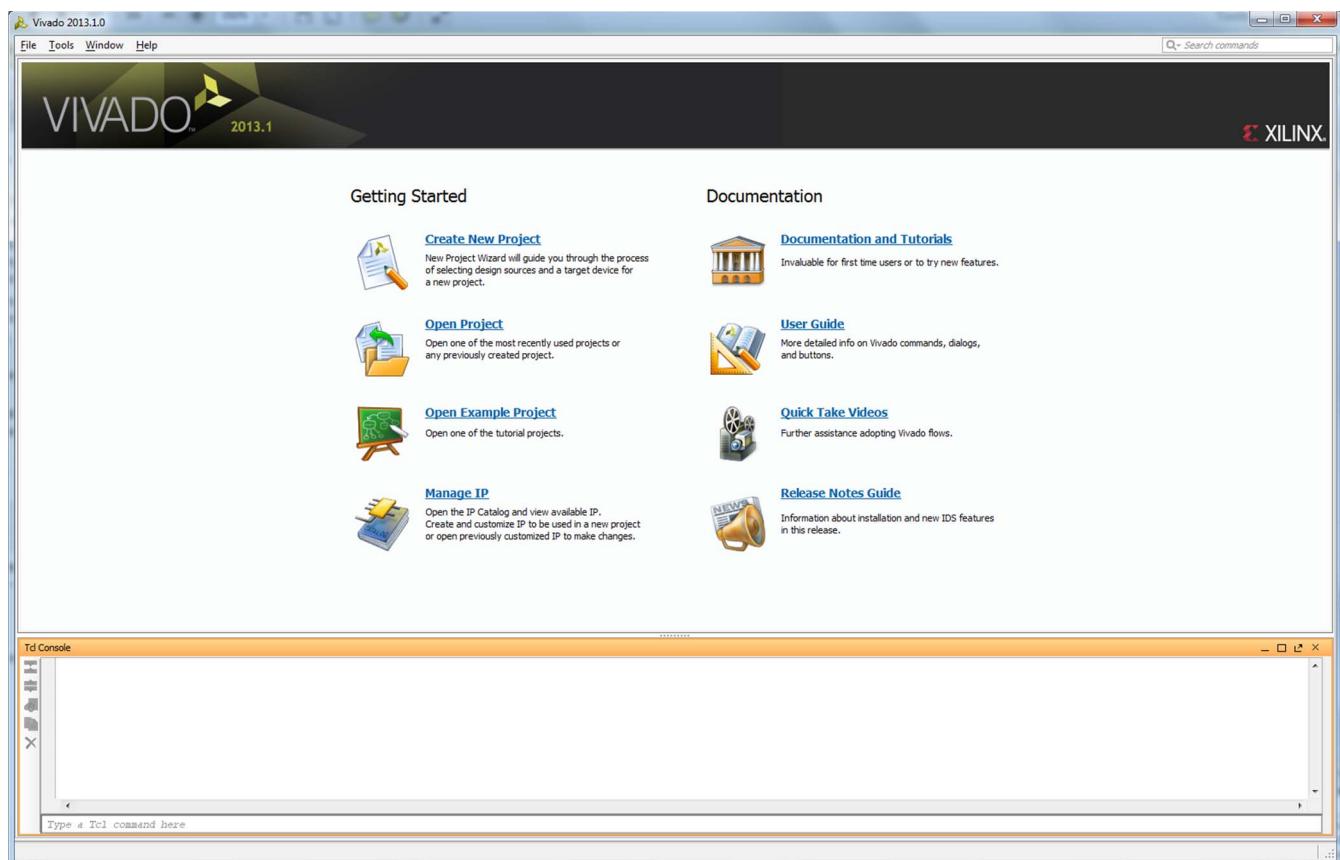
---

Enhancements to the Xilinx 7 series FPGA memory interface solutions from the 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 575](#)).
- Pinout rules changed due to the hardware blocks (see [Design Guidelines, page 632](#)).
- Controller and user interface operate at 1/2 of the memory clock frequency.

This section provides the steps to generate the Memory Interface Generator (MIG) IP core using the Vivado® Design Suite and run implementation.

1. Start the Vivado Design Suite (see [Figure 4-1](#)).



*Figure 4-1:*

2. To create a new project, click the **Create New Project** option shown in [Figure 4-1](#) to open the page as shown in [Figure 4-2](#).

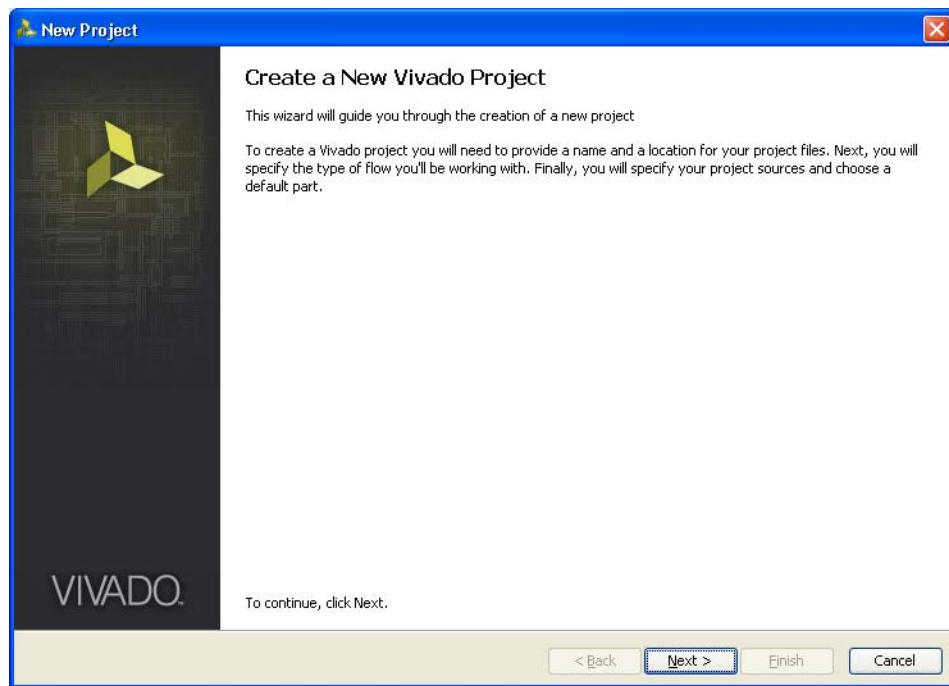


Figure 4-2:

3. Click **Next** to proceed to the **Project Name** page (Figure 4-3). Enter the **Project Name** and **Project Location**. Based on the details provided, the project is saved in the directory.

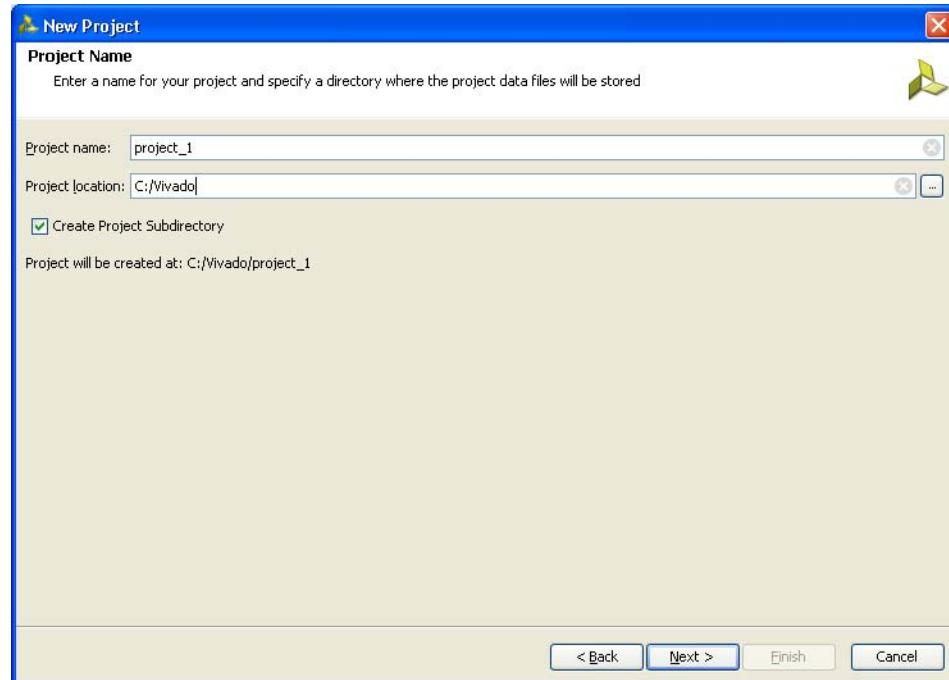
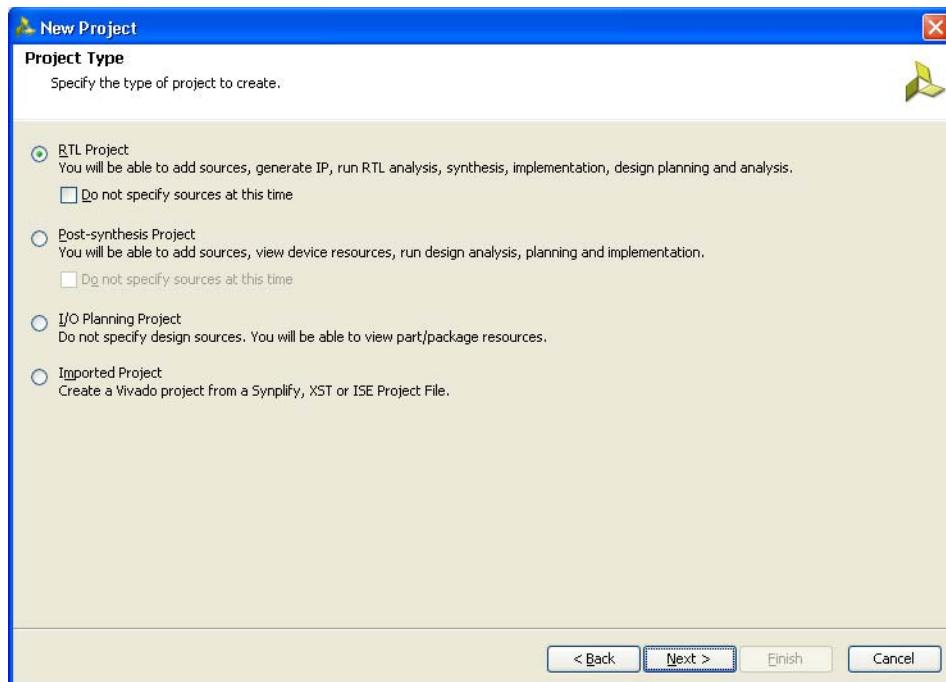


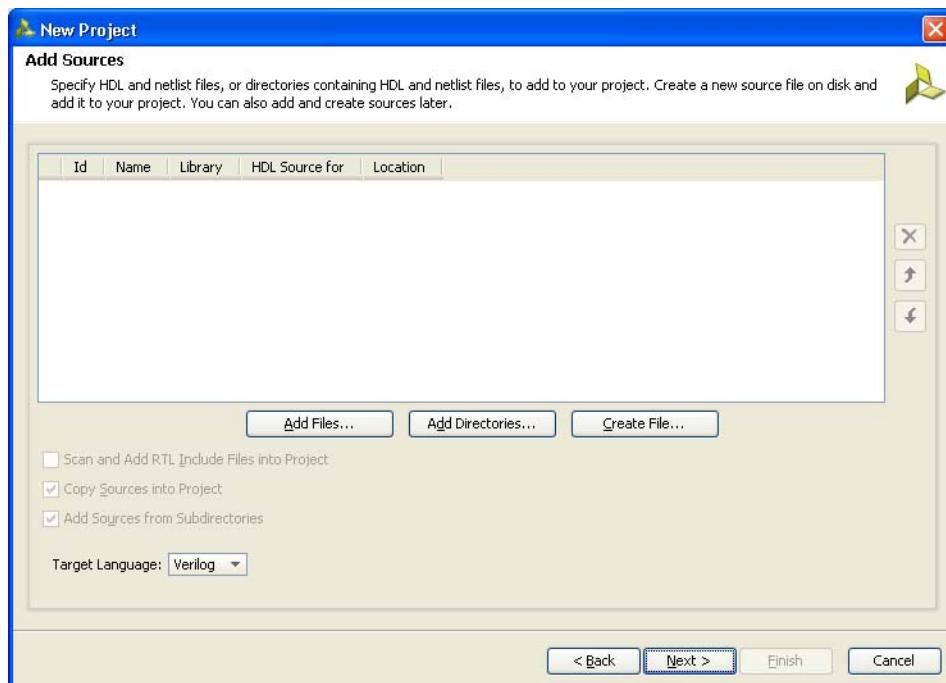
Figure 4-3:

4. Click **Next** to proceed to the **Project Type** page (Figure 4-4). Select the **Project Type** as **RTL Project** because MIG deliverables are RTL files.



*Figure 4-4:*

5. Click **Next** to proceed to the **Add Sources** page (Figure 4-5). RTL files can be added to the project in this page. If the project was not created earlier, proceed to the next page.



*Figure 4-5:*

6. Click **Next** to open the **Add Existing IP (Optional)** page (Figure 4-6). If the IP is already created, the XCI file generated by the IP can be added to the project and the previous created IP files are automatically added to the project. If the IP was not created earlier, proceed to the next page.

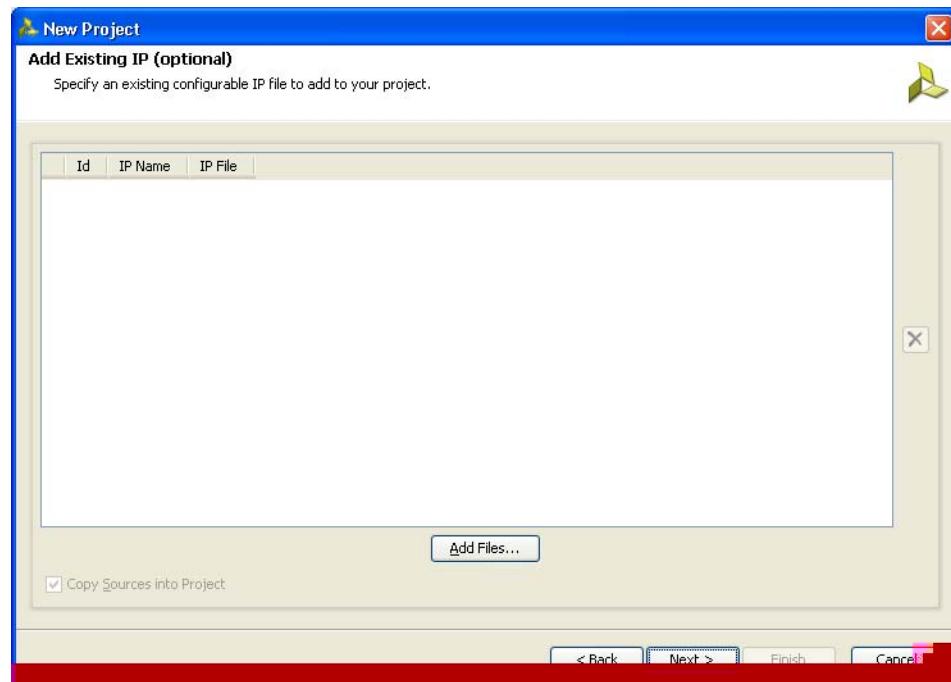


Figure 4-6:

7. Click **Next** to open the **Add Constraints (Optional)** page (Figure 4-7). If the constraints file exists in the repository, it can be added to the project. Proceed to the next page if the constraints file does not exist.

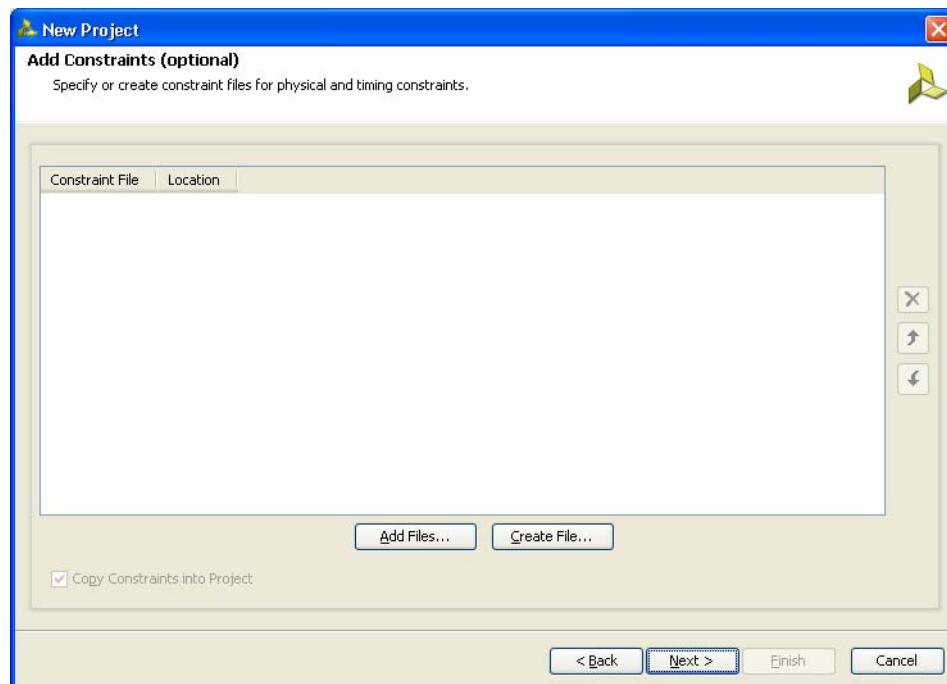


Figure 4-7:

- Click **Next** to proceed to the **Default Part** page (Figure 4-8) where the device that needs to be targeted can be selected. The **Default Part** page appears as shown in Figure 4-8.

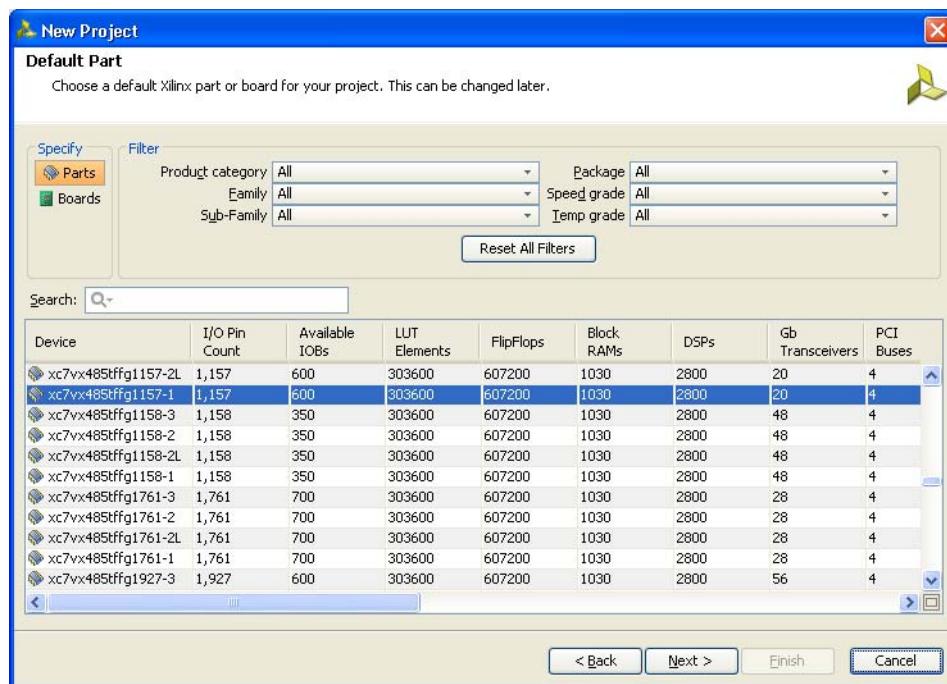
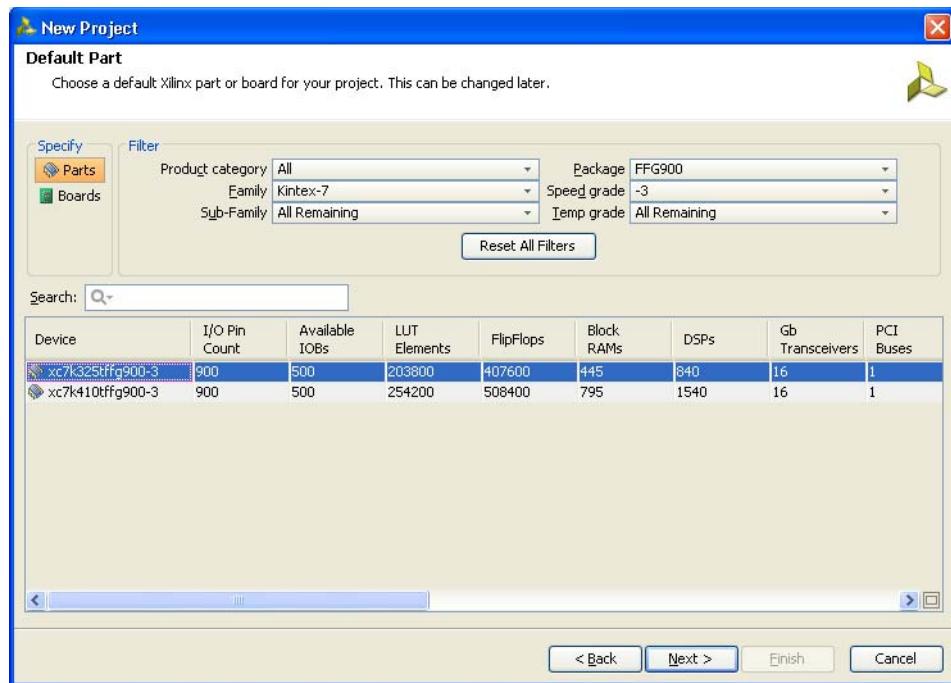


Figure 4-8:

Select the target **Family**, **Package**, and **Speed Grade**. The valid devices are displayed in the same page, and the device can be selected based on the targeted device ([Figure 4-9](#)).



*Figure 4-9:*

Apart from selecting the parts by using **Parts** option, parts can be selected by choosing the **Boards** option, which brings up the evaluation boards supported by Xilinx ([Figure 4-10](#)). With this option, design can be targeted for the various evaluation boards. If the XCI file of an existing IP was selected in an earlier step, the same part should be selected here.

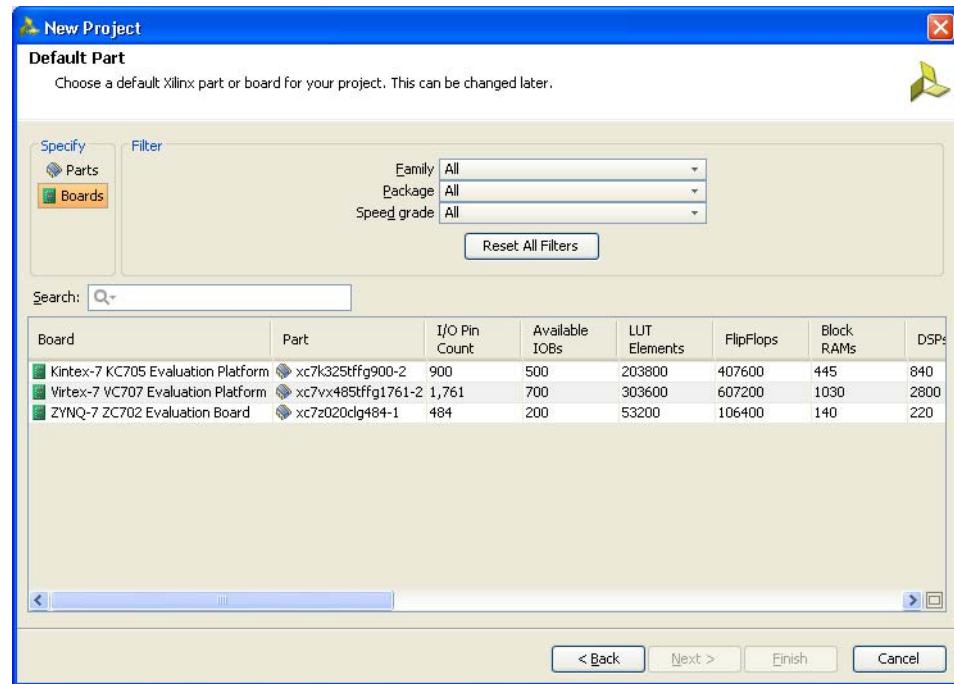


Figure 4-10:

- Click **Next** to open the **New Project Summary** page (Figure 4-11). This includes the summary of selected project details.

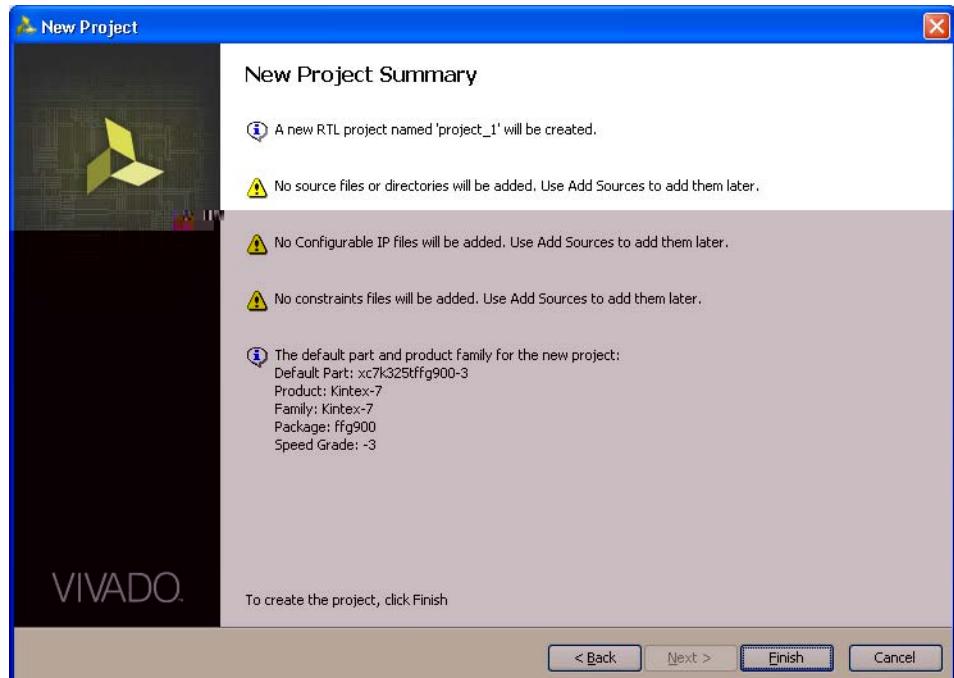
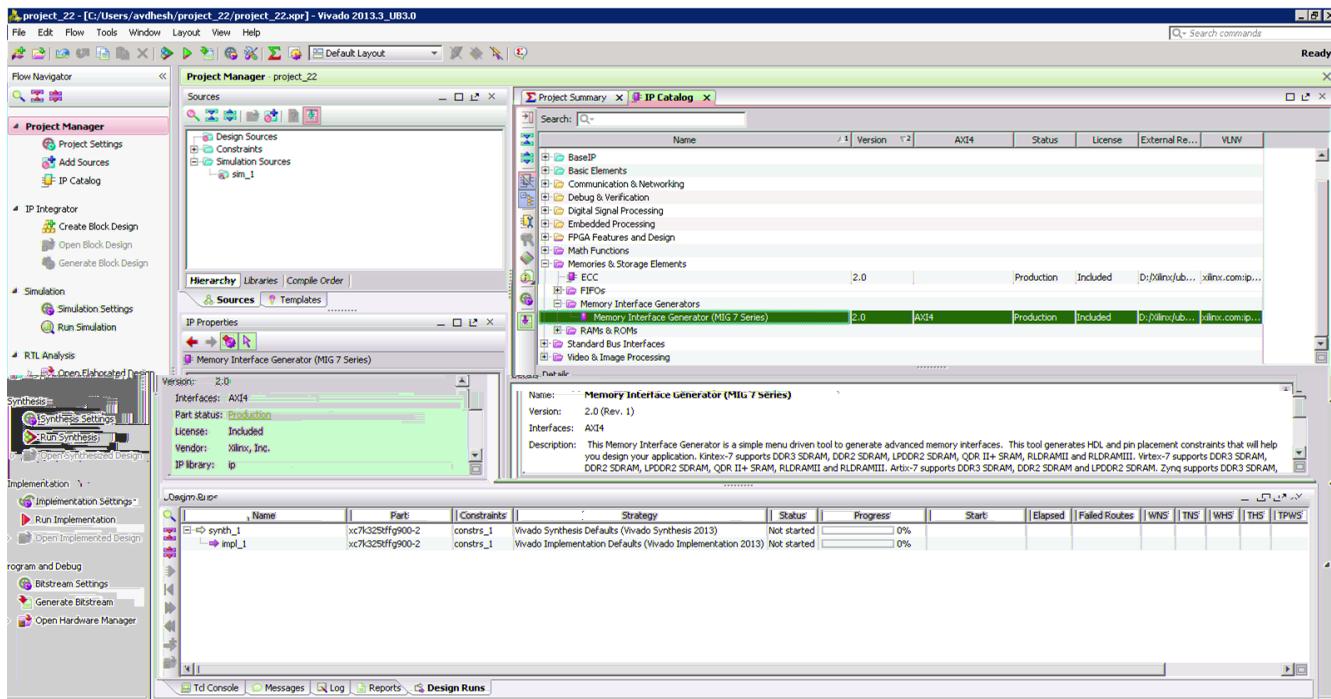


Figure 4-11:

- Click **Finish** to complete the project creation.

11. Click **IP Catalog** on the **Project Manager** window to open the IP catalog window. The IP catalog window appears on the right side panel (see [Figure 4-12](#), highlighted in a red circle).
12. The MIG tool exists in the **Memories & Storage Elements > Memory Interface Generators** section of the IP catalog window ([Figure 4-12](#)) or you can search from the Search tool bar for the string "MIG."



*Figure 4-12:*

13. Select **MIG 7 Series** to open the MIG tool (Figure 4-13).

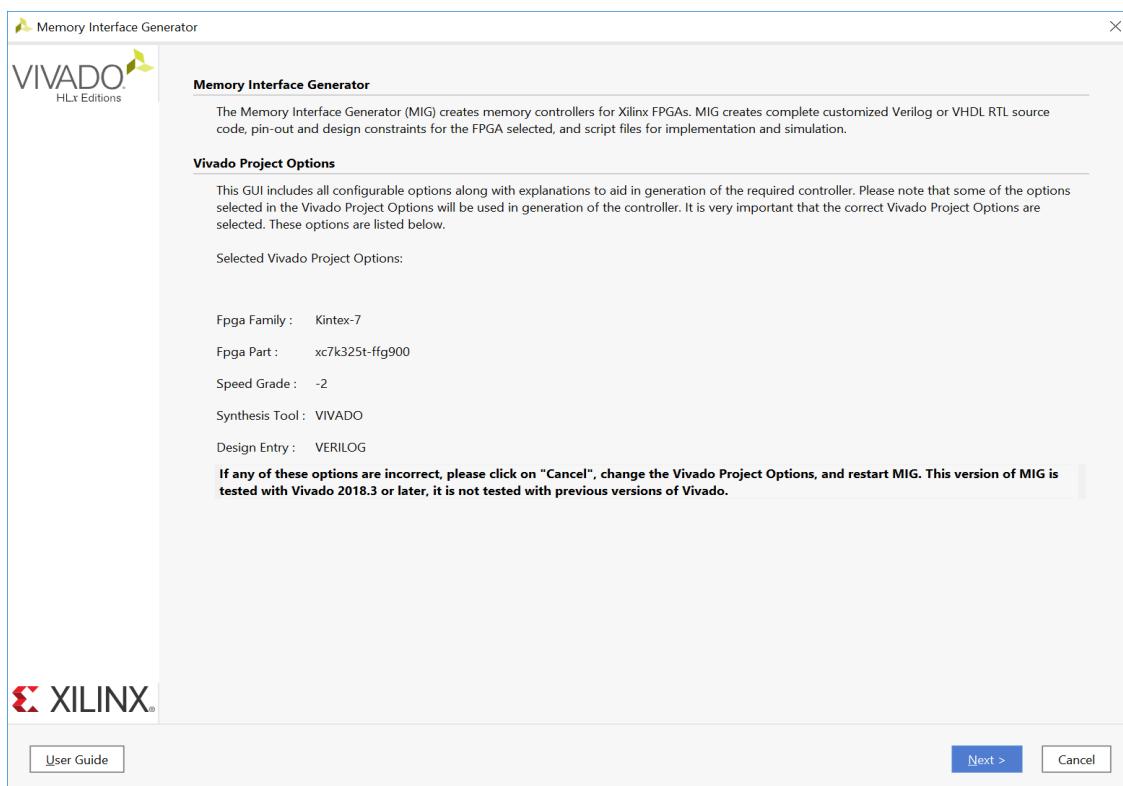


Figure 4-13:

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



**CAUTION!** *The Windows operating system has a 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs.*

## MIG Output Options

1. Select **Create Design** to create a new Memory Controller design. Enter a component name in the Component Name field (Figure 4-14).
2. Number of controllers supported for LPDDR2 SDRAM is 1.

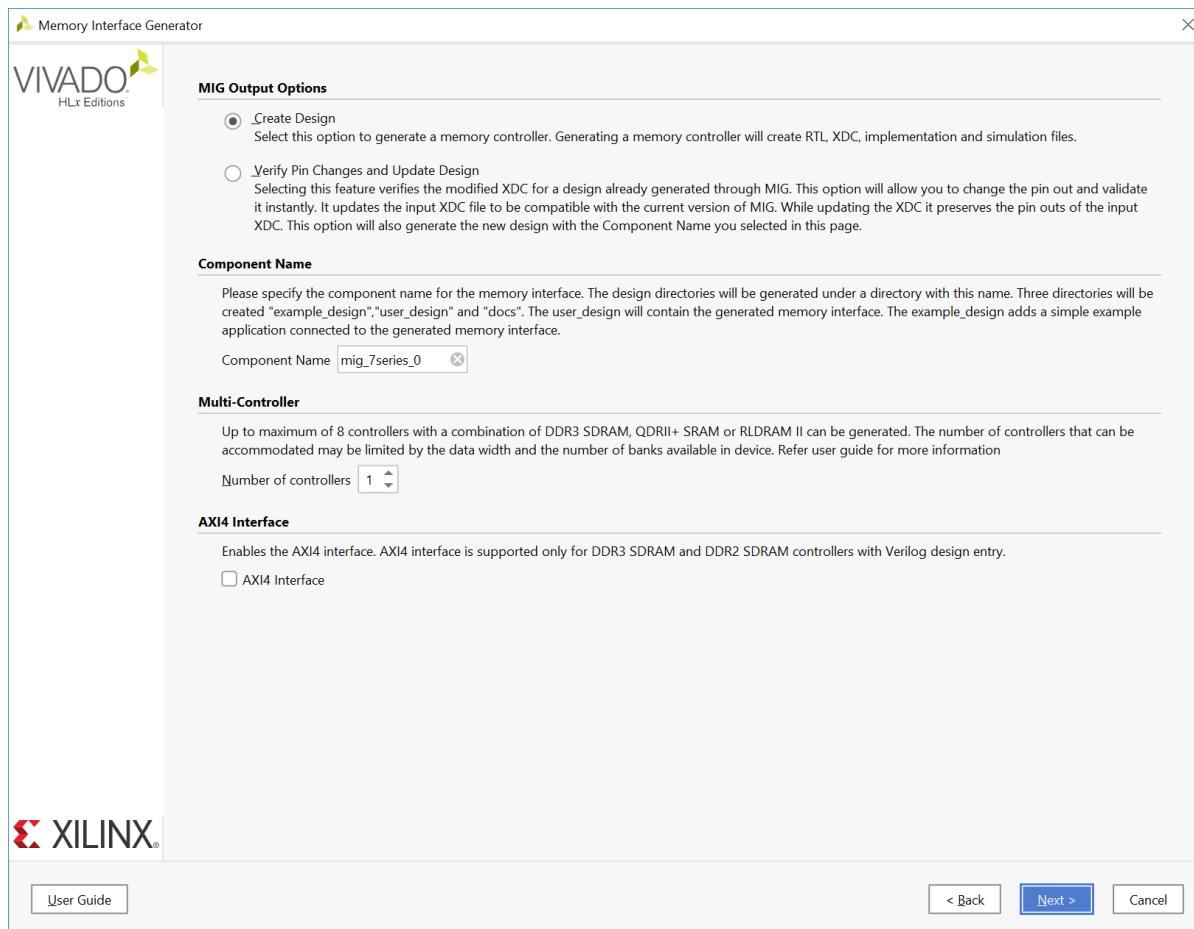


Figure 4-14:

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



**IMPORTANT:** 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.

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 4-15).

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.

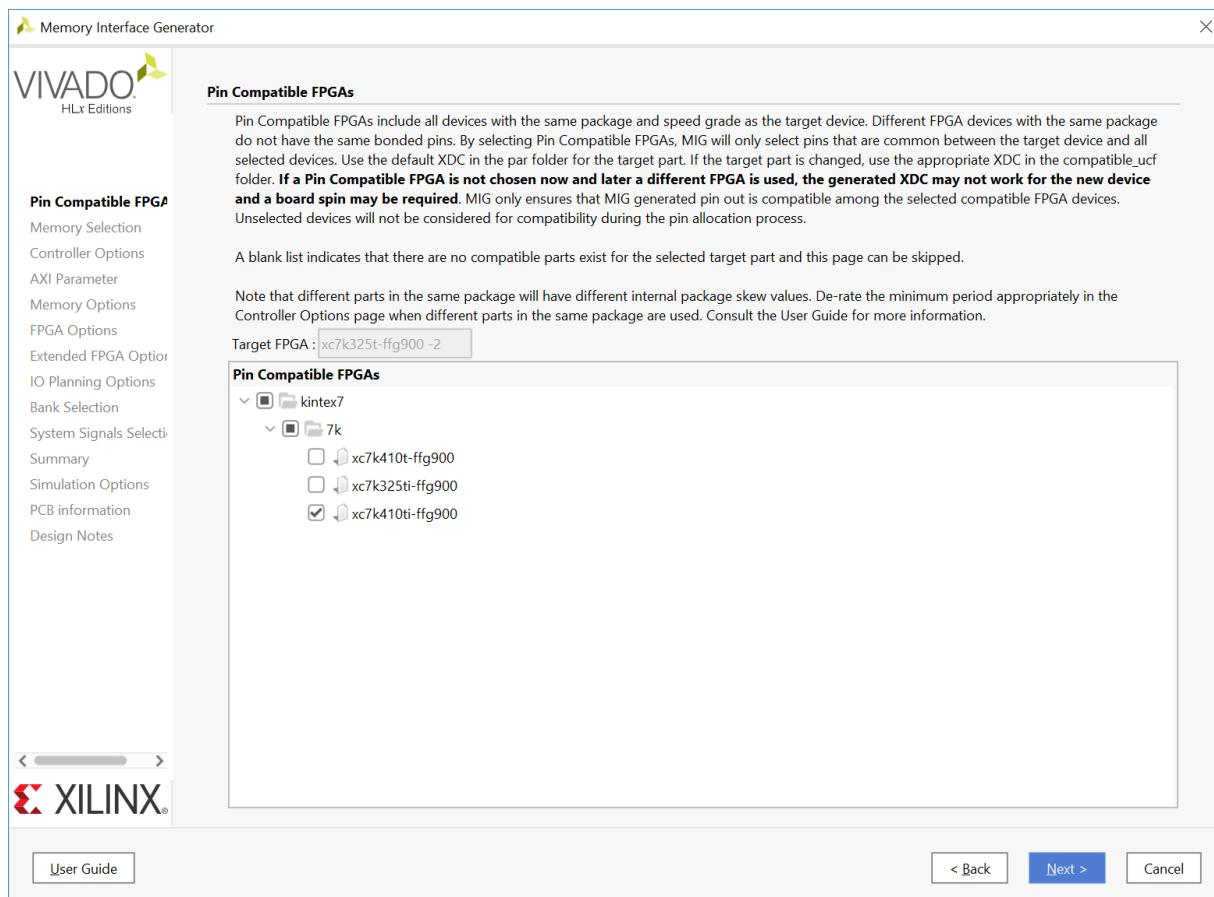


Figure 4-15:

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 LPDDR2 SDRAM Memory Controller Block Design

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

1. Select the **LPDDR2 SDRAM** controller type.
2. Click **Next** to display the **Controller Options** page (Figure 4-16).

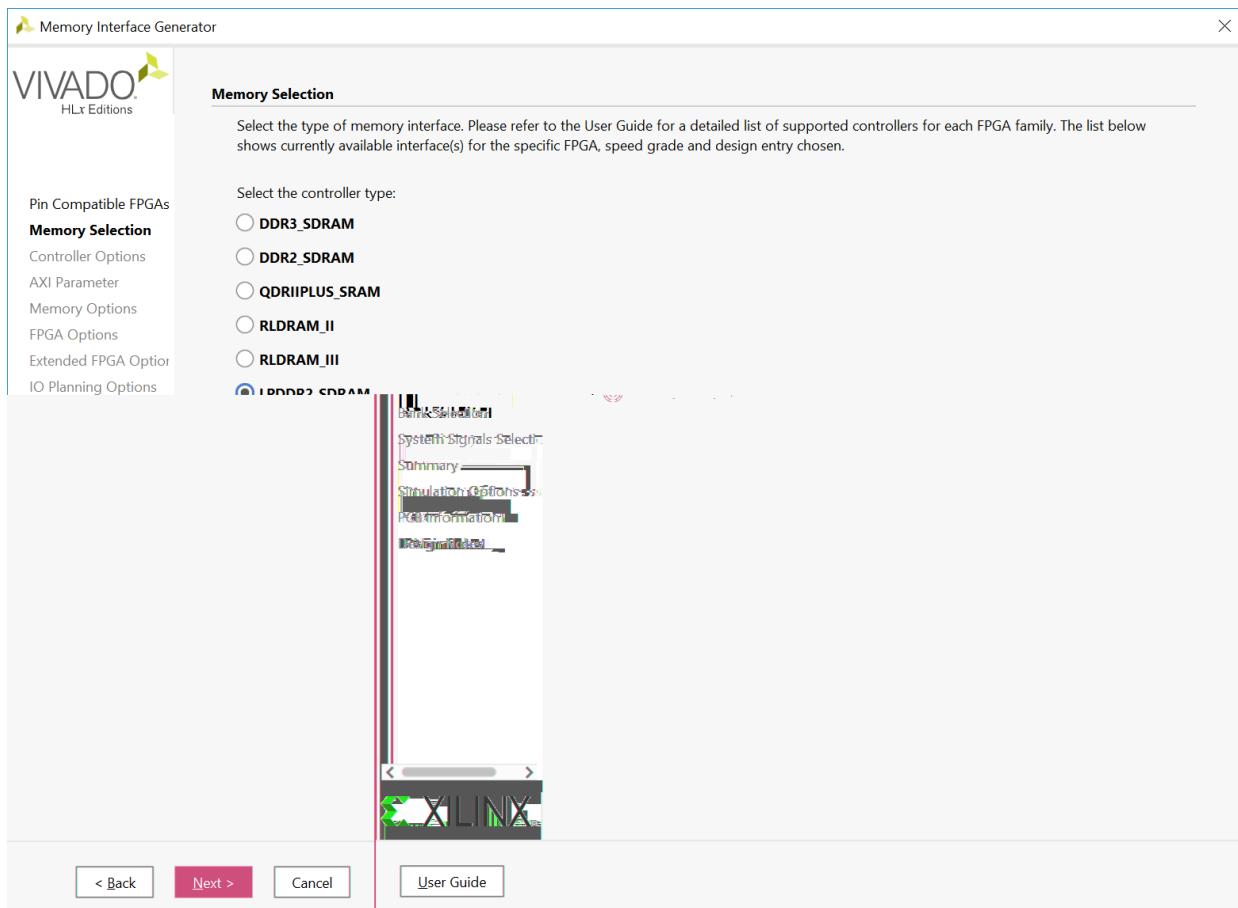


Figure 4-16:

This page shows the various controller options that can be selected (Figure 4-17).



**TIP:** *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. 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.*

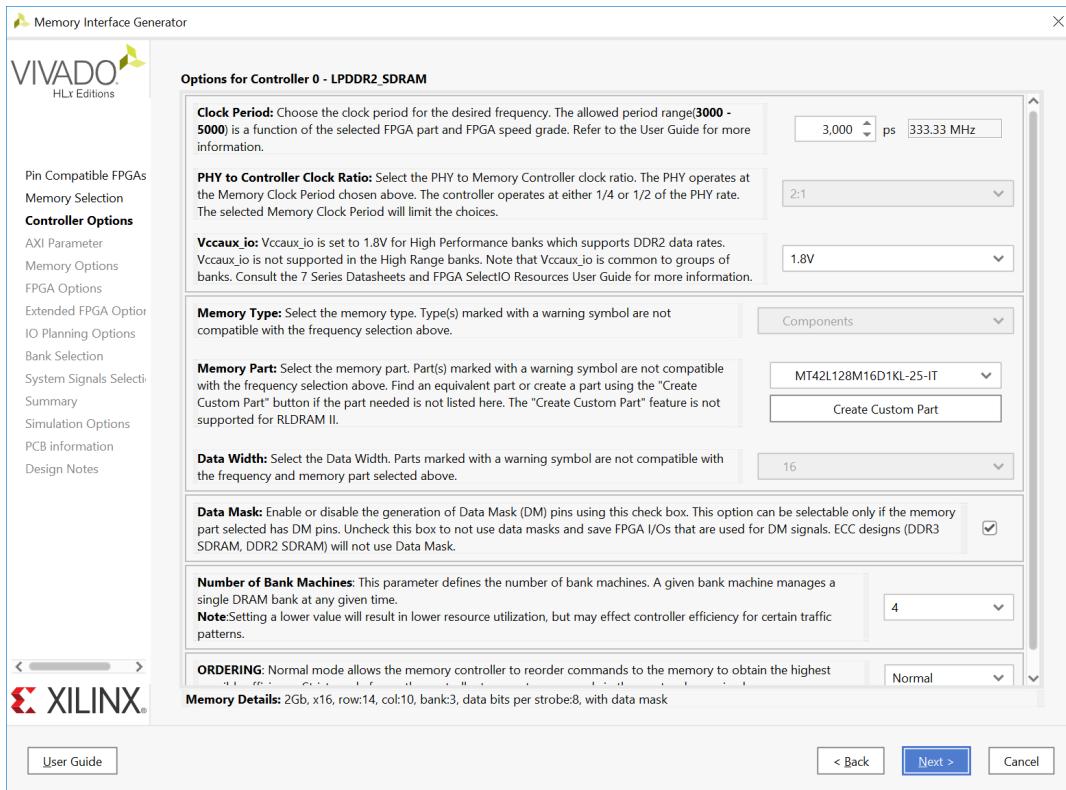


Figure 4-17:

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.
- **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 user interface data bus width of the 2:1 ratio is four times the width of the physical memory interface width.
- **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.
- **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 4-17, page 530) displays the details for the selected memory configuration (Figure 4-18).

**Memory Details:** 2Gb, x16, row:14, col:10, bank:3, data bits per strobe:8, with data mask

Figure 4-18:

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 **Create Custom Part** below the **Memory Part** pull-down menu. A new page appears, as shown in Figure 4-19.

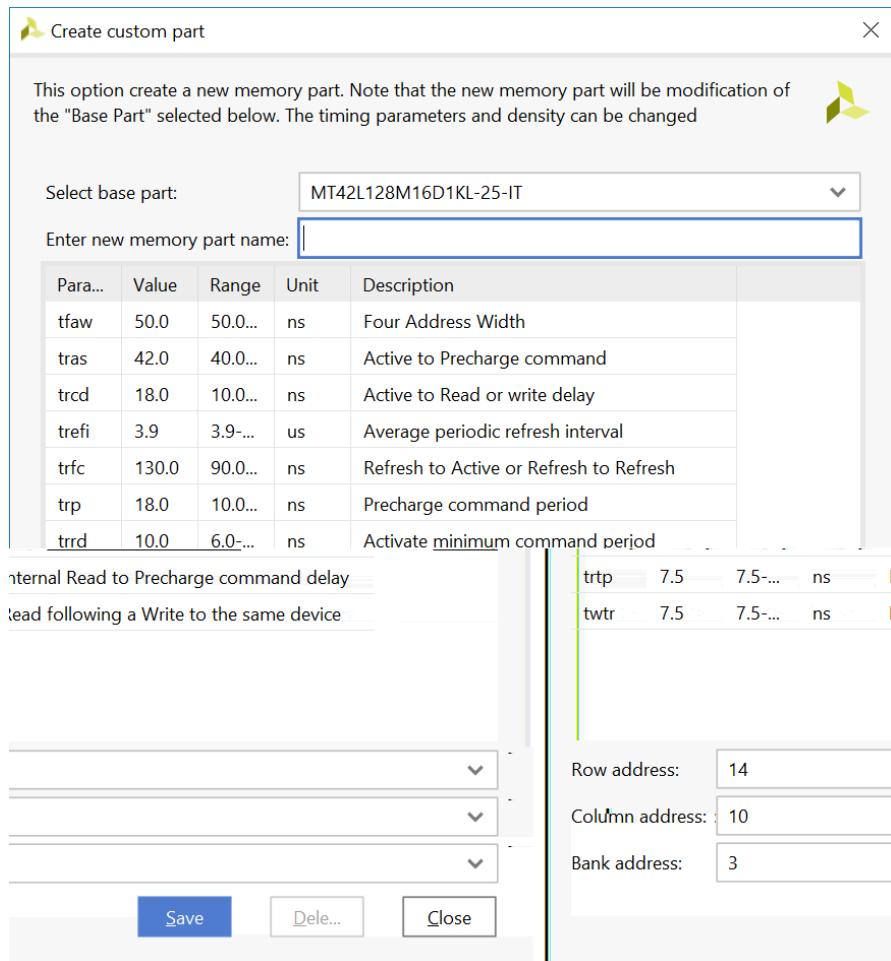
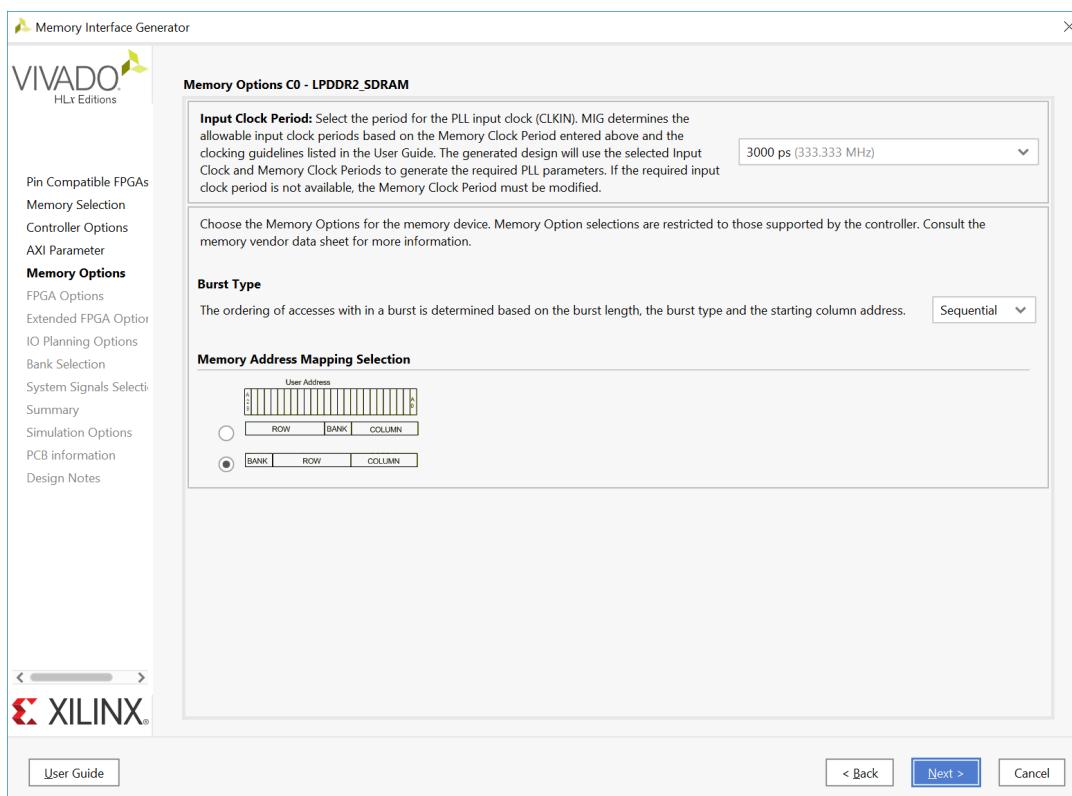


Figure 4-19:

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 **Save**. 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.

This feature allows the selection of various memory mode register values, as supported by the controller specification ([Figure 4-20](#)).



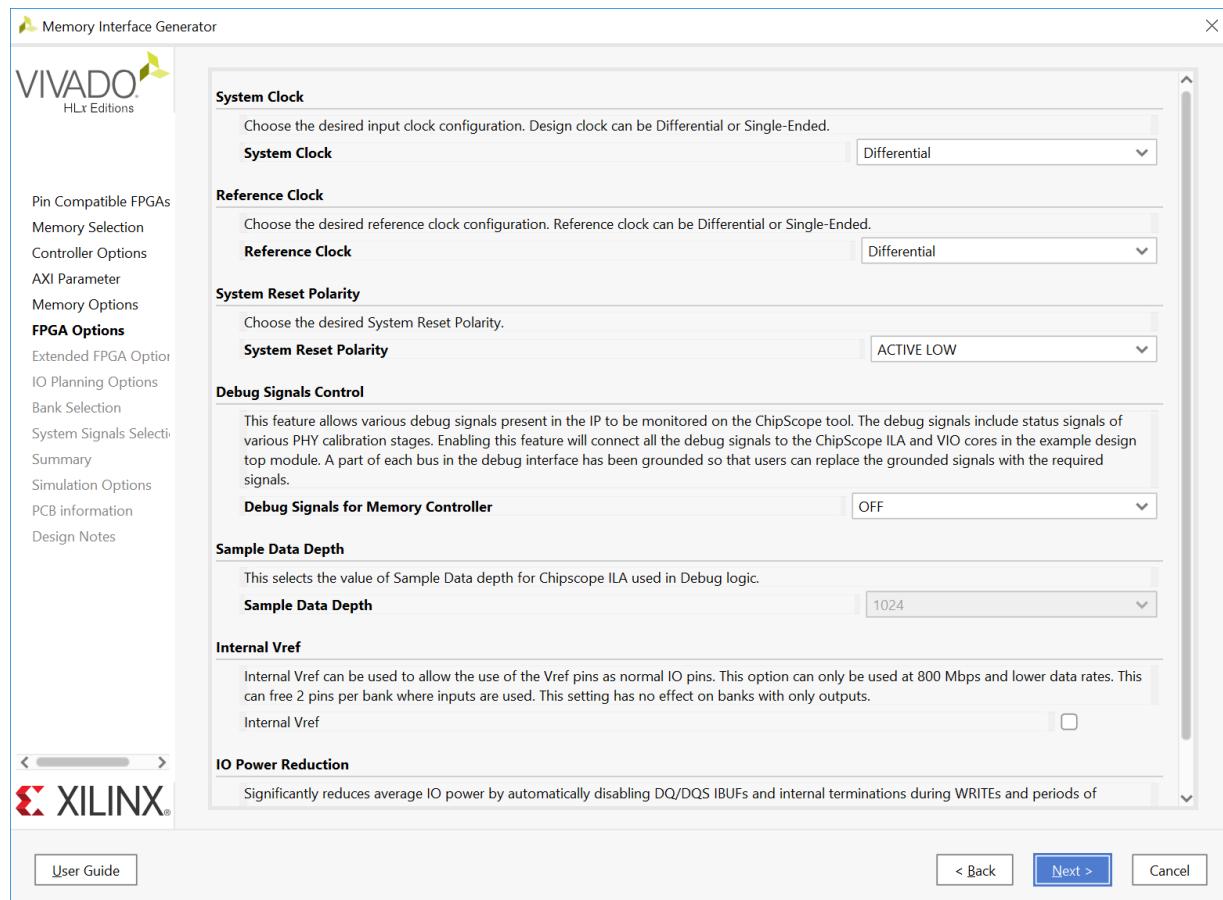
*Figure 4-20:*

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

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. For more information on the MMCM parameter limits, see [Design Guidelines, page 632](#).

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

[Figure 4-21](#) shows the **FPGA Options** page.



[Figure 4-21:](#)

- **System Clock** – This option selects the clock type (Single-Ended, Differential, or No Buffer) for the **sys\_clk** signal pair. When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the system clock.

If the designs generated from MIG for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the **sys\_clk\_i** signal. So for No Buffer scenarios, **sys\_clk\_i** signal needs to be connected to an internal clock.

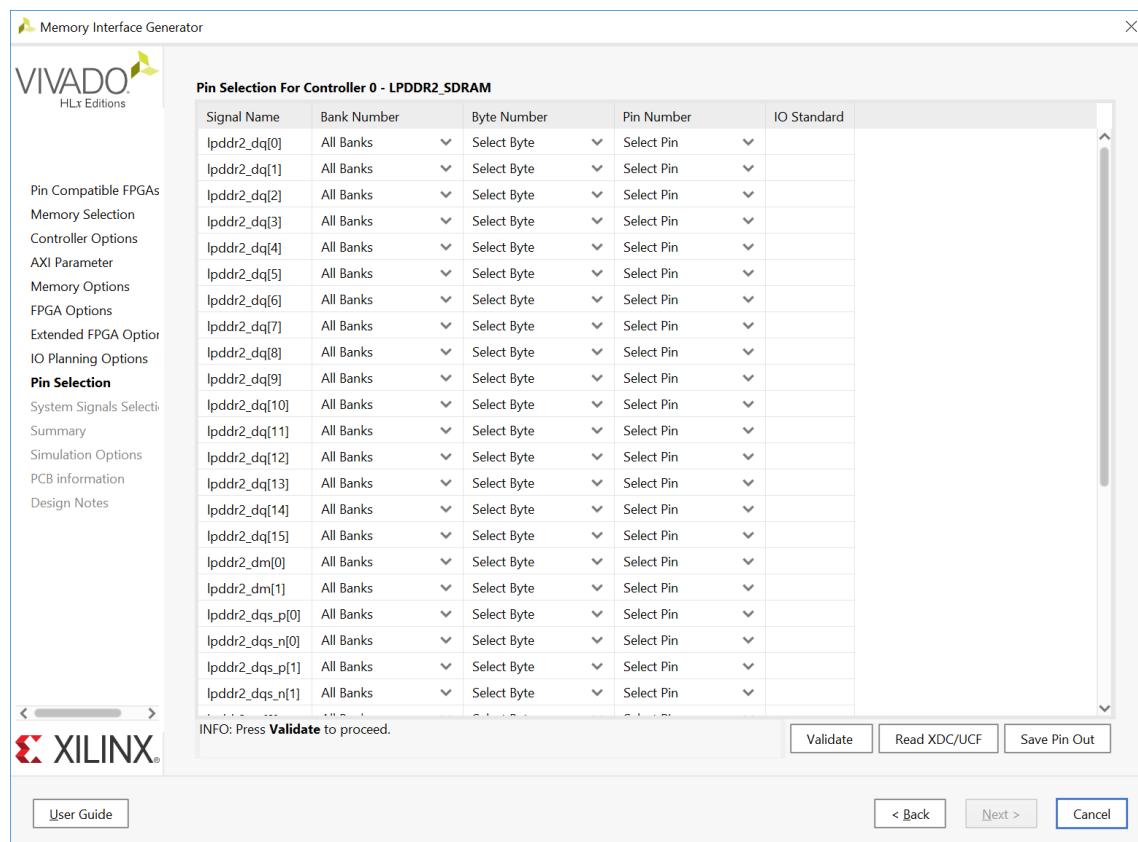
- **Reference Clock** – This option selects the clock type (Single-Ended, Differential, No Buffer, or Use System Clock) for the `clk_ref` signal pair. The **Use System Clock** option appears when the input frequency is between 199 and 201 MHz (that is, the Input Clock Period is between 5,025 ps (199 MHz) and 4,975 ps (201 MHz). When the **No Buffer** option is selected, IBUF primitives are not instantiated in RTL code and pins are not allocated for the reference clock.

If the designs generated from MIG for the **No Buffer** option are implemented without performing changes, designs can fail in implementation due to IBUFs not instantiated for the `ref_clk_i` signal. So for **No Buffer** scenarios, `ref_clk_i` signal needs to be connected to an internal clock.

- **System Reset Polarity** – The polarity for system reset (`sys_rst`) can be selected. If the option is selected as active-Low, the parameter `RST_ACT_LOW` is set to 1 and if set to active-High the parameter `RST_ACT_LOW` is set to 0.
- **Debug Signals Control** – Selecting this option enables calibration status and user port signals to be port mapped to the ILA and VIO in the `example_top` module. This helps in monitoring traffic on the user interface port with the Vivado Design Suite debug feature. Deselecting the **Debug Signals Control** option leaves the debug signals unconnected in the `example_top` module and no ILA/VIO modules are generated by the IP catalog. Additionally, the debug port is always disabled for functional simulations.
- **Sample Data Depth** – This option selects the Sample Data depth for the ILA module used in the Vivado debug logic. This option can be selected when the **Debug Signals for Memory Controller** option is ON.
- **Internal V<sub>REF</sub> Selection** – Internal V<sub>REF</sub> can be used for data group bytes to allow the use of the V<sub>REF</sub> pins for normal I/O usage. Internal V<sub>REF</sub> should only be used for data rates of 800 Mb/s or below.

Click **Next** to display the **Pin/Bank Selection Mode** page.

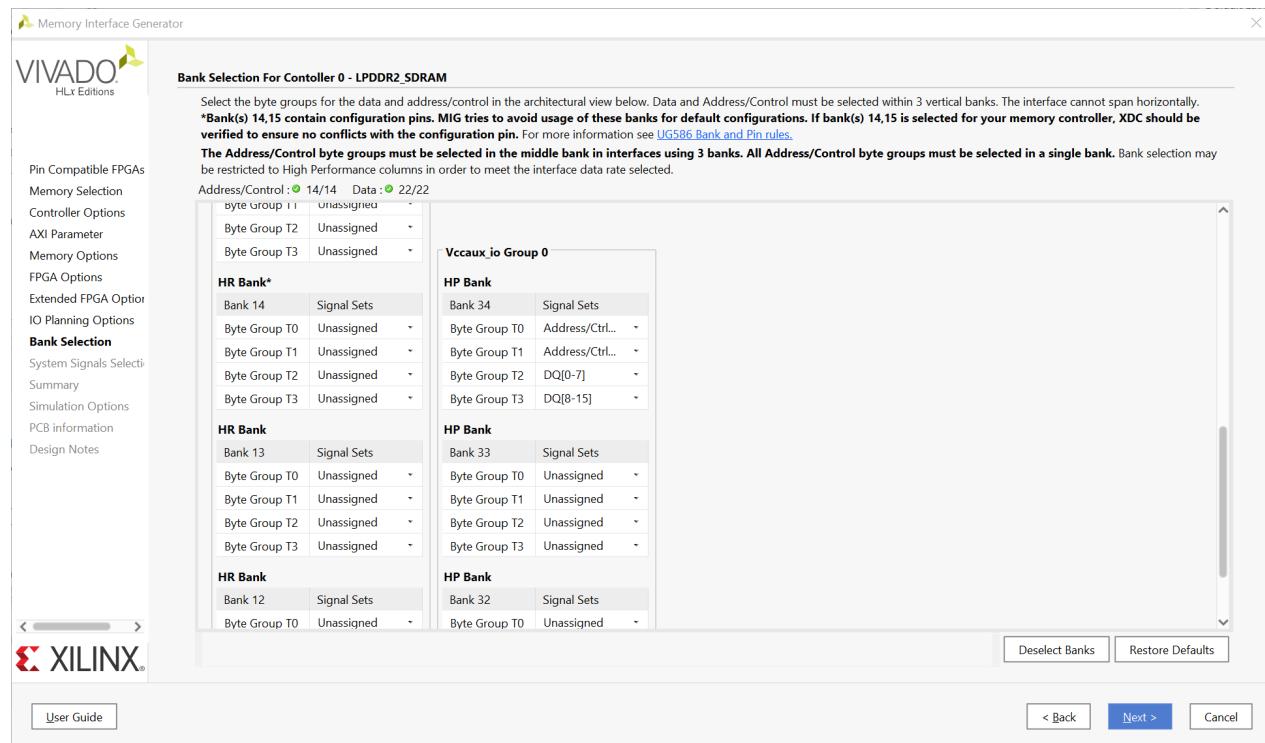
- **Pin/Bank Selection Mode** – This allows you to specify an existing pinout and generate the RTL for this pinout, or pick banks for a new design. [Figure 4-22](#) shows the options for using an existing pinout. You must assign the appropriate pins for each signal. A choice of each bank is available 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. You cannot proceed until the MIG DRC has been validated by clicking **Validate**.



*Figure 4-22:*

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 4-23:*

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 **Deselect Banks**. To restore the defaults, click **Restore Defaults**.

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 4-24](#)). The MIG tool allows the selection of either external pins or internal connections, as desired.

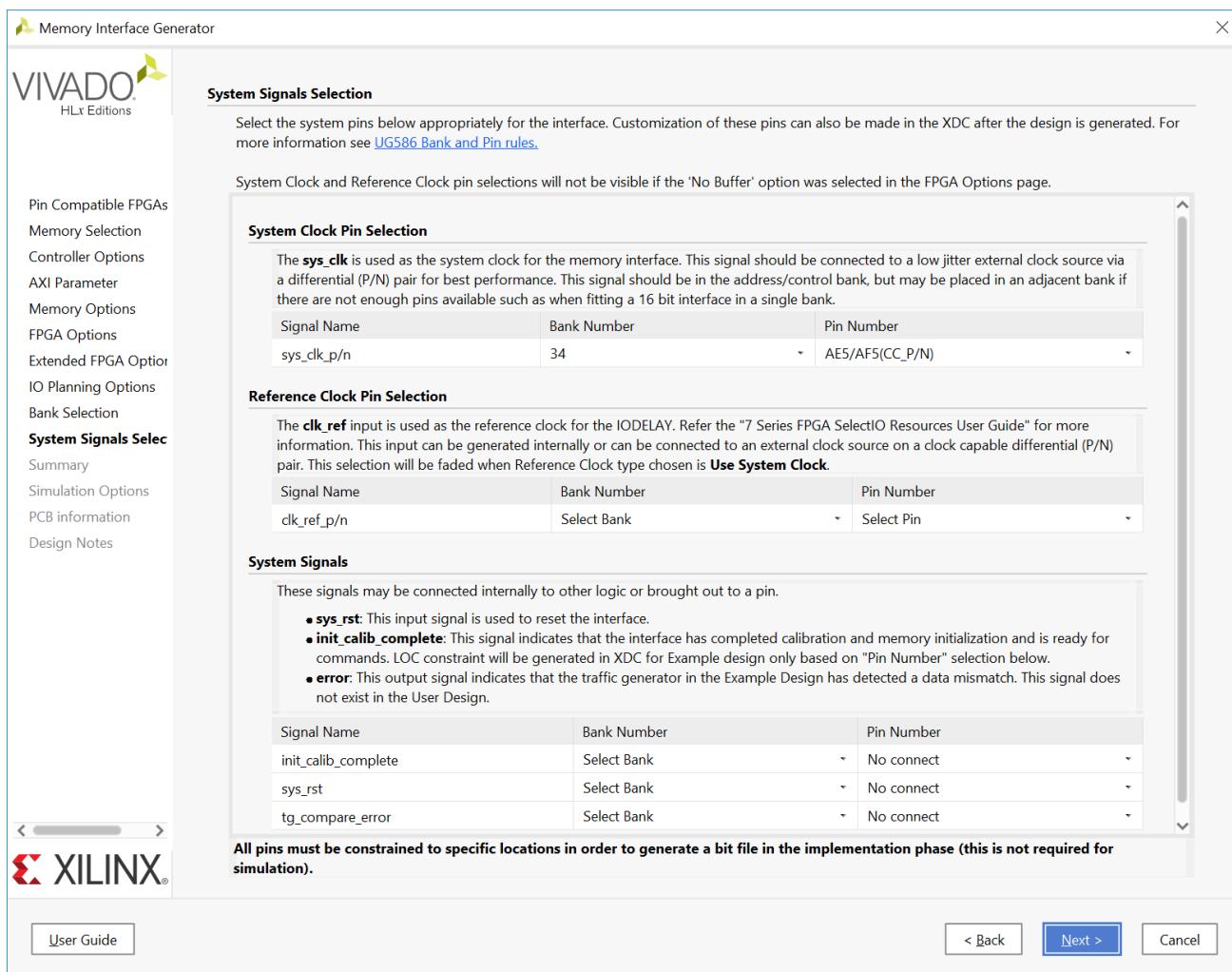


Figure 4-24:

- **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 4-21). 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\_SSTL12 or SSTL12. If **sys\_clk** is not connected in a memory interface bank, the MIG tool selects an appropriate standard such as LVCMOS18 or LVDS. The XDC 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 4-21). The I/O standard is selected in a similar way as **sys\_clk**.

- **sys\_rst** – This is the asynchronous 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 LVCMOS18 and LVCMOS25 for HP and HR banks, respectively. The default polarity of **sys\_rst** pin is active-Low. The polarity of **sys\_rst** pin varies based on the **System Reset Polarity** option chosen in **FPGA Options** page (Figure 4-21).
- **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.

This page provides the complete details about the 7 series FPGA memory core selection, interface parameters, Vivado tool options, and FPGA options of the active project (Figure 4-25).

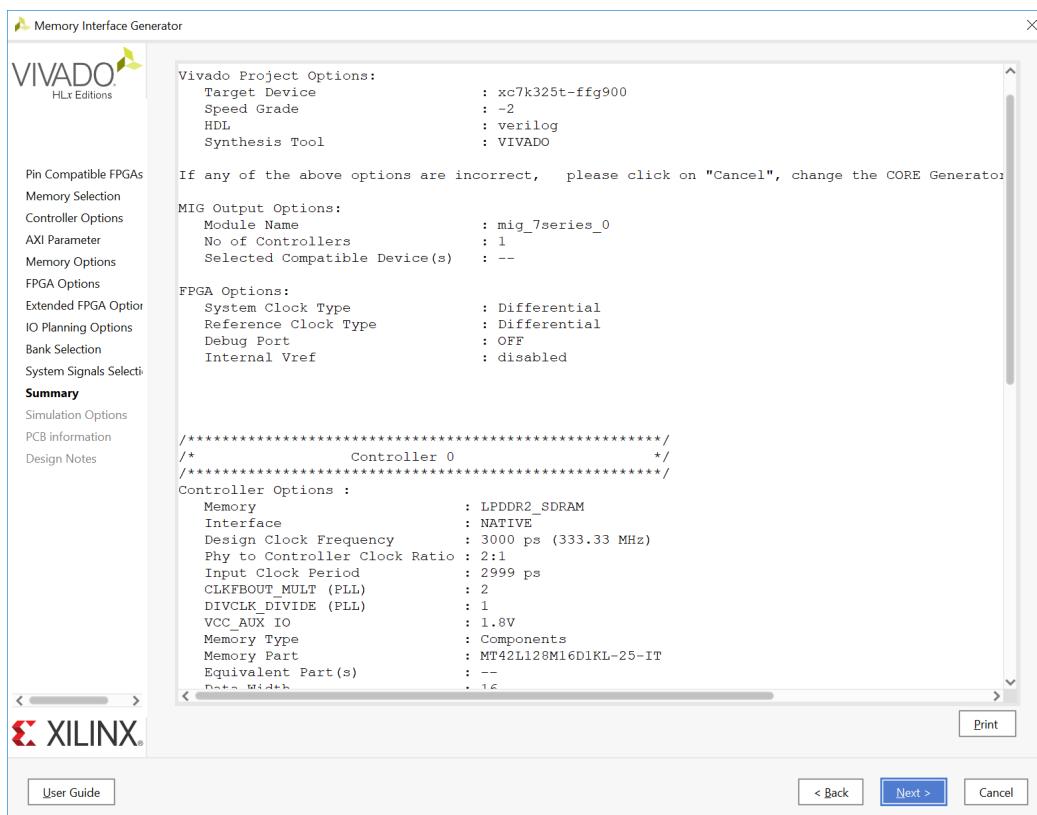


Figure 4-25:

The MIG tool can output a chosen vendor's memory model for simulation purposes for memories such as LPDDR2 SDRAMs. To access the models in the output **sim** folder, click the license agreement (Figure 4-26). 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.

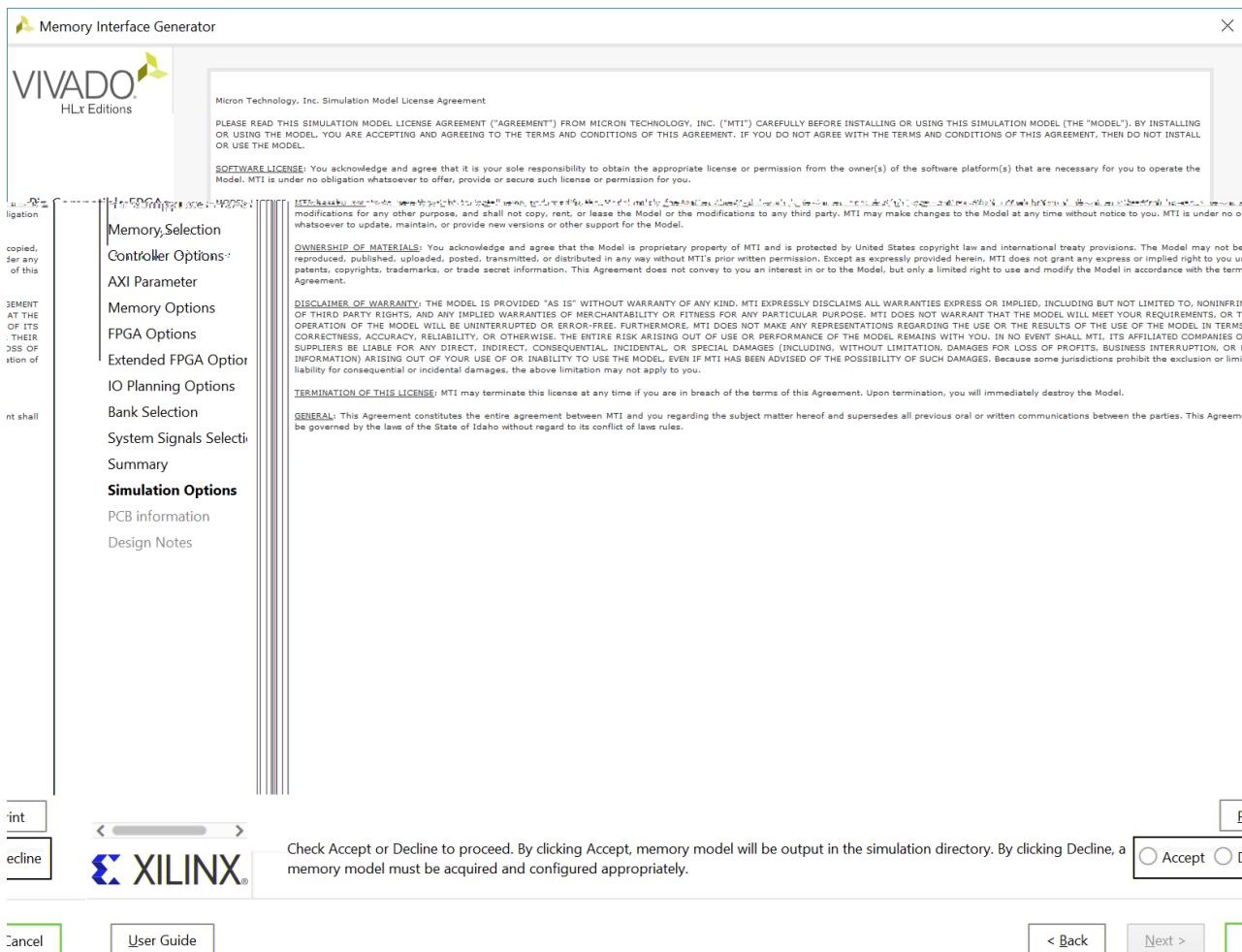


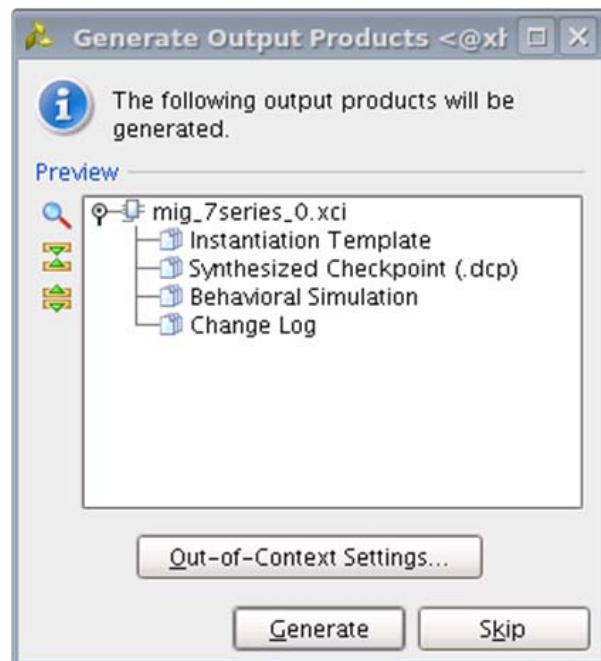
Figure 4-26:

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

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.

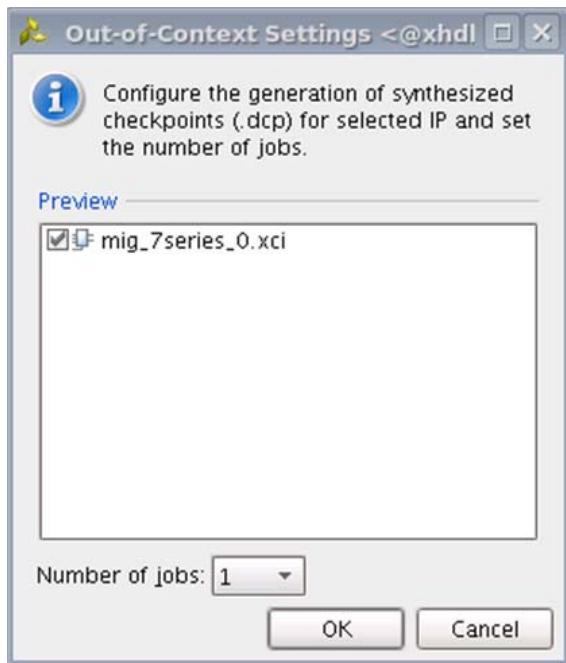
Click **Generate** to generate the design files. The MIG tool generates two output directories: **example\_design** and **user\_design**. After generating the design, the MIG GUI closes.

1. After clicking **Generate**, the **Generate Output Products** window appears. This window has the **Out-of-Context Settings** as shown in [Figure 4-27](#).



*Figure 4-27:*

2. Click **Out-of-Context Settings** to configure generation of synthesized checkpoints. To enable the **Out-of-Context** flow, enable the check box. To disable the **Out-of-Context** flow, disable the check box. The default option is “enable” as shown in [Figure 4-28](#).



*Figure 4-28:*

3. MIG designs comply with “Hierarchical Design” flow in Vivado. For more information, see the *Vivado Design Suite User Guide: Hierarchical Design* (UG905) [\[Ref 5\]](#) and the *Vivado Design Suite Tutorial: Hierarchical Design* (UG946) [\[Ref 6\]](#).
4. After generating the MIG design, the project window appears as shown in [Figure 4-29](#).

◆

*Chapter 4:*

Design generation from MIG can be generated using the **Create Design** flow or the **Verify Pin Changes** and **Update Design** flows. There is no difference between the flow when generating the design from the MIG tool. Irrespective of the flow by which designs are generated from the MIG tool, the XCI file is added to the Vivado tool project. The implementation flow is the same for all scenarios because the flow depends on the XCI file added to the project.

6. All MIG generated user design RTL and XDC files are automatically added to the project. If files are modified and you wish to regenerate them, right-click the XCI file and select **Generate Output Products** (Figure 4-31).

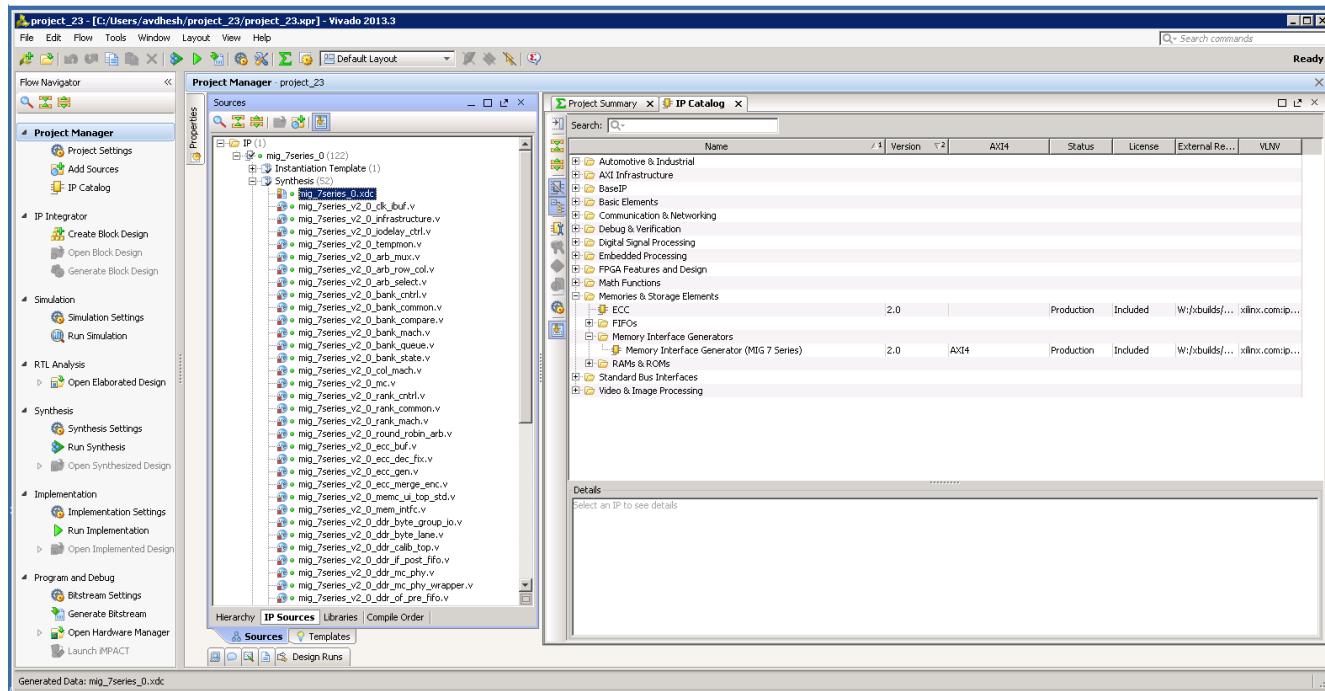
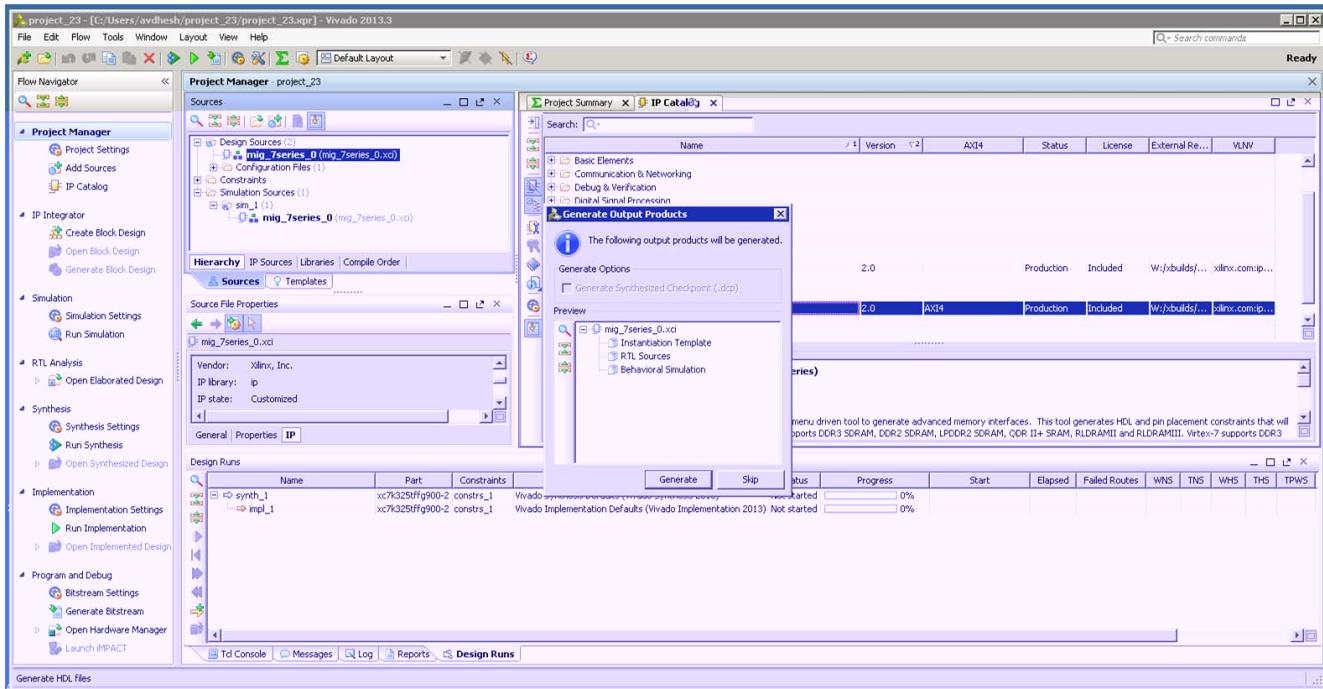


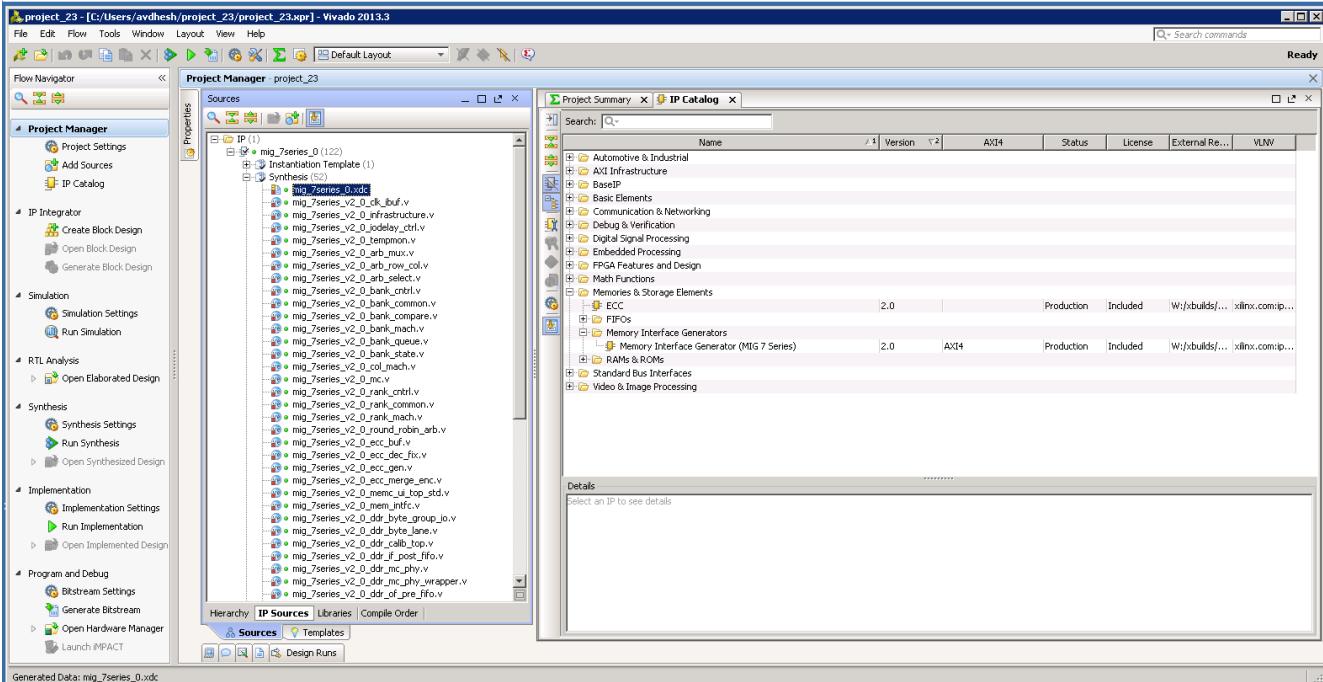
Figure 4-31:

7. Clicking **Generate Output Products** option brings up the **Manage Outputs** window (Figure 4-32).



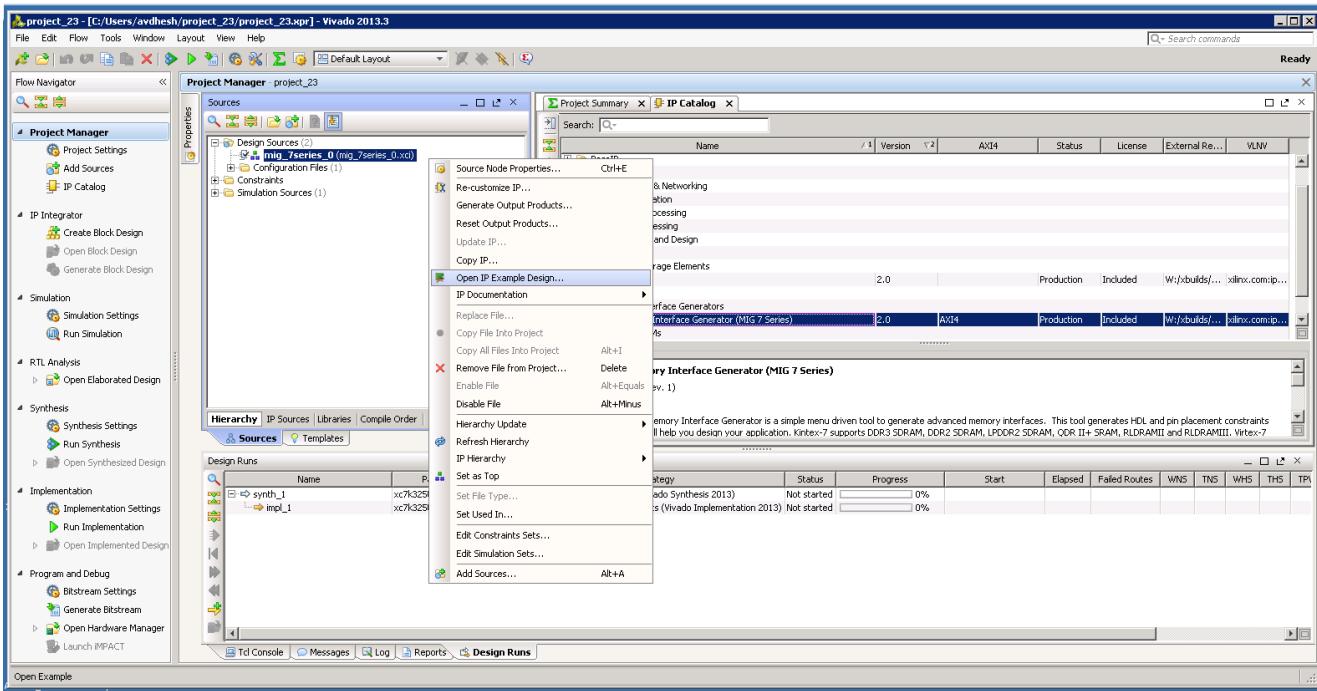
*Figure 4-32:*

8. All user-design RTL files and constraints files (XDC files) can be viewed in the Sources > Libraries tab (Figure 4-33).



*Figure 4-33:*

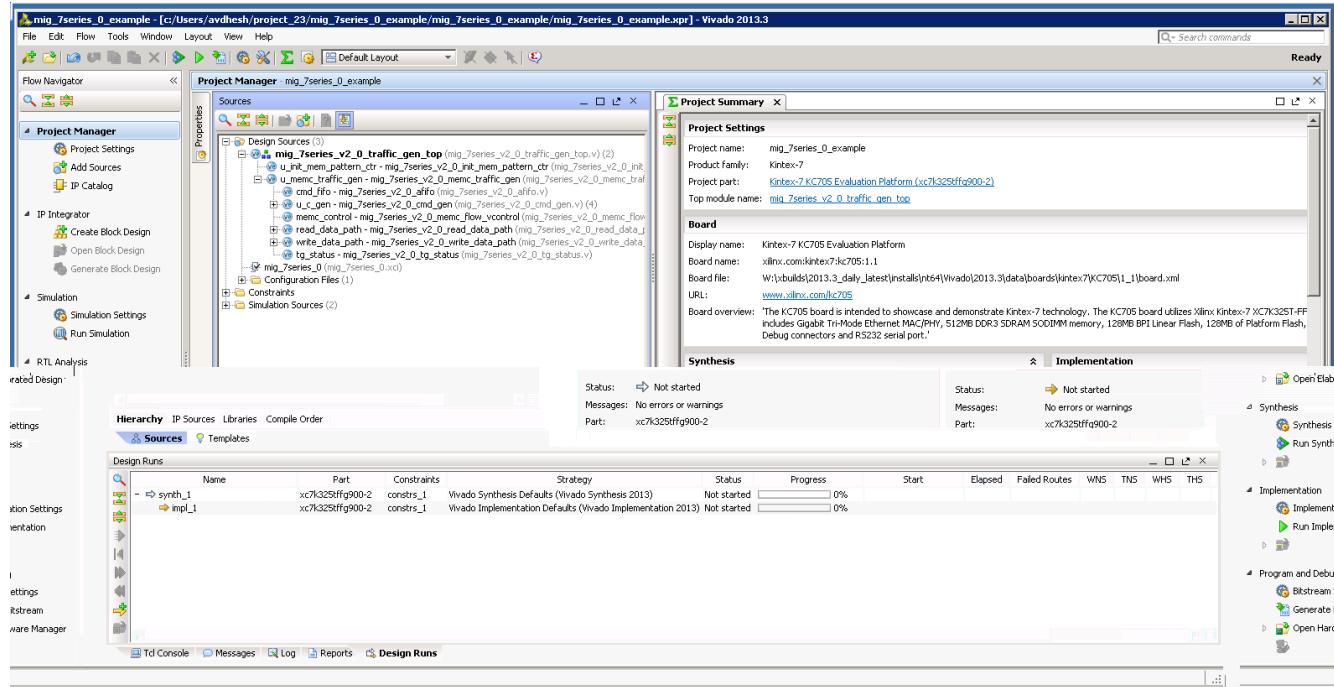
9. The Vivado Design Suite supports **Open IP Example Design** flow. To create the example design using this flow right-click the IP in the **Source Window**, as shown in [Figure 4-34](#) and select.



*Figure 4-34:*

10. This option creates a new Vivado project. Selecting the menu brings up a dialog box, which guides you to the directory for a new design project. Select a directory (or use the defaults) and click **OK**.

This launches a new Vivado project with all example design files and a copy of the IP. This project has **example\_top** as the Implementation top directory, and **sim\_tb\_top** as the Simulation top directory, as shown in [Figure 4-35](#).



*Figure 4-35:*

11. Click **Generate Bitstream** under **Project Manager > Program and Debug** to generate the BIT file for the generated design.

The `<project directory>/<project directory>.runs/ impl_1` directory includes all report files generated for the project after running the implementation. It is also possible to run the simulation in this project.

12. Recustomization of the MIG IP core can be done by using the **Recustomize IP** option. It is not recommended to recustomize the IP in the `example_design` project. The correct solution is to close the `example_design` project, go back to original project and customize there. Right-click the XCI file and click **Recustomize IP** (Figure 4-36) to open the MIG GUI and regenerate the design with the preferred options.

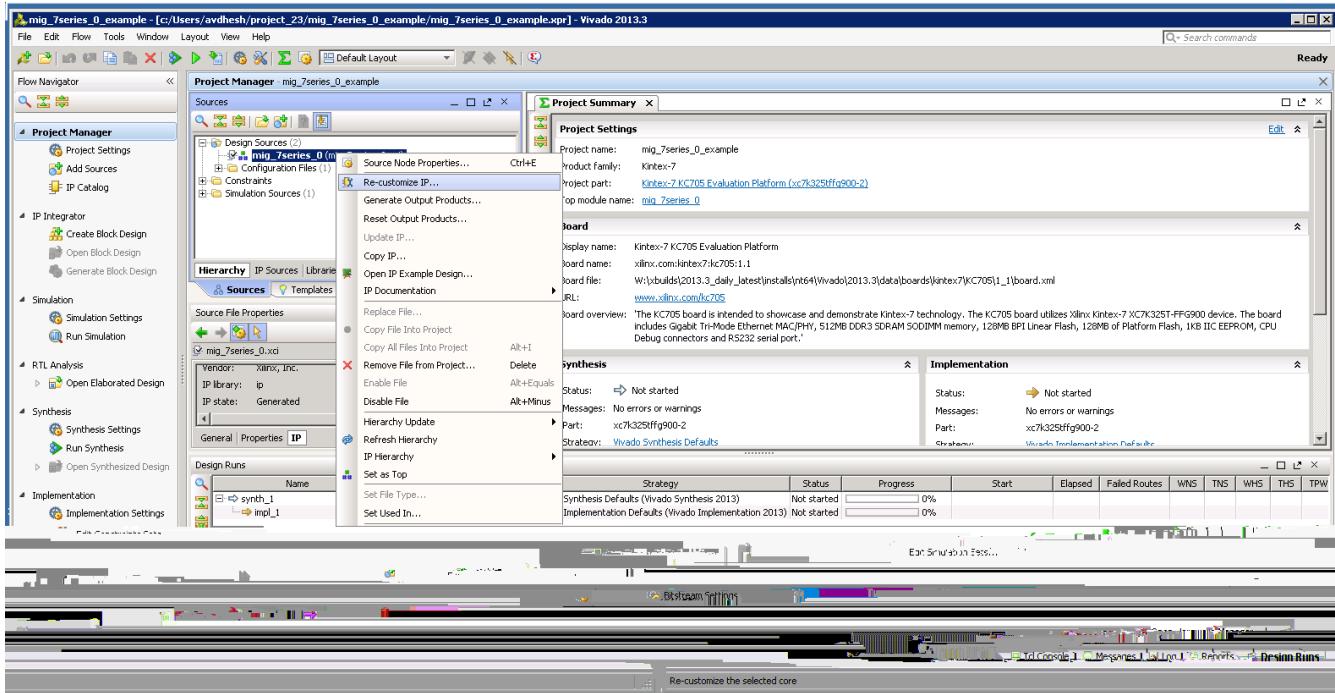


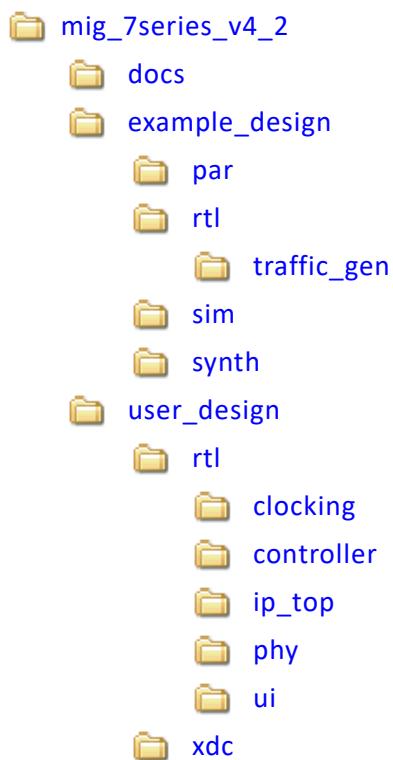
Figure 4-36:

### Directory Structure and File Descriptions

The MIG tool outputs are generated with folder name <component\_name>.

The output directory structure of the selected Memory Controller (MC) design from the MIG tool is shown here. In the <component\_name> directory, three folders are created:

- **docs**
- **example\_design**
- **user\_design**



The 7 series FPGAs core directories and their associated files are listed in this section for Vivado implementations.

**<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 4-1](#)).

*Table 4-1:*

example_top.v	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 4-2](#)).

*Table 4-2:*

memc_traffic_gen.v	This is the top-level of the traffic generator.
cmd_gen.v	This is the command generator. This module provides independent control of generating the types of commands, addresses, and burst lengths.
cmd_prbs_gen.v	This is a pseudo-random binary sequence (PRBS) generator for generating PRBS commands, addresses, and burst lengths.
memc_flow_vcontrol.v	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	This is the top-level for the read datapath.
read_posted_fifo.v	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	This module generates timing control for reads and ready signals to memc_flow_vcontrol.v.
write_data_path.v	This is the top-level for the write datapath.
wr_data_g.v	This module generates timing control for writes and ready signals to memc_flow_vcontrol.v.
s7ven_data_gen.v	This module generates different data patterns.
a_fifo.v	This is a synchronous FIFO using LUT RAMs.
data_prbs_gen.v	This is a 32-bit linear feedback shift register (LFSR) for generating PRBS data patterns.
init_mem_pattern_ctr.v	This module generates flow control logic for the traffic generator.
traffic_gen_top.v	This module is the top-level of the traffic generator and comprises the memc_traffic_gen and init_mem_pattern_ctr modules.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, the MIG 4.2 release module name of cmd\_gen in generated output is now mig\_7series\_v4\_2\_cmd\_gen.

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

[Table 4-3](#) lists the modules in the **example\_design/par** directory.

*Table 4-3:*

example_top.xdc	This is the XDC for the core and the example design.

### <component name>/example\_design/sim

Table 4-4 lists the modules in the `example_design/sim` directory.

Table 4-4:

mobile_ddr2_model.v	These are the LPDDR2 SDRAM models.
mobile_ddr2_model_parameters.vh	These files contain the LPDDR2 SDRAM model parameter setting.
ies_run.sh <sup>(1)</sup>	Linux Executable file for simulating the design using IES simulator.
vcs_run.sh <sup>(1)</sup>	Linux Executable file for simulating the design using VCS simulator.
readme.txt <sup>(1)</sup>	Contains the details and prerequisites for simulating the designs using Mentor Graphics Questa Advanced Simulator, IES, and VCS simulators.
sim_tb_top.v	This is the simulation top file.

**Notes:**

1. The `ies_run.sh` and `vcs_run.sh` files are generated in the folder `mig_7series_0_ex/imports` when the example design is created using **Open IP Example Design** for the design generated with **Component Name** entered in Vivado IDE as `mig_7series_0`.

### <component name>/user\_design

The `user_design` folder contains the following:

- `rtl` and `xdc` folders
- Top-level wrapper module `<component_name>.v/vhd`
- Top-level modules `<component_name>_mig.v/vhd` and `<component_name>_mig_sim.v/vhd`

The top-level wrapper file `<component_name>.v/vhd` has an instantiation of top-level file `<component_name>_mig.v/vhd`. Top-level wrapper file has no parameter declarations and all the port declarations are of fixed width.

Top-level files `<component_name>_mig.v/vhd` and `<component_name>_mig_sim.v/vhd` have the same module name as `<component_name>_mig`. These two files are same in all respects except that the file `<component_name>_mig_sim.v/vhd` has parameter values set for simulation where calibration is in fast mode `viz., SIM_BYPASS_INIT_CAL = "FAST"` etc.




---

**IMPORTANT:** The top-level file `<component_name>_mig.v/vhd` is used for design synthesis and implementation, whereas the top-level file `<component_name>_mig_sim.v/vhd` is used in simulations.

---

The top-level wrapper file 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 4-5](#)).

*Table 4-5:*

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

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of `clk_ibuf` in generated output is now `mig_7series_v4_2_clk_ibuf`.

### **`user_design/rtl/controller`**

This directory contains the Memory Controller that is instantiated in the example design ([Table 4-6](#)).

*Table 4-6:*

<code>arb_mux.v</code>	This is the top-level module of arbitration logic.
<code>arb_row_col.v</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</code>	This module selects a row and column command from the request information provided by the bank machines.
<code>bank_cntrl.v</code>	This structural block instantiates the three subblocks that comprise the bank machine.
<code>bank_common.v</code>	This module computes various items that cross all of the bank machines.
<code>bank_compare.v</code>	This module stores the request for a bank machine.
<code>bank_mach.v</code>	This is the top-level bank machine block.
<code>bank_queue.v</code>	This is the bank machine queue controller.
<code>bank_state.v</code>	This is the primary bank state machine.
<code>col_mach.v</code>	This module manages the DQ bus.
<code>mc.v</code>	This is the top-level module of the Memory Controller.
<code>mem_intf.v</code>	This top-level memory interface block instantiates the controller and the PHY.
<code>rank_cntrl.v</code>	This module manages various rank-level timing parameters.

Table 4-6:

(Cont'd)

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

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of arb\_mux in generated output is now mig\_7series\_v4\_2\_arb\_mux.

**user\_design/rtl/ip\_top**

This directory contains the user design ([Table 4-7](#)).

Table 4-7:

mem_intf.v	This is the top-level memory interface block that instantiates the controller and the PHY.
memc_ui_top.v	This is the top-level Memory Controller module.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of mem\_intf in generated output is now mig\_7series\_v4\_2\_mem\_intf.

**user\_design/rtl/phy**

This directory contains the 7 series FPGA memory interface PHY implementation ([Table 4-8](#)).

Table 4-8:

ddr_byte_group_io	This module contains the parameterizable I/O logic instantiations and the I/O terminations for a single byte lane.
ddr_byte_lane	This module contains the primitive instantiations required within an output or input byte lane.
ddr_calib_top	This is the top-level module for the memory physical layer interface.
ddr_mc_phy	This module is a parameterizable wrapper instantiating up to three I/O banks, each with 4-lane PHY primitives.
ddr_mc_phy_wrapper	This wrapper file encompasses the MC_PHY module instantiation and handles the vector remapping between the MC_PHY ports and your LPDDR2 ports.
ddr_of_pre_fifo	This module extends the depth of a PHASER OUT_FIFO up to four entries.
ddr_phy_4lanes	This module is the parameterizable 4-lane PHY in an I/O bank.
ddr_phy_init_lpddr2	This module contains the memory initialization and overall master state control during initialization and calibration.

Table 4-8:

(Cont'd)

ddr_phy_rdlvl	This module contains the Read leveling Stage1 calibration logic.
ddr_phy_top	This is the top-level module for the physical layer.
ddr_phy_wrlvl_off_delay.v	This module sets up the command and write datapath delays.
ddr_bitslip.v	This module contains the shift registers and MUXes to compensate the bitslip and align the read data.
ddr_phy_pd.v	This module contains the Phase detector logic to compensate any drift over the voltage and temperature variations.
ddr_phy_pd_top.v	This module is the top instance of phy_pd. This is used to instantiate Phase detector based on different calibration mode of parallel or sequential detection.
ddr_phy_prbs_rdlvl.v	This module contains calibration logic to perform data valid window detection and capture clock alignment using PRBS data pattern.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of ddr\_byte\_group\_io in generated output is now mig\_7series\_v4\_2\_ddr\_byte\_group\_io.

**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 4-9](#)).

Table 4-9:

ui_cmd.v	This is the user interface command port.
ui_rd_data.v	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	This is the user interface write buffer.
ui_top.v	This is the top-level of the Memory Controller user interface.

**Notes:**

1. All file names are prefixed with the MIG version number. For example, for the MIG 4.2 release module name of ui\_cmd in generated output is now mig\_7series\_v4\_2\_ui\_cmd.

**<component\_name>/user\_design/xdc**

[Table 4-10](#) lists the modules in the **user\_design/xdc** directory.

Table 4-10:

<component_name>.xdc	This is the XDC for the core and the user design.

This feature verifies the input XDC for bank selections, byte selections, and pin allocation. It also generates errors and warnings in a separate dialog box when you click **Validate** on the page. This feature is useful to verify the XDC for any pinout changes made after the design is generated from the MIG tool. You must load the MIG generated **.prj** file, the original **.prj** file without any modifications, and the XDC that needs to be verified. In the Vivado tool, the Re-customize IP 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 XDC is not sufficient; it is mandatory to proceed with design generation to get the XDC with updated clock and phaser related constraints and RTL top-level module for various updated Map parameters.

The Update Design feature is required in the following scenarios:

- A pinout is generated using an older version of MIG and the design is to be revised to the current version of MIG. In MIG the pinout allocation algorithms have been changed for certain MIG designs.
- A pinout is generated independent of MIG or is modified after the design is generated. When a design is generated from MIG, the XDC and HDL code are generated with the correct constraints.

Here are the rules verified from the input XDC:

- If a pin is allocated to more than one signal, the tool reports an error. Further verification is not done if the XDC 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.
  - $V_{REF}$  I/Os should be used as GPIOs when an internal  $V_{REF}$  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 DQS I/O pair.

- An FPGA byte lane should not contain pins related to two different strobe sets.
- $V_{REF}$  I/O can be used only when the internal  $V_{REF}$  is chosen.
- Verified address pin rules:
  - Address signals cannot mix with data bytes.
  - It can use any number of isolated byte lanes.
  - Memory clock pins should be allocated to DQS I/O only.
  - Except memory clock pins, any other Address/Control pin should not be allocated to DQS.
- 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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  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, you need to check whether the reference voltage pins are unallocated in the bank or the internal  $V_{REF}$  is used.
  - Status signals:
    - The sys\_rst signal should be allocated in the bank where the  $V_{REF}$  I/O is unallocated or the internal  $V_{REF}$  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.

## Overview

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

## *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 4-37](#).

lpddr2\_sim\_tb\_top

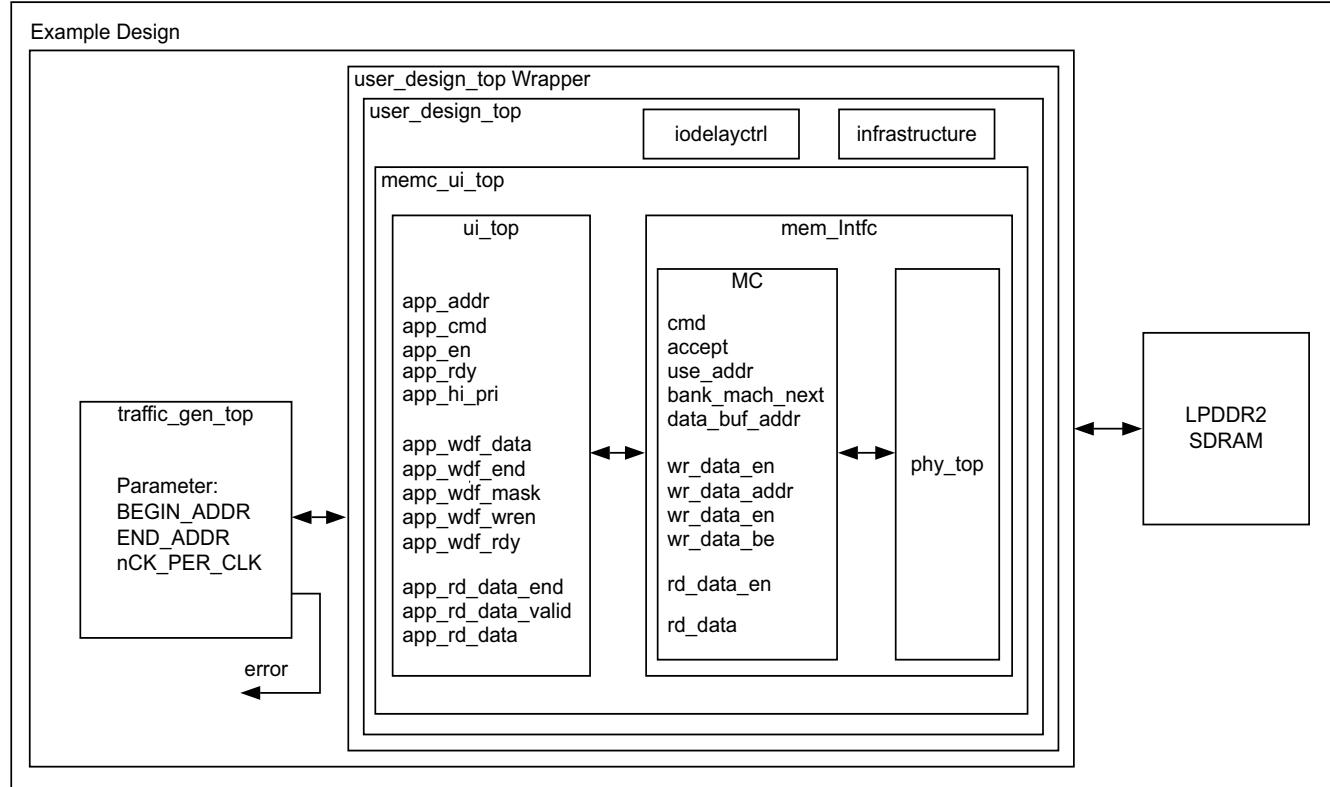


Figure 4-37:

[Figure 4-38](#) shows the simulation result of a simple read and write transaction between the **tb\_top** and **memc\_intfc** modules.

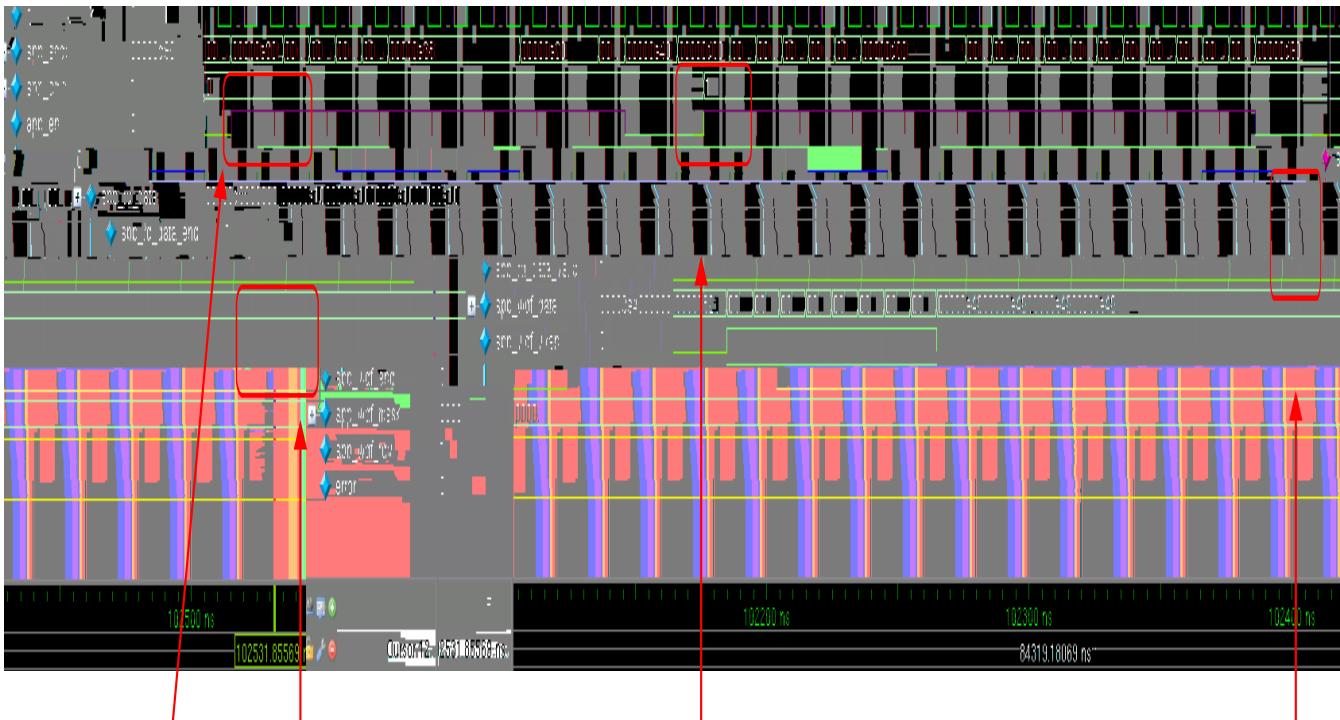


Figure 4-38:

### 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.

You 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 Vivado logic analyzer feature.

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) and addresses are determined by PRBS generator logic in the traffic generator module.

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

The 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 4-11](#) describes these parameters.

*Table 4-11:*

FAMILY	Indicates the family type.	"VIRTEX7"
MEMORY_TYPE	Indicate the Memory Controller type.	"LPDDR2"
nCK_PER_CLK	This is the Memory Controller clock to DRAM clock ratio. This parameter should <b>not</b> be changed.	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 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.	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.

Table 4-11:

(Cont'd)

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.
PRBS_SADDR_MASK_POS	Sets the 32-bit OR MASK position.	This parameter is used with the PRBS address generator to shift random addresses up into the port address space. The START_ADDRESS value is ORed with the PRBS address for bit positions that have a 1 in this mask
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 increased 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 powers 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 4-11:

(Cont'd)

		Valid settings for this parameter are: <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>Ox1: FIXED – 32 bits of fixed_data.</li> <li>Ox2: ADDRESS – 32 bits address as data.</li> <li>Ox3: HAMMER</li> <li>Ox4: SIMPLE8 – Simple 8 data pattern that repeats every 8 words.</li> <li>Ox5: WALKING1s – Walking 1s are on the DQ pins.</li> <li>Ox6: WALKING0s – Walking 0s are on the DQ pins.</li> <li>Ox7: PRBS – A 32-stage LFSR generates random data.</li> <li>Ox9: SLOW HAMMER – This is the slow MHz hammer data pattern.</li> <li>OxA: PHY_CALIB pattern – OxFF, O0, AA, 55, 55, AA, 99, 66. This mode only generates READ commands at address zero.</li> </ul> </li> </ul>
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.	Valid values: 0 to 32.
CMDS_GAP_DELAY	This parameter allows pause delay between each user burst command.	This parameter only applies to the Hammer pattern. Valid settings for this parameter are 0 to NUM_DQ_PINS. When value = NUM_DQ_PINS, all DQ pins have the same Hammer pattern.
SEL_VICTIM_LINE	Select a victim DQ line whose state is always at logic High.	

Table 4-11:

(Cont'd)

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.
----------	---	--

**Notes:**

1. 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 you to modify the command pattern in real-time using the Vivado logic analyzer feature 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 4-12](#) shows some examples of setting the two mask parameters.

*Table 4-12:*

0x1000	0xFFFF	0x00001000	0xFFFF0000
0x2000	0xFFFF	0x00002000	0xFFFF0000
0x3000	0xFFFF	0x00003000	0xFFFF0000
0x4000	0xFFFF	0x00004000	0xFFFF0000
0x5000	0xFFFF	0x00005000	0xFFFF0000
0x2000	0x1FFF	0x00002000	0xFFFFE000
0x2000	0x2FFF	0x00002000	0xFFFFD000
0x2000	0x3FFF	0x00002000	0xFFFFC000
0x2000	0x4FFF	0x00002000	0xFFFF8000
0x2000	0x5FFF	0x00002000	0xFFFF8000
0x2000	0x6FFF	0x00002000	0xFFFF8000
0x2000	0x7FFF	0x00002000	0xFFFF8000
0x2000	0x8FFF	0x00002000	0xFFFF0000
0x2000	0x9FFF	0x00002000	0xFFFF0000

Table 4-12:

(Cont'd)

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 4-13](#).

Table 4-13:

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 QDR II+ user interface.

Table 4-13:

(Cont'd)

vio_modify_enable	Input	Allow vio_xxxx_mode_value to alter traffic pattern.
vio_data_mode_value[3:0]	Input	<p>Valid settings for this signal are:</p> <ul style="list-style-type: none"> <li>• Ox0: Reserved.</li> <li>• Ox1: FIXED – 32 bits of fixed_data as defined through fixed_data_i inputs.</li> <li>• Ox2: ADDRESS – 32 bits address as data. Data is generated based on the logical address space. If a design has a 256-bit user data bus, each write beat in the user bus would have a 256/8 address increment in byte boundary. If the starting address is 1,300, the data is 1,300, followed by 1,320 in the next cycle. To simplify the logic, the user data pattern is a repeat of the increment of the address value Bits[31:0].</li> <li>• Ox3: 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>• Ox4: SIMPLE8 – Simple 8 data pattern that repeats every 8 words. The patterns can be defined by the "simple_datax" inputs.</li> <li>• Ox5: 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>• Ox6: 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>• Ox7: 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>• Ox9: SLOW HAMMER – This is the slow MHz hammer data pattern.</li> <li>• OxA: PHY_CALIB pattern – OxFF, OO, 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>• Ox1: FIXED address mode. The address comes from the fixed_addr_i input bus. With FIXED address mode, the data_mode is limited to the fixed_data_input. No PRBS data pattern is generated.</li> <li>• Ox2: 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>• Ox3: SEQUENTIAL address mode. The address is generated from the internal address counter. The increment is determined by the user interface port width.</li> </ul>

		Valid settings for this signal are:
vio_instr_mode_value[3:0]	Input	<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	Valid settings for this signal are:
		<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>
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 address as data. 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.

c a w 5 e

3/4nbq

ssstw

Table 4-13:

(Cont'd)

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_b1_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.

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. The **bl\_mode\_i** is set to 01 to get the burst length from the **fixed\_b1\_i** input.
5. The **fixed\_b1\_i** input is set to either 16 or 32.
6. The **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. The **addr\_mode\_i** is set to 11 for the sequential address mode to fill up the memory space.
9. The **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).

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 4-13, page 563](#).

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.

### ***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

The Questa Advanced Simulator, Vivado Simulator, IES, and VCS simulation tools are used for verification of the MIG IP core at each software release. Script files to run simulations with IES and VCS simulators are generated in MIG generated output. Simulations using Questa Advanced Simulator and Vivado simulators can be done through the Vivado Tcl Console commands or in the Vivado IDE.



**IMPORTANT:** *Other simulation tools can be used for MIG IP core simulation but are not specifically verified by Xilinx.*

---

To run the simulation, go to this directory:

```
<project_dir>/<Component_Name>_ex/imports
```

For a project created with the name set as **project\_1** and the Component Name entered in Vivado IDE as **mig\_7series\_0**, go to the directory as follows:

```
project_1/mig_7series_0_ex/imports
```

IES and VCS simulation scripts are meant to be executed only in Linux operating systems.

The **ies\_run.sh** and **vcs\_run.sh** files are the executable files for running simulations using IES and VCS simulators respectively. Library files should be added to the **ies\_run.sh** and **vcs\_run.sh** files respectively. See the **readme.txt** file for details regarding simulations using IES and VCS.

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings** (Figure 4-39).

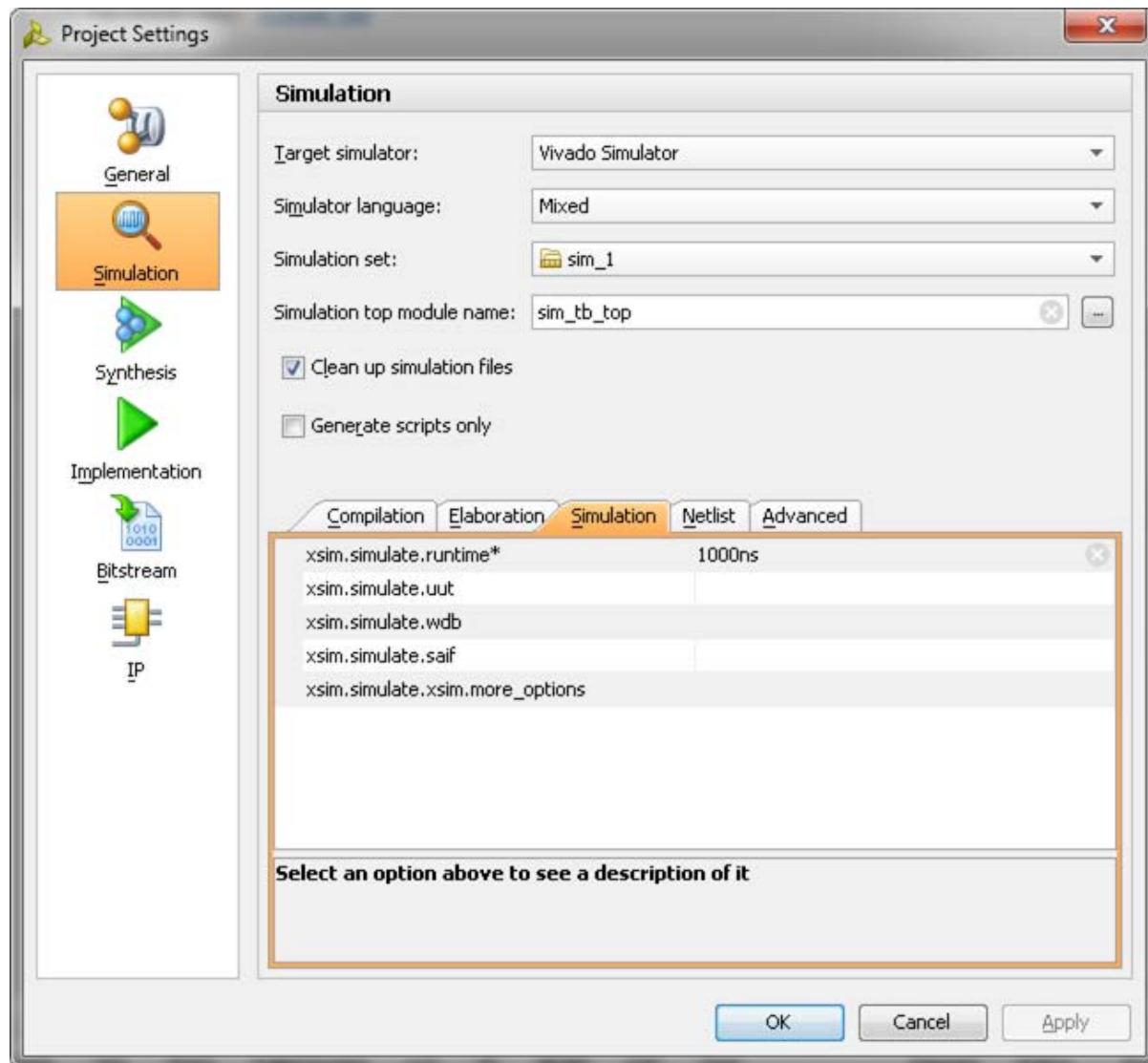
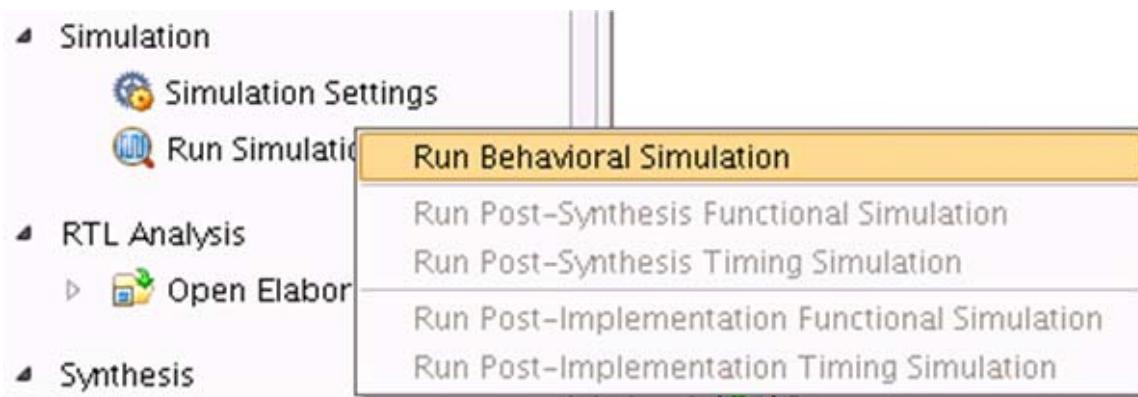


Figure 4-39:

2. Under the **Simulation** tab as shown in Figure 4-39, set the **xsim.simulate.runtime** as 1 ms (there are simulation RTL directives which stop the simulation after a certain period of time, which is less than 1 ms). Apply the settings and select **OK**.

3. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 4-40](#).



*Figure 4-40:*

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Questa Advanced Simulator/ModelSim.
  - a. Browse to the **Compiled libraries location** and set the path on **Compiled libraries location** option.
  - b. Under the **Simulation** tab, set the **modelsim.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms), set **modelsim.simulate.vsim.more\_options** to **-novopt** as shown in [Figure 4-39](#).
3. Apply the settings and select **OK**.

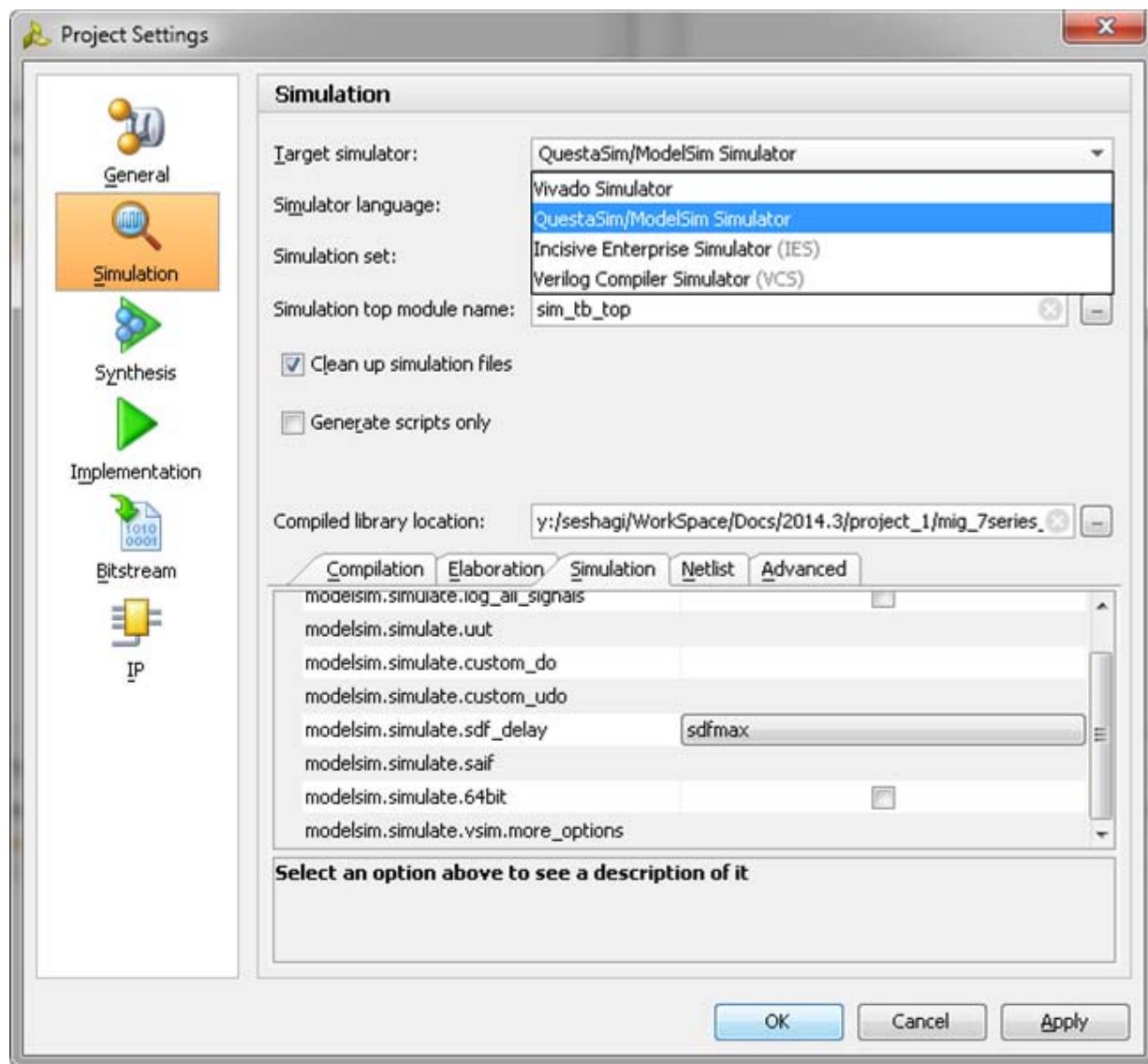


Figure 4-41:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 4-40](#).
5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG 900)* [\[Ref 8\]](#).
  
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Verilog Compiler Simulator (VCS).



5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [Ref 8].
  
1. In the **Open IP Example Design Vivado** project, under **Flow Navigator** select **Simulation Settings**.
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
  - a. Browse to the **Compiled libraries location** and set the path on **Compiles libraries location** option.
  - b. Under the **Compilation** tab, set the **ies.compile.ncvlog.more\_options** to **-sv**.
  - c. Under the **Elaboration** tab, set the **ies.elaborate.ncelab.more\_options** to **-namemap\_mixgen**.
  - d. Under the **Simulation** tab, set the **ies.simulate.runtime** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time which is less than 1 ms) as shown in [Figure 4-43](#).

3. Apply the settings and select **OK**.

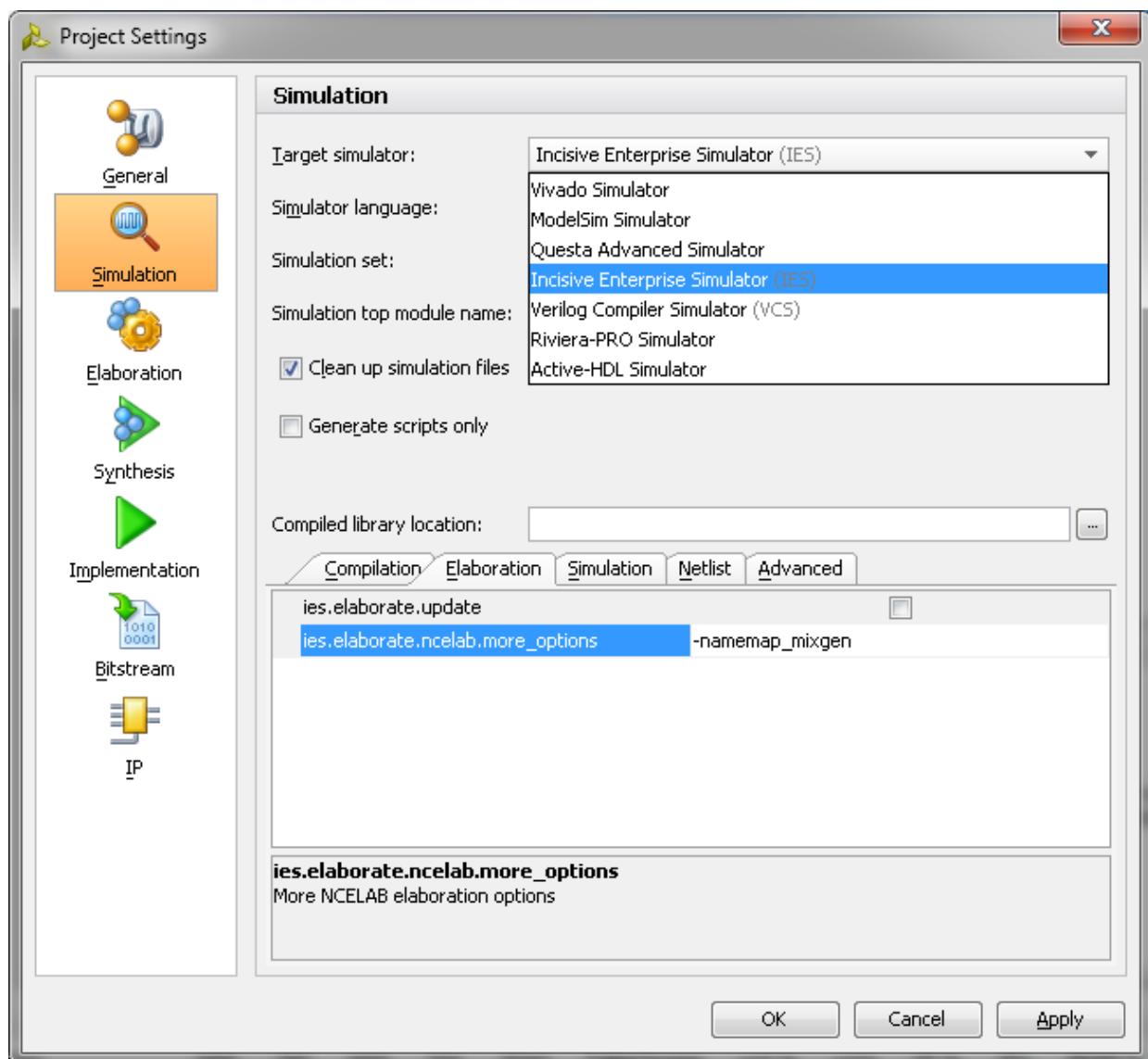
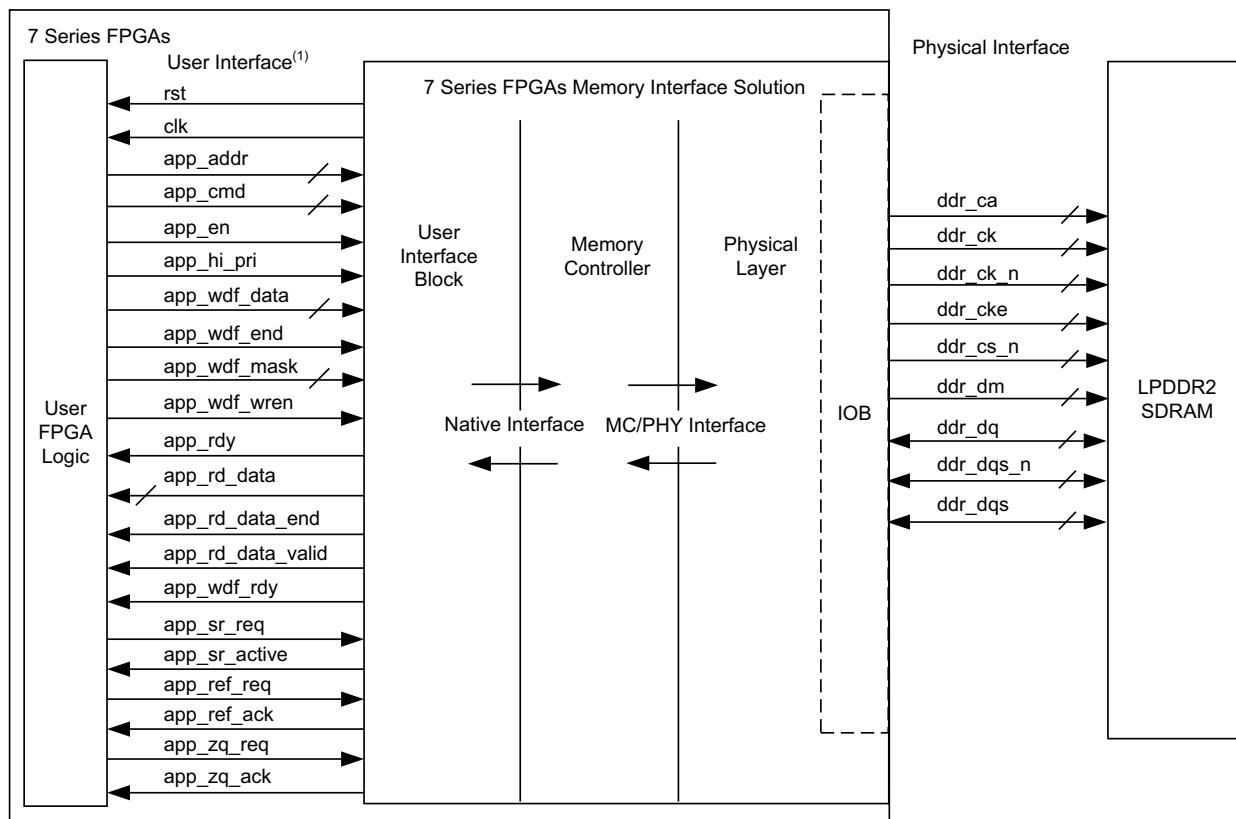


Figure 4-43:

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** as shown in [Figure 4-40](#).
5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation (UG900)* [\[Ref 8\]](#).

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

The 7 series FPGAs memory interface solutions core is shown in [Figure 4-44](#).



1. System clock (sys\_clk\_p and sys\_clk\_n/sys\_clk\_i), Reference clock (clk\_ref\_p and clk\_ref\_n/clk\_ref\_i), and system reset (sys\_rst\_n) port connections are not shown in block diagram.

*Figure 4-44:*

### User FPGA Logic

The user FPGA logic block shown in [Figure 4-44](#) is any FPGA design that requires to be connected to an external LPDDR2 SDRAM. The user FPGA logic connects to the Memory Controller through the user interface. An example user FPGA logic is provided with the core.

## ***User Interface Block and User Interface***

The UI block presents the UI to the user FPGA logic 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 backend 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 backend 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 IDELAYCTRL and then uses the IODELAY\_GROUP attribute (see the `iodelay_ctrl.v` module). Based on this attribute, the Vivado tool properly replicates IDELAYCTRLs 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 Vivado tool replicates the IDELAYCTRLs for each region where the IDELAY blocks exist. When a user creates a multicontroller design on their own, each MIG output has the component instantiated with the primitive. This violates the rules for IDELAYCTRLs and the usage of the IODELAY\_GRP attribute. IDELAYCTRLs need to have only one instantiation of the component with the attribute set properly, and allow the tools to replicate as needed.

The UI is shown in [Table 4-14](#) and connects to an FPGA user design to allow access to an external memory device.

*Table 4-14:*

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], 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.
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_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_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.
ui_clk	Output	This UI clock must be a half or quarter of the DRAM clock.
init_calib_complete	Output	PHY asserts init_calib_complete when calibration is finished.
ui_clk_sync_RST	Output	This is the active-High UI reset.

***app\_addr[ADDR\_WDTH - 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 you.

***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 4-15](#).

*Table 4-15:*

Read	001
Write	000

***app\_en***

This input strobes in a request. You 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\_WDTH - 1:0]***

This bus indicates which bytes of **app\_wdf\_data[]** are written to the external memory and which bytes remain in their current state. The bytes are masked by setting a value of 1 to the corresponding bits in **app\_wdf\_mask**. For example, if the application data width is 256, the mask width takes a value of 32. The least significant byte [7:0] of **app\_wdf\_data** is masked using Bit[0] of **app\_wdf\_mask** and the most significant byte [255:248] of **app\_wdf\_data** is masked using Bit[31] of **app\_wdf\_mask**. Hence if you have to mask the last DWORD, that is, bytes 0, 1, 2, and 3 of **app\_wdf\_data**, the **app\_wdf\_mask** should be set to 32'h0000\_000F.

***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\_WDTH - 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 you 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\_WDTH - 1:0]*

This output contains the data read from the external memory.

### *app\_rd\_data\_end*

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

### *app\_rd\_data\_valid*

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

### *app\_wdf\_rdy*

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

### *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.

### *ui\_dk\_sync\_rst*

This is the reset output from the UI which is in synchronous with **ui\_clk**.

### *ui\_dk*

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

### *init\_calib\_complete*

The 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.

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

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 4-16](#).

*Table 4-16:*

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: Locates data while processing write commands. Places data while processing read commands.
hi_priority	Input	This input is reserved and should be connected to logic 0.
rank[]	Input	This input is reserved and should be connected to logic 0.
row[ROW_WIDTH - 1:0]	Input	This input selects the row address for the current request.
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 4-17](#).

Table 4-17:

Memory read	000
Memory write	001
Reserved	All other codes

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.

The user design asserts the **use\_addr** signal to strobe the request that was submitted to the native interface on the previous cycle.

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 4-18). These signals connect to the control, address, and data signals of a buffer in the user design.

Table 4-18:

wr_data[2 × nCK_PER_CLK × 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 × nCK_PER_CLK × DATA_WIDTH/8 – 1:0]	Input	This input provides the byte enable for the write data.

Table 4-18:

(Cont'd)

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.

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.

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.

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.

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.

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 4-19). 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 4-19:

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.

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.

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.

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.

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 4-20 lists the native interface maintenance command signals.

Table 4-20:

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 4-20:

(Cont'd)

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.

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.

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.

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.

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.

The PHY design requires that a MMCM module be used to generate various clocks, and both global and local clock networks are used to distribute the clock throughout the design. The PHY also requires one PLL in the same bank as the PLL. This MMCM compensates for the insertion delay of the BUFG to the PHY.

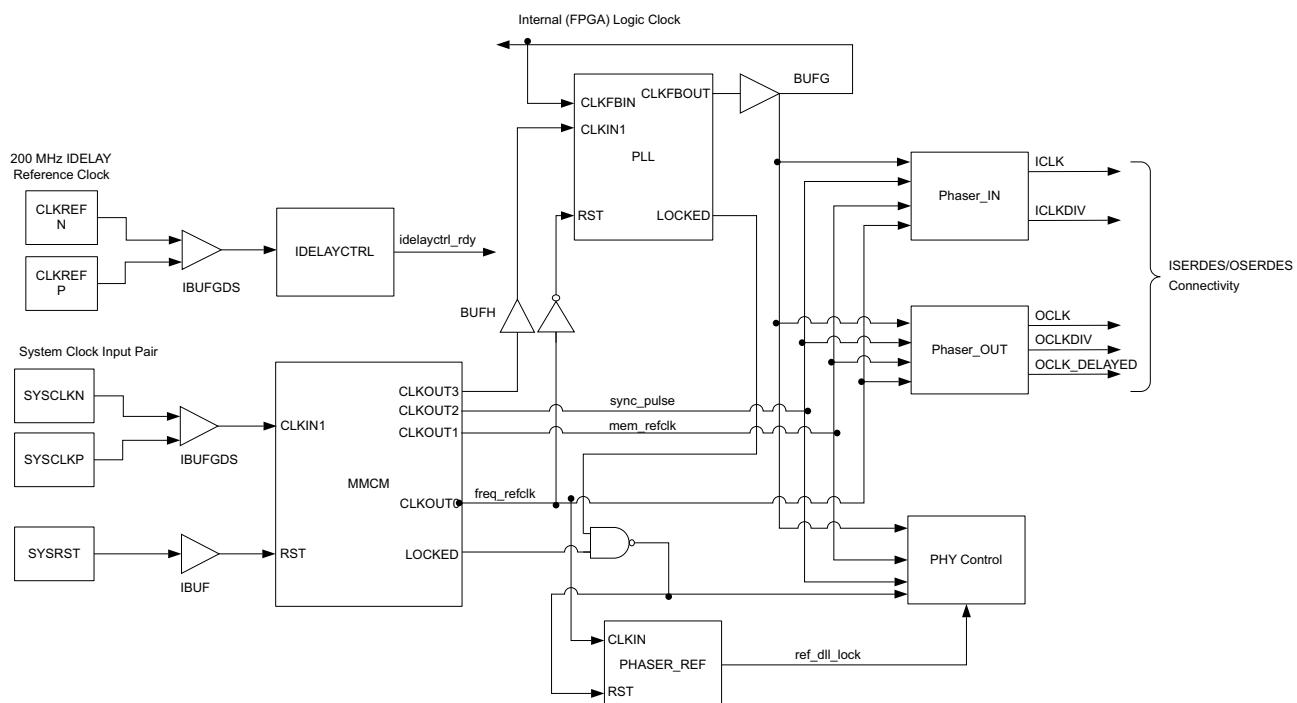
The clock generation and distribution circuitry and networks drive blocks within the PHY that can be divided roughly into four separate, general functions:

- Internal (FPGA) logic
- Write path (output) I/O logic
- Read path (input) and delay I/O logic
- IDELAY reference clock (200 MHz)

One MMCM and one PLL are required for the PHY. The MMCM 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 LPDDR2 SDRAM clock frequency range of 200 MHz to 333 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. The two phaser frequency reference clocks must be generated by the same MMCM so they are in phase with each other. The block diagram of the clocking architecture is shown in [Figure 4-45](#).

The default setting for the MMCM multiply (M) and divide (D) values is for the system clock input frequency to be equal to the memory clock frequency. This 1:1 ratio is not required. The MMCM input divider (D) can be any value listed in the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] as long as the MMCME2\_ADV operating conditions are met and the other constraints listed here are observed. The MMCM multiply (M) value must be between 1 and 16 inclusive. The MMCM output driver (O) for the memory clock must be 2. The MMCM 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 MMCM and the System Clock CCIO input, see [Design Guidelines, page 632](#).



*Figure 4-45:*

The details of the ISERDES/OSERDES connectivity are shown in [Figure 4-50](#) and [Figure 4-52](#).

## *Internal (FPGA) Logic Clock*

The internal FPGA logic is clocked by a global clocking resource at a half frequency of the LPDDR2 SDRAM clock frequency. This MMCM also outputs the high-speed LPDDR2 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° phase offset between the write DQS and its associated DQ bits.

The PHASER\_OUT also drives the signaling 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 4-50](#) and [Figure 4-52](#).

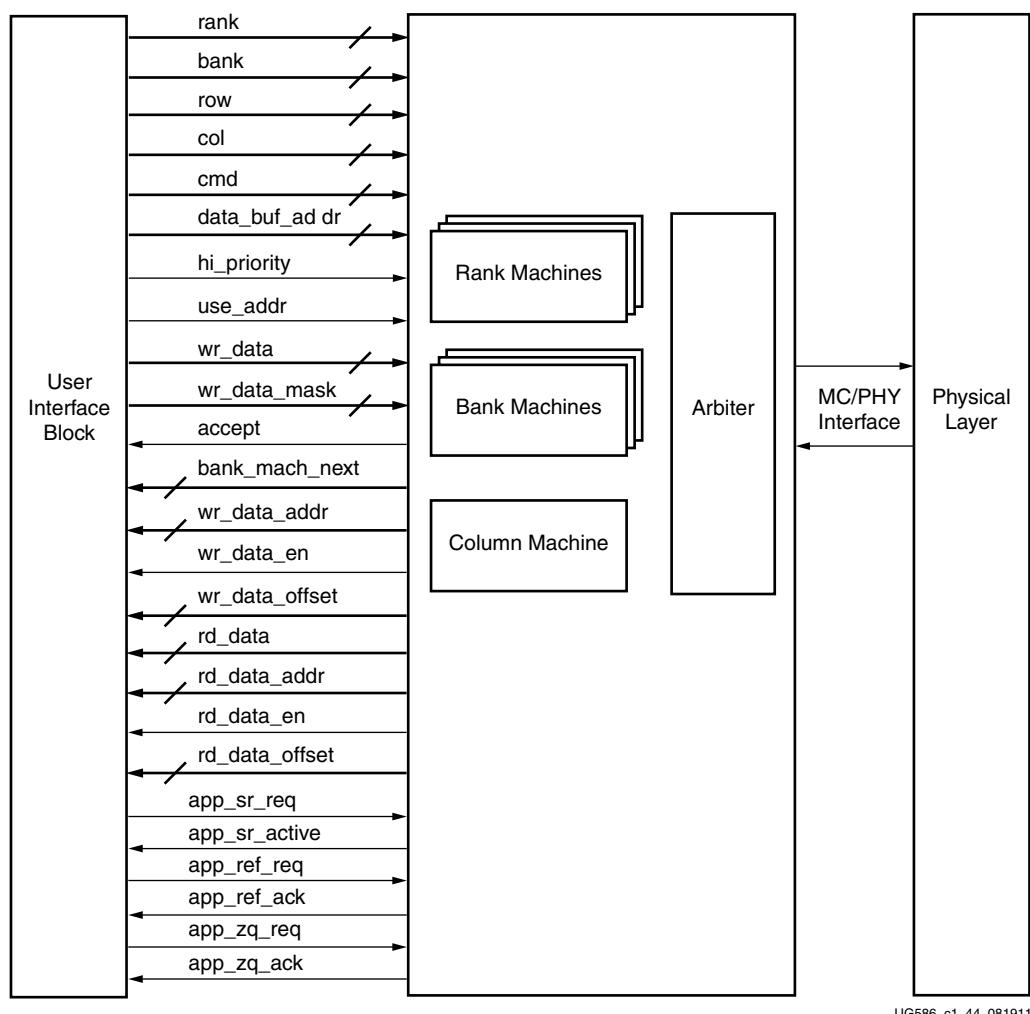
## *Read Path (Input) I/O Logic Clock*

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 generates two delayed clocks for LPDDR2 SDRAM data captures: read byte clock (ICLK) and read divided byte clock (ICLKDIV). ICLK is the delayed version of the frequency reference clock. 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 clocking details of the read path using PHASER\_IN is shown in [Figure 4-52](#).

## *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.

In the core default configuration, the Memory Controller (MC) resides between the UI block and the physical layer. This is depicted in [Figure 4-46](#).



*Figure 4-46:*

The Memory Controller is the primary logic block of the memory interface. It 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:

- Configurable number of “bank machines”
- Configurable number of “rank machines”
- Column machine
- Arbitration block

## Bank Machines

Most of the Memory Controller logic resides in the bank machines. 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.

The duration of a bank machine 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 your 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.

The controller supports three ordering modes:

- **STRICT** – In this mode the controller always issues commands to the memory in the exact order received at the native interface. This mode can be useful in situations that do not benefit from reordering and the lowest latency is desired. Because the read data comes back in order, the user interface layer might not be needed thus reducing latency. This mode is also useful for debugging.
- **NORM** – In this mode the controller reorders reads but not writes as needed to improve efficiency. All write requests are issued in the request order relative to all other write requests, and requests within a given rank-bank retire in order. This ensures that it is not possible to observe the result of a later write before an earlier write completes.

**Note:** This reordering is only visible at the native interface. The user interface reorders the read requests back into the original request order.

- **RELAXED** – This is the most efficient mode of the controller. Writes and reads can be reordered as needed for maximum efficiency between rank-bank queues. Thus in this mode it is possible to observe the reordering of writes. However, this behavior is not observable at the user interface layer because the requests are retired in order within a rank-bank and the user interface layer returns the read requests in order. Therefore the RELAXED mode is recommended for use with the user interface layer.

**Note:** This option is not selectable in the MIG GUI. To enable, generate the design with the synthesis options "Global" in the **Generate Output Products** settings. After generating the design, the design top-level RTL file should be edited and the ORDERING parameter should be changed to "RELAXED."

## *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, FPGA logic 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 command behavior to determine the optimum setting.

**Note:** The overall read latency of the MIG 7 series LPDDR2 core is dependent on how the Memory Controller is configured, but most critically on the target traffic/access pattern and the number of commands already in the pipeline before the read command is issued. Read latency is measured from the point where the read command is accepted by the user or native interface. Simulation should be run to analyze read latency.

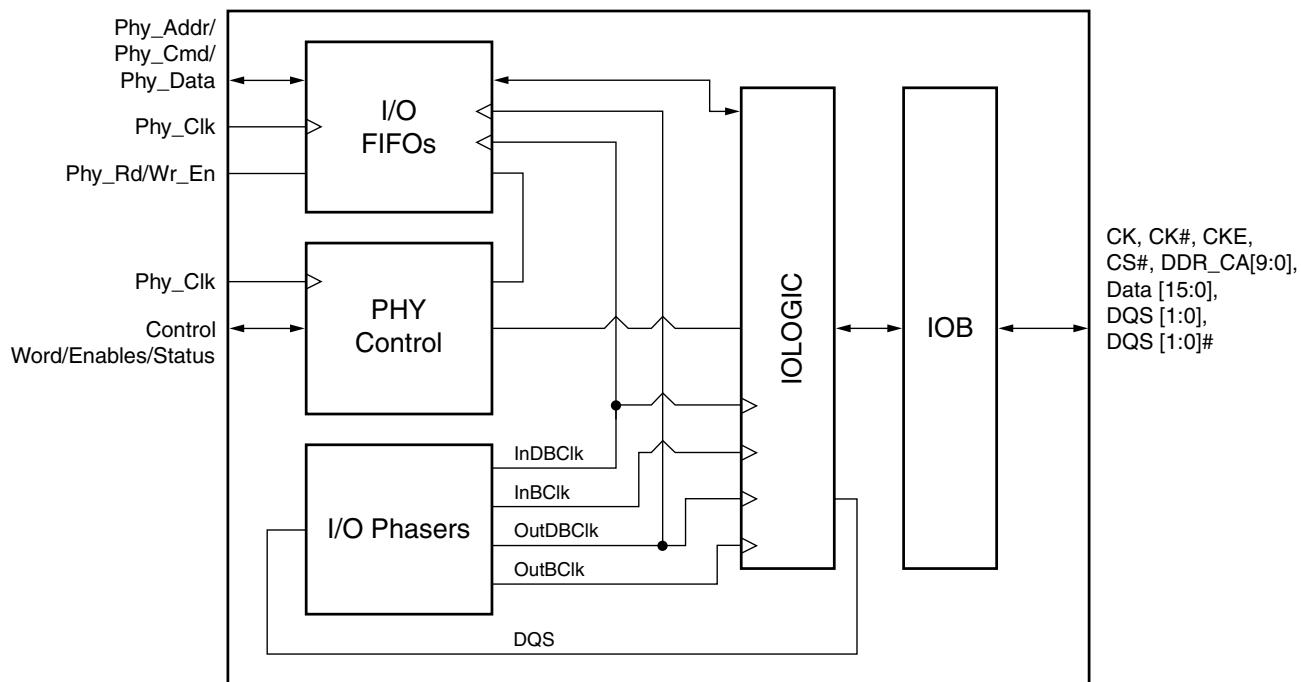
The PHY provides a physical interface to an external LPDDR2 SDRAM. The PHY generates the signal timing and sequencing required to interface to the memory device. It contains the clock-, address-, and control-generation logic, write and read datapaths, and state logic for initializing the SDRAM after 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 LPDDR2 SDRAMs. The MIG tool customizes the SDRAM type and numerous other design-specific parameters through top-level HDL parameters and constraints contained in a XDC file.

## *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 4-47](#) 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.



*Figure 4-47:*

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 LPDDR2 memory clock. A block diagram of the PHY design is shown in [Figure 4-48](#).

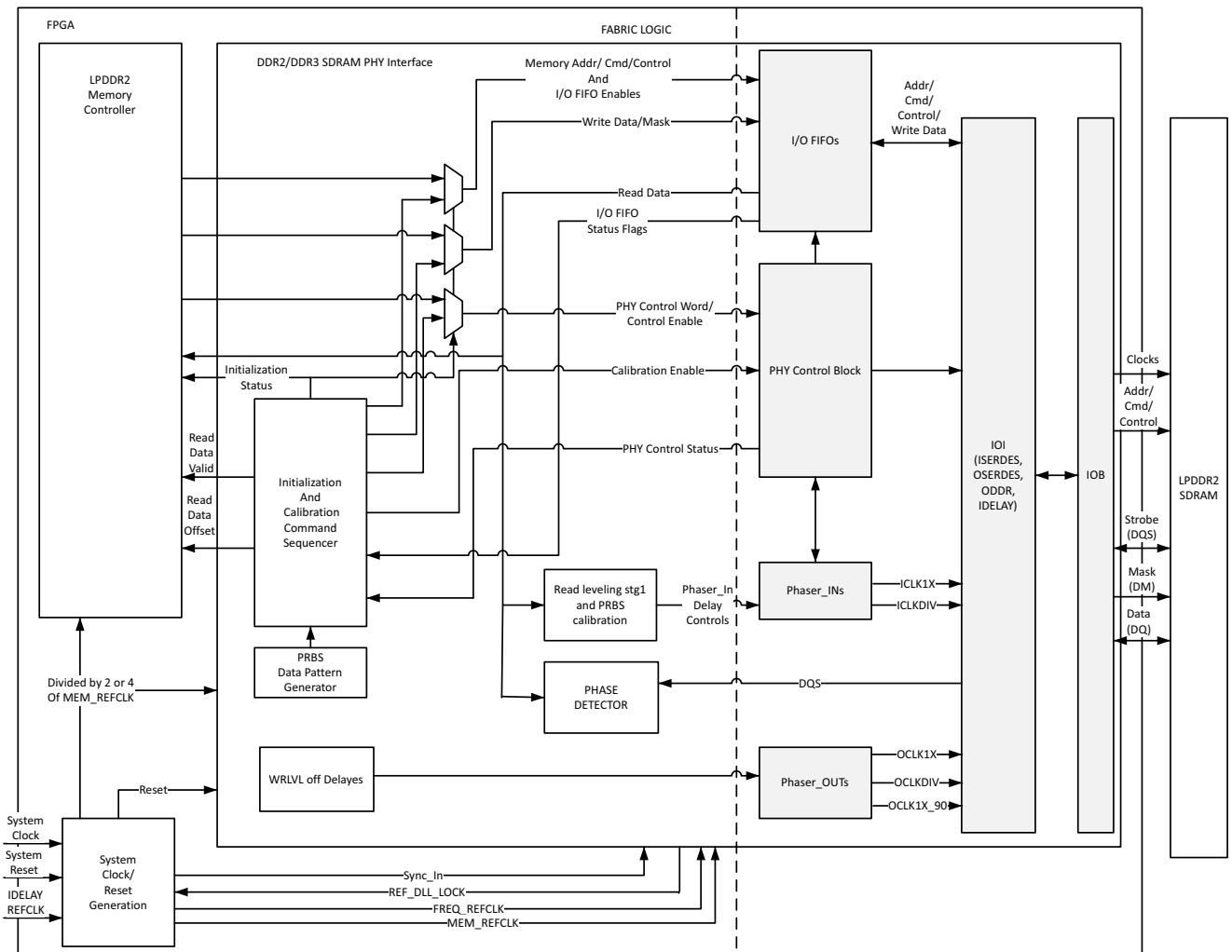


Figure 4-48:

### 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 4-49 shows the overall flow of memory initialization and the different stages of calibration.

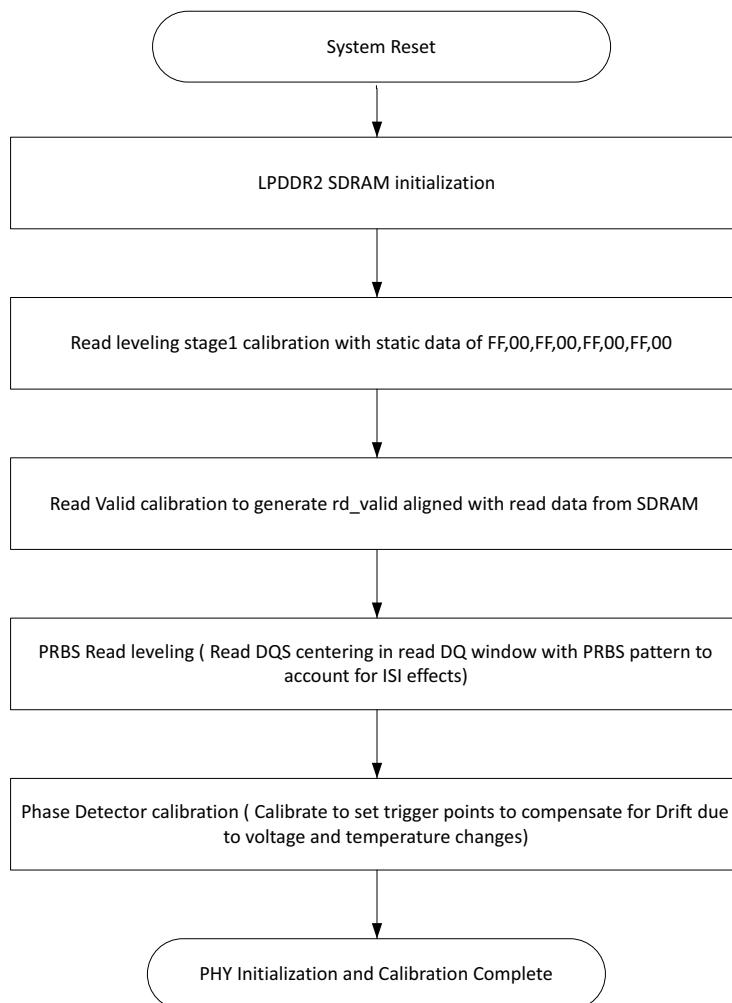


Figure 4-49:

The calibration stages in Figure 4-49 correspond to these sections:

- [Memory Initialization, page 604](#)
- [Read Leveling, page 604](#)
- [Read Valid Calibration, page 607](#)
- [PRBS Read Leveling, page 607](#)
- [Phase Detector, page 608](#)

## 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.

The PHY control block is the central 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 (1/2 the frequency of the LPDDR2 SDRAM clock) PHY\_Clk rate and processes the control words at the LPDDR2 SDRAM clock rate (CK frequency).

The calibration logic or the Memory Controller initiates a LPDDR2 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 LPDDR2 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 4-21](#). 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 4-21:*

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 4-22</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 anymore 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 4-22](#).

**Table 4-22:**

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	Reser	Low	Aux_Out	Control Offset	PHY Cmd	Pre	Delay	Slot																			

- **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 LPDDR2 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 4-23, page 598](#). These outputs are not used by the LPDDR2 interface generated by the MIG tool; they are set to 0.
- **Low Index (Bank)** – The dedicated PHY has internal counters that require this field to specify which of the eight LPDDR2 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.
- **Reserved** – This field must always be set to 2'b00.
- **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 LPDDR2 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 LPDDR2 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 LPDDR2 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 4-23:*

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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 SDRAM clock cycles after the associated read command is executed that the auxiliary output becomes active.

Table 4-23:

(Cont'd)

RD_DURATION_1	Vector[5:0]	This attribute specifies how long in LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 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 LPDDR2 SDRAM clock cycles the auxiliary output remains active for a read command.
CMD_OFFSET	Vector[5:0]	This attribute specifies how long in LPDDR2 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 several counters that are not enabled because the synchronous mode is used where PHY\_Clk is 1/2 the frequency of the LPDDR2 SDRAM clock frequency.

At every rising edge of PHY\_Clk, a PHY control word is sent to the PHY control block with information for two 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 field must be increased with every command sequence of four. The Seq field is used to synchronize PHY control blocks across multiple I/O banks.

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 two 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 two 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 set to zero. 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.

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 two memory clock cycles because each PHY\_Clk cycle entails two 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 CA[0] bit on falling edge of CK 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 CA[0] bit on falling edge of CK 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, and Deselect. The PHY command values in the PHY control word for all these commands are the same (**0x100**). The ca value inputs to the OUT\_FIFOs associated with these commands differ.

Figure 4-50 shows the block diagram of the address/control/command path. The OSERDES is used in single data rate mode because address/control/commands are DDR 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.

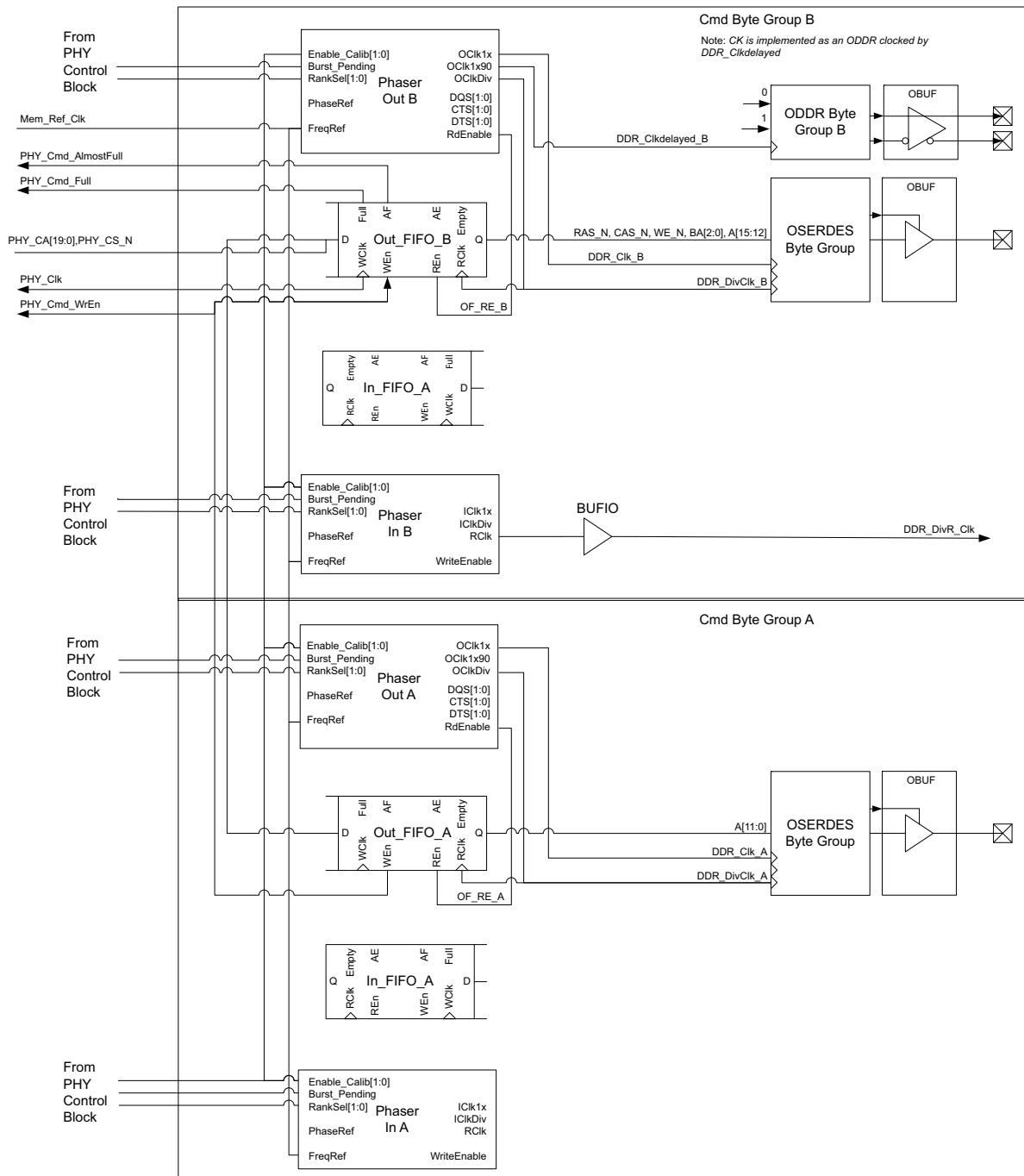


Figure 4-50:

The timing diagram of the address/command path from the output of the OUT\_FIFO to the FPGA pins is shown in [Figure 4-51](#).

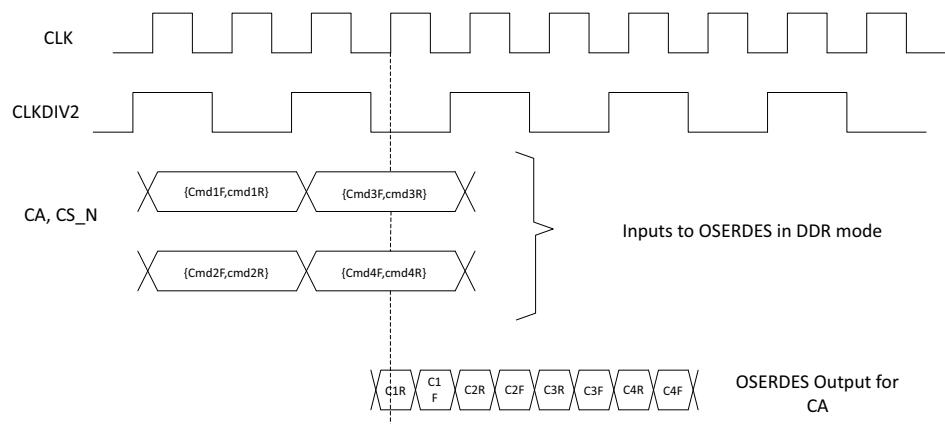


Figure 4-51:

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 1/2 the frequency of the LPDDR2 SDRAM clock. [Figure 4-52](#) shows the block diagram of the datapath.

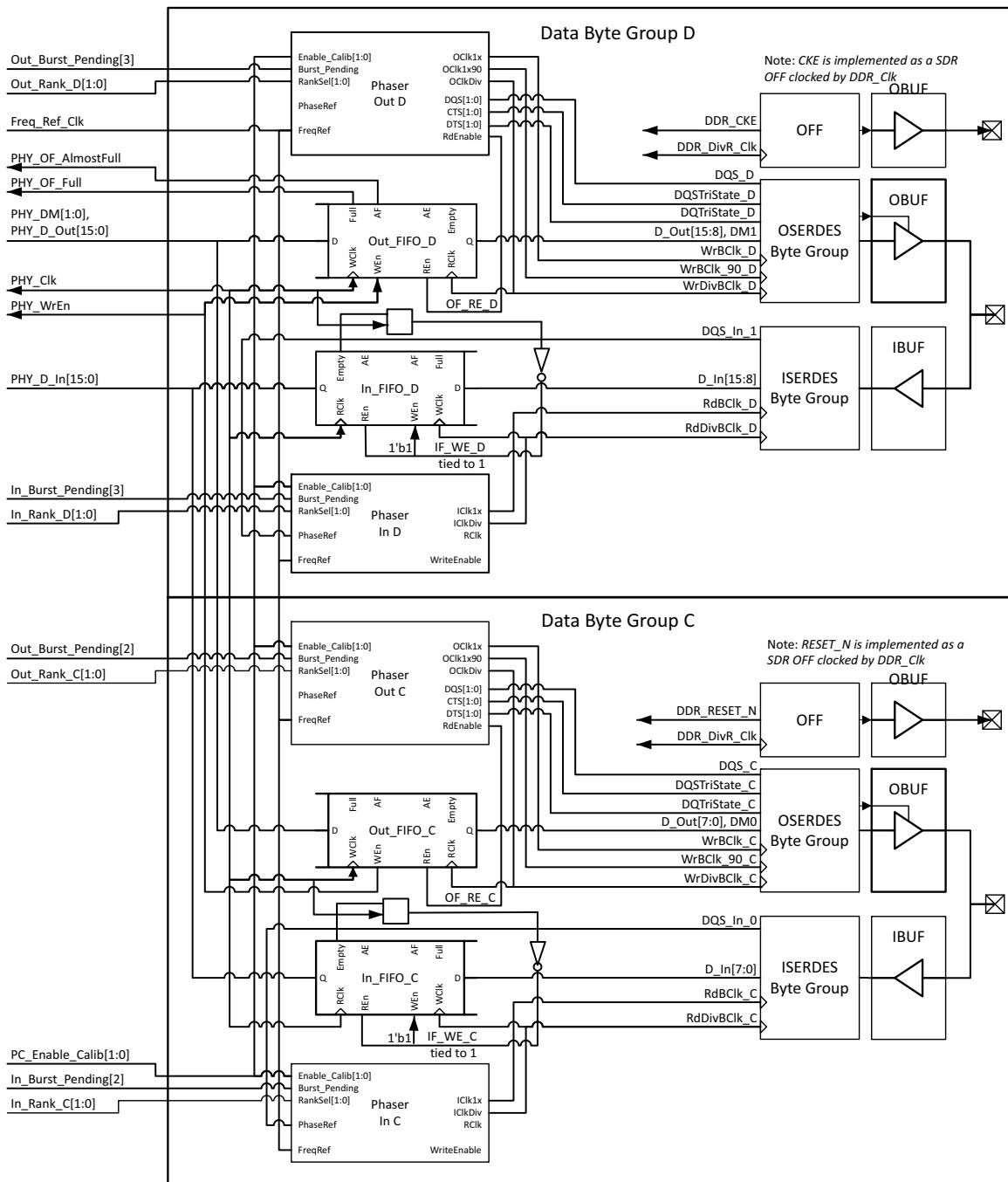


Figure 4-52:

Each IN/OUT\_FIFO has a storage array of memory elements arranged as 10 groups eight bits wide and eight entries deep. During a write, the OUT\_FIFO receives four 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 1/2 the frequency of the LPDDR2 SDRAM clock.

The OUT\_FIFO outputs the 4-bit data to the OSERDES in the OCLKDIV domain that is half the frequency of the LPDDR2 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° phase offset between DQS and its associated DQ bits during writes.

The IN\_FIFO shown in [Figure 4-51](#) receives 4-bit data from each DQ bit ISERDES in a given byte group and writes them into the storage array. This 4-bit parallel data is output in the PHY\_CLK clock domain which is 1/2 the frequency of the LPDDR2 SDRAM clock. Each read cycle from the IN\_FIFO contains half the byte data read during a burst length eight memory read transaction. Therefore, two cycles are required to get burst length eight worth of data. The data bus width input to the dedicated PHY is 4x that of the LPDDR2 SDRAM when running the FPGA logic at 1/2 the frequency of the LPDDR2 SDRAM clock.

### ***Calibration and Initialization Stages***

The PHY executes a JEDEC®-compliant LPDDR2 initialization sequence for memory following deassertion of system reset. Each LPDDR2 SDRAM has a series of mode registers, accessed through mode register write (MRW) commands. These mode registers determine various SDRAM behaviors, such as burst length, read and write CAS latency, and others. 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), write recovery for auto precharge (tWR).

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 LPDDR2 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. 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 LPDDR2 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 LPDDR2 SDRAM address location with a predefined data pattern. The write data pattern used is static pattern of FF, OO, FF, OO, FF, OO, FF, OO. 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 PHASER\_IN stg2 fine delay taps for all DQ bits in a byte simultaneously until an edge is detected. The STG2 fine tap are increased until a edge is found. If no edge is found then the STG2 fine taps are decreased to their initial positions.

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 DQ using the IDELAY taps 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 IDELAY tap values 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 IDELAY 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.)

First, the PHASER\_IN fine delay lines are used to find the start of right noise region. In the first two cases, right noise region would be found with PHASER\_IN fine tap increments less than 64 taps. After the right noise region is found with FINE taps, proceed to use IDELAY taps to find the start of left noise region by delaying the DATA.

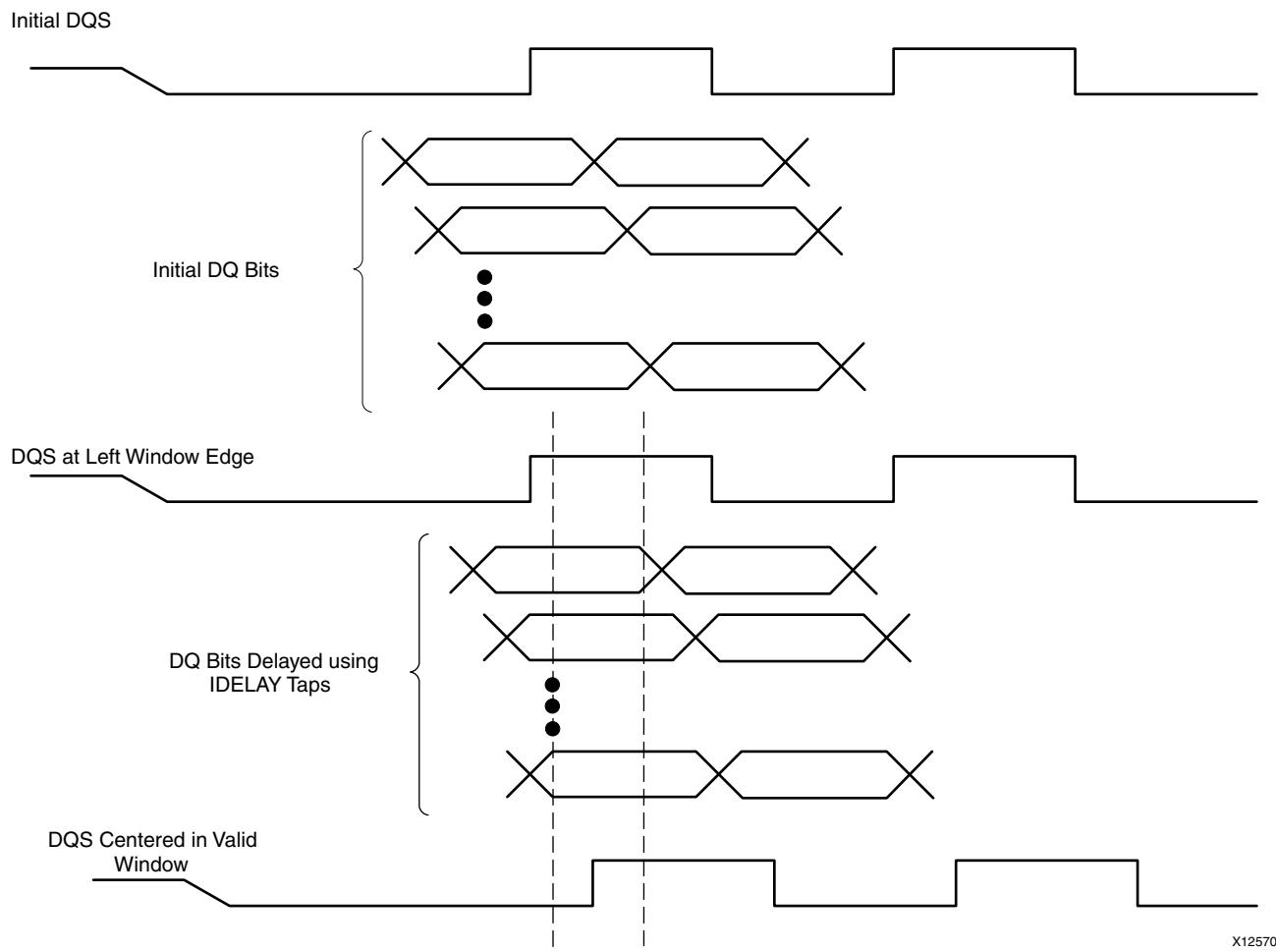
In the third scenario, because operating in frequencies < 400 MHz and PHASER\_IN is operating in DIV2 mode, you would not be able to find the right noise region with 64 taps. Thus, assume that you are close to left noise region and bring back the PHASER\_IN fine taps values to their initial position. After the PHASER\_IN fine taps increments/decrements, use IDELAY taps to delay the DQ to find both the edges (third case). 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, the final DQS tap value is computed as:

$$(\text{first\_edge\_taps} + (31 - \text{first\_edge\_taps})/2)$$

[Figure 4-53](#) shows the timing diagram for DQS center alignment in the data valid window.



*Figure 4-53:*

During read valid calibration, PHY generates delay count to align the internal Read Valid with the correct read data. In LPDDR2, WREN to IN\_FIFO is tied to 1, which means it is always writing to the IN\_FIFO. The RDEN is tied to inverted registered version of IF\_EMPTY, which means it is always reading once and there is a single entry in IN\_FIFO. This soft calibration is preformed to generate RD\_VALID to validate the correct read data from the SDRAM.

During this stage, PHY perform two sets of three back-to-back writes to a particular bank, row, and incremental column address. The write data pattern written for the first set of writes is FF, FF, FF, FF, FF, FF, F F and write data pattern written for the second set of writes is 55, 55, 55, 55, 55, 55, 55, 55. The data used to perform pattern matching is 55, 55, 55, 55, 55, 55, 55, 55.

During reads, PHY performs three back-to-back reads, with the first and third address used are the second column address for the first set of writes. The second address used is the second column address for the second set of writes performed earlier to SDRAM.

The Read Valid state machine waits for a read to occur and then checks if the pattern is matched. In doing so it also counts the number of cycles used to see the pattern match (correct data of 55, 55, 55, 55, 55, 55, 55, 55). If the state machine does not see a pattern match for delay count of 31, it increments the bitslip count, waits and issues another set of three reads. This process is repeated until either the correct pattern is found and read delay count is finalized or the bitslip count  $\leq 3$ . If the bitslip count exceeds three, it is considered a calibration failure.

The bitslip count is fed to select inputs of a MUX which shifts the read data captured from the output of ISERDESE2 and aligns it to match the entire rise and fall data in the same cycle.

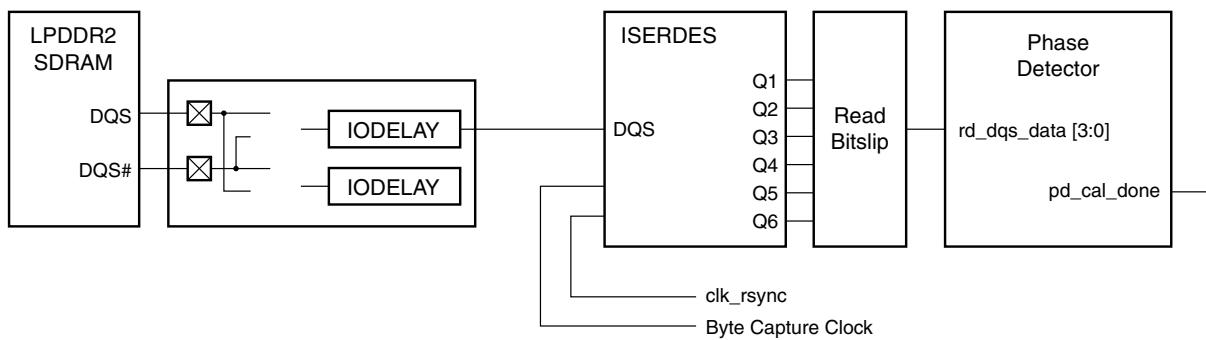
This stage of read calibration follows the Read Leveling calibration stage. The IDELAY tap setting determined during the Read Leveling calibration stage is used as the starting point for this stage of calibration. The PRBS read leveling stage does not change the PHASER\_IN fine tap settings determined during the Read Leveling calibration stage.

A 64-bit LFSR generates a 128 long PRBS sequence that is written to the LPDDR2 SDRAM at the start of this calibration stage. This sequence is then read back continuously to determine the read data valid window. 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 36'hFFFFFF. The algorithm starts at the IDELAY tap setting determined during the Read Leveling calibration stage (initial tap value) and decrements one tap at time until a data mismatch is found when comparing read data with the expected data.

Note that the expected data is generated using the same 64-bit LFSR logic that was used to write the 128 long PRBS sequence to the SDRAM. The data mismatch tap value is recorded as the left edge.

The algorithm then increments to the initial tap value and edge detection begins with every increment after the initial tap value until a data mismatch is found or the tap value is 31. The algorithm then computes the center of the read data valid window based on the detected edges.

In the 7 series FPGA memory interface design, read DQ is not sampled by the corresponding DQS signal. Instead, read DQ is sampled by a free-running clock operating at the same frequency as the differential SDRAM CK/CK# signals. The free-running clock has a single source for all DQ bits, but the phase of each byte capture clock output can be separately adjusted using IODELAY elements. The phase detector initially locks the phase of each byte-capture clock such that it is in phase with the corresponding DQS signal (Figure 4-54).



UG406\_c1\_51\_022610

*Figure 4-54:*

Subsequent changes in capture timing delays after initial calibration due to voltage and temperature changes can be compensated for by maintaining the phase relationship between the byte-capture clock and the corresponding DQS. Periodic dummy reads are required from the Memory Controller to dynamically maintain phase lock between the byte-capture clock and DQS (Figure 4-55).

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

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

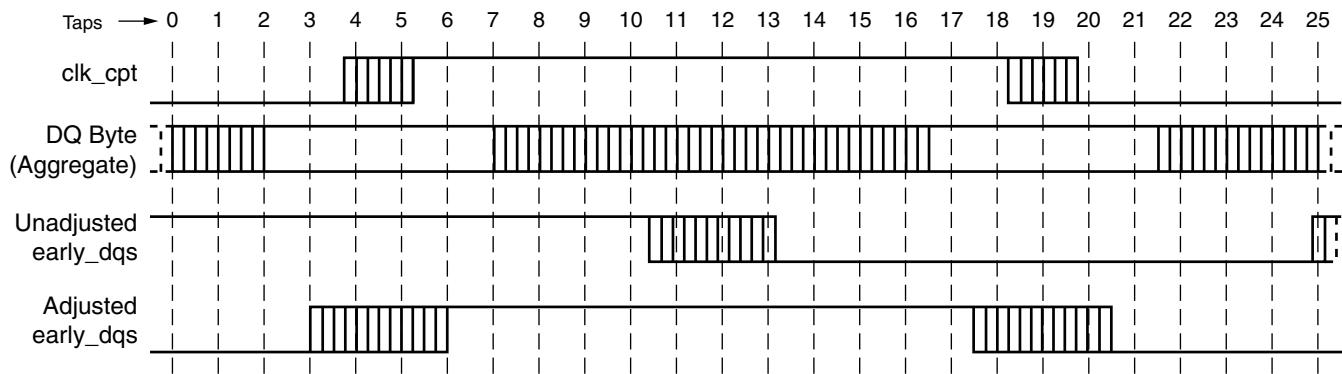


Figure 4-55:

### 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 4-24](#) describes the Memory Controller to PHY interface signals. These signals are synchronous to the FPGA logic clock.

Table 4-24:

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 LPDDR2 clock.
mem_refclk	1	Input	-	This is the LPDDR2 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_ca	[CA_WIDTH × 2 × nCK_PER_CLK – 1:0]	Input	-	mc_ca[2 × CA_WIDTH – 1:0] is the first command address in the sequence of four.

Table 4-24:

(Cont'd)

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_cke	[nCK_PER_CLK – 1:0]	Input	-	mc_cke [nCK_PER_CLK – 1:0] is the CKE associated with the DRAM interface. This signal is valid when the CKE_ODT_AUX parameter is set to FALSE.
mc_wrdata	[2 × nCK_PER_CLK × DQ_WIDTH – 1:0]	Input	-	This is the write data to the dedicated PHY. It is 4x the memory DQ width.
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 4x the memory DM width.
mc_wrdata_en	1	Input	Active-High	This signal is the WREN input to the DQ OUT_FIFO.
mc_cmd_wren	1	Input	Active-High	This signal is the write enable input of the address/command OUT_FIFOs.
mc_ctl_wren	1	Input	Active-High	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: Ox04: Non-data command (No column command in the sequence of commands) Ox01: Write command Ox03: Read command
mc_data_offset	[5:0]	Input	-	This signal is used for PHY_Ctl_Wd configuration: Ox00: Non-data command (No column command in the sequence of commands) CWL + COL cmd position: Write command Ox00: 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.

Table 4-24:

(Cont'd)

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	[ $2 \times \text{nCK\_PER\_CLK} \times \text{DQ\_WIDTH} - 1:0$ ]	Output	-	This is the read data from the dedicated PHY. It is 4x the memory DQ width.
phy_rddata_valid	1	Output	Active-High	This signal is asserted when valid read data is available.
calib_rd_data_offset	[ $6 \times \text{RANKS} - 1:0$ ]	Output	-	This signal is the calibrated read data offset value with respect to command 0 in the sequence of four commands.
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 LPDDR2 SDRAM clock cycles per PHY\_Clk cycle.
2. The parameter ROW\_WIDTH is the number of LPDDR2 SDRAM ranks.
3. The parameter BANK\_WIDTH is the number of LPDDR2 SDRAM banks.
4. The parameter CS\_WIDTH is the number of LPDDR2 SDRAM cs\_n signals.
5. The parameter CKE\_WIDTH is the number of LPDDR2 SDRAM CKE signals.
6. The parameter DQ\_WIDTH is the width of the LPDDR2 SDRAM DQ bus.

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 621](#) for supported configuration parameters.

The Memory Controller can be connected using either the UI or the native interface. 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 reo

[Figure 4-56](#) and [Figure 4-57](#) show that the address map is controlled by the string parameter MEM ADDR ORDER. This parameter can take the following values:

- **BANK\_ROW\_COLUMN** – Address map is as shown in [Figure 4-56](#).
  - **ROW\_BANK\_COLUMN** – Address map is as shown in [Figure 4-57](#).
  - **TG\_TEST** – Address map is used for testing purpose only. It enables the address remap to test address access to different portions of the DRAM. It remaps the address as explained in the following examples. The remap is done within the UI portion of the controller.

**Note:** The row width, column width, and bank width value settings are assumed for the following examples:

- **Row Width** – 15
  - **Bank Width** – 3
  - **Column Width** – 10

**Example (1)** - When the selected option in the MIG GUI is BANK\_ROW\_COLUMN and the address to the controller is mapped accordingly.

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
B2	B1	BO	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	RO	C9	C8	C7	C6	C5	C4	C3	C2	C1	CO
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
RO	C9	C8	R4	R3	B2	B1	BO	R14	R13	R12	R11	R10	R9	R8	C7	C6	C5	R2	R1	R7	R6	R5	C4	C3	C2	C1	CO

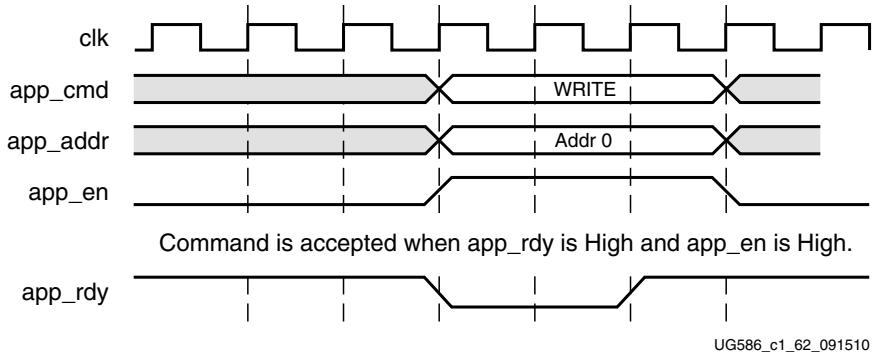
**Example (2)** – When the selected option in the MIG GUI is ROW\_BANK\_COLUMN and the address to the controller is mapped accordingly.

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
R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	RO	B2	B1	B0	C9	C8	C7	C6	C5	C4	C3	C2	C1	CO

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
RO	C9	C8	R4	R3	B2	B1	B0	R14	R13	R12	R11	R10	R9	R8	C7	C6	C5	R2	R1	R7	R6	R5	C4	C3	C2	C1	CO

### 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 4-58](#).



[Figure 4-58:](#)

A non back-to-back write command can be issued as shown in [Figure 4-59](#). 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.

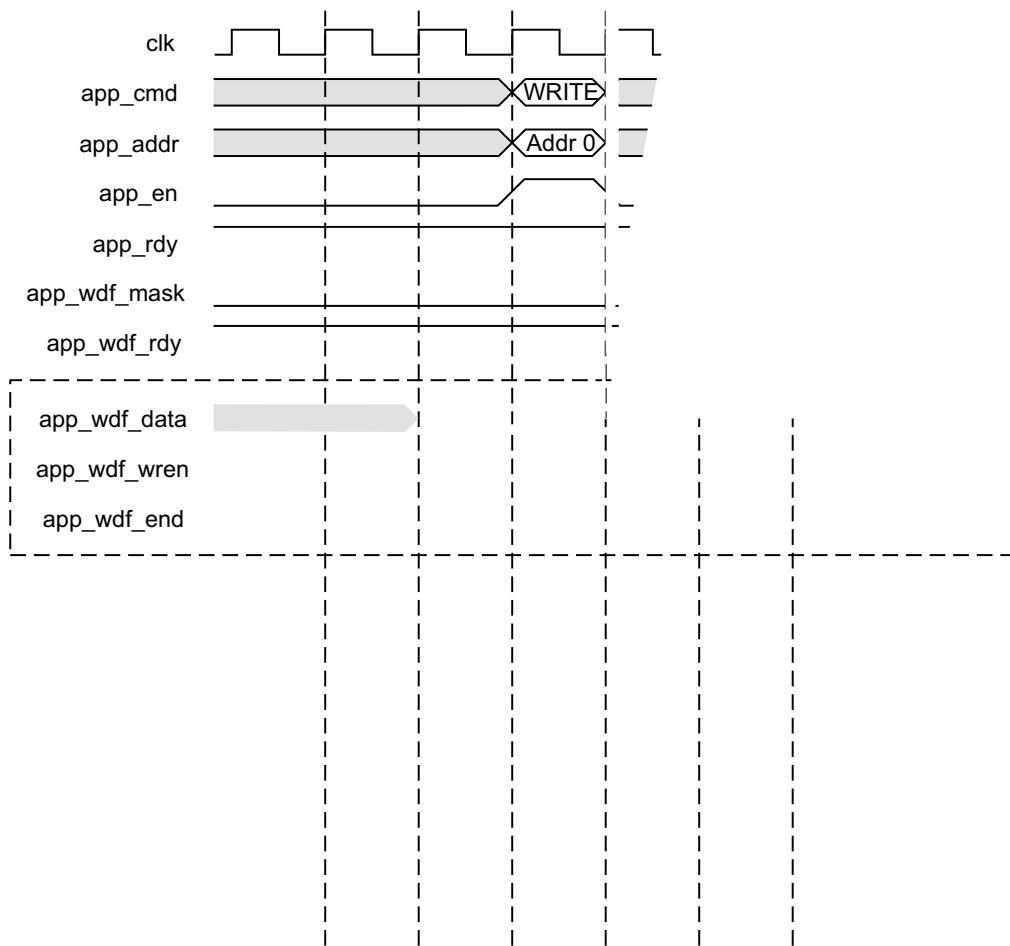


Figure 4-59:

## Write Path

The write data is registered in the write FIFO when **app\_wdf\_wren** is asserted and **app\_wdf\_rdy** is High (Figure 4-60). 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.

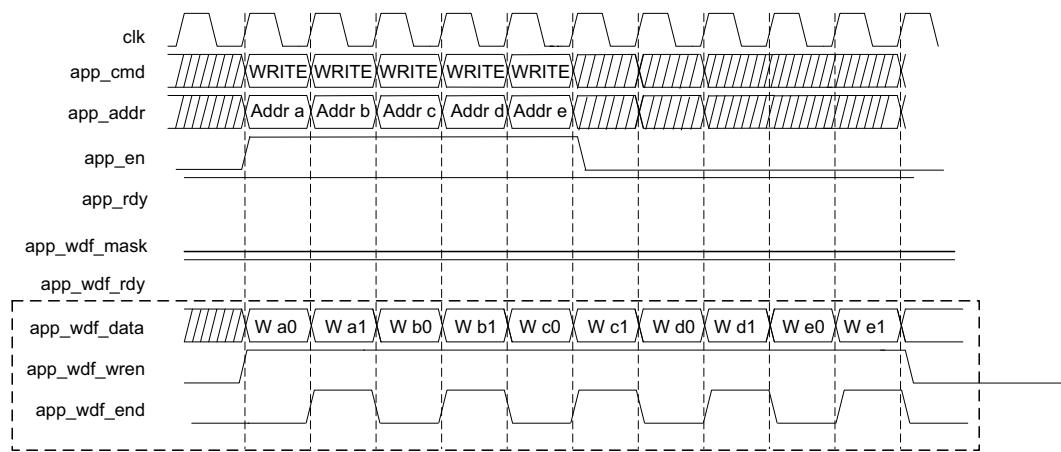


Figure 4-60:

As shown in Figure 4-59, 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 4-61.

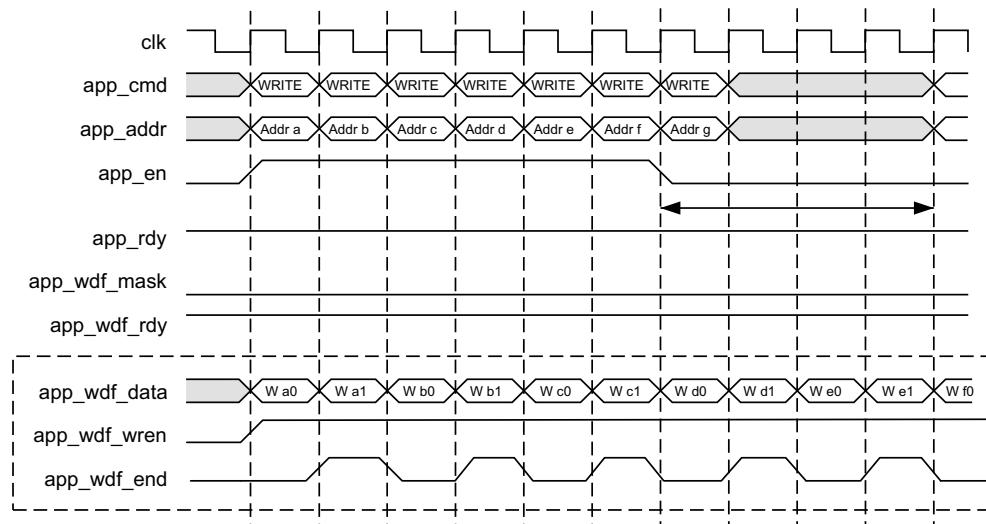
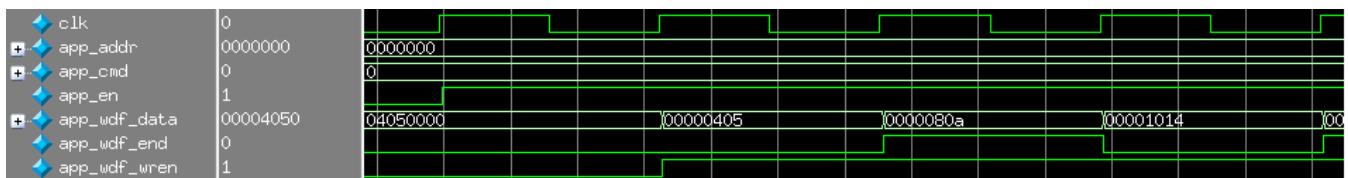


Figure 4-61:

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.

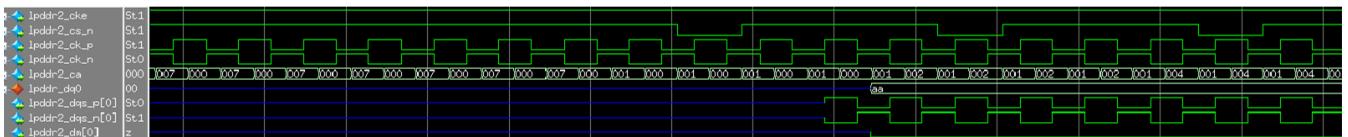
The map of the application interface data to the DRAM output data can be explained with an example.

For a 2:1 Memory Controller, the User Interface clock to DRAM clock ratio is 2:1. For an 8-bit wide memory interface, the User Interface data width is 32-bit wide ( $8 \times 2 \times 2 = \text{mem\_data\_width} \times \text{nck\_per\_clk} \times \text{ddr\_rate}$ ). To perform a BL8 transaction, write 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 4-62](#). 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 4-62:*

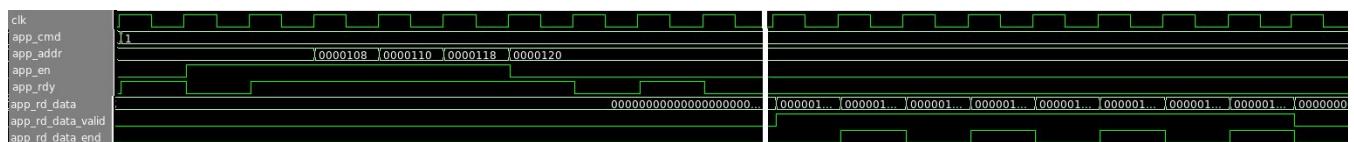
[Figure 4-63](#) shows the corresponding data at the DRAM interface.



*Figure 4-63:*

## 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 4-64](#) and [Figure 4-65](#)). The **app\_rd\_data\_end** signal indicates the end of each read command burst.



*Figure 4-64:*

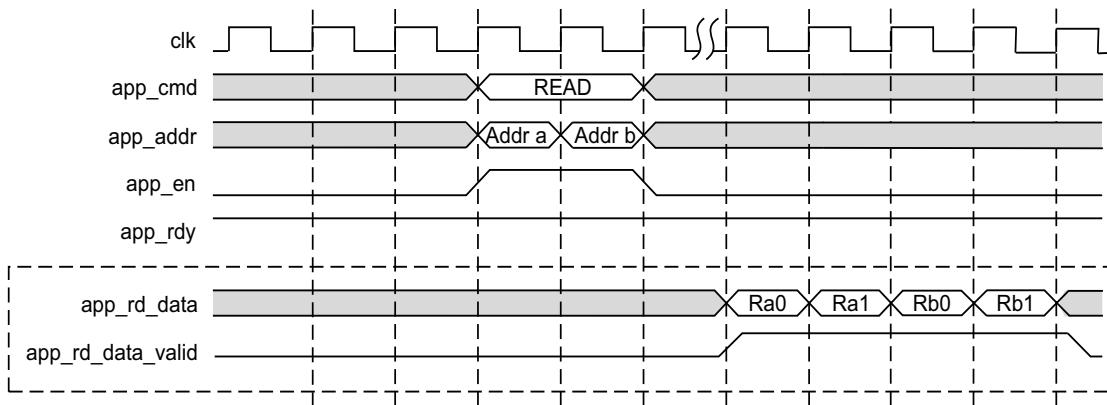


Figure 4-65:

In Figure 4-65, the read data returned is always in the same order as the requests made on the address/control bus.

### 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 strobos **app\_zq\_ack** for one cycle, after which another request can be sent. Figure 4-66 illustrates the interface.

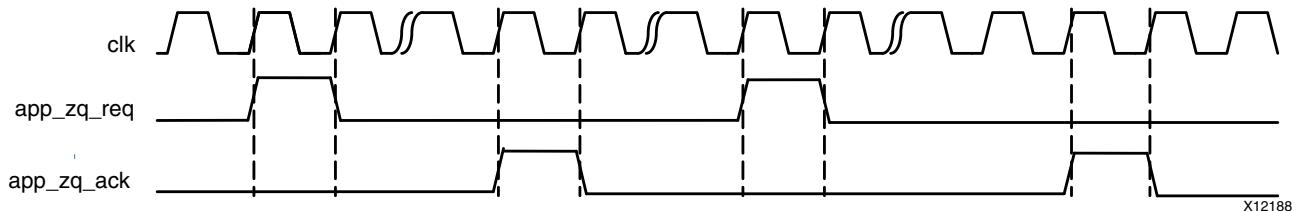


Figure 4-66:

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.

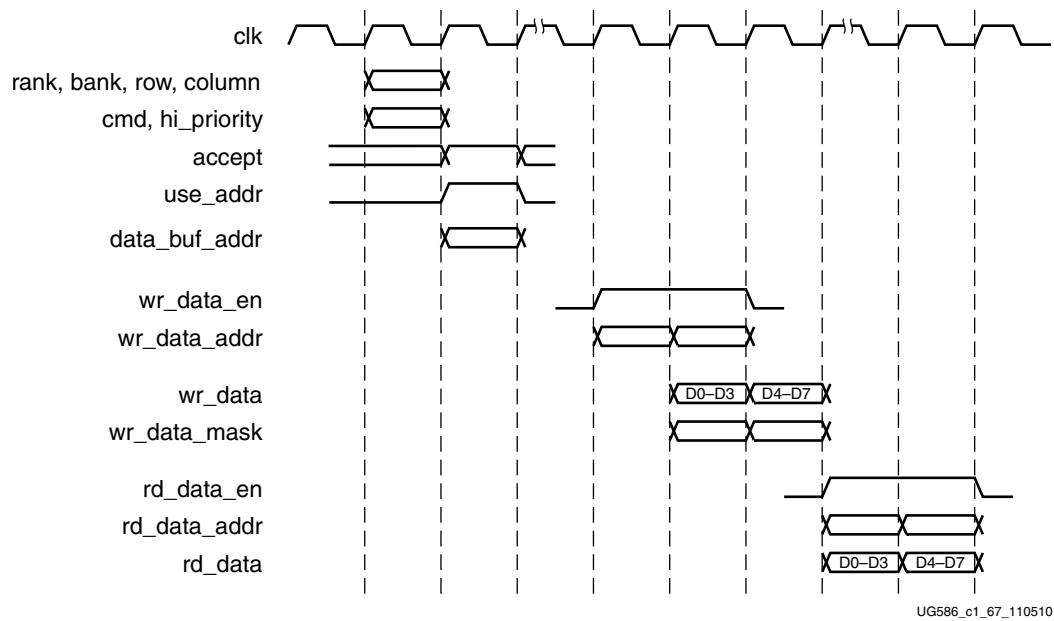
Timing parameters must be considered for each pending request when determining when to strobe **app\_zq\_req** to achieve the desired interval if precision timing is desired. To account for the worst case, subtract tRCD, CL, the data transit time and tRP for each bank machine to ensure that all transactions can complete before the target tZQI expires. Equation 4-1 shows the ZQ request interval maximum.

$$(tZQI - (tRCD + ((CL + 4) \times tCK) + tRP) \times nBANK\_MACHS)$$

Equation 4-1

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

The native interface protocol is shown in [Figure 4-67](#).



*Figure 4-67:*

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 4-68](#).

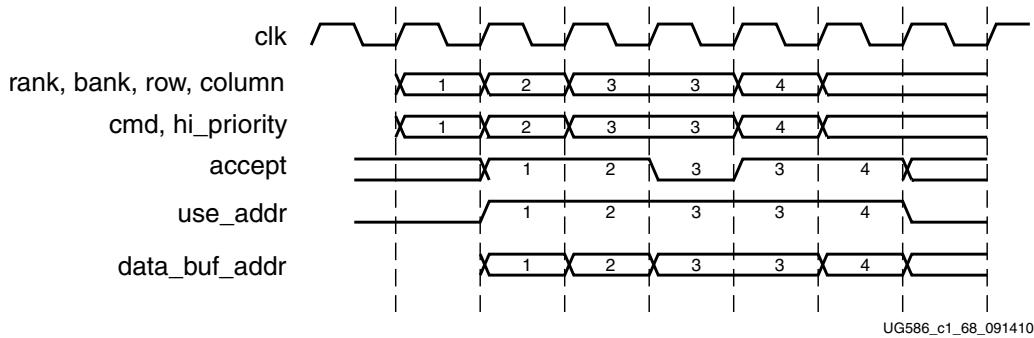


Figure 4-68:

In Figure 4-68, 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 through **wr\_data\_addr** for write commands and **rd\_data\_addr** for read commands. This behavior is shown in Figure 4-69. Write data must be supplied in the same clock cycle that **wr\_data\_en** is asserted.

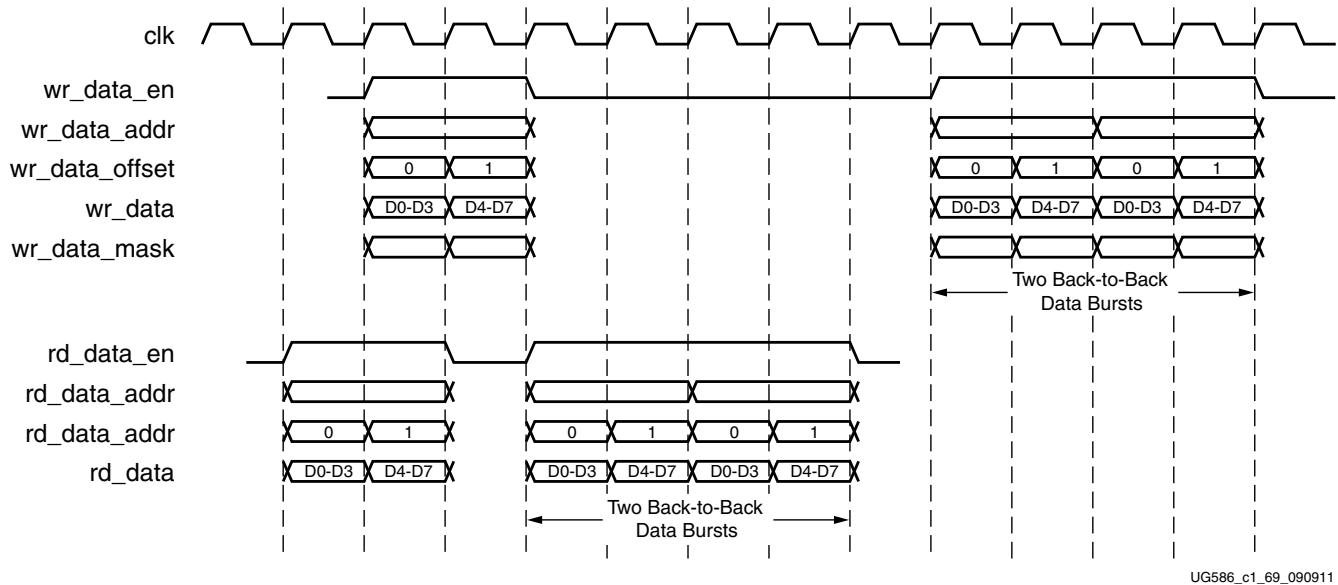


Figure 4-69:

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 4-69](#).

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 ZQ

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

---

The 7 series FPGAs memory interface solution supports several configurations for LPDDR2 SDRAM devices. The specific configuration is defined by Verilog parameters in the top-level of the core. As per the OOC flow, none of the parameter values are passed down to the user design RTL file from the example design top RTL file. So, any design related parameter change is not reflected in the user design logic. 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 4-25](#) to [Table 4-27](#).

Table 4-25:

REFCLK_FREQ <sup>(1)</sup>	This is the reference clock frequency for IDELAYCTRLs. This can be set to 200.0 for any speed grade device. For more information, see the IDELAYE2 (IDELAY) and ODELAYE2 (ODELAY) Attribute Summary table in the <i>7 Series FPGAs SelectIO™ Resources User Guide</i> [Ref 2]. This parameter should <b>not</b> be changed.	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" "FAST" "SIM_FULL"
nCK_PER_CLK	This is the number of memory clocks per clock. This parameter should <b>not</b> be changed.	2
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.
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.	This parameter value is 1 for both Single and Dual rank devices.
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 4-25:

(Cont'd)

DO_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.	DO_WIDTH/8
BURST_MODE	This is the memory data burst length.	LPDDR2: "8"
BM_CNT_WIDTH	This is the number of bits required to index a bank machine and is given by $\text{ceil}(\log_2(n\text{BANK\_MACHS}))$ .	
ADDR_CMD_MODE	This parameter is used by the controller to calculate timing on the memory addr/cmd bus. This parameter should <b>not</b> be changed.	"1T"
ORDERING <sup>(3)</sup>	This option reorders received requests to optimize data throughput and latency.	"NORM": Allows the Memory Controller to reorder read but not write commands to the memory. "RELAXED": Allows the Memory Controller to reorder commands to the memory for maximum efficiency. Strong ordering is not preserved at the native interface in this mode. "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
IODELAY_GRP <sup>(4)</sup>	This is an ASCII character string to define an IDELAY group used in a memory design. This is used by the Vivado 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"
PAYLOAD_WIDTH	This is the actual DQ bus used for user data.	PAYLOAD_WIDTH = DATA_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.

Table 4-25:

(Cont'd)

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. 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 or RELAXED, ORDERING enables the reordering algorithm in the Memory Controller. When set to STRICT, request reordering is disabled, which might limit 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.
4. This parameter is prefixed with the module name entered in MIG during design generation. If the design is generated with the module name as mig\_7series\_0, then IODELAY\_GRP parameter name is mig\_7series\_0\_IODELAY\_MIG.

The parameters listed in [Table 4-26](#) depend on the selected memory clock frequency, memory device, memory configuration, and FPGA speed grade. The values for these parameters are integrated in the `memc_ui_top` IP core and should not be modified in the top-level.



**RECOMMENDED:** Xilinx strongly recommends that the MIG tool be rerun for different configurations.

Table 4-26:

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.

(Cont'd)

Table 4-26:

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 ZQ short calibration interval. This value is system dependent and should be based on the expected rate of change of voltage and temperature in the system. Consult the memory vendor for more information on ZQ calibration.	This value is set in nanoseconds. Set to 0, if you manage 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.	LPDDR2: 3, 4, 5, 6
CWL	This is the write CAS latency. The available option is frequency dependent in the MIG tool.	
BURST_TYPE	This is an option for the ordering of accesses within a burst.	"Sequential" "Interleaved"
RST_ACT_LOW	Active-Low or active-High reset. This is set to 1 when System Reset Polarity option is selected as active-Low and set to 0 when the option is selected as active-High.	0, 1
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"
DATA_IO_IDLE_PWRDWN	This option is set to ON valid when I/O Power reduction option is enabled.	"ON," "OFF"

(Cont'd)

Table 4-26:

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 ZQ short calibration interval. This value is system dependent and should be based on the expected rate of change of voltage and temperature in the system. Consult the memory vendor for more information on ZQ calibration.	This value is set in nanoseconds. Set to 0, if you manage 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.	LPDDR2: 3, 4, 5, 6
CWL	This is the write CAS latency. The available option is frequency dependent in the MIG tool.	
BURST_TYPE	This is an option for the ordering of accesses within a burst.	"Sequential" "Interleaved"
RST_ACT_LOW	Active-Low or active-High reset. This is set to 1 when System Reset Polarity option is selected as active-Low and set to 0 when the option is selected as active-High.	0, 1
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"
DATA_IO_IDLE_PWRDWN	This option is set to ON valid when I/O Power reduction option is enabled.	"ON," "OFF"

**SYSCLK\_TYPE**

This parameter indicates whether the system uses single-ended system clocks, differential system clocks, or is driven from an internal clock (No Buffer). 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

(Cont'd)

Table 4-26:

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 4. This parameter should <b>not</b> be changed.	4
RANKS	This is the number of ranks.	
DATA_WIDTH	This parameter determines the write data mask width and depends on whether or not ECC is enabled.	DATA_WIDTH = DQ_WIDTH
APP_DATA_WIDTH	This UI_INTFC parameter specifies the payload data width in the UI.	APP_DATA_WIDTH = $2 \times nCK\_PER\_CLK \times PAYLOAD\_WIDTH$
APP_MASK_WIDTH	This UI_INTFC parameter specifies the payload mask width in the UI.	
USER_REFRESH	This parameter indicates if you manage refresh commands. Can be set for either the User or Native interface.	"ON," "OFF"

Table 4-27 contains parameters set up by the MIG tool based on the pinout selected. When making pinout changes, Xilinx recommends rerunning the MIG tool to set up the parameters properly. See [Bank and Pin Selection Guides for LPDDR2 Designs, page 632](#).

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 4-27:

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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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.

Table 4-27:

(Cont'd)

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. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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 as per bank. Except CK/CK# pins, all Data and Address/Control pins are considered for this parameter generation. DQS pins are excluded when used for DQS pins in data byte groups. One of the unused pins where Data byte group is allocated should be set to 1 which is used for DQSO_MAP (DQS still allocated to DQS I/O only and extra bit is used internally in the PHY). This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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_BYT_E_MAP	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> This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	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_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.

Table 4-27:

(Cont'd)

ADDR_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>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	<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>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 with in a byte group by excluding DQS I/Os. DQS_N and DQS_P pins of the byte group are numbered as A and B, respectively.</p> <p>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. One important change w.r.t DDR3/DDR2 designs is that LPDDR2 have only 10 bits available for Address mapping in byte lane. The reason for that is in LPDDR2, the CA is DDR.</p> <p>12'h235: Address pin placed in bank 2, byte lane 3, at location 5.</p>
CS_MAP	<p>Bank and byte lane position information for the chip select. See the <a href="#">ADDR_MAP</a> description.</p> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	See the <a href="#">ADDR_MAP</a> example.
DQS_BYTE_MAP	<p>Bank and byte lane position information for the strobe. See the <a href="#">CK_BYTE_MAP</a> description.</p> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	See the <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	<p>Bank and byte lane position information for the data bus. See the <a href="#">ADDR_MAP</a> description.</p> <p>This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.</p>	See the <a href="#">ADDR_MAP</a> example.

(Cont'd)

Table 4-27:

MASKO_MAP	Bank and byte lane position information for the data mask. See the <a href="#">ADDR_MAP</a> description. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">ADDR_MAP</a> example.
DQSO_MAP	Bank and byte lane position information for the DQS of respective data lanes. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">DQS_MAP</a> example.
ADDR_O_BITLANES, ADDR_1_BITLANES, ADDR_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 where Address/Control are selected. Except CK/CK# and Data pins, only the Address/Control pins are considered for this parameter generation. DQS pins are excluded when used for CK/CK# in command/address byte group. This parameter varies based on the pinout and should <b>not</b> be changed manually in generated design.	See the <a href="#">PHY_O_BIT_LANES</a> example.

Guidelines for LPDDR2 SDRAM designs are covered in this section.

For general PCB routing guidelines, see [Appendix A, General Memory Routing Guidelines](#).

This section describes guidelines for LPDDR2 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 LPDDR2 Designs***

Xilinx 7 series FPGAs are designed for very high-performance memory interfaces, and certain rules must be followed to use the LPDDR2 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 LPDDR2 configuration, 8 of these 10 I/Os are used for the **DQS**: one is used for the data mask (DM), and the remaining one is used for **DQS** sampling. However, there would not be any physical connect on this pin because it would be internally used to capture the **DQS** for the phase detector.

Xilinx 7 series FPGAs have dedicated clock routing for high-speed synchronization that is routed vertically within the I/O banks. Thus, LPDDR2 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 LPDDR2 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 (**CA**, **CS\_N**, **CKE**) and address lines must be connected to byte groups not used for the data byte groups.
- 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 **DQS** pair in one of the control byte groups. 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. **CK** pairs are generated for each component, and a maximum of four pairs only are allowed due to I/O pin limitations. This varies based on **Memory Clock Selection** in the **Memory Options** page in the MIG GUI. Except **CK/CK#**, any of the Address/Control pin should not be allocated to **DQS**.
- **CS\_N** pins are generated for each component and a maximum of four ports/pairs only are allowed due to I/O pin limitations.
- Only one **CKE** port is generated.
- **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



---

**RECOMMENDED:** Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.

---

- 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.

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.

- 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 clock-capable **DQS** 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 V<sub>REF</sub>*

Internal V<sub>REF</sub> can only be used for data rates of 800 Mb/s or below.

### *System Clock, MMCM Location, and Constraints*

The MMCM 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 MMCM. The system clock input to the MMCM 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 MMCM can be used as clock outputs. Only the settings for these outputs can be changed. Settings related to the overall MMCM behavior and the used outputs must not be disturbed.

A MMCM cannot be shared among interfaces.

See [Clocking Architecture, page 585](#) for information on allowed MMCM parameters.

## Configuration

The XDC 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:

```
create_clock -period 1.875 [get_ports sys_clk_p]
```

The **clk\_ref** constraint sets the frequency for the IDELAY reference clock, which is typically 200 MHz. For example:

```
create_clock -period 5 [get_ports clk_ref_p]
```

The I/O standards are set appropriately for the LPDDR2 interface with HSUL\_12, as appropriate. If system clock (**sys\_clk**<sup>\*</sup>) and I/O delay reference clock (**clk\_ref**<sup>\*</sup>) are allocated in LPDDR2 memory interface allocated bank, then the I/O Standards would need to be DIFF\_HSLU\_12 or HSUL\_12 depending on whether these clocks are differential or single-ended. If these clocks are placed outside the LPDDR2 interface banks, then the I/O Standards are LVDS\_25 or LVCMS25 (depending on whether these clocks are differential or single-ended). 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 (asynchronous).
- **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 traffic generator, if read data does not match the write data.

These signals are all set to LVCMS25 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 XDC. An example is shown here:

```
set_property CLOCK_DEDICATED_ROUTE_BACKBONE [get_nets sys_clk_p]  
set_property CLOCK_DEDICATED_ROUTE_BACKBONE [get_pins -hierarchical *pll*CLKIN1]
```

This results in a warning listed 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 I/O 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 might want to analyze why this issue 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 might lead to very poor timing results. It is recommended that this error condition be corrected in the design.

Do not drive user clocks through 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. For more information, see the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10].

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

```
# PadFunction: IO_L13P_T2_MRCC_34
set_property VCCAUX_IO NORMAL [get_ports {lpddr2_dq[0]}]
set_property SLEW FAST [get_ports {lpddr2_dq[0]}]
set_property IOSTANDARD HSUL_12 [get_ports {lpddr2_dq[0]}]
set_property PACKAGE_PIN AJ11 [get_ports {lpddr2_dq[0]}]

# PadFunction: IO_L13N_T2_MRCC_34
set_property VCCAUX_IO NORMAL [get_ports {lpddr2_dq[1]}]
set_property SLEW FAST [get_ports {lpddr2_dq[1]}]
set_property IOSTANDARD HSUL_12 [get_ports {lpddr2_dq[1]}]
set_property PACKAGE_PIN AK11 [get_ports {lpddr2_dq[1]}]
```

For more information, see the *Xilinx Timing Constraints User Guide* (UG612) [Ref 15].

For LPDDR2 SDRAM interfaces that have the memory system input clock (*sys\_clk\_p*/*sys\_clk\_n*) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_HSUL\_12 I/O standard ( $V_{CCO} = 1.2V$ ) to the CCIO pins.

## Termination

These rules apply to termination for LPDDR2 SDRAM:

- Simulation (using IBIS or other) is highly recommended. The loading of command address and control (**cs\_n**) signals depends on various factors, such as speed requirements, and termination topology. Loading can be a limiting factor in reaching a performance target.
- If termination is used, unidirectional signals should be terminated with a resistor to  $V_{TT}$  at the load. A split termination to  $V_{CCO}$  and GND can be used, but takes more power. Bidirectional signals might need termination at both ends of the signal.
- If termination is used, differential signals should be terminated with a differential termination at the load. Bidirectional signals might need termination at both ends of the signal.

- If 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.
- The **CKE** signal should be pulled down during memory initialization with a  $4.7\text{ k}\Omega$  resistor connected to GND.
- **DM** should be pulled to GND if **DM** is not driven by the FPGA (for scenarios where the data mask is not used or is disabled).
- LPDDR2 does not have a GUI option to configure the SDRAM output drive strength like DDR2/DDR3. For LPDDR2, use the default setting of  $40\Omega$  for output drive strength.

## I/O Standards

These rules apply to the I/O standard selection for LPDDR2 SDRAMs:

- Designs generated by the MIG tool use the HSUL\_12 and DIFF\_HSUL\_12 standards for all bidirectional I/O (**DQ**, **DQS**) in the High-Performance banks.
- The HSUL\_12 and DIFF\_HSUL\_12 standards are used for unidirectional outputs, such as control/address and forward memory clocks.

## 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 generate the package lengths using Vivado Design Suite. The following commands generate a **csv** file that contains the package delay values for every pin of the device under consideration.

```
link_design -part <part_number>
write_csv <file_name>
```

For example, to obtain the package delay information for the 7 series FPGA XC7K160T-FF676, this command should be issued:

```
link_design -part xc7k160tfg676
write_csv flight_time
```

This generates a file named **flight\_time.csv** in the current directory with package trace delay information for each pin. While applying specific trace-matching guidelines for the LPDDR2 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 LPDDR2 SDRAM signals:

- The maximum electrical delay between any **DQ** or **DM** and its associated **DQS/DQS#** must be  $\leq \pm 15$  ps.
- The maximum electrical delay between any address and control signals and the corresponding **CK/CK#** must be  $\leq \pm 25$  ps, with 15 ps being the optimum target.
- The maximum electrical delay between any **DQS/DQS#** and **CK/CK#** must be  $< \pm 25$  ps.

The 7 series FPGA MIG LPDDR2 SDRAM design has two clock inputs, the reference clock and the system clock. The reference clock drives the IODELAYCTRL components in the design, while the system clock input is used to create all MIG design clocks that are used to clock 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 more information on clocking architecture, see [Clocking Architecture, page 585](#).

The MIG tool allows you to input the Memory Clock Period and then lists available Input Clock Periods that follow the supported clocking guidelines. Based on these two clock periods selections, the generated MIG core appropriately sets the MMCM parameters. The MIG tool enables automatic generation of all supported clocking structures. For information on how to use the MIG tool to set up the desired clocking structure including input clock placement, input clock frequency, and IDELAYCTRL **ref\_clk** generation, see [Creating 7 Series FPGA LPDDR2 SDRAM Memory Controller Block Design, page 529](#).

## ***Input Clock Guidelines***



---

**IMPORTANT:** *The input system clock cannot be generated internally.*

---

- MMCM Guidelines
  - CLKFBOUT\_MULT\_F (M) must be between 1 and 16 inclusive.
  - DIVCLK\_DIVIDE (D, Input Divider) can be any value supported by the MMCME2 parameter.
  - CLKOUT\_DIVIDE (O, Output Divider) must be 2 for 400 MHz and up operation and 4 for below 400 MHz operation.

- The above settings must ensure the minimum MMCM VCO frequency (FVCOMIN) is met. For specifications, see the appropriate DC and Switching Characteristics Data Sheet. The *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 10] includes the equation for calculating FVCO.
- The relationship between the input period and the memory period is  $\text{InputPeriod} = (\text{MemoryPeriod} \times M) / (D \times D1)$ .
- The clock input (**sys\_clk**) can be input on any CCIO in the column where the memory interface is located; this includes CCIO in banks that do not contain the memory interface, but must be in the same column as the memory interface. The MMCM must be located in the bank containing the clock sent to the memory. To route the input clock to the memory interface MMCM, the CMT backbone must be used. With the MIG implementation, one spare interconnect on the backbone is available that can be used for this purpose.
  - MIG versions 1.4 and later allow this input clocking setup and properly drive the CMT backbone.
  - **CLOCK\_DEDICATED\_ROUTE = BACKBONE** constraint is used to implement CMT backbone, following warning message is expected. It can be ignored safely.

**WARNING:** [Place 30-172] Sub-optimal placement for a clock-capable IO pin and PLL pair. The flow will continue as the **CLOCK\_DEDICATED\_ROUTE** constraint is set to **BACKBONE**.

```
u_mig_7series_0/c0_u_clk_ibuf/diff_input_clk.u_ibufg_sys_clk (IBUFDS.O) is locked
to IOB_X0Y176
u_mig_7series_0/c0_u_infrastructure/mmcm2_i (MMCME2_ADV.CLKIN1) is locked to
MMCME2_ADV_X0Y1
u_mig_7series_0/c1_u_infrastructure/mmcm2_i (MMCME2_ADV.CLKIN1) is locked to
MMCME2_ADV_X0Y5
....
```

- For LPDDR2 SDRAM interfaces that have the memory system input clock (**sys\_clk**) placed on CCIO pins within one of the memory banks, MIG assigns the DIFF\_HSUL\_12 I/O standard (VCCO = 1.2V) to the CCIO pins. Because the same differential input receiver is used for both DIFF\_HSUL\_12 and LVDS inputs, an LVDS clock source can be connected directly to the DIFF\_HSTL\_I CCIO pins.
- It is acceptable to have differential inputs such as LVDS and LVDS\_25 in I/O banks that are powered at voltage levels other than the nominal voltages required for the outputs of those standards (1.8V for LVDS outputs, and 2.5V for LVDS\_25 outputs). However, these criteria must be met:
  - a. The optional internal differential termination is not used (DIFF\_TERM = FALSE, which is the default value).
 

**Note:** This might require manually changing DIFF\_TERM parameter located in the top-level module or setting this in the UCF or XDC.
  - b. The differential signals at the input pins meet the VIN requirements in the Recommended Operating Conditions table of the specific device family data sheet.

- c. The differential signals at the input pins meet the VIDIFF (min) requirements in the corresponding LVDS or LVDS\_25 DC specifications tables of the specific device family data sheet.

One way to accomplish the above criteria is to use an external circuit that both AC-couples and DC-biases the input signals. The figure shows an example circuit for providing an AC-coupled and DC-biased circuit for a differential clock input. RDIFF provides the  $100\Omega$  differential receiver termination because the internal DIFF\_TERM is set to FALSE. To maximize the input noise margin, all RBIAS resistors should be the same value, essentially creating a VCM level of  $VCCO/2$ . Resistors in the 10k to  $100\text{ k}\Omega$  range are recommended. The typical values for the AC coupling capacitors CAC are in the range of  $100\text{ nF}$ . All components should be placed physically close to the FPGA inputs.

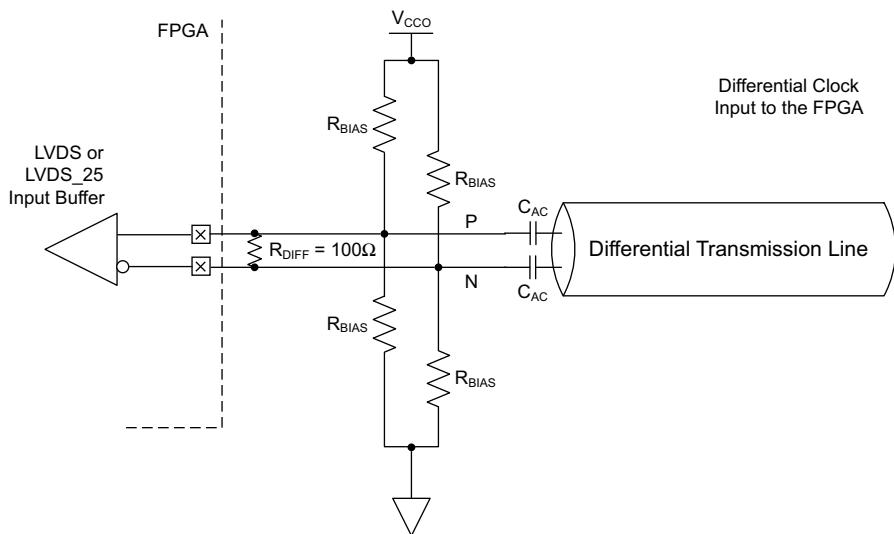


Figure 4-70:

**Note:** The last set of guidelines on differential LVDS inputs are added within the LVDS and LVDS\_25 (Low Voltage Differential Signaling) section of the *7 Series SelectIO Resources User Guide* (UG471) [Ref 2] in the next release of the document.

These guidelines are irrespective of Package, Column (HR/HP), or I/O Voltage.

### Sharing sys\_clk between Controllers

MIG 7 series FPGA designs require `sys_clk` to be in the same I/O bank column as the memory interface to minimize jitter.

- **Interfaces Spanning I/O Columns** – A single `sys_clk` input cannot drive memory interfaces spanning multiple I/O columns. The input clock input must be in the same column as the memory interface to drive the MMCM using the CMT Backbone, which minimizes jitter.

- **Interfaces in Single I/O Column** – If the memory interfaces are entirely contained within the same I/O column, a common **sys\_clk** can be shared among the interfaces. The **sys\_clk** can be input on any CCIO in the column where the memory interfaces are located. This includes CCIO in banks that do not contain the memory interfaces, but must be in the same column as the memory interfaces.

### ***Information on Sharing BUFG Clock (phy\_clk)***

The MIG 7 series LPDDR2 SDRAM design includes a PLL which outputs the **phy\_clk** on a BUFG route. It is not possible to share this clock amongst multiple controllers to synchronize the user interfaces. This is not allowed because the timing from the FPGA logic to the PHY Control block must be controlled. This is not possible when the clock is shared amongst multiple controllers. The only option for synchronizing user interfaces amongst multiple controllers is to create an asynchronous FIFO for clock domain transfer.

### ***Information on Sync\_Pulse***

The MIG 7 series LPDDR2 SDRAM design includes one MMCM that generates the necessary design clocks. One of these outputs is the **sync\_pulse**. The sync pulse clock is 1/16 of the **mem\_refclk** frequency and must have a duty cycle distortion of 1/16 or 6.25%. This clock is distributed across the low skew clock backbone and keeps all PHASER\_IN/\_OUT and PHY\_Control blocks in sync with each other. The signal is sampled by the **mem\_refclk** in both the PHASER\_INs/\_OUTs and PHY\_Control blocks. The phase, frequency, and duty cycle of the **sync\_pulse** is chosen to provide the greatest setup and hold margin across PVT.

Table 4-28 shows an example of a 16-bit LPDDR2 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.



**TIP:** *Termination is not required for LPDDR2 memory interfaces. For more information, contact your memory vendor. The termination guidelines can be used in case termination is required.*

Internal V<sub>REF</sub> is used in this example.

*Table 4-28:*

1	VRP	-	SE	49	-
1	DQ15	D_11	P	48	-
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	DQS-P
1	DQS1_N	D_06	N	43	DQS-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	DQSO_P	C_07	P	32	DQS-P
1	DQSO_N	C_06	N	31	DQS-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	-	C_00	N	25	-
1	RAS_N	B_11	P	24	CCIO-P
1	-	B_10	N	23	-
1	-	B_09	P	22	-
1	-	B_08	N	21	-
1	CK_P	B_07	P	20	DQS-P
1	CK_N	B_06	N	19	DQS-N
1	-	B_05	P	18	-
1	-	B_04	N	17	-
1	CS_N	B_03	P	16	-

Table 4-28:

(Cont'd)

1	-	B_02	N	15	-
1	CKE	B_01	P	14	-
1	A12	B_00	N	13	-
1	-	A_11	P	12	-
1	-	A_10	N	11	-
1	A9	A_09	P	10	-
1	A8	A_08	N	9	-
1	A7	A_07	P	8	DQS-P
1	A6	A_06	N	7	DQS-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	AO	A_00	N	1	-
1	VRN	-	SE	0	-

## System Clock

If the SRCC/MRCC I/O pin and PLL are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint must be set to BACKBONE. LPDDR2 SDRAM manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on the SRCC/MRCC I/O and PLL allocation needs to be handled manually for the IP flow. LPDDR2 SDRAM does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

## Reference Clock

If the SRCC/MRCC I/O pin and MMCM are not allocated in the same bank, the CLOCK\_DEDICATED\_ROUTE constraint is set to FALSE. Reference clock is a 200 MHz clock source used to drive IODELAY CTRL logic (through an additional MMCM). This clock is not utilized, CLOCK\_DEDICATED\_ROUTE (as they are limited in number), hence the FALSE value is set. LPDDR2 SDRAM manages these constraints for designs generated with the **System Clock** option selected as **Differential/Single-Ended** (at **FPGA Options > System Clock**).

If the design is generated with the **System Clock** option selected as **No Buffer** (at **FPGA Options > System Clock**), the CLOCK\_DEDICATED\_ROUTE constraints based on SRCC/MRCC I/O and MMCM allocation needs to be handled manually for the IP flow. LPDDR2 SDRAM does not generate clock constraints in the XDC file for the **No Buffer** configurations. You must take care of the clock constraints for the **No Buffer** configurations in the IP flow.

# Multicontroller Design

---

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 eight controllers
  - Multi-interface support includes the combination of all memory interfaces as DDR3 SDRAM (Native only), QDR II+ SRAM, and RLDRAM II up to total of eight 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 eight 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, etc.) 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, QDR II+ SRAM, and RLDRAM II) remain the same as for single controller designs.

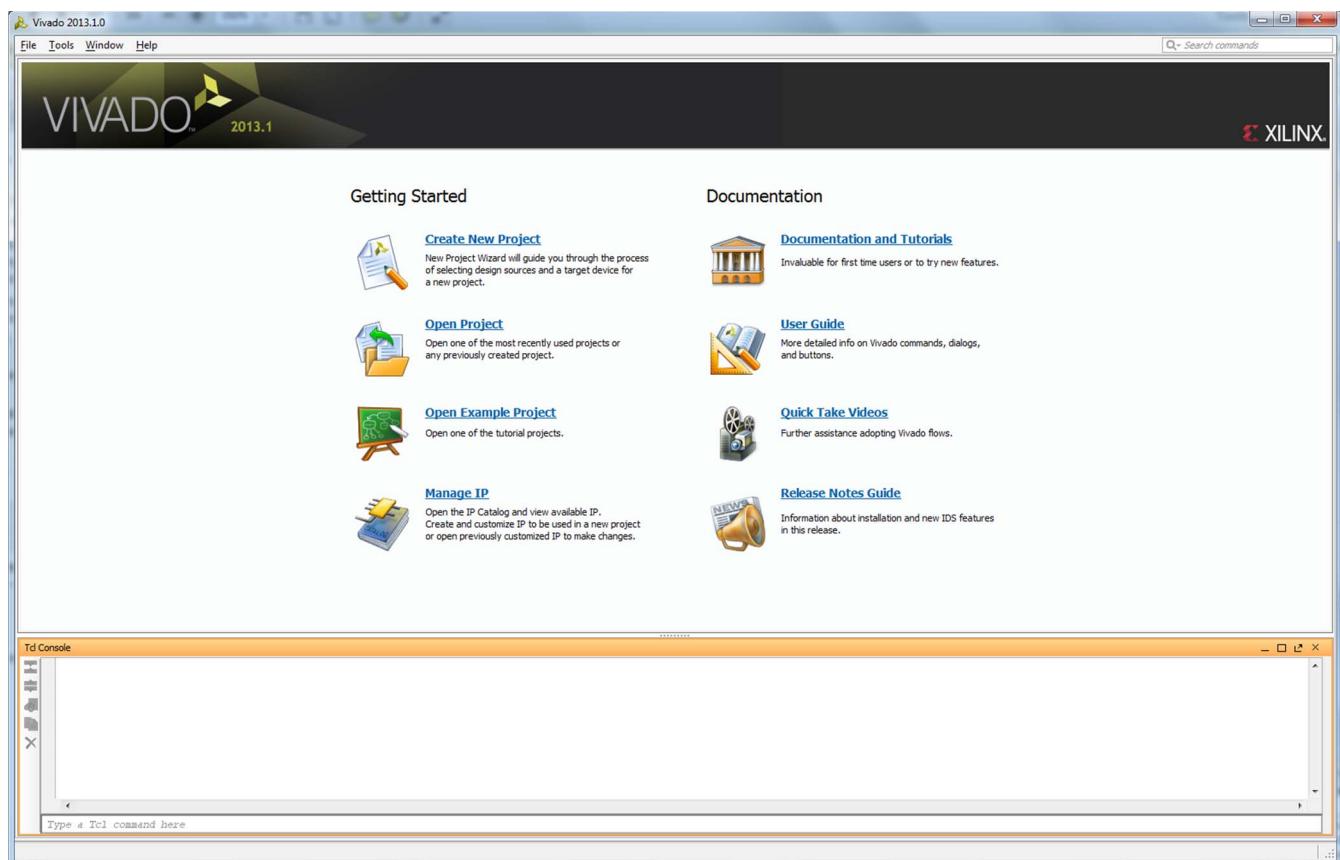


**IMPORTANT:** *Memory Interface Solutions v4.2 only supports the Vivado® Design Suite. The ISE® Design Suite is not supported in this version.*

---

This section provides the steps to generate the Memory Interface Generator (MIG) IP core using the Vivado Design Suite and run implementation.

1. Start the Vivado Design Suite (see [Figure 5-1](#)).



*Figure 5-1:*

2. To create a new project, click the **Create New Project** option shown in [Figure 5-1](#) to open the page as shown in [Figure 5-2](#).

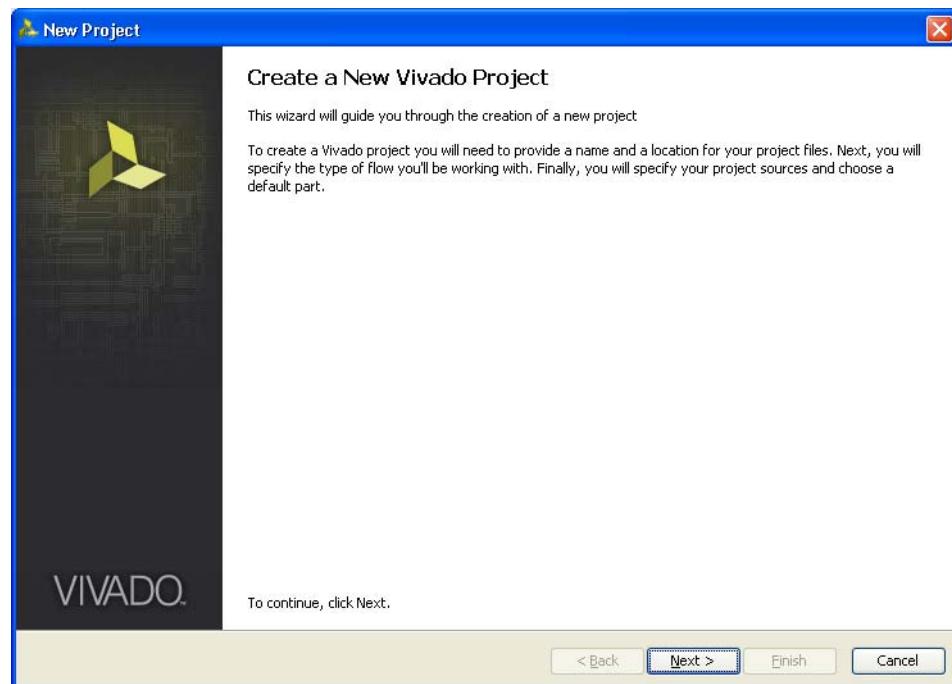


Figure 5-2:

3. Click **Next** to proceed to the **Project Name** page (Figure 5-3). Enter the **Project Name** and **Project Location**. Based on the details provided, the project is saved in the directory.

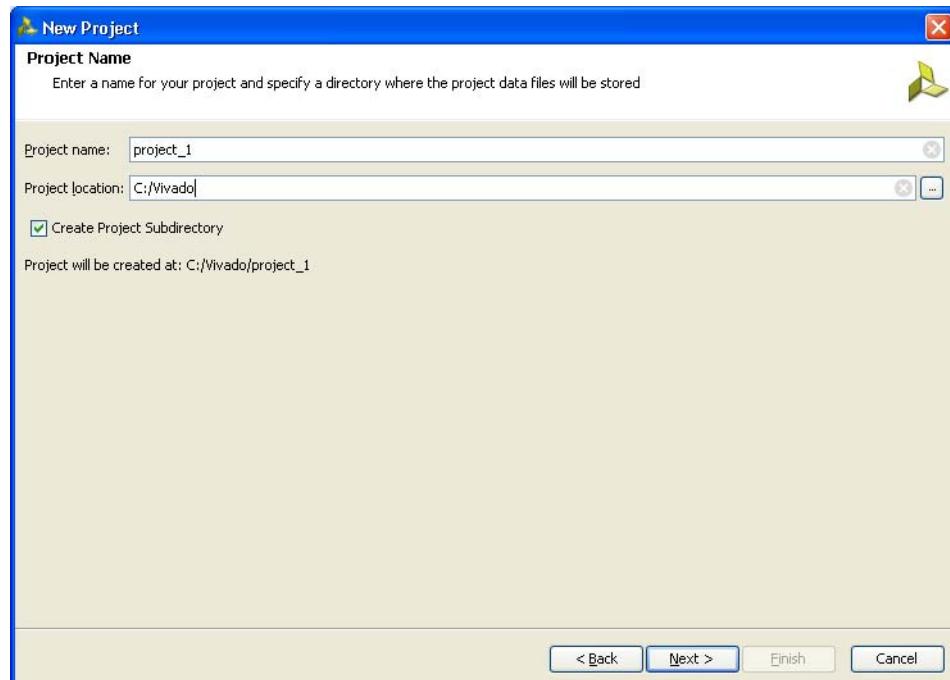
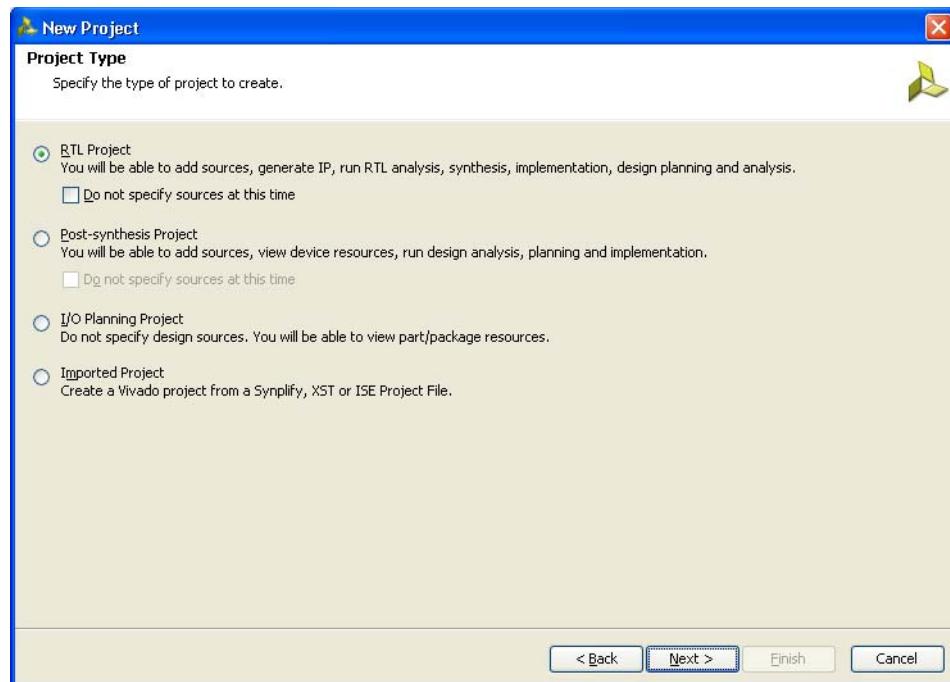


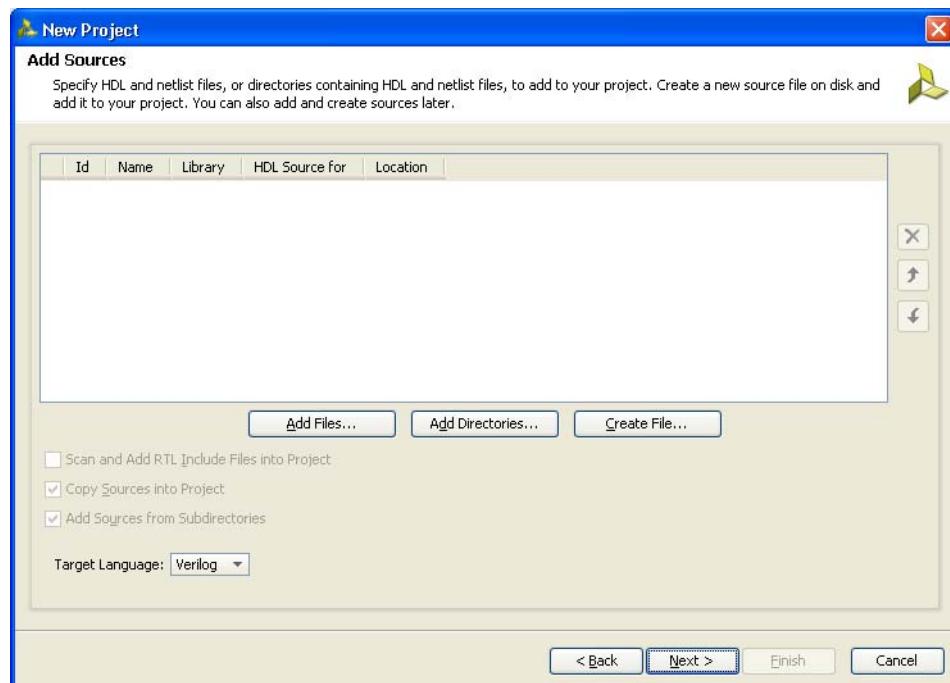
Figure 5-3:

4. Click **Next** to proceed to the **Project Type** page (Figure 5-4). Select the **Project Type** as **RTL Project** because MIG deliverables are RTL files.



*Figure 5-4:*

5. Click **Next** to proceed to the **Add Sources** page (Figure 5-5). RTL files can be added to the project in this page. If the project was not created earlier, proceed to the next page.



*Figure 5-5:*

6. Click **Next** to open the **Add Existing IP (Optional)** page (Figure 5-6). If the IP is already created, the XCI file generated by the IP can be added to the project and the previous created IP files are automatically added to the project. If the IP was not created earlier, proceed to the next page.

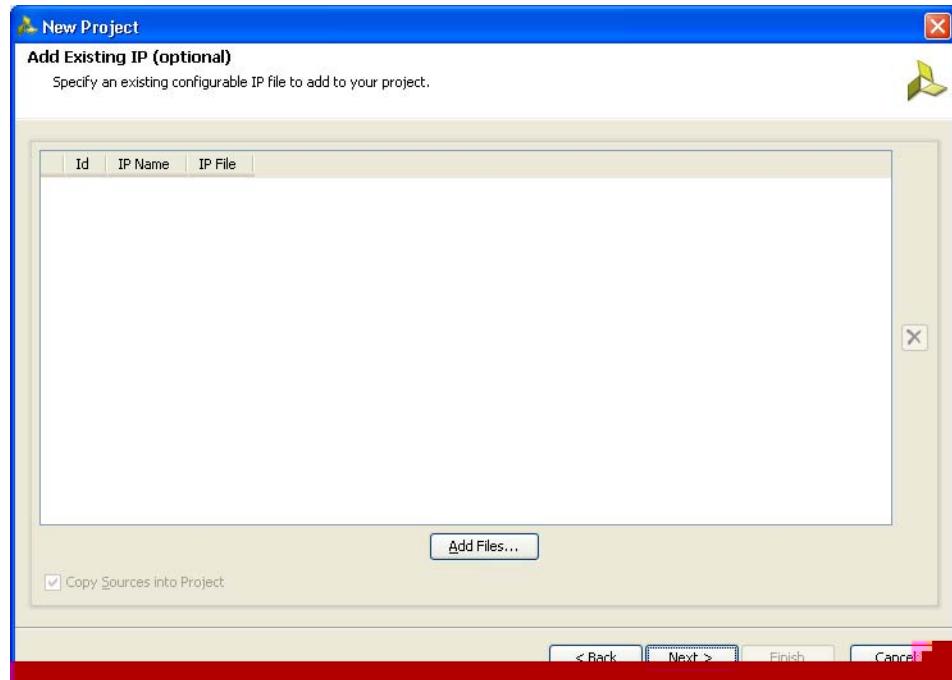


Figure 5-6:

7. Click **Next** to open the **Add Constraints (Optional)** page (Figure 5-7). If the constraints file exists in the repository, it can be added to the project. Proceed to the next page if the constraints file does not exist.

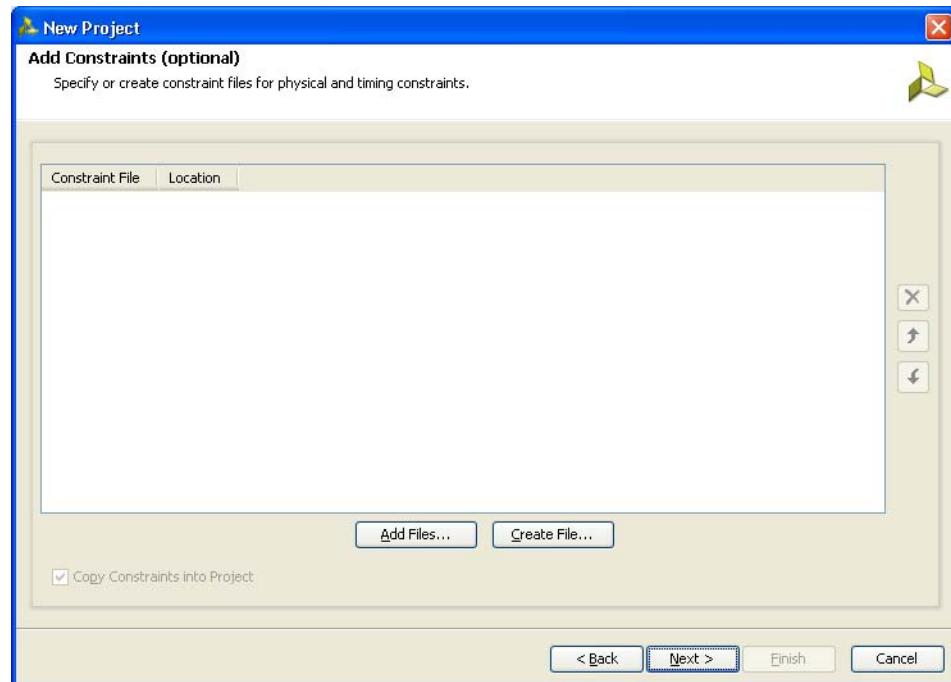


Figure 5-7:

- Click **Next** to proceed to the **Default Part** page (Figure 5-8) where the device that needs to be targeted can be selected. The **Default Part** page appears as shown in Figure 5-8.

Device	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	PCI Buses
xc7vx485tffg1157-2	1,157	600	303600	607200	1030	2800	20	4
xc7vx485tffg1157-1	1,157	600	303600	607200	1030	2800	20	4
xc7vx485tffg1158-3	1,158	350	303600	607200	1030	2800	48	4
xc7vx485tffg1158-2	1,158	350	303600	607200	1030	2800	48	4
xc7vx485tffg1158-2L	1,158	350	303600	607200	1030	2800	48	4
xc7vx485tffg1158-1	1,158	350	303600	607200	1030	2800	48	4
xc7vx485tffg1761-3	1,761	700	303600	607200	1030	2800	28	4
xc7vx485tffg1761-2	1,761	700	303600	607200	1030	2800	28	4
xc7vx485tffg1761-2L	1,761	700	303600	607200	1030	2800	28	4
xc7vx485tffg1761-1	1,761	700	303600	607200	1030	2800	28	4
xc7vx485tffg1927-3	1,927	600	303600	607200	1030	2800	56	4

Figure 5-8:

Select the target **Family**, **Package**, and **Speed Grade**. The valid devices are displayed in the same page, and the device can be selected based on the targeted device (Figure 5-9).

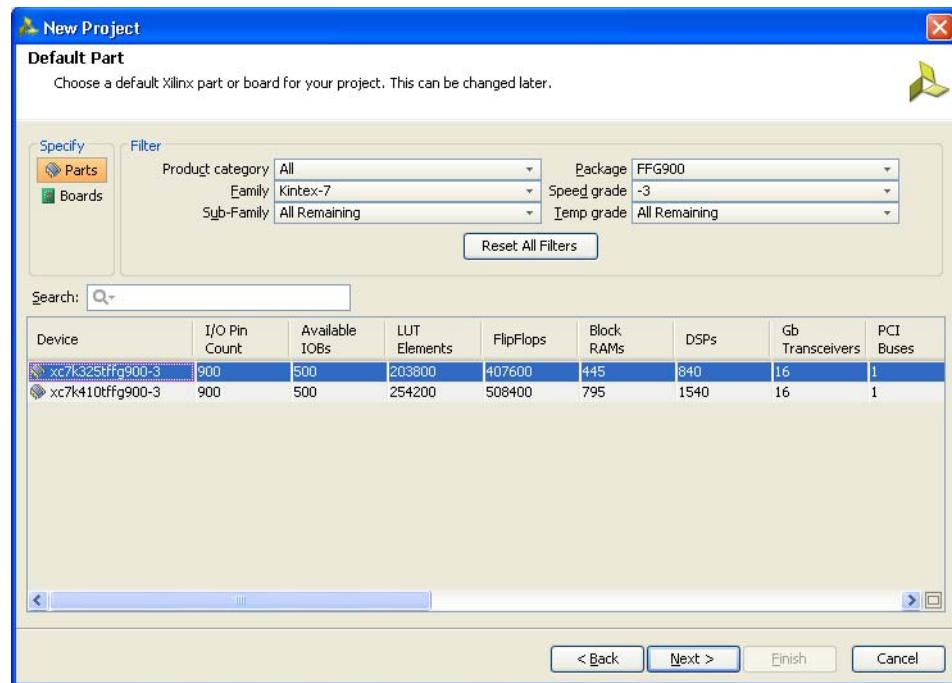


Figure 5-9:

Apart from selecting the parts by using the **Parts** option, parts can be selected by choosing the **Boards** option, which brings up the evaluation boards supported by Xilinx® (Figure 5-10). With this option, designs can be targeted for the various evaluation boards. If the XCI file of an existing IP was selected in an earlier step, the same part should be selected here.

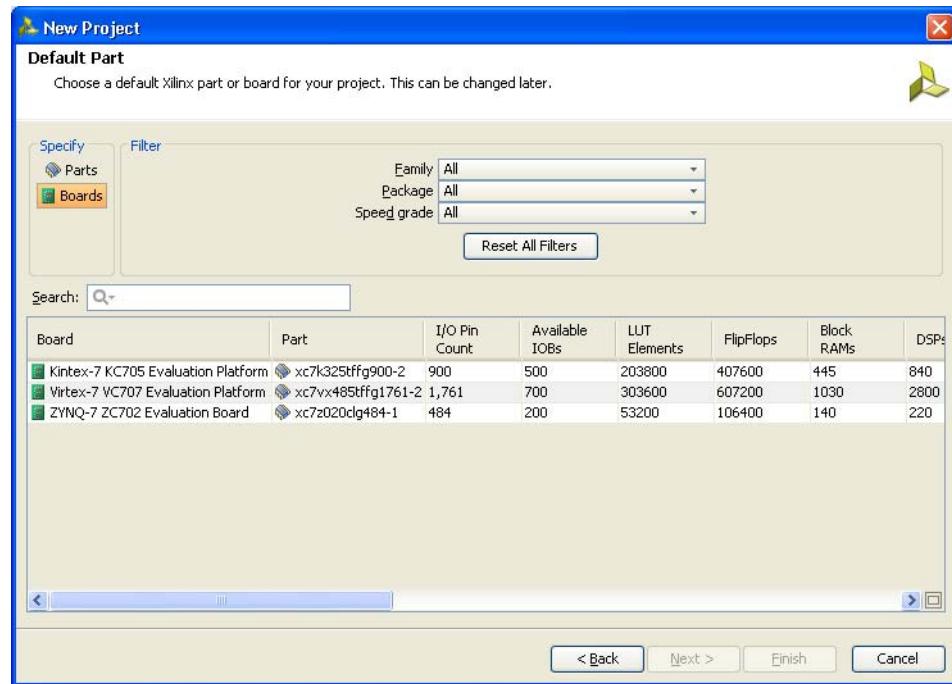


Figure 5-10:

- Click **Next** to open the **New Project Summary** page (Figure 5-11). This includes the summary of selected project details.

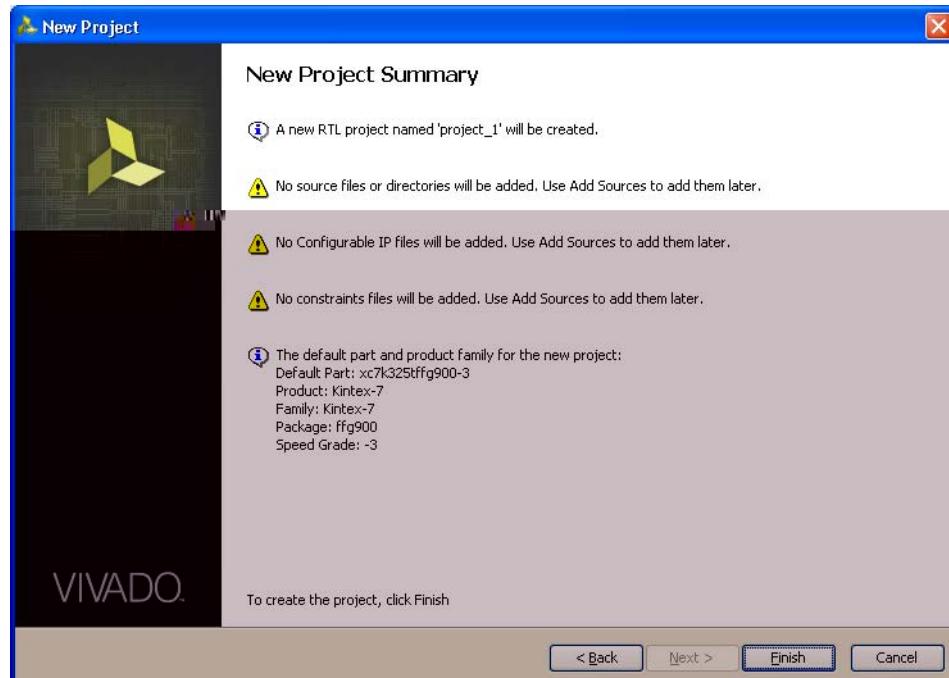
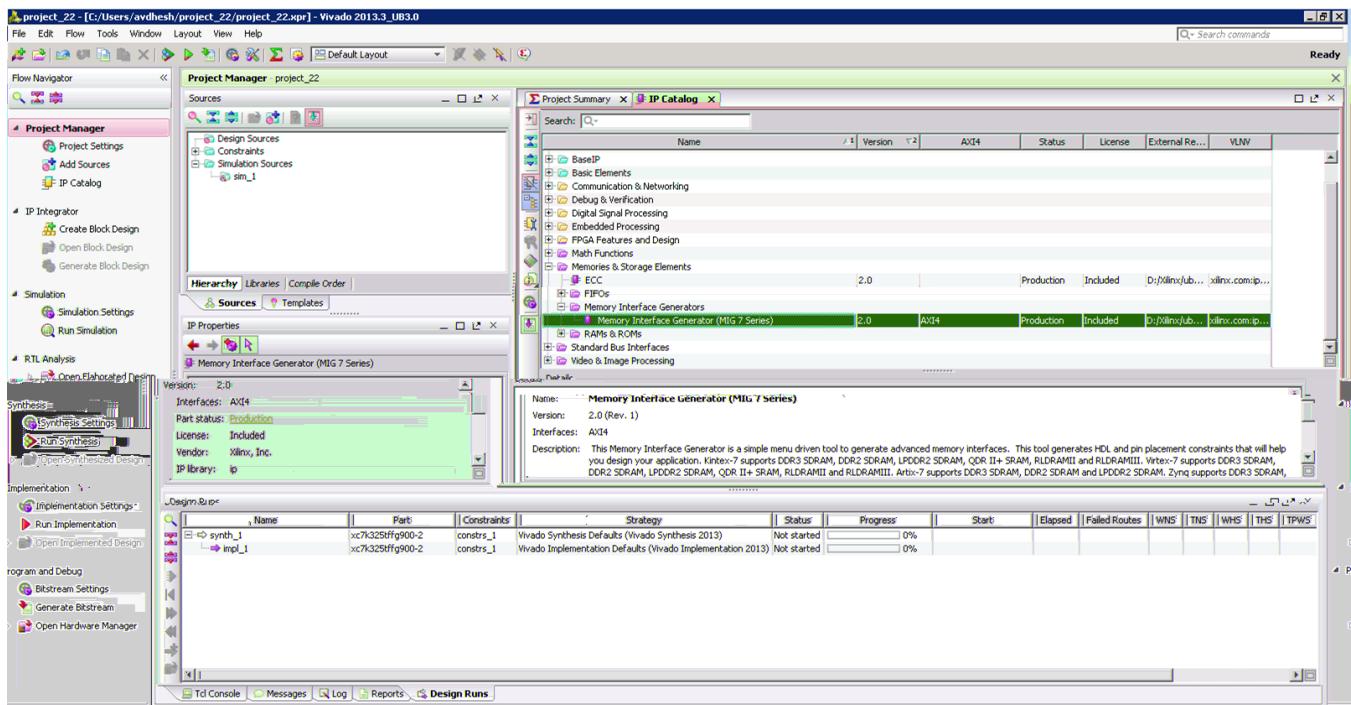


Figure 5-11:

- Click **Finish** to complete the project creation.

11. Click **IP Catalog** on the **Project Manager** window to open the IP catalog window. The Vivado IP catalog window appears on the right side panel (see [Figure 5-12](#)).

12. The MIG tool exists in the **Memories & Storage Elements > Memory Interface Generators** section of the IP catalog window ([Figure 5-12](#)) or you can search from the Search tool bar for the string “MIG.”



*Figure 5-12:*

13. Select **MIG 7 Series** to open the MIG tool (Figure 5-13).

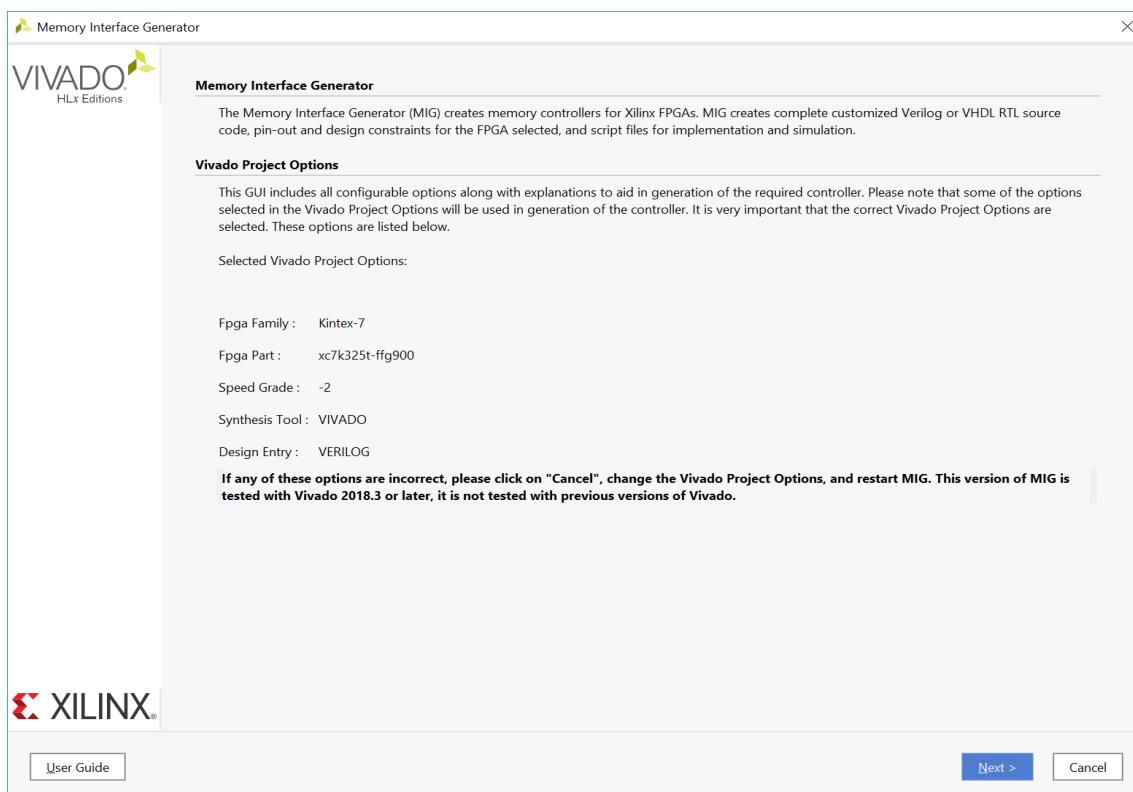


Figure 5-13:



**CAUTION!** *The Windows operating system has a 260-character limit for path lengths, which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, and creating block designs.*

## Multiple Controllers

Select the number of controllers from the **MIG Output Options** page (Figure 5-14). 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).

The screenshot shows the Xilinx Vivado Memory Interface Generator (MIG) configuration interface. The top navigation bar includes the Xilinx logo, the title "Memory Interface Generator", and a close button. The main content area is titled "MIG Output Options". It contains two radio button options: "Create Design" (selected) and "Verify Pin Changes and Update Design". The "Create Design" option is described as generating a memory controller with RTL, XDC, implementation, and simulation files. The "Verify Pin Changes and Update Design" option is described as verifying modified XDC files for an already generated design, updating the input XDC file to be compatible with the current version of MIG, and preserving pin outs while generating a new design with the selected component name. Below this is a "Component Name" section with a text input field containing "mig\_7series\_0" and a clear button. The next section is "Multi-Controller", which specifies up to 8 controllers with a combination of DDR3 SDRAM, QDRII+ SRAM, or RLDRAM II. A dropdown menu shows "Number of controllers" set to 1. The final section is "AXI4 Interface", which enables the AXI4 interface for supported DDR3 and DDR2 SDRAM controllers. A checkbox labeled "AXI4 Interface" is present. At the bottom of the interface are buttons for "User Guide", "< Back" (disabled), "Next >" (highlighted in blue), and "Cancel".

Figure 5-14:

## Creating 7 Series FPGA Multicontroller Block Design

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 5-15).

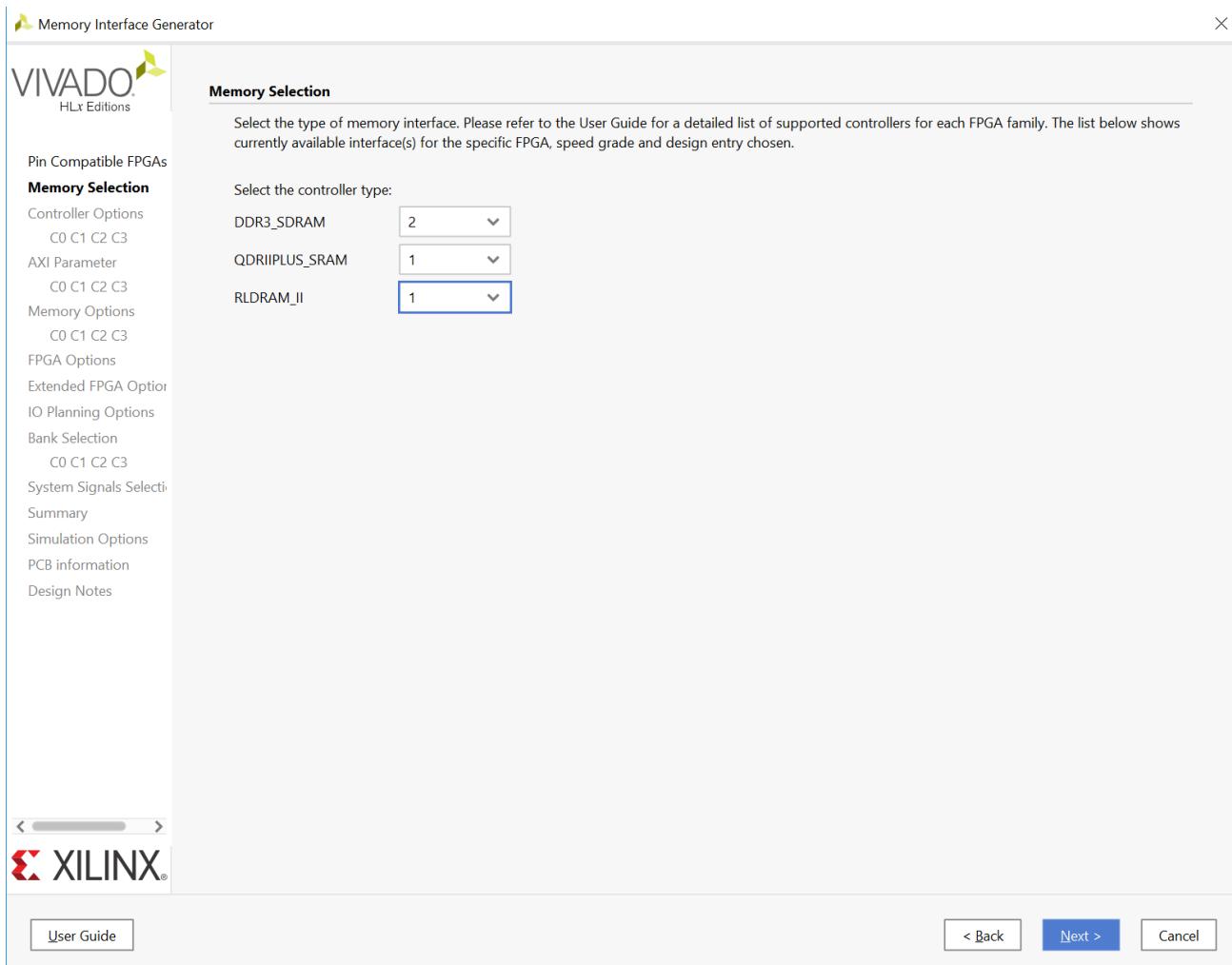
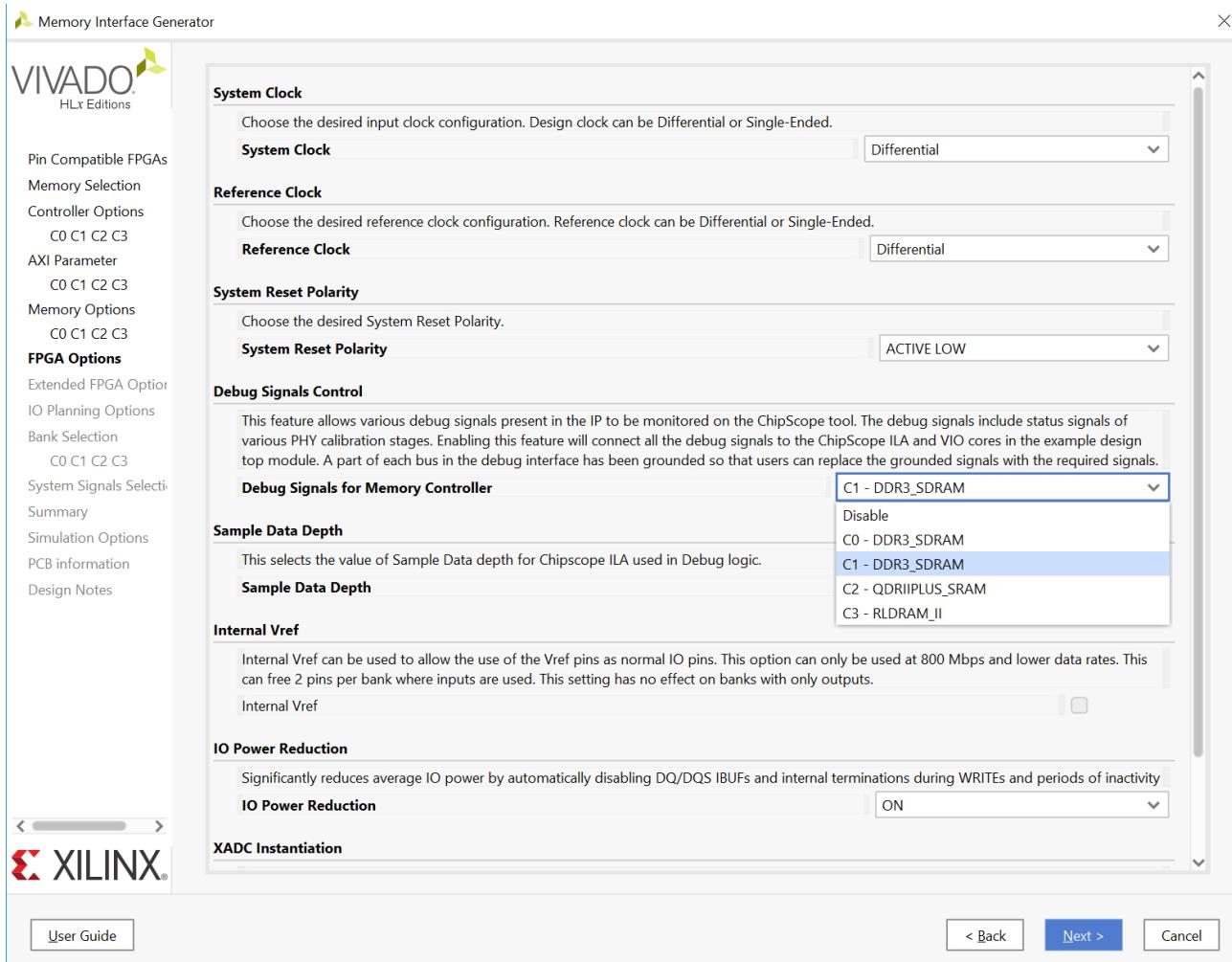


Figure 5-15:

The Debug option can be selected for one controller only. Debug logic is generated for the selected controller ([Figure 5-16](#)).



*Figure 5-16:*

Figure 5-17 shows the **Extended FPGA Options** page for a multicontroller design with all three memory inter

4

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 5-18).

The screenshot shows the 'System Signals Selection' interface. At the top, there is a note: 'Select the system pins below appropriately for the interface. Customization of these pins can also be made in the XDC after the design is generated. For more information see [UG586 Bank and Pin rules](#).'. Below this, a note states: 'System Clock and Reference Clock pin selections will not be visible if the 'No Buffer' option was selected in the FPGA Options page.' The main section is titled 'System Clock Pin Selection' and contains a table with four rows. The table has three columns: 'Signal Name', 'Bank Number', and 'Pin Number'. The data is as follows:

Signal Name	Bank Number	Pin Number
c0_sys_clk_p/n	39	J13/H13(CC_P/N)
c1_sys_clk_p/n	38	K19/J18(CC_P/N)
c2_sys_clk_p/n	37	C28/C29(CC_P/N)
c3_sys_clk_p/n	36	K23/J23(CC_P/N)

Figure 5-18:

The criteria for sharing a system clock pin is as follows:

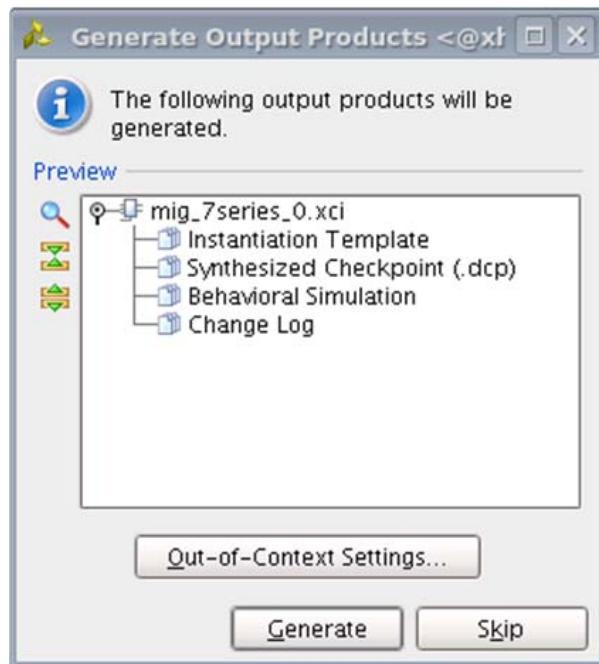
- System clock pins can be shared across the controllers when the input frequency is the same for the controllers.
- Pins can be shared across the controllers as long as the memory interface chosen banks are in the same column.



**RECOMMENDED:** *Although the MIG allows system clock selection to be in different super logic regions (SLRs), it is not recommended due to the additional clock jitter in this topology.*

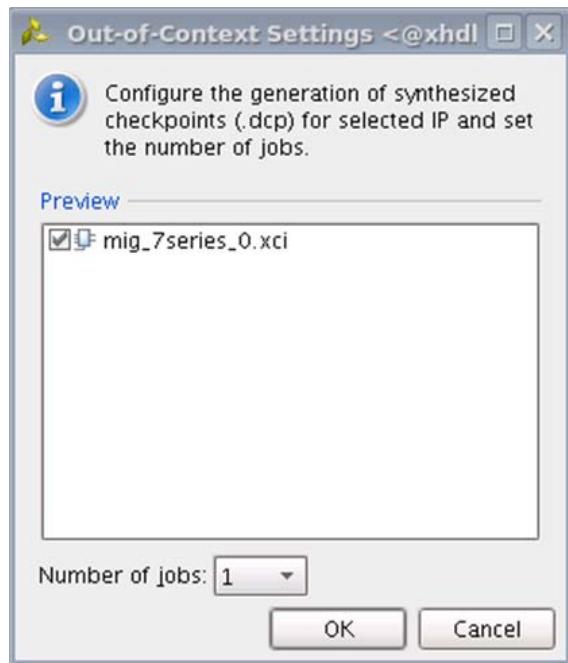
- One CCIO port can drive any number of PLLs and there is no restriction on maximum number of PLLs that a system clock pin can drive. So the same pin can be used for any number of controllers.
- MIG validates the rules after clicking **Next** and following the selection for System Clock pins is done.
- Selecting the same pin indicates the same pin is shared across the controllers.
- One PLL and one MMCM are needed for each controller regardless of system clock pin is shared or not. System clock pin can only be shared and no other resources (PLL or MMCM) are shared across controllers.

1. After clicking **Generate**, the **Generate Output Products** window appears. This window has the **Out-of-Context Settings** as shown in [Figure 5-19](#).



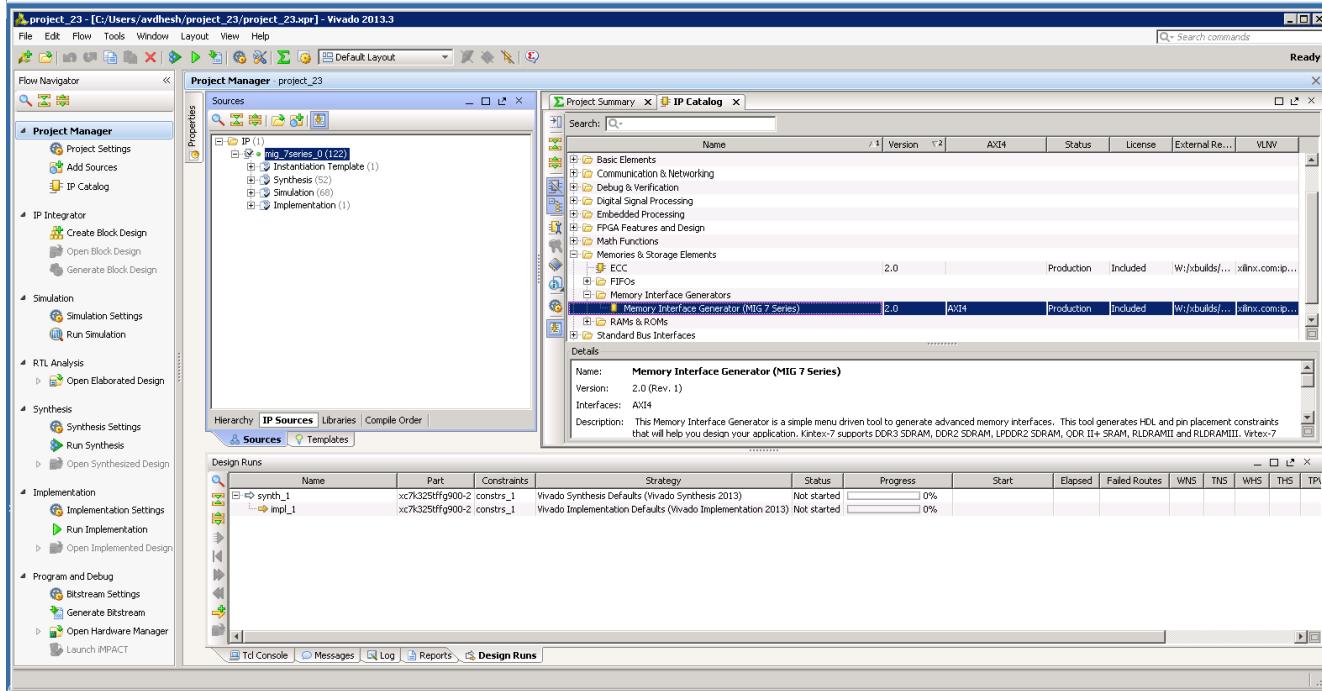
*Figure 5-19:*

2. Click **Out-of-Context Settings** to configure generation of synthesized checkpoints. To enable the **Out-of-Context** flow, enable the check box. To disable the **Out-of-Context** flow, disable the check box. The default option is **Enable** as shown in [Figure 5-20](#).



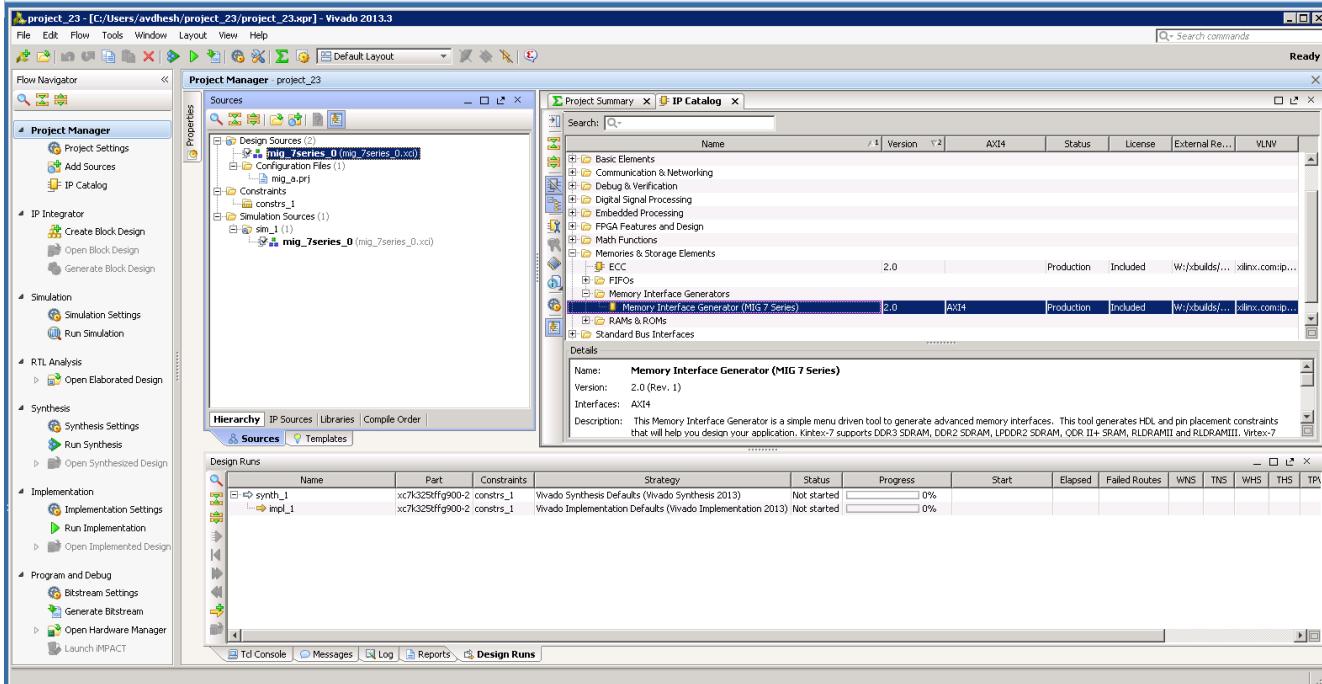
*Figure 5-20:*

3. MIG designs comply with "Hierarchical Design" flow in Vivado. For more information, see the *Vivado Design Suite User Guide: Hierarchical Design* (UG905) [\[Ref 5\]](#) and the *Vivado Design Suite Tutorial: Hierarchical Design* (UG946) [\[Ref 6\]](#).
4. After generating the MIG design, the project window appears as shown in [Figure 5-21](#).



*Figure 5-21:*

- After project creation, the XCI file is added to the Project Hierarchy. The same view also displays the module hierarchies of the user design. The list of HDL and XDC files is available in the **IP Sources** view in the **Sources** window. Double-clicking on any module or file opens the file in the Vivado Editor. These files are read only.



*Figure 5-22:*

Design generation from MIG can be generated using the **Create Design** flow or the **Verify Pin Changes** and **Update Design** flow. There is no difference between the flow when generating the design from the MIG tool. Irrespective of the flow by which designs are generated from the MIG tool, the XCI file is added to the Vivado project. The implementation flow is the same for all scenarios because the flow depends on the XCI file added to the project.

6. All MIG generated user design RTL and XDC files are automatically added to the project. If files are modified and you wish to regenerate them, right-click the XCI file and select **Generate Output Products** (Figure 5-23).

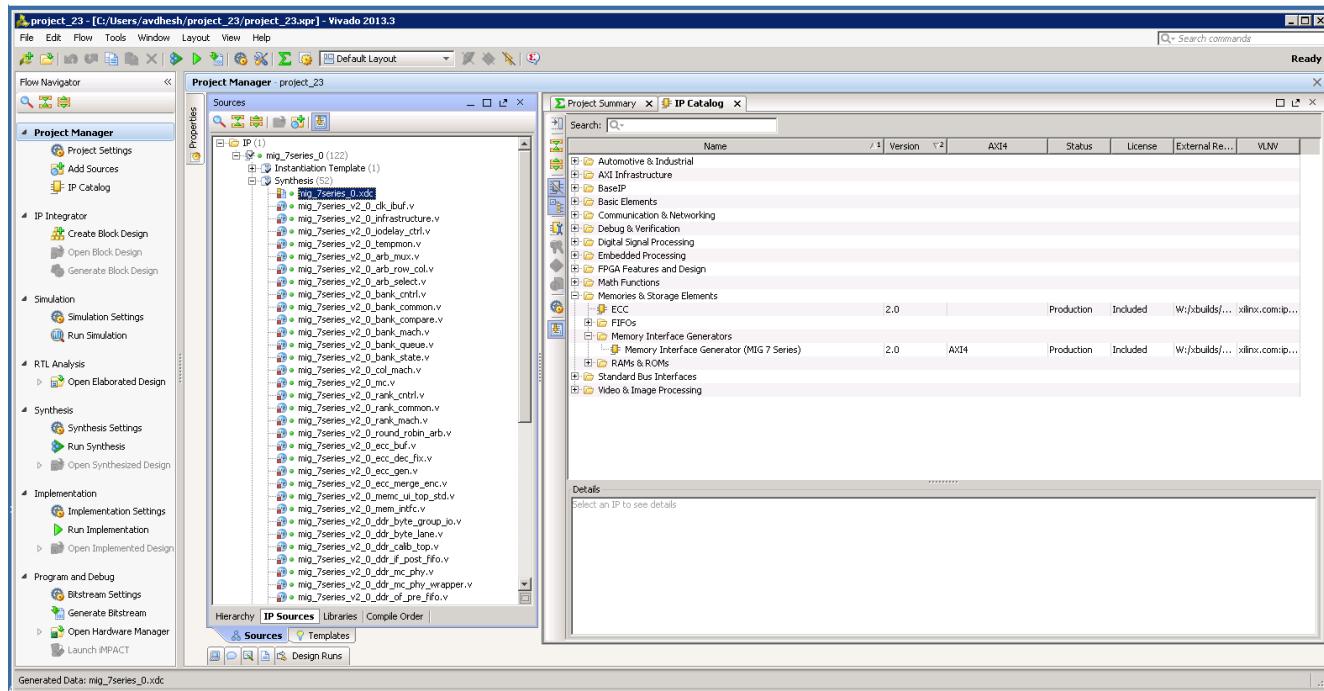
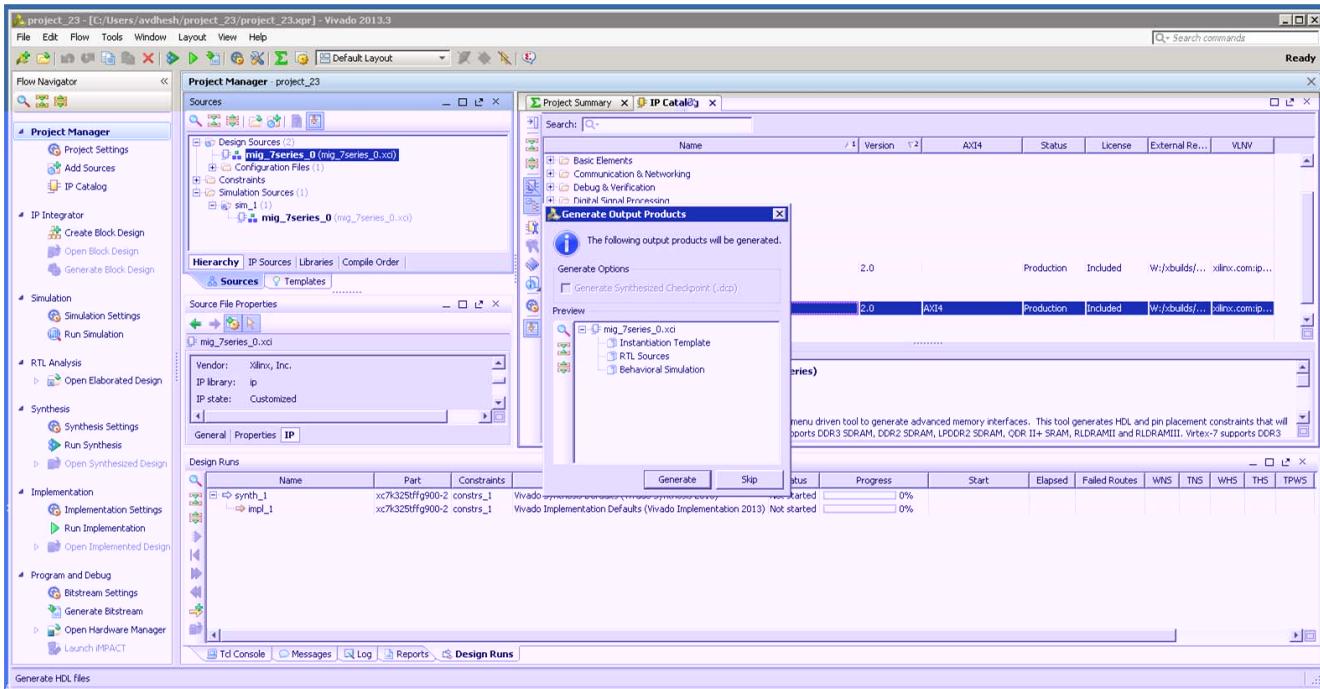


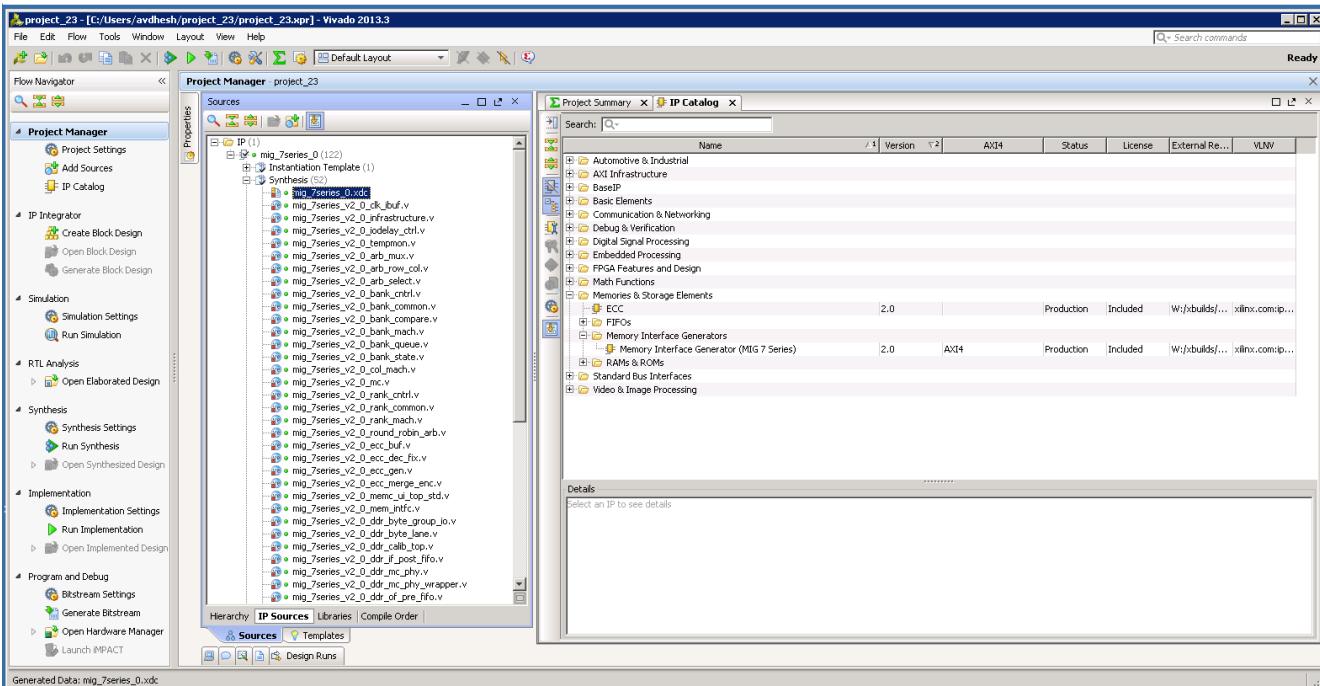
Figure 5-23:

7. Clicking **Generate Output Products** option brings up the **Manage Outputs** window (Figure 5-24).



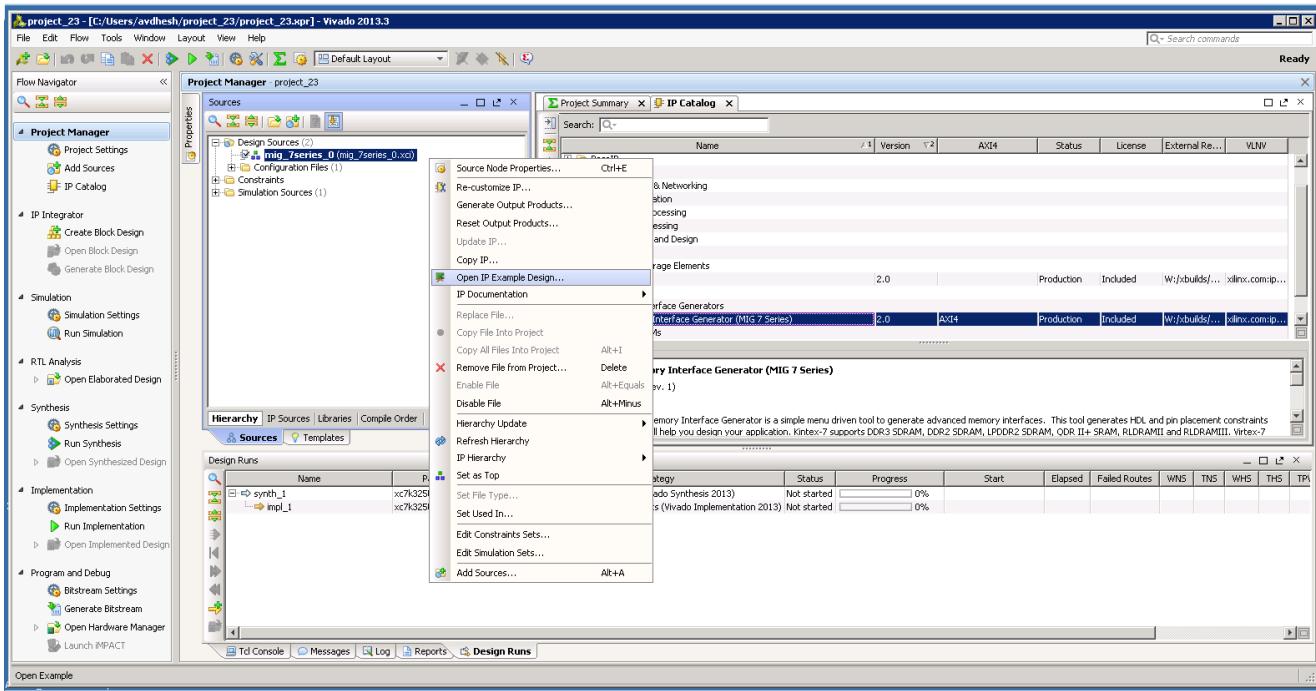
*Figure 5-24:*

8. All user-design RTL files and constraints files (XDC files) can be viewed in the **Sources > Libraries** tab (Figure 5-25).



*Figure 5-25:*

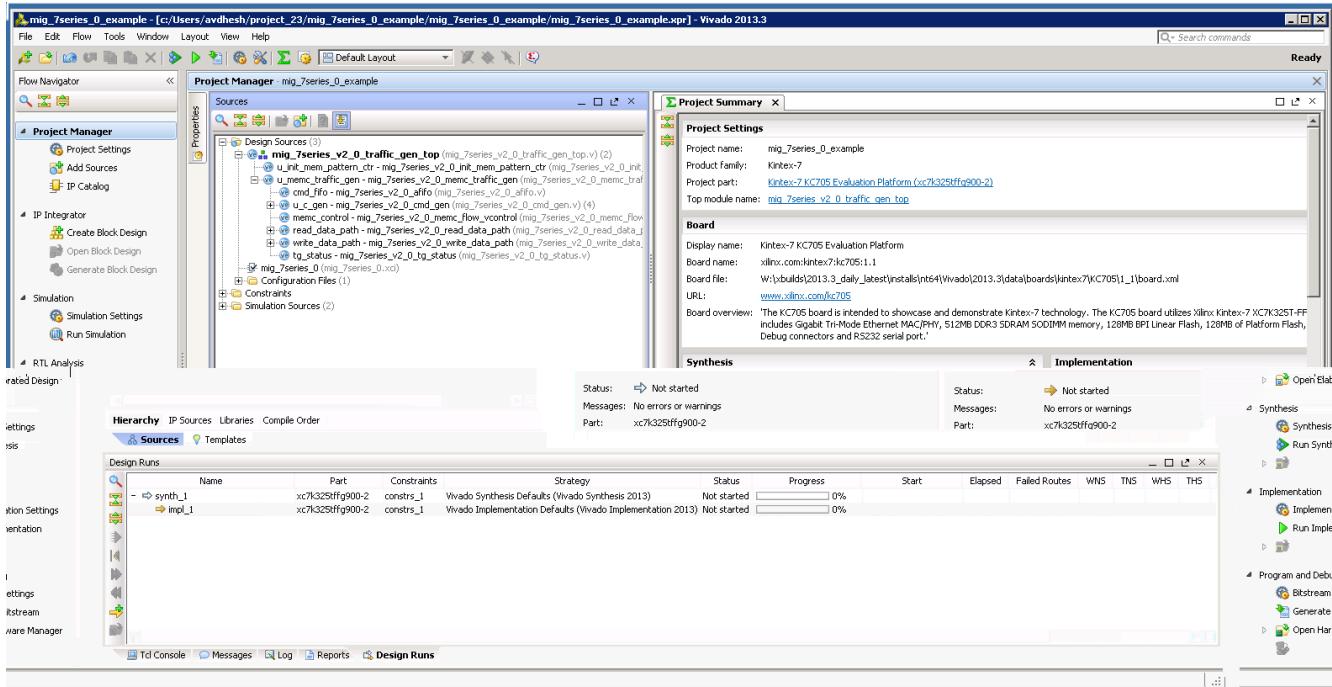
9. The Vivado Design Suite supports **Open IP Example Design** flow. To create the example design using this flow right-click the IP in the **Source Window**, as shown in [Figure 5-26](#) and select.



*Figure 5-26:*

10. This option creates a new Vivado project. Selecting the menu brings up a dialog box, which guides you to the directory for a new design project. Select a directory (or use the defaults) and click **OK**.

This launches a new Vivado project with all example design files and a copy of the IP. This project has **example\_top** as the Implementation top directory, and **sim\_tb\_top** as the Simulation top directory, as shown in [Figure 5-27](#).



*Figure 5-27:*

11. Click **Generate Bitstream** under **Project Manager > Program and Debug** to generate the BIT file for the generated design.

The `<project_directory>/<project_directory>.runs/ impl_1` directory includes all report files generated for the project after running the implementation. It is also possible to run the simulation in this project.

12. Recustomization of the MIG IP core can be done by using the **Recustomize IP** option. It is not recommended to recustomize the IP in the `example_design` project. The correct solution is to close the `example_design` project, go back to original project and customize there. Right-click the XCI file and click **Recustomize IP** (Figure 5-28) to open the MIG GUI and regenerate the design with the preferred options.

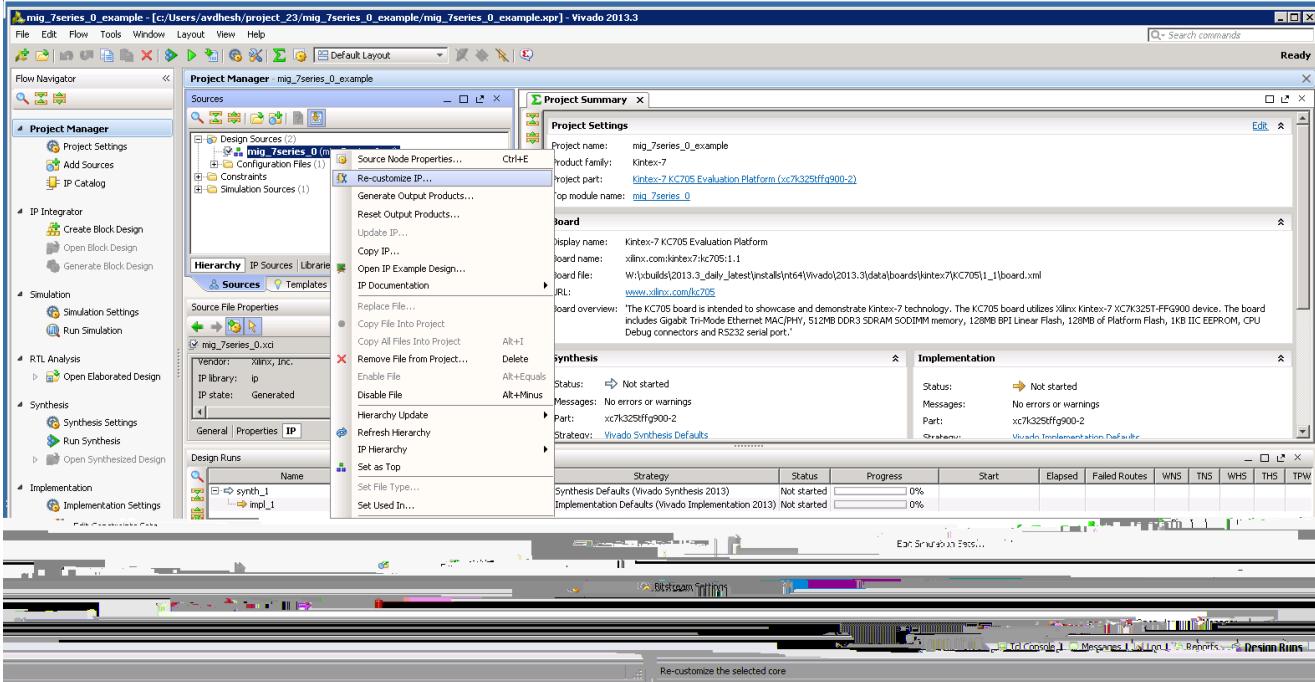
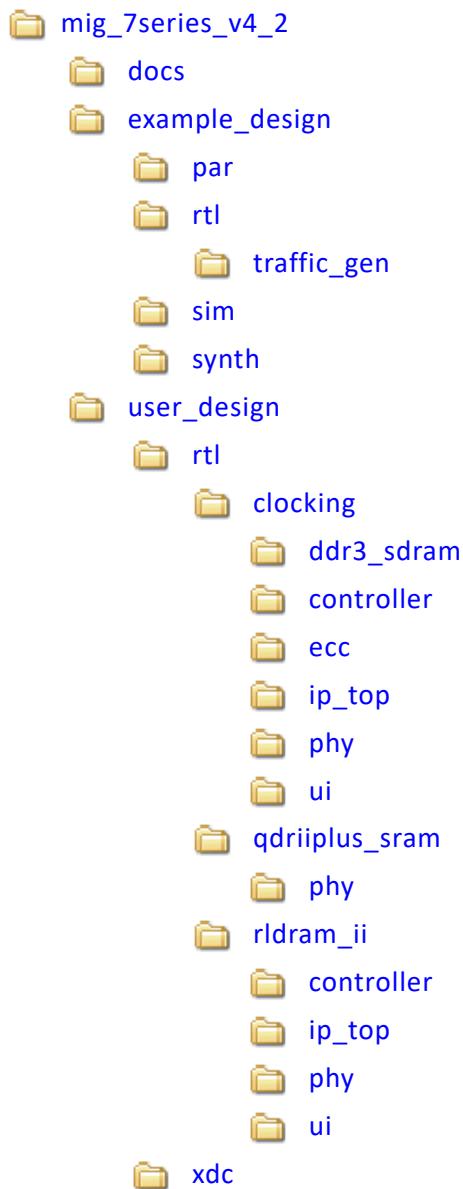


Figure 5-28:

Invoking the MIG tool from the Vivado Design Suite 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.

## 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 here.



# Upgrading the ISE/CORE Generator MIG Core in Vivado

To upgrade the previous version of the Memory Interface Generator (MIG) IP cores which are generated using either ISE® or CORE Generator™, tools cannot be upgraded in a direct manner similar to other IPs. Here is the process to upgrade the ISE/CORE Generator MIG core in Vivado®.

1. This requires the **mig.prj** file of the MIG core generated using ISE or CORE Generator.
2. Invoke Vivado with the same FPGA part settings that the earlier core is generated with.
3. Apply the following command in the Tcl Console of Vivado to create the IP:

```
create_ip -name mig_7series -version <latest version> -vendor xilinx.com -library ip  
-module_name <component_name>
```

For example,

```
create_ip -name mig_7series -version 4.2 -vendor xilinx.com -library ip -module_name  
mig_7series_0
```

4. Apply the following command to generate the core with the previous MIG project settings:

```
set_property CONFIG.XML_INPUT_FILE {<absolute path of the old core mig.prj>}  
[get_ips <ip_name>]
```

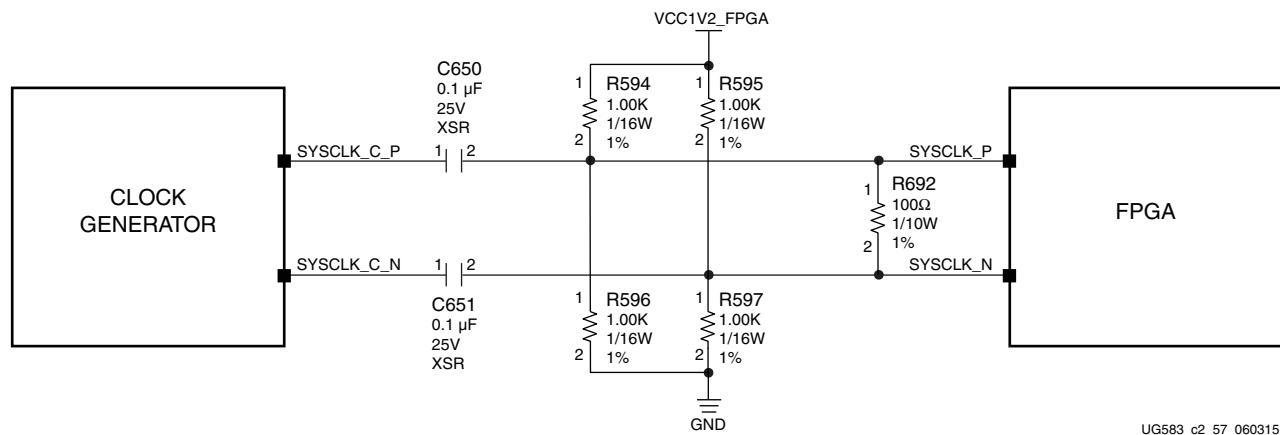
For example,

```
set_property CONFIG.XML_INPUT_FILE  
{/proj/mig/users/coregen_core/mig_7series_v4_2/mig.prj} [get_ips mig_7series_0]
```

5. You can see the core created in the Hierarchy.

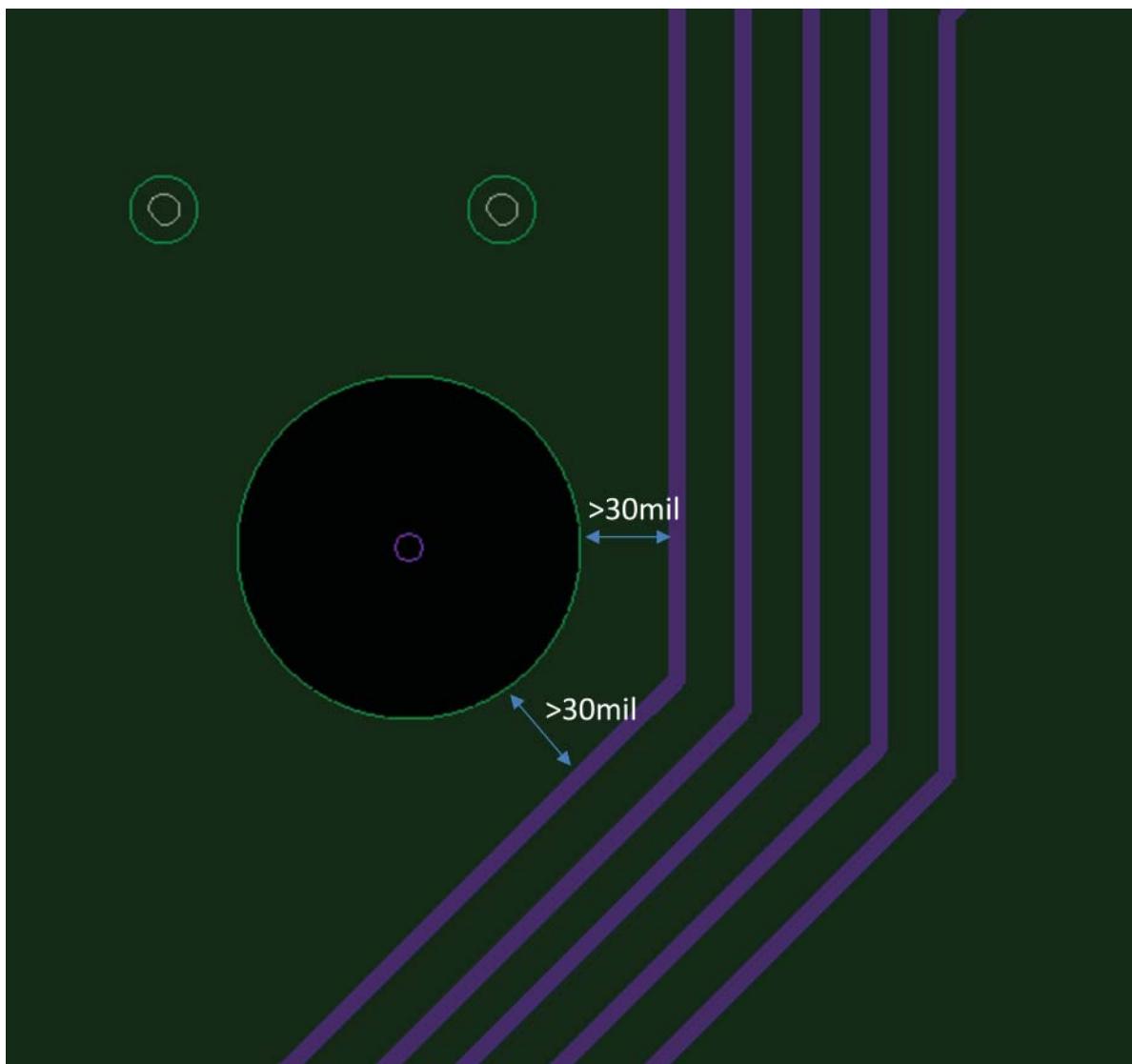
# General Memory Routing Guidelines

1. Include package delay in routing constraints when determining signal trace lengths. When minimum and maximum values are available for the package delay, use the midpoint between the minimum and maximum values.
2. DQ and DQS signals in the same byte should be routed in the same layer from FPGA to DRAM, except in the breakout areas.
3. For fly-by routing, address, command, control, and clock signals can be routed on different layers but each signal needs to be routed consistently in one layer across all DRAMs. Any signal layer switching via needs to have one ground via within a 50 mil perimeter range.
4. Ensure the memory ODT settings match transmission line impedance.
5. ck as shown in [Figure A-1](#).



*Figure A-1:*

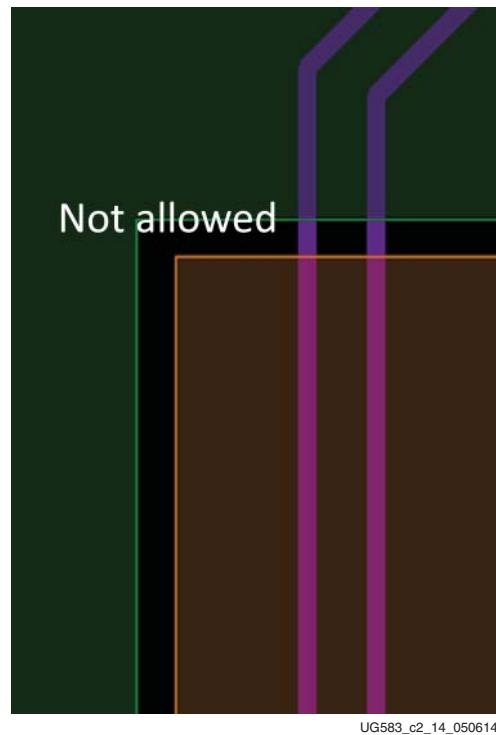
6. Signal lines must be routed over a solid reference plane. Avoid routing over voids (Figure A-2).



UG583\_c2\_13\_050614

Figure A-2:

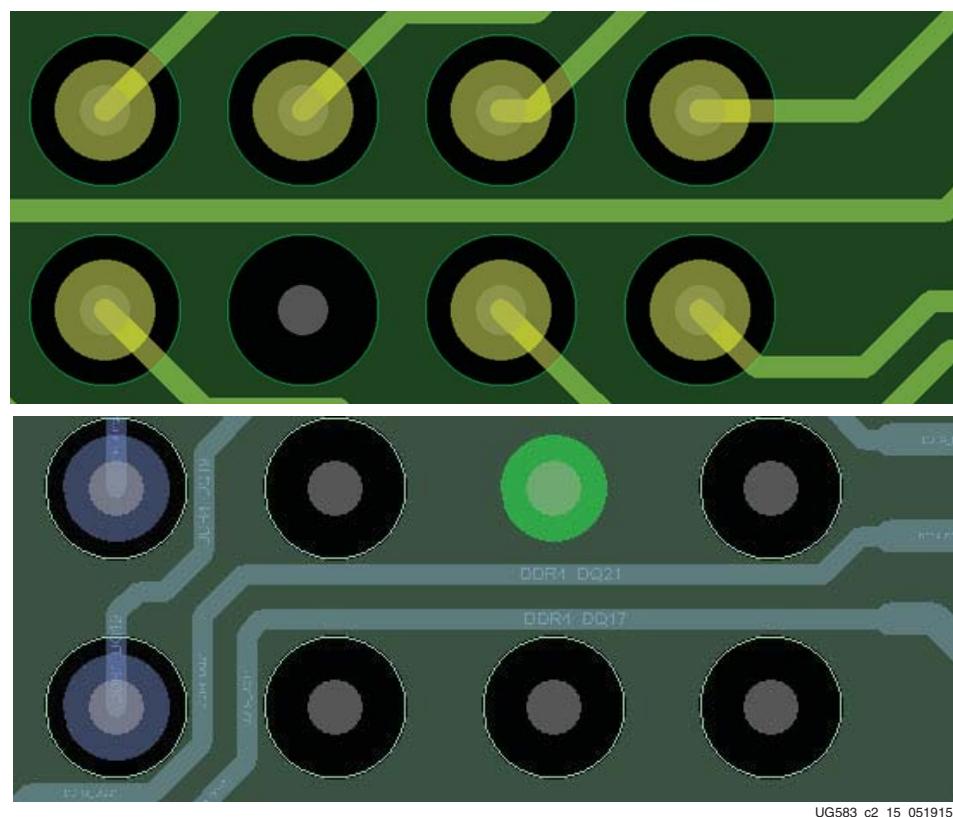
7. Avoid routing over reference plane splits ([Figure A-3](#)).



*Figure A-3:*

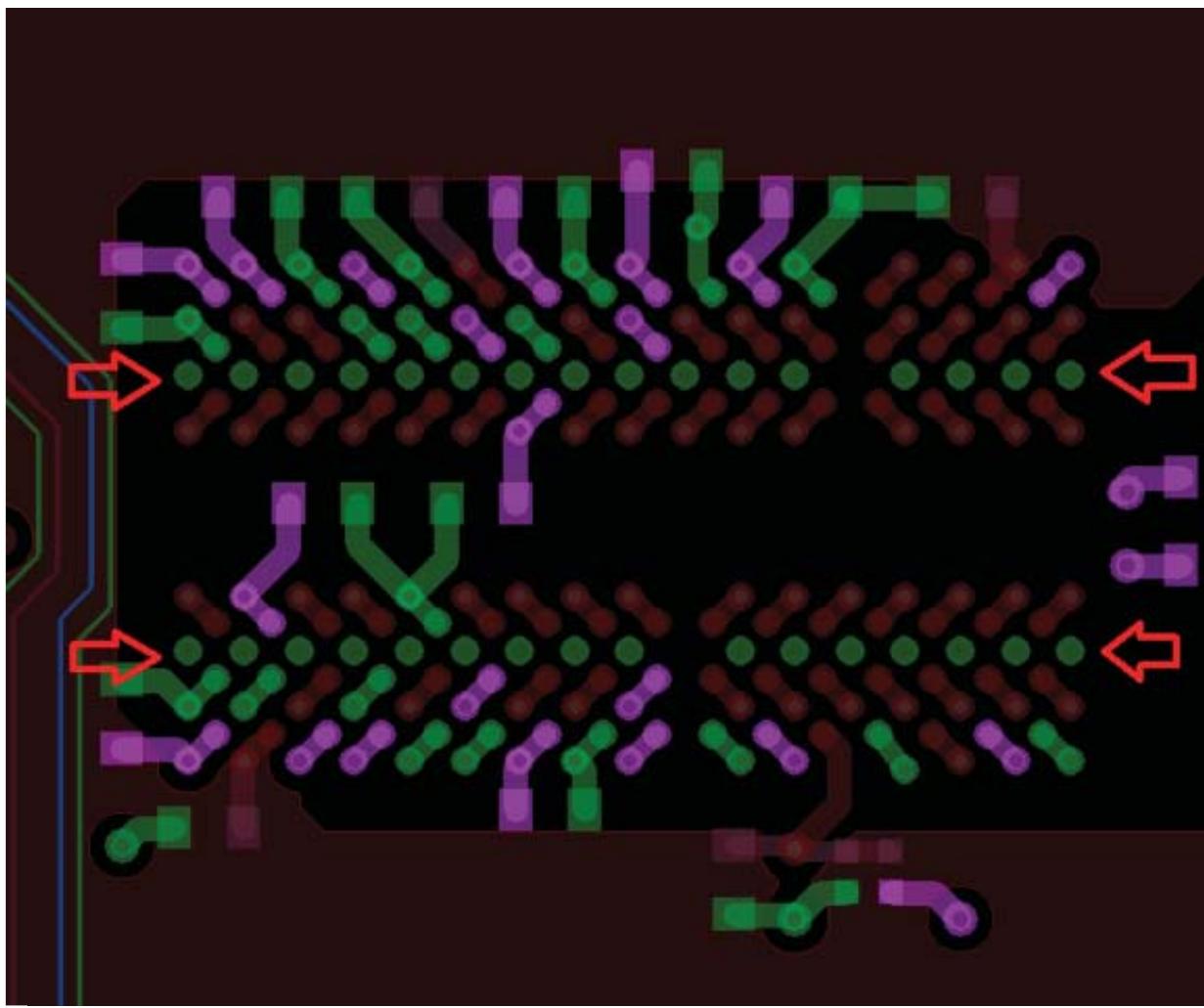
8. Keep the routing at least 30 mils away from the reference plane and void edges with the exception of breakout regions ([Figure A-2](#)).

9. In the breakout region, route signal lines in the middle of the via void aperture. Avoid routing at the edge of via voids ([Figure A-4](#)).



*Figure A-4:*

10. Use chevron-style routing to allow for ground stitch vias (Figure A-5).

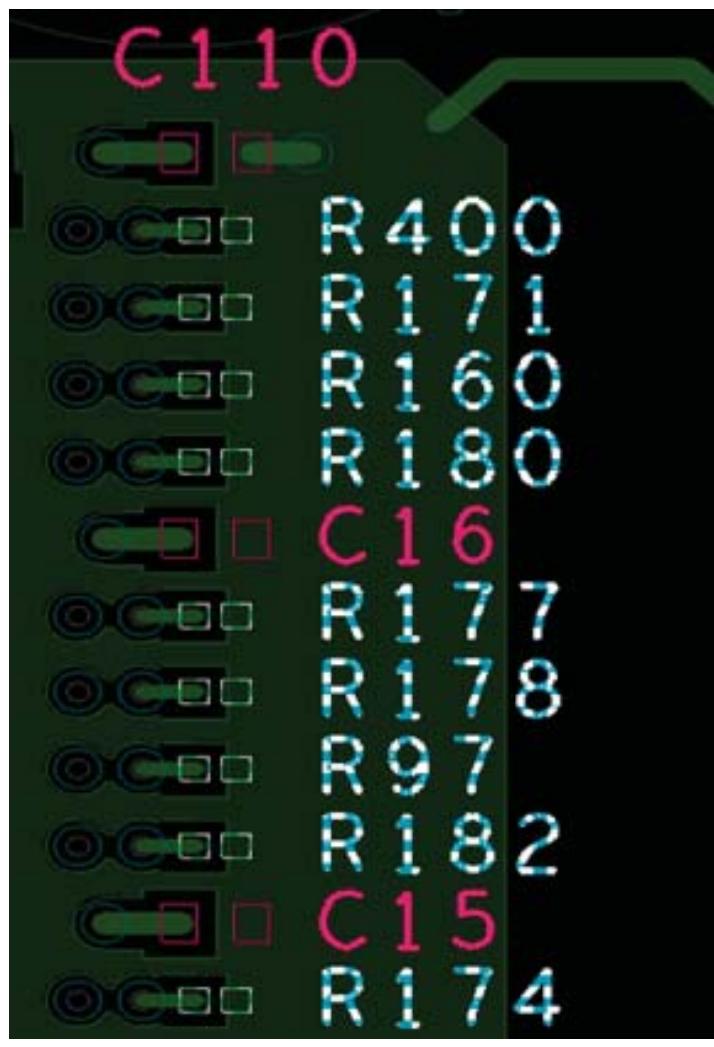


UG583\_c2\_16\_050614

Figure A-5:

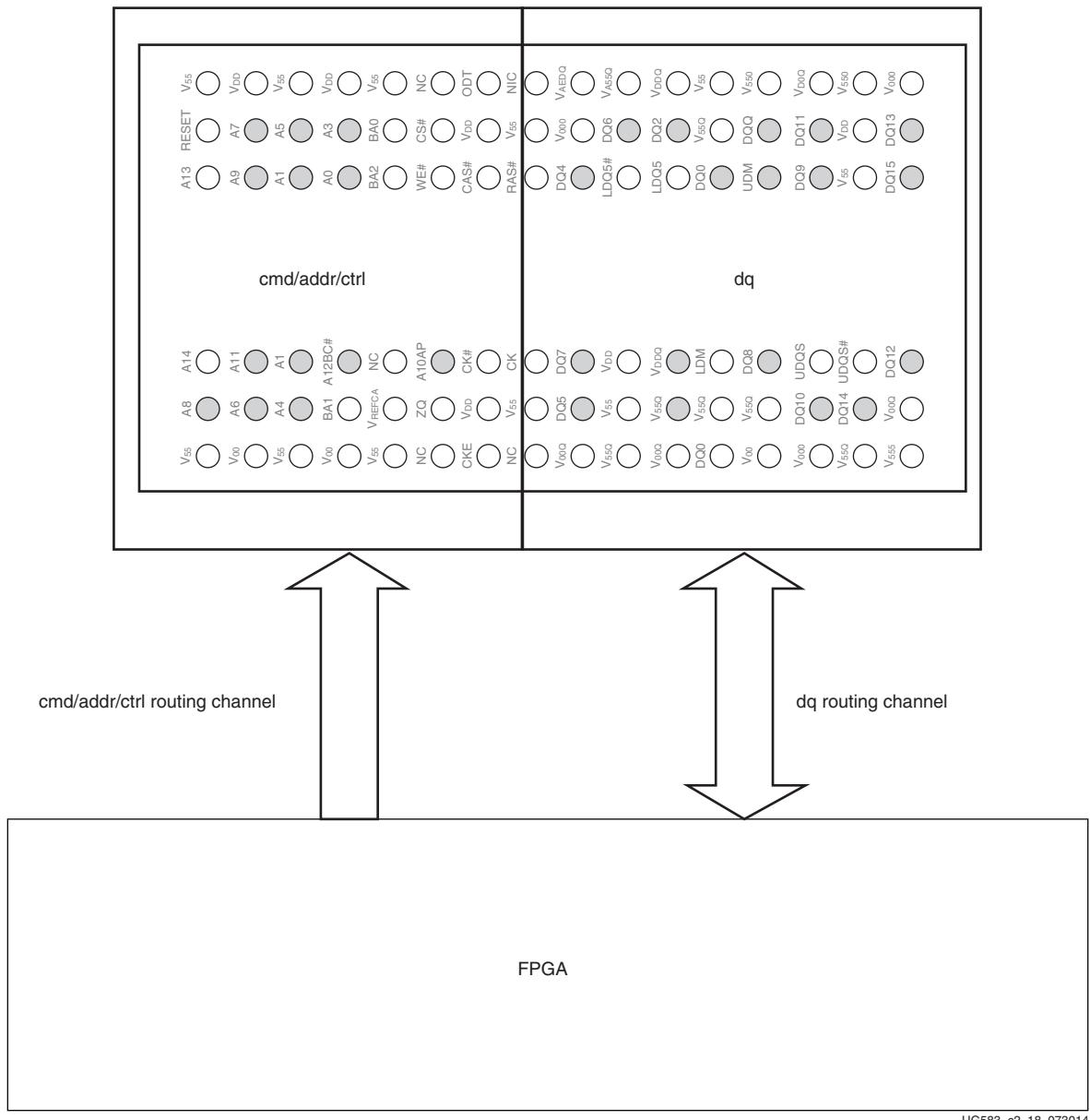
11. Add ground vias as much as possible around the edges of the device and inside the device to make a better ground return path for signals and power, especially corners. Corner or edge balls are generally less populated as grounds.

12. For ADDR/CMD/CTRL V<sub>TT</sub> termination, every four termination resistors should be accompanied by one 1.0 µF capacitor, physically interleaving among resistors, as shown in [Figure A-6](#).



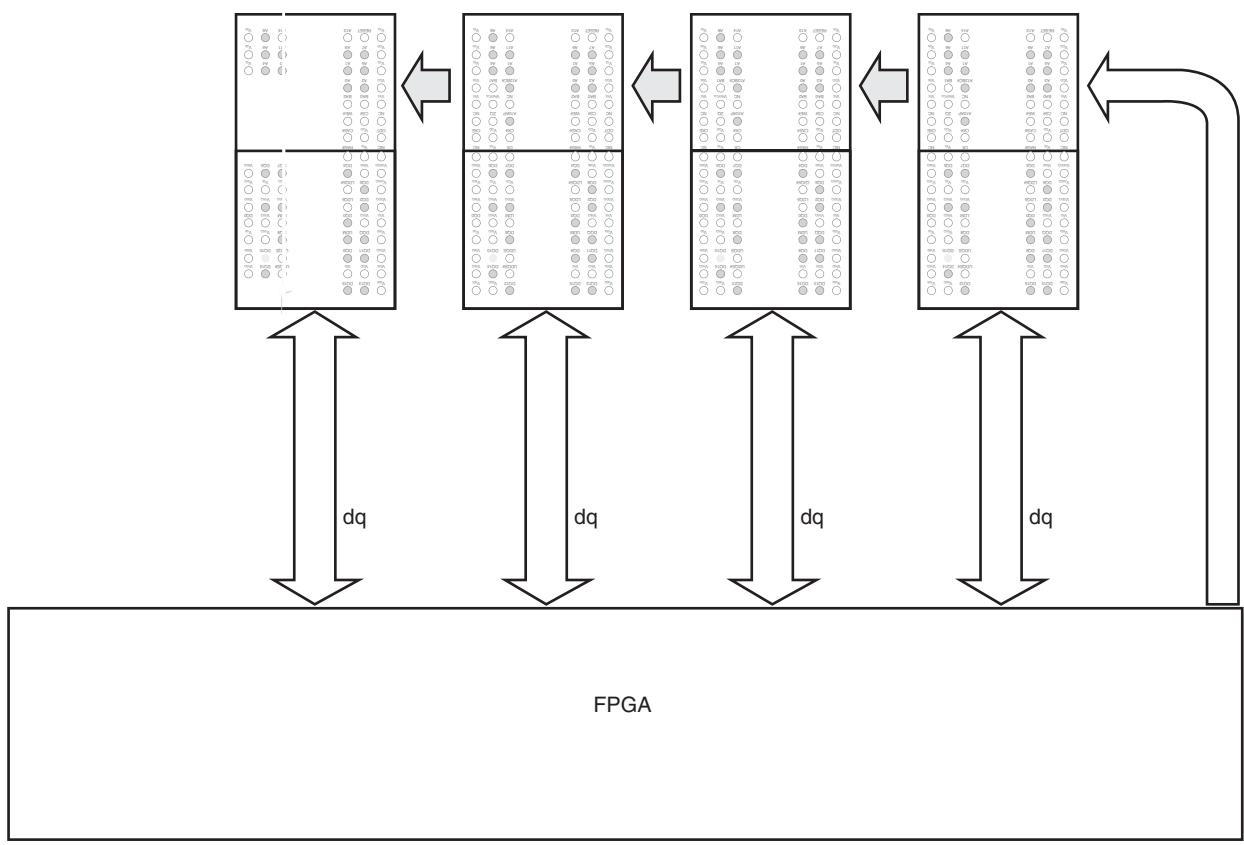
*Figure A-6:*

13. To optimize the signal routing, the recommendation for one component placement is shown in [Figure A-7](#).



*Figure A-7:*

For five components, the recommendation is shown in [Figure A-8](#).


UG583\_c2\_19\_073014

*Figure A-8:*

**Note:** For more information on routing constraints for total length/delay constraints, see the corresponding memory in *UltraScale Architecture PCB Design User Guide* (UG583) [\[Ref 25\]](#).

# Additional Resources and Legal Notices

---

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

---

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

**Note:** For more information on Documentation Navigator, see the [Documentation Navigator](#) page on the Xilinx website.

Unless otherwise noted, IP references are for the product documentation page. These references provide supplemental information useful for this document:

1. *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions Data Sheet* ([DS176](#))
2. *7 Series FPGAs SelectIO™ Resources User Guide* ([UG471](#))
3. *7 Series FPGAs Packaging and Pinout Specification* ([UG475](#))
4. [Arm® AMBA® Specifications](#)
5. *Vivado® Design Suite User Guide: Hierarchical Design* ([UG905](#))
6. *Vivado Design Suite Tutorial: Hierarchical Design* ([UG946](#))
7. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
8. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
9. *Embedded System Tools Reference Manual* ([UG111](#))
10. *7 Series FPGAs Clocking Resources User Guide* ([UG472](#))
11. *XADC Wizard v2.4 LogiCORE IP Product Guide* ([PG091](#))
12. *7 Series FPGAs PCB Design and Pin Planning Guide* ([UG483](#))
13. [7 Series FPGAs Data Sheets](#)
14. *DDR2-533 Memory Design Guide For Two-DIMM Unbuffered Systems*. Micron Technology, Inc., ([TN-47-01](#))
15. *Xilinx Timing Constraints User Guide* ([UG612](#))
16. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
17. *Command Line Tools User Guide*, COMPXLIB ([UG628](#))
18. *Synthesis and Simulation Design Guide* ([UG626](#))
19. [PlanAhead™ Design Analysis tool](#)
20. *Virtex®-5 FPGA ML561 Memory Interfaces Development Board User Guide* ([UG199](#))
21. JESD79-3E, *DDR3 SDRAM Standard*, JEDEC® Solid State Technology Association
22. "Improving DDR SDRAM Efficiency with a Reordering Controller", XCELL Journal Issue 69
23. *AXI Multi-Ported Memory Controller* ([XAPP739](#))
24. *ISE® to Vivado Design Suite Migration Guide* ([UG911](#))
25. *UltraScale™ Architecture PCB Design User Guide* ([UG583](#))

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 Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; 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 such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

#### AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2011–2018 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, 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, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.