Spartan-6 FPGA Memory Interface Solutions

User Guide

UG416 July 25, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/2/09	1.0	Initial Xilinx release.
2/23/10	1.1	Updated Figure 1-30. Revised the text below the Calibrated Input Termination bullet on page 26. Added Xilinx ISim to first paragraph in Functional Simulation, page 44.
3/3/10	1.2	Added note about device migration to page 27.
10/5/10	1.3	Added ARM® AMBA® specifications web link to References. Added paragraph about creating an MCB base memory interface in the EDK environment to the first page of Chapter 1, Getting Started. Added note after step 1 and updated step 5 in Setting up a New Project, page 9. Added information to step 1 in Selecting a Memory Standard. Updated Figure 1-10 and Figure 1-23. Added description of axi_s6_ddrx instances after Figure 1-11. Updated Frequency bullet in Setting Controller Options, page 15. Added information to step 8, updated step 9, and added step 10 and Figure 1-27 in Multi-Port Configuration. Updated first paragraph, step 14, and step 14 of Setting FPGA Options, page 26. Updated and added note after Figure 1-37. Added Table 1-1. Added <component name>/example_design/rtl/mcb_controller, page 33. Updated and added note after Table 1-6. Updated parameter description of CMD_PATTERN in Table 1-11. Added last paragraph (about changing command patterns) to Custom Command Sequences. Added Chapter 2, EDK Flow Details. Changed “ctrl_state[4:0]“ to “ctrl_state[144:0]”, “ctrl_cmd” to “ctrl_cmd[2:0]”, and “cal_state[3:0]” to ”cal_state[144:0]” in Table 3-1. Added “ASCII radix”to descriptions of ctrl_state[144:0] and cal_state[144:0] in Table 3-1.
10/18/10	1.3.1	Revised document file name.

Date	Version	Revision
06/22/11	1.4	Chapter 1: Removed step 14 about calibration memory address and associated figure from Setting FPGA Options, page 26. Revised notes on page 33 and page 34. Revised Functional Simulation, page 44. Chapter 2: Revised Interface Clock, page 60. Added Simulation Considerations, page 63. Updated ui_clk in Table 2-2. Chapter 3: Added Figure 3-3.
10/19/11	1.5	ISE 13.3 tool release. • Removed Preface and added Appendix A, Additional Resources. • Added note to Debug Signals.
01/18/12	1.6	ISE 13.4 tool release. • Changed bus width of addr_mode_i from [1:0] to [2:0] in Table 1-12. • Added CORE Generator Tool with AXI4 Interface Only, page 53.
07/25/12	1.7	ISE 14.2 tool release. Corrected addr_mode_i setting to 011 in step 8, page 52.

Table of Contents

XILINX®

Chapter 1

Getting Started

This document describes the design tool flow and debug procedures for external memory interfaces implemented with the memory controller block (MCB) in Spartan®-6 FPGAs. For more information on the functionality and operation of the MCB, refer to *Spartan-6 FPGA Memory Controller User Guide* [Ref 1].

This chapter describes how to use the MIG tool available in the CORE Generator™ tool or in the Embedded Development Kit (EDK) environment to implement a memory interface based on the memory controller block (MCB) in Spartan-6 FPGAs. It contains these sections:

- MIG Overview
- Supported Tools and System Requirements
- Using the MIG Tool
- MIG Directory Structure and File Descriptions
- MIG Example Design with Traffic Generator

If the MIG tool is invoked from the CORE Generator tool, an MCB design can be configured with either a standard (native) interface or an advanced extensible interface (AXI4). When invoking the MIG tool in the EDK environment, only the AXI4 interface option is supported. For additional details on creating an MCB based memory interface in the EDK environment, see Chapter 2, EDK Flow Details.

MIG Overview

The MIG tool guides the user through a series of steps that define all the necessary attributes required to implement the desired memory interface. In general, these attributes determine the following types of memory interface characteristics:

- Memory device attributes

  Memory type, data width, timing characteristics and other memory behavior
- User (fabric side) Interface configuration

  Number of ports, port type, and port width
- Controller configuration

  Address ordering, arbitration schemes, and other controller behavior

When the required steps are completed, the MIG tool generates RTL code and a user constraints file (UCF) for implementing the desired memory interface. For Spartan-6 devices, the RTL code includes a top-level "wrapper" file that incorporates the MCB hard block and any other FPGA resources required to implement the requested memory solution. When invoked from the CORE Generator tool, the MIG tool also produces the

necessary script files for simulation and design implementation (synthesis, map, par) of the interface.

Finally, the MIG tool produces a "Traffic Generator" synthesizable test bench to verify or demonstrate the MCB based memory interface in a simulation or hardware environment (CORE Generator tool native interface only). The bitstream created from this example design flow can be targeted to a Spartan-6 FPGA SP601 or SP605 hardware evaluation board to demonstrate DDR2 or DDR3 interfaces, respectively. The example design can also be targeted to any hardware environment with an MCB based memory interface. See MIG Example Design with Traffic Generator, page 37 for more information.

Supported Tools and System Requirements

Operating System Requirements

For a list of system requirements, see the Xilinx Design Tools: Release Notes Guide.

Tools

- ISE® Design Suite, version 14.2

Using the MIG Tool

This section provides a walkthrough of the MIG GUI, detailing all necessary steps for implementing a Spartan-6 FPGA memory interface based on the MCB.

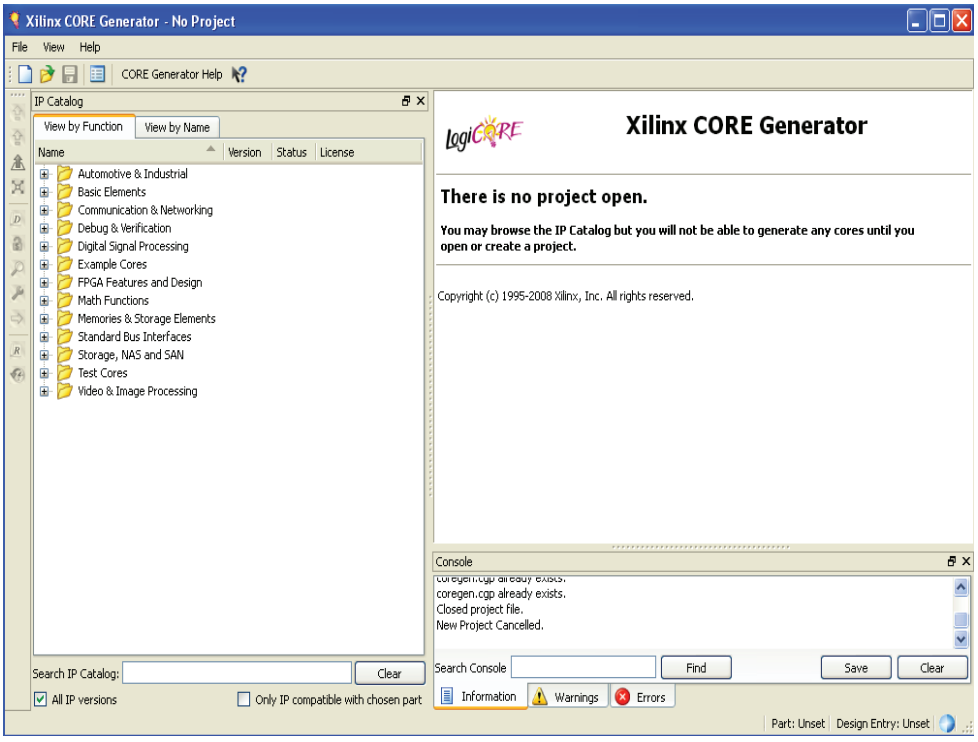
Note: The screen captures in this chapter are conceptual representatives of their subjects and provide general information only. For the latest information, see the MIG GUI tool. The exact behavior of the MIG tool and the appearance of some pages might differ depending on what type of user interface is selected and whether the MIG tool is invoked from the CORE Generator tool or from XPS (EDK). These differences are noted where appropriate.

Setting up a New Project

These steps set up a new CORE Generator tool project in preparation for launching the MIG tool:

1. The CORE Generator system is launched by selecting **Start** > **Xilinx ISE Design Suite *<Release #>*** > **ISE** > **Accessories** > **CORE Generator**.

Note: To invoke the MIG tool from XPS, select **Memory and Memory Controller** > **AXI S6 Memory Controller** from the XPS IP catalog (when adding new IP to the system) or right-click the **axi_s6_ddrx** component in the XPS System Assembly View and select **Configure IP...**, then skip to Creating an MCB Design, page 14.



UG416_c1_01_091409

Figure 1-1: **CORE Generator Tool**

2. Choose **File** > **New Project** to open a New Project dialog box.

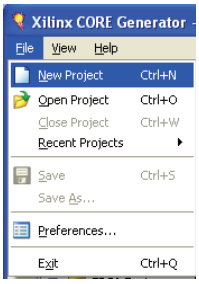


Figure 1-2: **Create a New CORE Generator Tool Project**

3. Enter a project name (for example, **`Spartan6_MIG_Example_Design`**) and location. Then click **Save**.

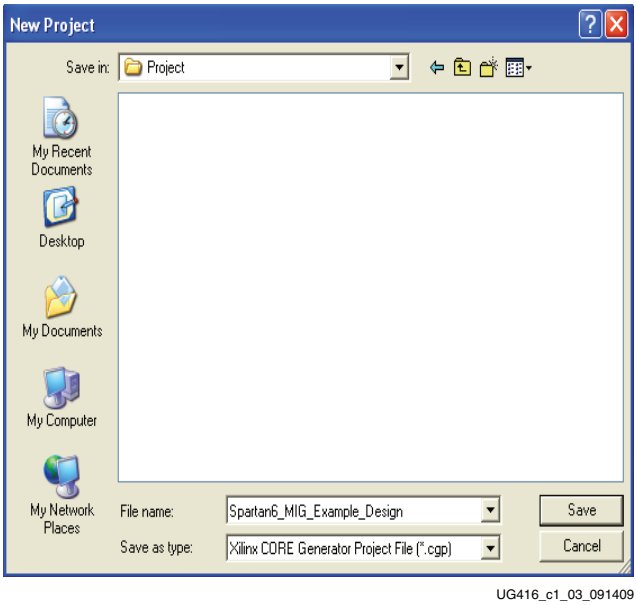


Figure 1-3: **Naming the New Project**

4. Select the Spartan-6 family, and then the target device, the package, and the speed grade (for example, **xc6slx16**, **cs324**, **-2**) on the Project Options page. Click on **Generation** in the left pane to access the design flow options.

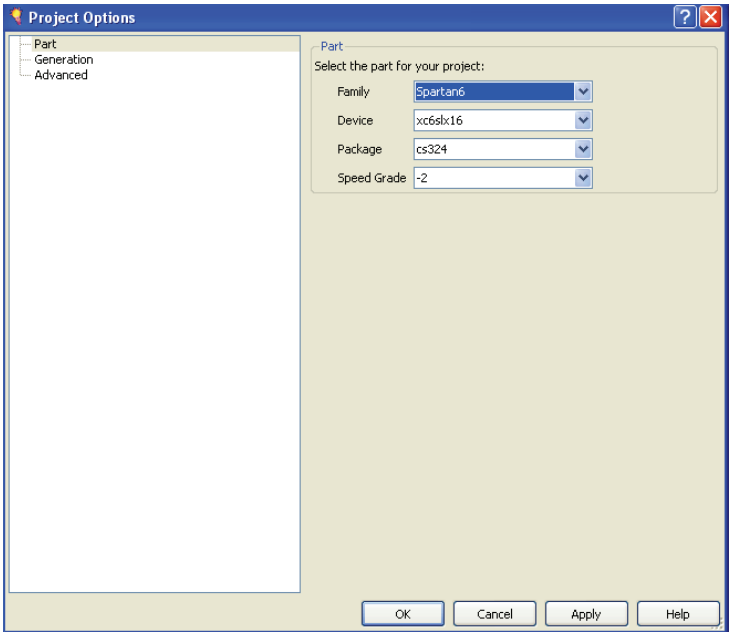


Figure 1-4: **Device Selection**

5. Select **Verilog** or **VHDL** as the Design Entry option and **ISE** for Vendor Flow Settings. Click **OK** to finish the Project Options setup.

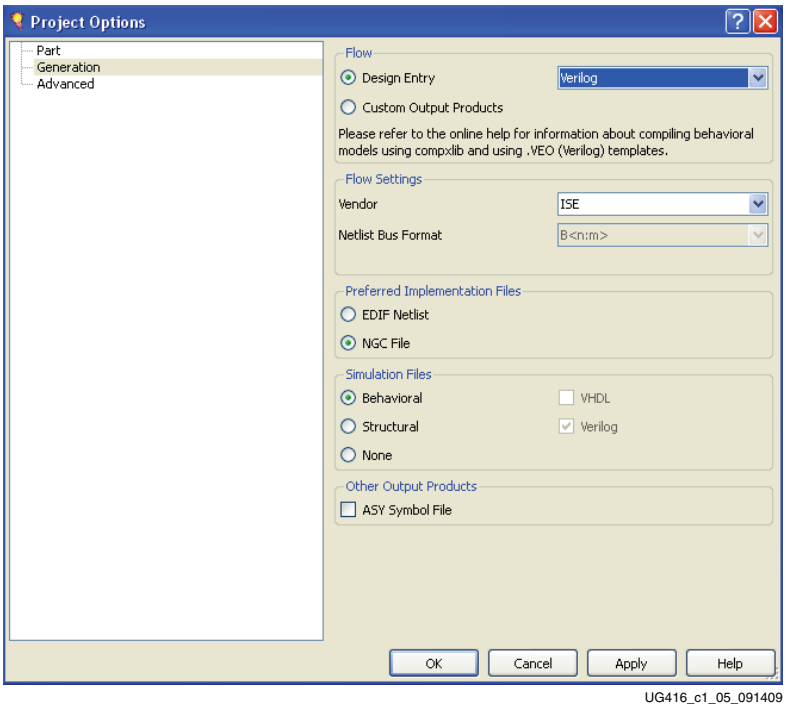
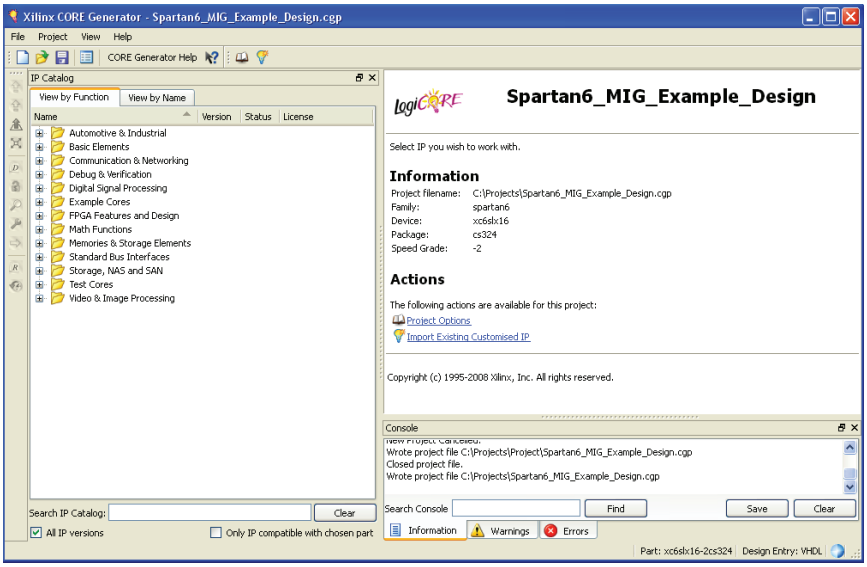


Figure 1-5: **Setting Up the Design Flow Options**

Launching MIG

The steps in this section launch the MIG tool in preparation for creating the desired memory interface based on the MCB:

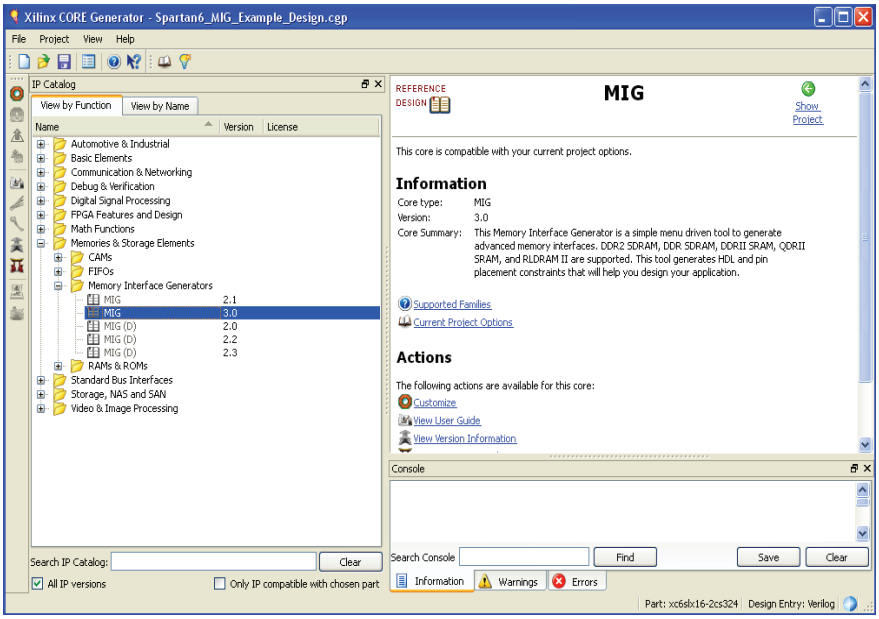
1. Review the project settings summarized in the Information section of the right window to make sure they are correct.



UG416_c1_06_091409

Figure 1-6: **Project Page**

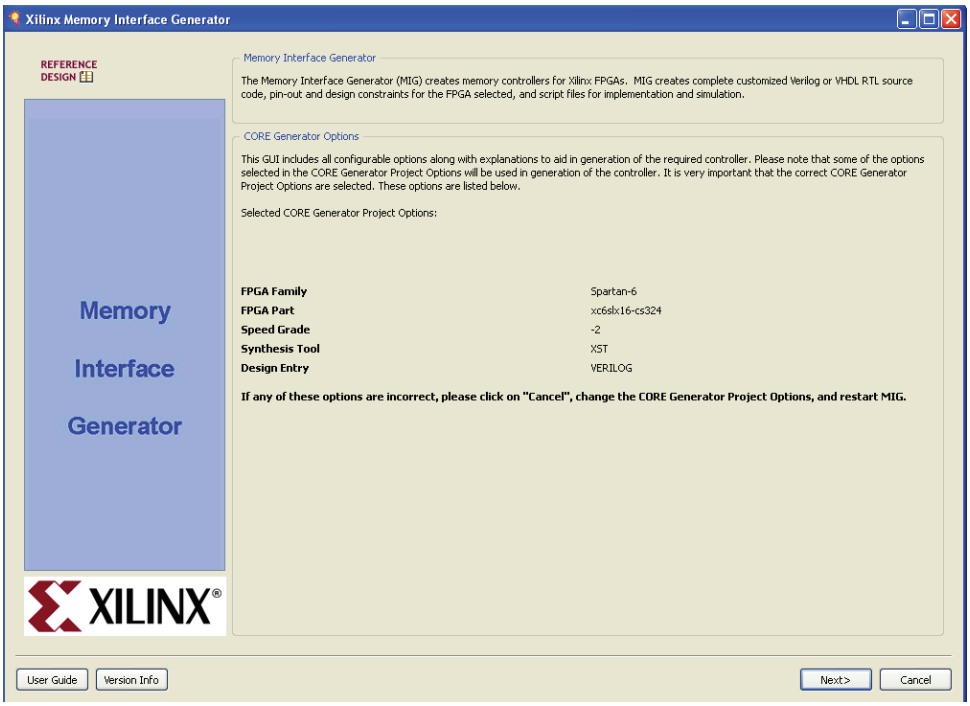
2. Expand the `Memories & Storage Elements` folder in the left window to access the MIG tool. Double-click the latest version under **Memories & Storage Elements** > **Memory Interface Generators** > **MIG** to launch the MIG tool.



UG416_c1_07_091409

Figure 1-7: **Launching the MIG Tool**

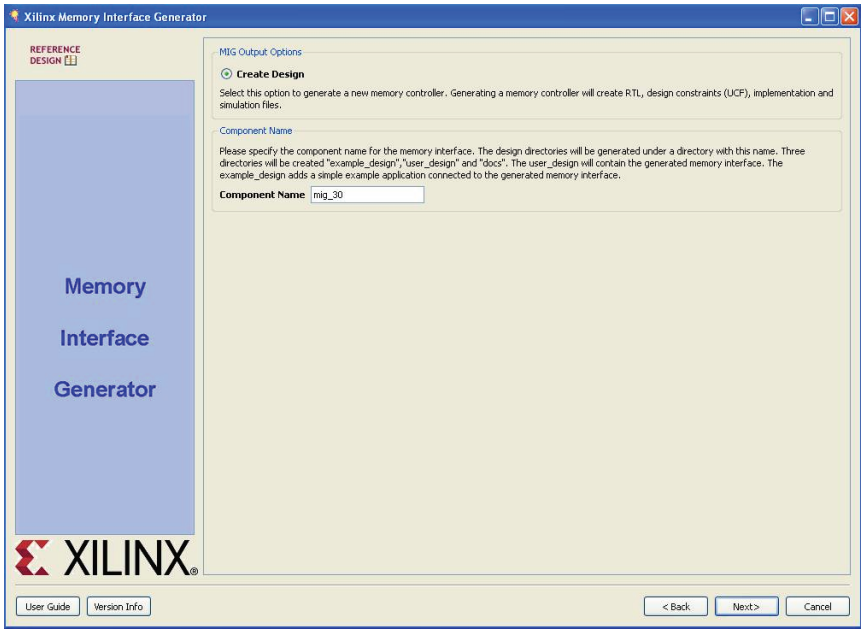
3. When the MIG tool startup page appears, click **Next** to advance to the MIG Output Options page.



UG416_c1_08_091409

Figure 1-8: **MIG Tool Startup Page**

4. Select **Create Design** to create a new MCB based memory interface. Enter a name for the memory interface in the Component Name field. Click **Next** to begin defining the MCB based memory interface.



UG416_c1_09_091409

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

The output files and directories generated by the MIG tool are placed in a folder named `<Component Name>`.

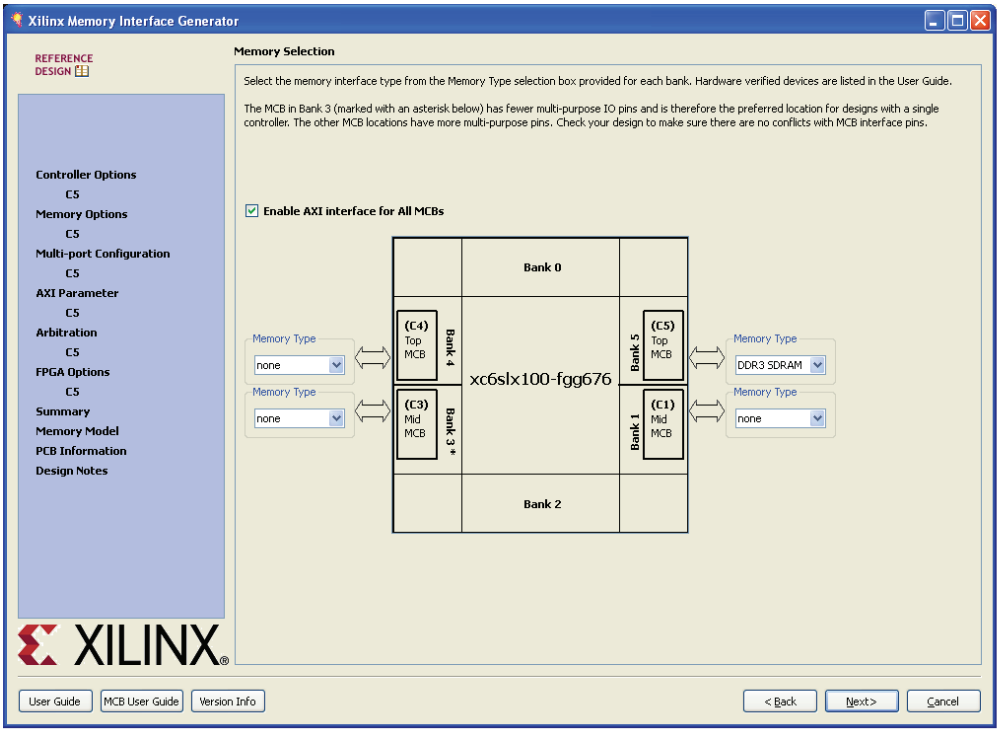
Note: <Component Name> cannot contain special characters. Only alphanumeric characters can be used to specify a component name, and the name should always start with a letter of the alphabet (a-z).

Creating an MCB Design

This section details the steps required to customize and generate an MCB based memory interface using the MIG tool.

Selecting a Memory Standard

The Memory Selection page is used to choose a supported memory standard (DDR SDRAM, DDR2 SDRAM, DDR3 SDRAM, LPDDR) for one or more of the memory controller blocks. Spartan-6 devices contain either two or four MCBs, depending on the size of the device. In general, the left or lower left memory controller block should be the first choice when implementing a single memory interface. The predefined MCB I/O locations for this core have fewer multi-function pins (for example, not shared with configuration related pins). In addition, this core location is given priority for supporting migration between different Spartan-6 devices in the same package type.



UG416_c1_10_081010

Figure 1-10: **Memory Selection Page**

When working with a Spartan-6 device that contains four MCBs, the two MCBs on the same side of the device must share the pair of system clocks generated from one of the on-chip PLL blocks. Therefore, the data rates of the memory interfaces implemented by these two MCBs must be the same. See the "Clocking" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for more details on MCB clocking requirements.

1. Click the Enable AXI check box to add an AXI4 user interface to all MCBs (this option is only available for Verilog designs). Otherwise, the MIG design is generated with the standard (or Native) user interface by default. Select a memory standard from the Memory Type drop-down menu for each MCB that implements a memory interface (only DDR2 and DDR3 are available when using an AXI4 interface). Click **Next**.

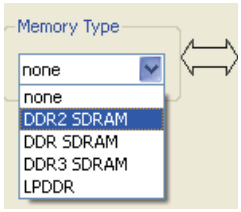


Figure 1-11: **Memory Standard Selection**

When invoked from XPS (EDK), only one MCB site can be active per axi_s6_ddrx instance. Multiple controllers are implemented as multiple separate instances of axi_s6_ddrx in XPS.

Setting Controller Options

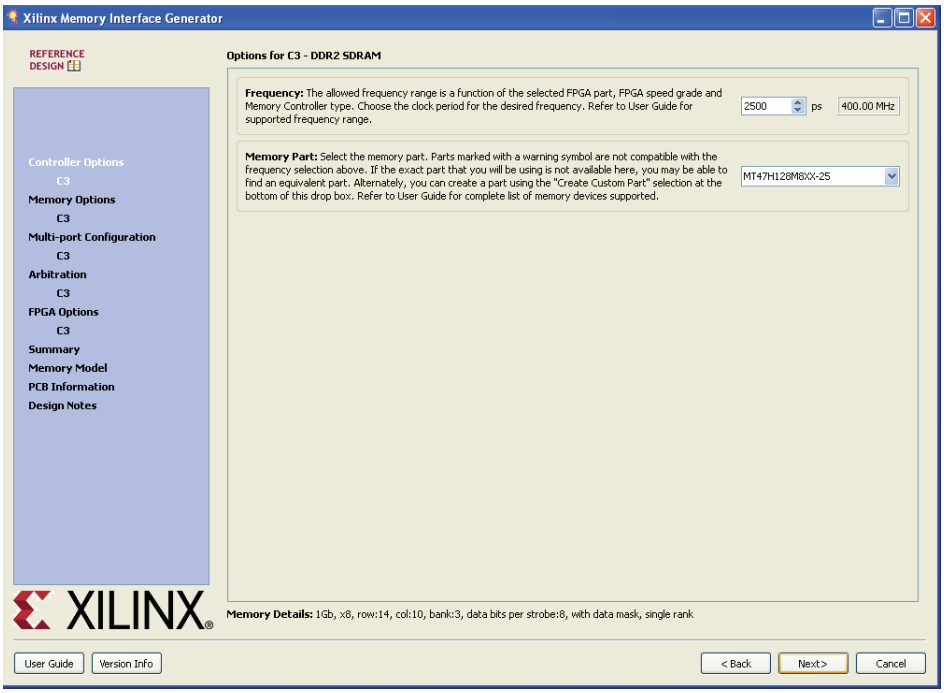
The Controller Options page is used to define some general characteristics of the memory interface. If the design has multiple controllers, this page is repeated for each controller. These characteristics of the memory solution can be defined on this page:

- Frequency

  This option indicates the operating frequency of the controller (equivalent to the memory clock frequency, for example, 400 MHz for an 800 Mb/s DDR interface). The frequency specified here should be half of the clock rate coming from the BUFPLL driver to the MCB (see the "Clocking" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for more details on the clocking requirements for the MCB). The controller frequency is limited by factors such as the selected FPGA device and speed grade. In the EDK flow, an extra check box (selected by default) allows the user to specify that the frequency information should be calculated automatically from EDK.

- Memory Part

  This option selects a target memory device for the design. The selection can be made from the existing list of supported devices or a new custom device can be created.



UG416_c1_12_102709

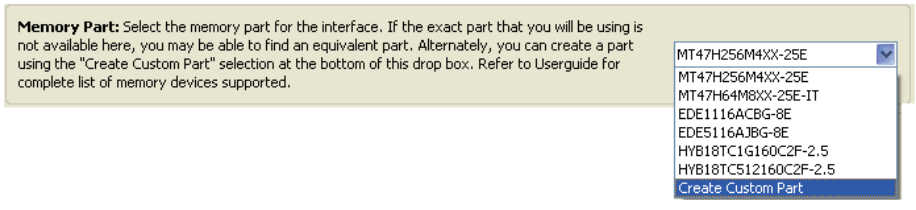
Figure 1-12: **Controller Options Page**

2. Enter the appropriate value for the Frequency selection by using the arrows on the spin box or directly typing in the value. Values entered are restricted based on the minimum and maximum frequencies supported.

UG416_c1_13_091409

Figure 1-13: **Entering Memory Interface Frequency**

3. Select the desired Memory Part from the drop-down menu (see Figure 1-14). If the required part or its equivalent is unavailable, a new memory part can be created. To create a custom part, select the **Create Custom Part** option from the list, which causes a new window to appear as shown in Figure 1-15. If not creating a custom part, skip to step 5.



UG416_c1_14_091409

Figure 1-14: **Selecting a Memory Device**

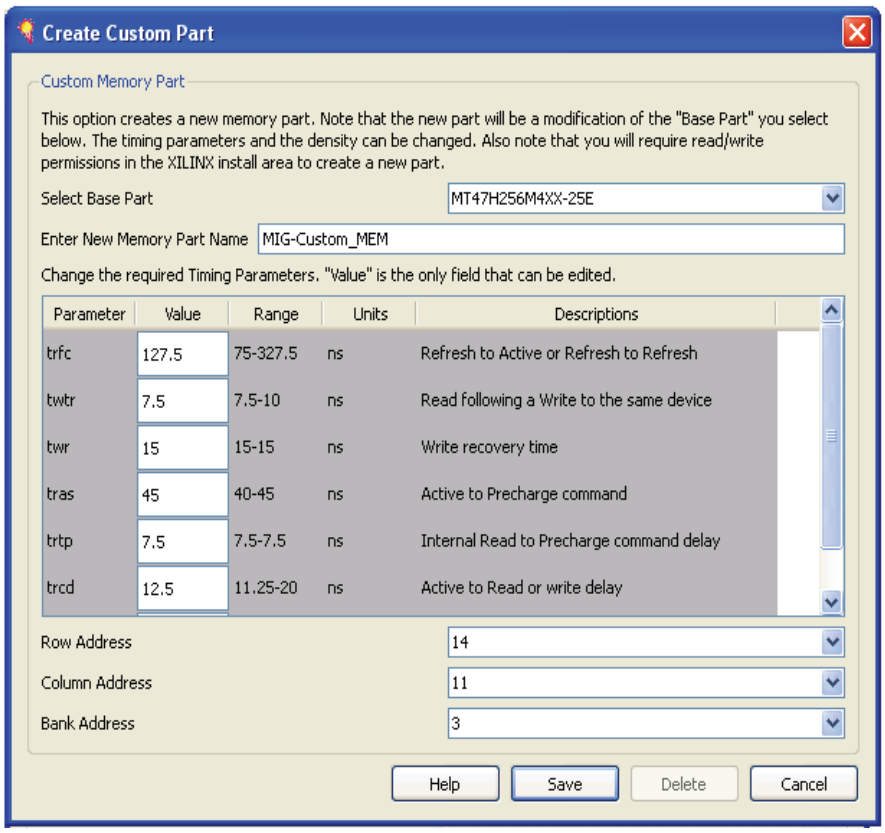


Figure 1-15: **Creating a Custom Memory Device**

The Create Custom Part window allows a new memory device to be defined by modifying parameters of an existing part from the list of supported devices.

4. To create a custom memory device:
   a. Select a supported device in the Select Base Part drop-down menu that has similar parameters to the device being created.
   b. Enter a name for the new device in the Enter New Memory Part Name box (for example, **MIG-Custom_MEM**).
   c. Edit the timing parameter values and row/column/bank address bit counts as needed.
   d. Click **Save** to add the new device to the list of supported devices as shown in Figure 1-16.

   This new device is saved in the local MIG database for future use.

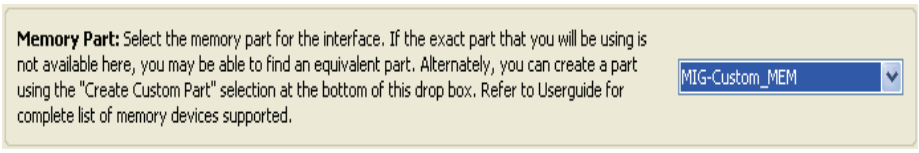


Figure 1-16: **Selecting a Custom Memory Device from the Memory Part List**

5. Select or unselect the Data Mask check box (see Figure 1-17) to determine whether or not the generated MIG design includes data mask pins. The bottom of the Controller Options page displays the details of the selected memory configuration as shown in Figure 1-18. Click **Next** to continue to the Memory Options page.

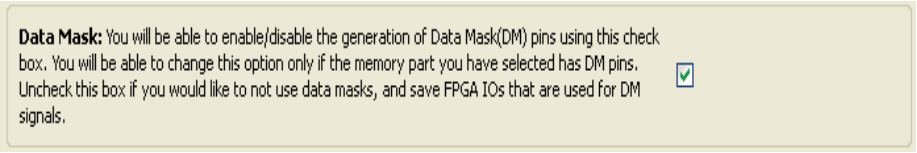


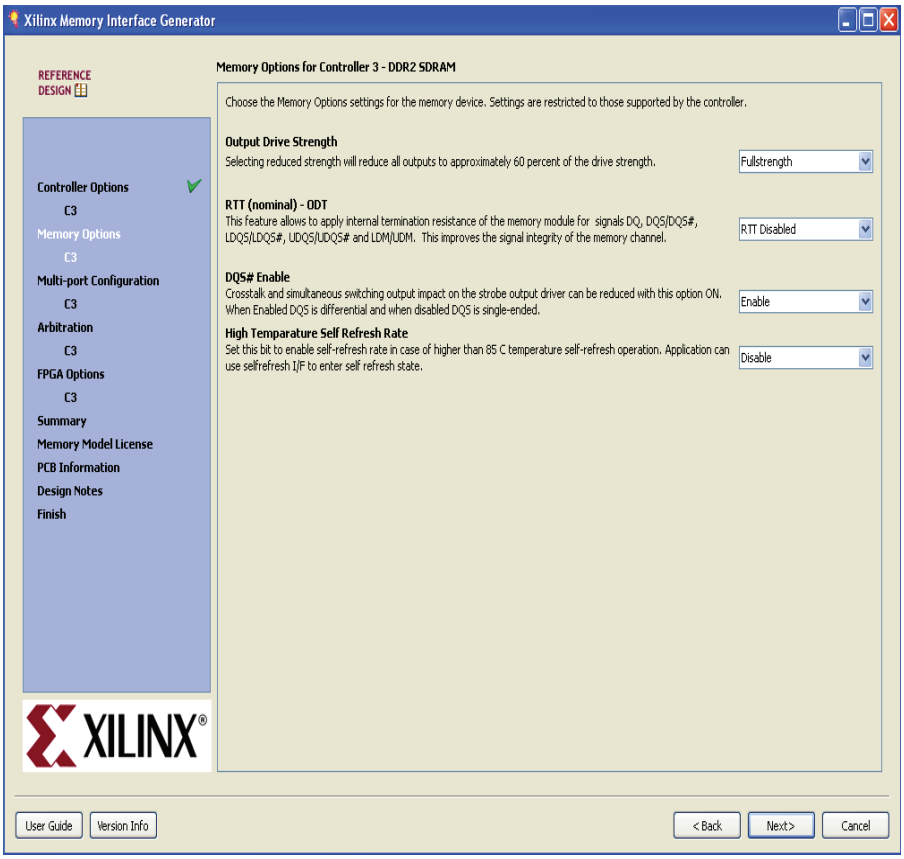
Figure 1-17: **Setting the Data Mask Option**

Figure 1-18: **Memory Configuration Details**

Setting Memory Device Options

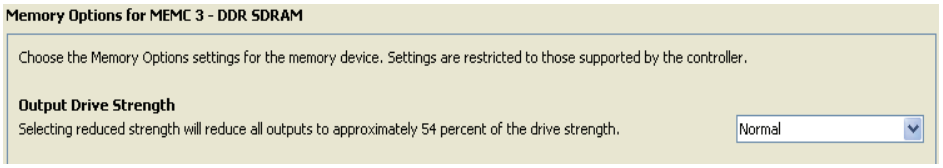
The Memory Options page is used to set up various mode register values that are loaded into the memory device during initialization. The list of available settings is determined by the memory standard selected for the interface. Burst length and CAS latency are automatically set by the MIG tool to offer the best performance.

Figure 1-19 shows the available settings for the DDR2 controller as an example. Figure 1-20 through Figure 1-22 show how these settings change for DDR, DDR3, and LPDDR controllers, respectively.



UG416_c1_19_091409

Figure 1-19: **Mode Register Settings for the DDR2 SDRAM Controller**



UG416_c1_20_091409

Figure 1-20: **Mode Register Settings for the DDR Controller**

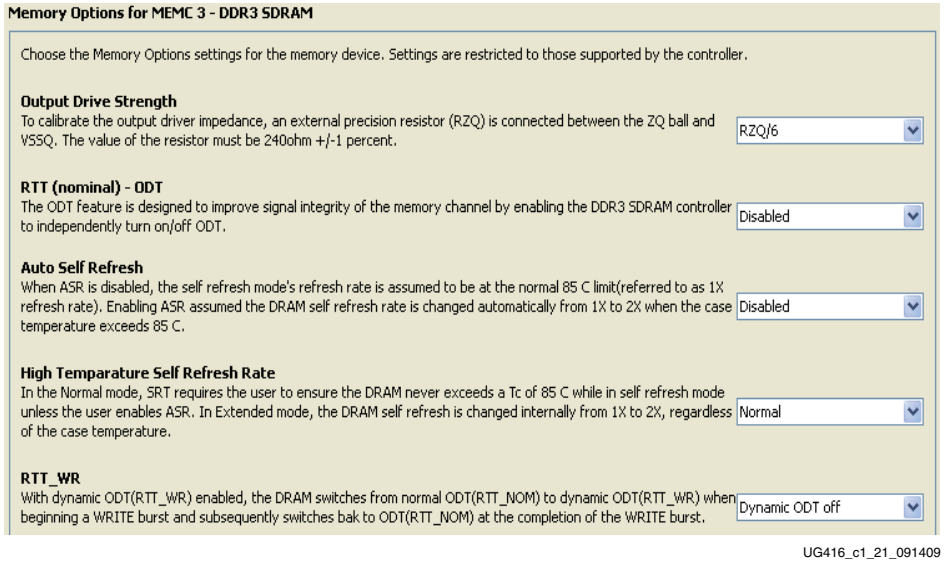


Figure 1-21: **Mode Register Settings for the DDR3 Controller**

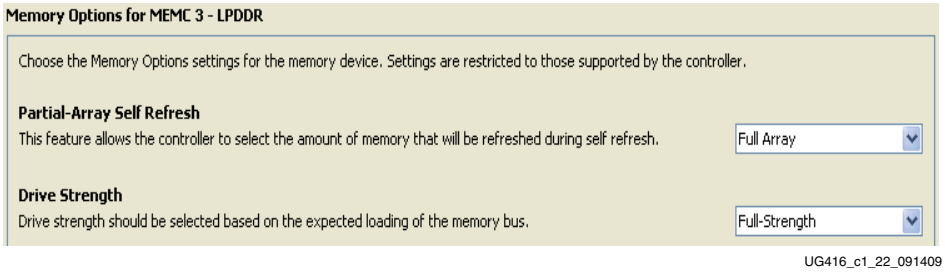
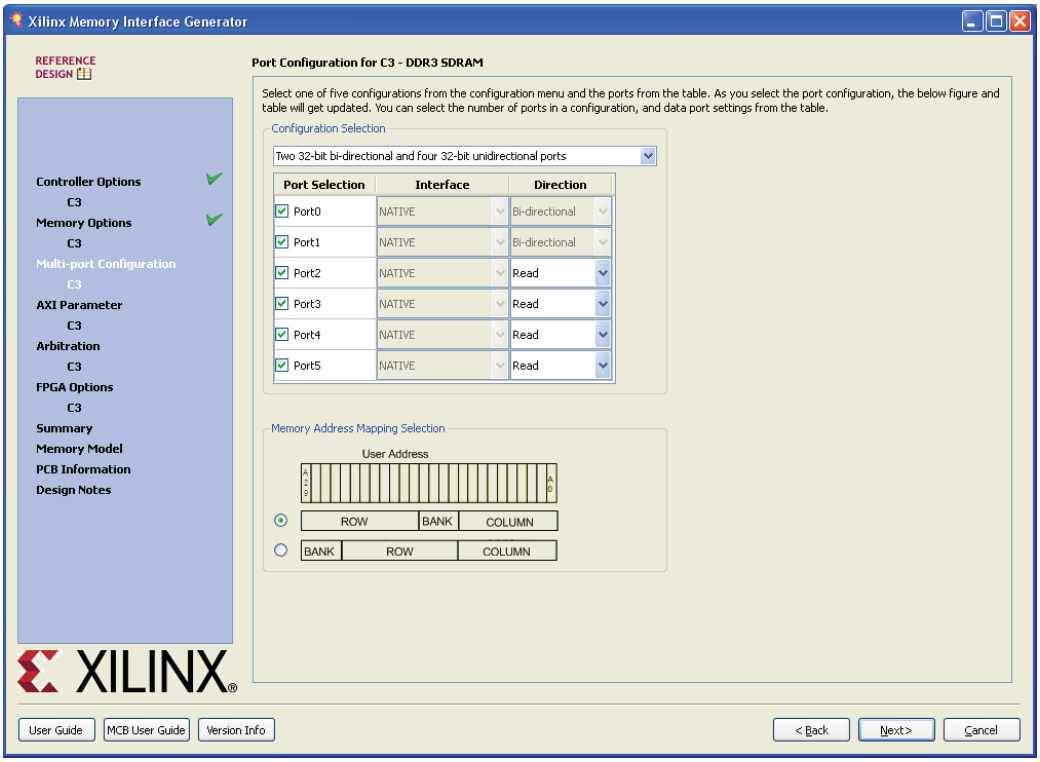


Figure 1-22: **Mode Register Settings for the LPDDR Controller**

6. Select the desired mode register setting for each entry. Then click **Next** to advance to multi-port configuration setup.

Multi-Port Configuration

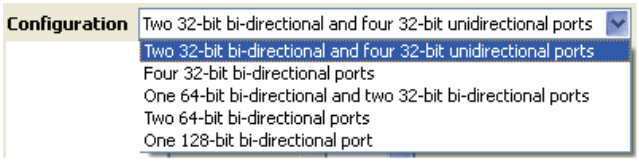
There are five possible port configurations for the MCB User Interface to the FPGA logic. For details on port configurations, see the "Port Configurations" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1]. Based on the selected configuration, the MIG tool generates the necessary user signal names and assignments in the top-level wrapper file.



UG416_c1_23_081010

Figure 1-23: **Port Configuration Page**

7. Select the desired port configuration from the Configuration Selection drop-down menu as shown in Figure 1-24. The port selection table is updated based on the chosen configuration.



UG416_c1_24_091409

Figure 1-24: **Selecting a Port Configuration**

8. Select the check box in front of each port used in the design under the Port Selection column.
   - Unchecked ports are disabled.
   - For unidirectional ports, the Direction column must be set to either Read or Write.
   - The Interface column indicates whether all ports are implemented with the Native (standard) or AXI4 user interface based on the earlier GUI selection. Selecting Native interface results in a standard MCB user interface for the port. Selecting AXI interface results in the addition of an AXI4 memory mapped slave

bridge to the native interface (the AXI4 interface is the only option in the EDK flow).

For more information on the AXI4 slave interface, see Chapter 2, EDK Flow Details.

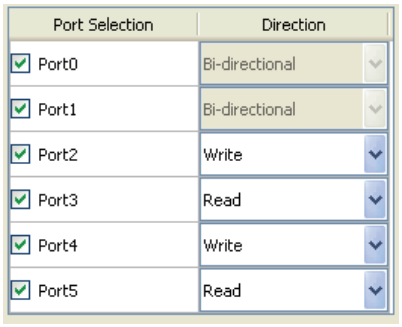


Figure 1-25: **Setting Port Selection and Direction**

The mapping between the User Interface address bus and the physical memory row, bank, and column is configurable. Depending on how the application data is organized, it might be desirable to change the addressing scheme to optimize controller efficiency. For example, in Row-Bank-Column addressing, when requesting a long burst transaction that extends beyond the end of an open row, the controller can open up a new bank to continue the burst and thereby avoid the penalty (efficiency loss) of closing the open row (precharge command) and issuing another row activate command in the same bank.

9. Select the Memory Address Mapping that works best for the application as shown in Figure 1-26. The User Configuration Interface diagram on the right of the page (not available for AXI4 interfaces) now shows a summary of the completed multi-port configuration (see Figure 1-27). Click **Next** to proceed.

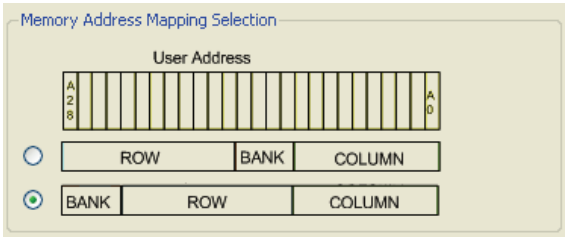


Figure 1-26: **Selecting the Memory Address Mapping Scheme**

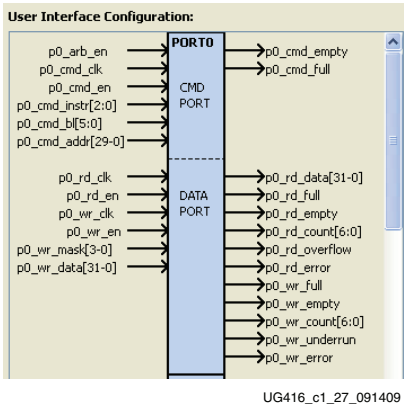


Figure 1-27: **User Interface Port Configuration**

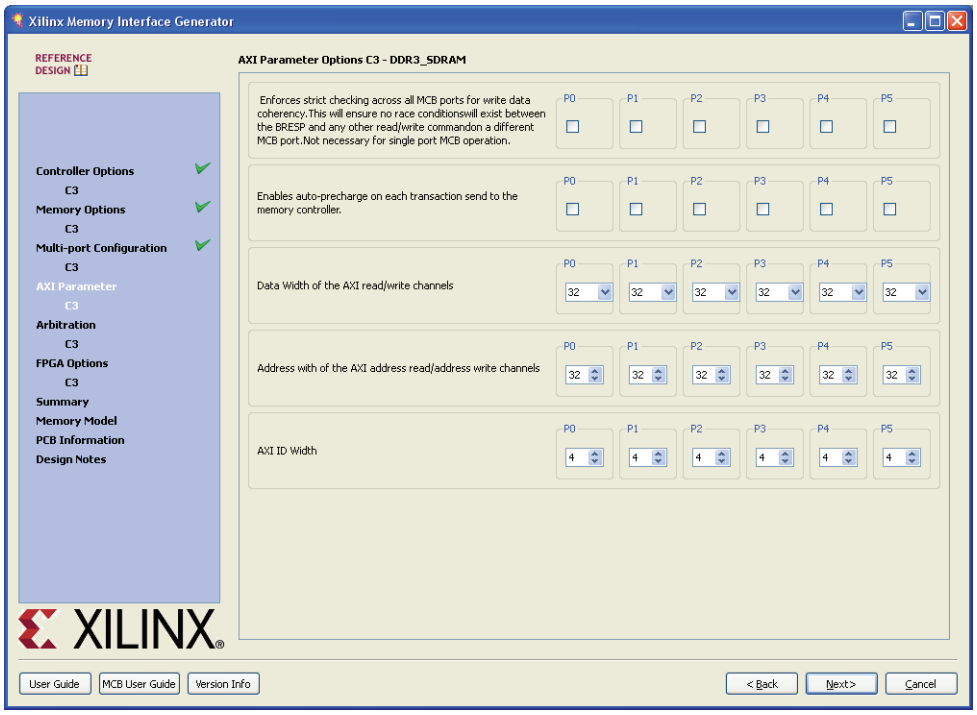
For designs using the AXI user interface option, a page appears to set the parameters for each AXI port (see Figure 1-28).

This feature allows the selection of AXI parameters for the controller. These are standard AXI parameters or parameters specific to the AXI4 interface. Details are available in the ARM® AMBA® specifications. [Ref 10]

Interconnect parameters are also available in the EDK flow. Details on the interconnect parameters are available in the EDK documentation. Some of these parameter options might exist in the EDK flow only:

- **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** (EDK flow only): 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** (EDK flow only): 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.
- **Enforce Strict of Write Coherency Across All Ports**: Sets whether BRESP (AXI Write Response) should be provided early when the transaction completes on that port (unchecked) or only after the transaction is known to have completed across all ports (checked).
- **Enable Auto-Precharge On Each Transaction**: When checked, all MCB read/write transactions on that port close the bank/page to be closed (with auto-precharge) after the transaction completes.

10. Choose whether to enforce strict checking for write coherency on each port. This is recommended for designs with more than one port. Choose whether to enable auto-precharge for each AXI port. Select the Data, Address, and AXI ID width for each port (see Figure 1-28). Click **Next**.



UG416_c1_45_081010

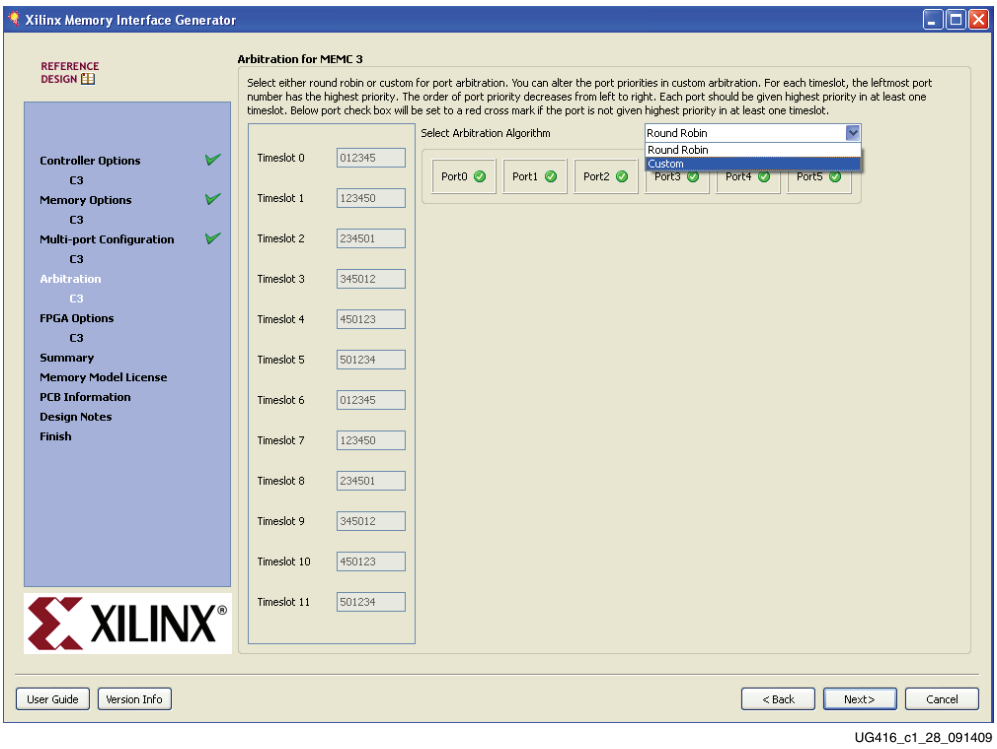
Figure 1-28: **Selecting AXI User Interface Parameters**

Arbitration Table Programming

The MIG tool uses a round-robin arbitration scheme by default. However, a custom arbitration scheme can also be defined. The port priority decreases from left to right in the time-slot entry boxes.

When using the custom option, care should be taken to make sure that all ports have some access to the memory device. In general, each port should receive the highest priority in at least one time slot. The MIG tool generates a RED warning indicator when this guideline is not followed, but it does not prevent such schemes. See the "Arbitration" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for more information on arbitration.

11. Select either **Round Robin** or **Custom** in the Select Arbitration Algorithm drop-down menu. If **Custom** is selected, enter the preferred port priority for each time slot. Then click **Next**.

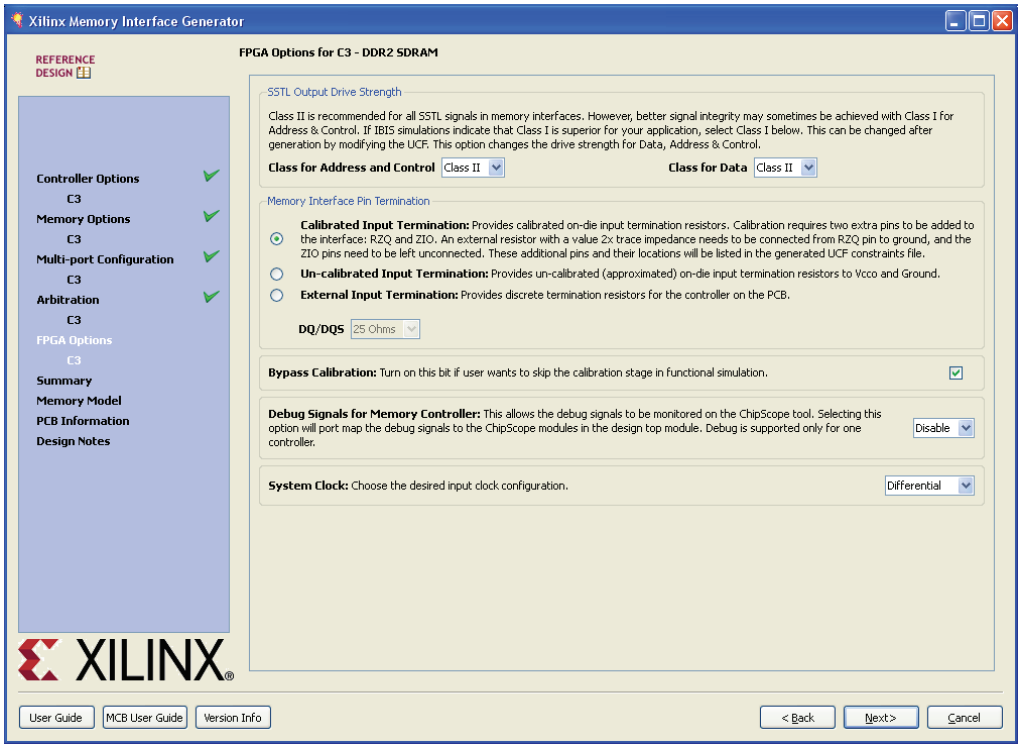


UG416_c1_28_091409

Figure 1-29: **Setting Up the Arbitration Scheme**

Setting FPGA Options

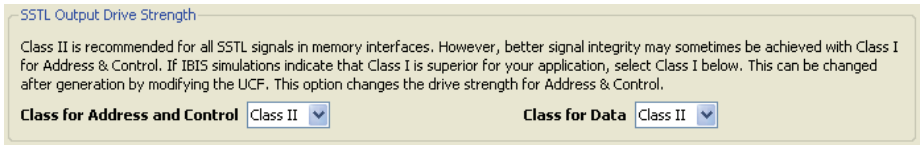
The FPGA Options page is used to configure some remaining aspects of the memory interface solution. In the EDK flow, this is the last page. Clicking the **Finish** button saves the changes and returns the user to XPS.



UG416_c1_29_021710

Figure 1-30: **FPGA Options Page**

12. Set the FPGA output driver strength to either SSTL **Class I** or **Class II** for both Address and Control and Data pins. Class II is recommended by default, but Class I can provide better signal integrity for some applications.



UG416_c1_30_091409

Figure 1-31: **Setting FPGA Output Driver Strength**

The MIG tool provides multiple options for FPGA input termination. The two basic forms of input termination offered are:

- Calibrated Input Termination

  This option uses the Soft Calibration module that is automatically generated by the MIG (or EDK) tool to match the input impedance of the memory interface pins to an external resistor value. Calibrated input termination provides for an on-chip, precisely calibrated termination, resulting in superior signal integrity and reduced component count compared to the other available termination options.

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

The ZIO pin is only required for designs using calibrated input termination and must be a no-connect pin (for example, not connected to any PCB trace) assigned to a valid package pin (for example, bonded I/O) location within the MCB bank. The default locations of the RZQ and ZIO pins can be found in the UCF constraints files. See the "Calibration" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for more details on using calibrated input termination.

Note: If device migration is desired (that is, migrating between device sizes in the same package type), the designer must verify that the chosen RZQ and ZIO locations are available (bonded out) as User I/Os for all planned devices. These pins can be relocated as necessary by modifying the UCF.

- Uncalibrated Input Termination

  This option provides two means of input termination:

  - The FPGA can create an "approximated" on-die input termination value of 25Ω, 50Ω, or 75Ω, as selected from the DQ/DQS and Address/Control drop-down boxes.

    or

  - These settings can be left as "none" for situations where external termination resistors are provided.

  In either case, no static or active calibration takes place to optimize the termination values.

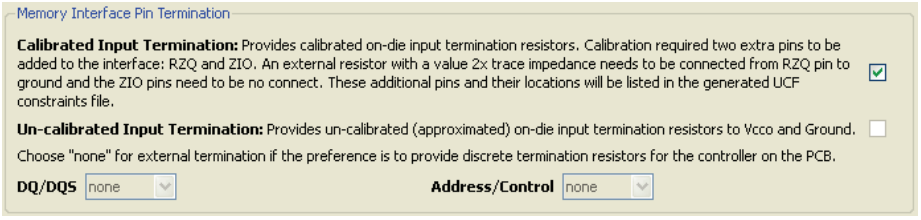
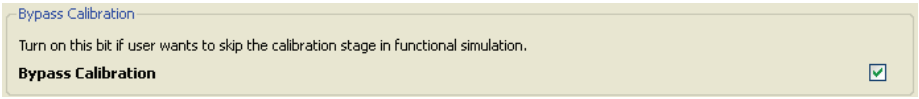


Figure 1-32: **Selecting the Method of Input Termination**

13. Select either Calibrated or Uncalibrated input termination with the appropriate check box. If uncalibrated termination is selected, make the desired selections from the DQ/DQS drop-down menu.

The MCB provides a user interface to allow the initial DQ and DQS calibration process to be retriggered (see the "Calibration" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for details on DQ/DQS calibration). This is especially important for applications that require suspend mode operation. The MIG tool allows a user-specified Calibration Memory Address to be reserved to avoid overwriting user application data during a recalibration process. A training pattern is written to the specified location when a recalibration is requested, and the MIG tool verifies the calibration address space to ensure that it does not cross a row boundary, an additional safeguard to protect the user application data.

14. Check the Bypass Calibration box to skip initial calibration during functional simulation (not shown in the EDK flow).

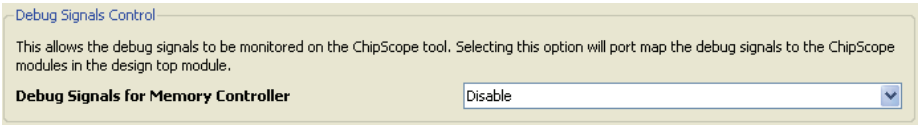


UG416_c1_33_091409

Figure 1-33: **Bypassing Calibration During Functional Simulation**

The MIG tool simplifies the process of setting up the memory design for ChipScope™ tool debug. If desired, the MIG tool can be directed to port map user debug signals to ChipScope tool modules in the top-level design (not available in the EDK flow), allowing the ChipScope tool to monitor traffic on the User Interface ports (see Debug Signals in Chapter 3 for more information on which User Interface signals are connected to the ChipScope tool modules). When the memory design is implemented using the `ise_flow.bat` batch mode script in the design's `PAR` folder, the CORE Generator tool is automatically called to generate ChipScope tool modules (that is, NGC files are generated) for monitoring the debug signals. If the debug option is not selected, the debug signals are left unconnected in the design top module, and no ChipScope tool modules are generated.

15. Set the Debug Signals for Memory Controller pull-down menu to **Enable** to monitor debug signals using the ChipScope tool. Otherwise, leave this option set to **Disable**. Click **Next** to see the summary of all options and settings for the current project.

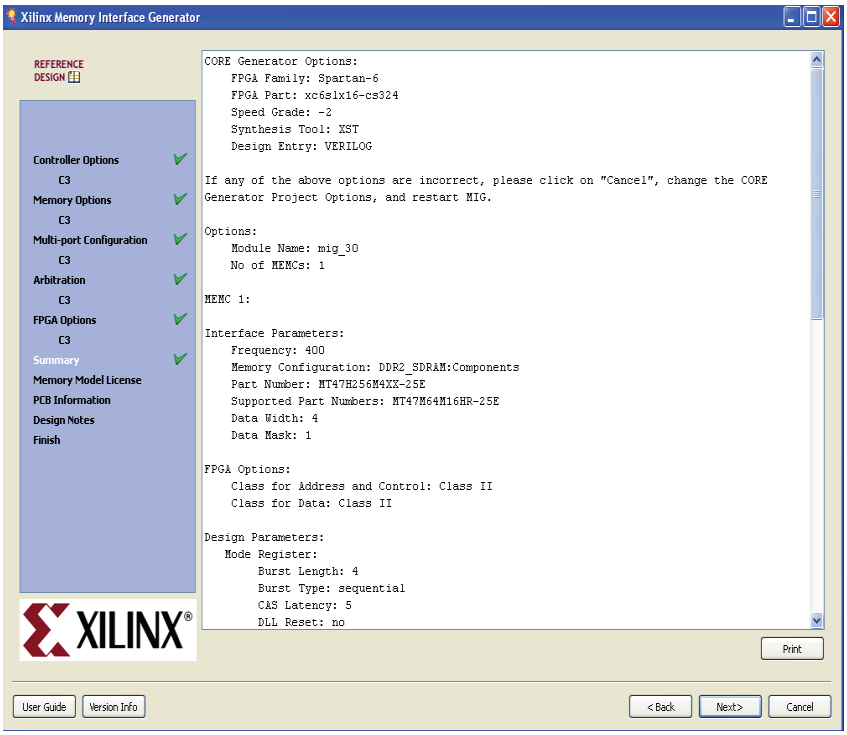


UG416_c1_34_091409

Figure 1-34: **Setting Up Debug Signal Control**

Design Summary

The Summary page (not shown in the EDK flow) provides a detailed summary of design parameters, interface parameters, CORE Generator tool options, and FPGA options for the memory interface design as shown in Figure 1-35.



UG416_c1_35_091409

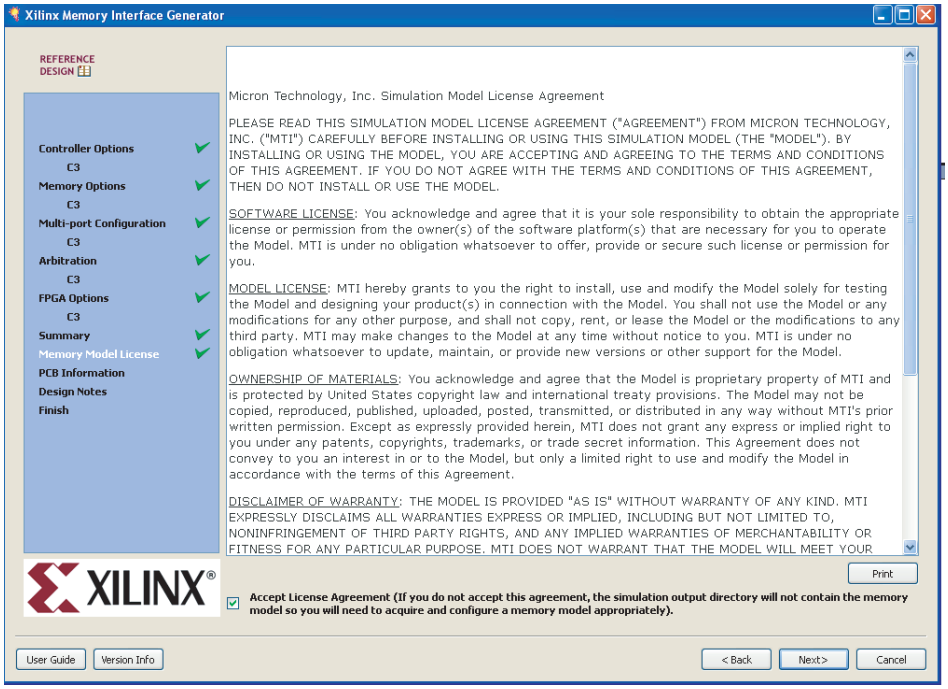
Figure 1-35: **Summary Page**

16. After reviewing the summary page to make sure all information is correct, click **Next** to move on to Memory Model License agreements.

Memory Model License

The MIG tool can output a vendor memory model to support simulation of DDR, DDR2, DDR3, and LPDDR devices (not available in the EDK flow). To access the models in the output SIM folder, the user must read and agree to the vendor license agreement. If the license agreement is not agreed to, no memory models are produced and it is not possible to simulate the design.

17. Read the vendor license agreement, and click the **Accept License Agreement** check box to have a memory model created for the design. Then click **Next**.



UG416_c1_36_021810

Figure 1-36: **Vendor Memory Model License Agreement**

PCB Information

The PCB Information page provides a list of PCB design guidelines for MIG generated designs.

18. Click **Next** to advance to the Design Notes page.

Design Notes

The Design Notes page contains information about the specific MIG release used to create the memory interface.

19. Click **Generate** to have the MIG tool create all the necessary design files for simulation and implementation of the specified memory interface solution.

The MIG tool generates two output directories: `example_design` and `user_design`.

Finish

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

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

MIG Directory Structure and File Descriptions

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

Output Directory Structure

The MIG tool (non-EDK flow) places all output files and directories in a folder named `<component name>`, where `<component name>` was specified on the MIG Output Options page in step 4, page 13 of the MCB design creation flow.

Figure 1-37 shows the output directory structure for the memory controller design. There are three folders created within the `<component name>` directory:

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

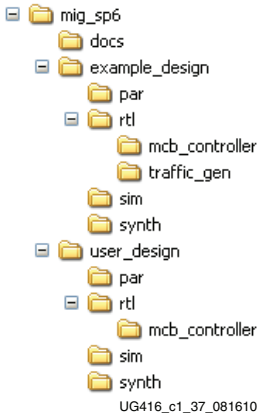


Figure 1-37: **MIG Directory Structure for Native Interface Designs**

Note: In the EDK flow, the MIG project file and related files are stored in `<EDK Project Directory>/__xps/<Instance Name>` and is regenerated when the XPS project is built. The MIG UCF with pin location information is written to `<EDK Project Directory>/__xps/<Instance Name>/mig.ucf` and is translated to an EDK core-level UCF at `<EDK Project Directory>/implementation/<Instance Name>_wrapper/<Instance Name>.ucf` during builds. Though these files are regenerated during builds, it is useful to retain or refer to them for debug purposes.

Directory and File Contents (CORE Generator Tool Flow Only)

<component name>/docs

The `docs` directory contains all PDF documentation related to the memory design, including this document.

<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 traffic generator. For more details on the example design, see MIG Example Design with Traffic Generator, page 37. The optional ChipScope tool module is also included in this directory structure.

<component name>/example_design/rtl

The `rtl` folder in this directory contains the MIG generated top-level files shown in Table 1-1.

Table 1-1: **example_design/rtl/ Directory File Descriptions**

File Name	Description
`example_top.v`	The top-level file generated by the MIG tool.
`infrastructure.v`	Spartan-6 FPGA PLL and clock network resources required for the memory design.
`memc_tb_top.v`	The test bench stimulus module with the Init Memory Control block and the Traffic Pattern Generator. All six MCB ports can be instantiated for the test bench even when not all ports are used. The unused interface ports can be ignored.
`memc_wrapper.v`	The top-level wrapper file containing an MCB and other FPGA resources necessary to create the desired memory interface. This file contains both Native and AXI4 interface port options regardless of selections made in the MIG tool. The unused interface port option can be ignored.

<component name>/example_design/rtl/traffic_gen

The `traffic_gen` subfolder contains the synthesizable HDL files for the traffic generator. Table 1-2 describes all files within this folder.

Table 1-2: **example_design/rtl/traffic_gen Directory File Descriptions**

File Name	Description
`a_fifo.v`	This module is the Synchronous FIFO using LUT RAM.
`cmd_gen.v`	This module is the command generator. It provides independent control for generating types of commands, addresses, and burst length.
`cmd_prbs_gen.v`	This module is the PRBS generator. It generates PRBS commands, PRBS address, and PRBS burst length.
`data_prbs_gen.v`	This module is a 32-bit LFSR for generating the PRBS data pattern.
`init_mem_pattern_ctr.v`	This module generates flow control logic for the traffic generator.
`mcb_flow_control.v`	This module generates flow control logic between the memory controller core and the cmd_gen, read_data_path and write_data_path modules.
`mcb_traffic_gen.v`	This module is the top level of the traffic generator.
`pipeline_inserter.v`	This module is used to insert pipeline stages.
`rd_data_gen.v`	This module generates timing control for read and ready signals to `mcb_flow_control.v`.
`read_data_path.v`	This module is the top level for the read datapath.

Table 1-2: **example_design/rtl/traffic_gen Directory File Descriptions *(Cont'd)***

File Name	Description
read_posted_fifo.v	This module stores the read command sent to the memory controller. Its FIFO output generates expected data for read data comparison.
sp6_data_gen.v	This data gen file generates different data patterns.
wr_data_gen.v	This module generates timing control for write and ready signals to mcb_flow_control.v.
write_data_path.v	This module is the top level for the write datapath.

<component name>/example_design/rtl/mcb_controller

This directory contains the soft calibration module and the lower level wrapper files for the MCB.

Note: If the AXI4 interface is enabled, the <component name>/example_design/ directory is empty because the traffic generator is not supported for AXI designs. For more information on the AXI4 interface, refer to Chapter 2, EDK Flow Details.

<component name>/example_design/par

The par folder contains the necessary constraint and script files for design implementation. Table 1-3 describes all files within this folder.

Table 1-3: **example_design/par Directory File Descriptions**

File Name	Description
example_top.ucf	This file is the UCF for the core and the example design.
create_ise.bat	The user double-clicks this file to create an ISE tool project. The generated ISE tool project contains the recommended build options for the design. To run the project in GUI mode, the user double-clicks the ISE tool project file to open up the ISE tool in GUI mode with all project settings.
ise_flow.bat	This script file runs the design through synthesis, build, map, and par. It sets all the required options. Users should refer to this file for the recommended build options for the design.

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

<component name>/example_design/synth

The synth folder contains the necessary tool constraints and script files for synthesizing the example design. Table 1-4 describes all files within this folder.

Table 1-4: **example_design/synth Directory File Descriptions**

File Name	Description
mem_interface_top_synp.sdc	The SDC file has design constraints for the Synplify Pro synthesis tool.
script_synp.tcl	These script files set various tool options.

<component name>/example_design/sim

The sim folder contains the vendor memory model, top-level simulation module, and other files necessary for simulating the example design. Table 1-5 describes all files within this folder.

Table 1-5: **example_design/sim Directory File Descriptions**

File Name	Description
ddr<n>_model_c<x>.v	This file is the DDR SDRAM memory model.
ddr<n>_model_parameters_c<x>.v	This file contains the DDR memory model parameter settings.
sim.do	This file is the ModelSim simulator script file for the example design.
sim.exe	The user double-clicks on this executable file to automatically simulate the design using the ModelSim simulator.
sim_tb_top.v	This file is the simulation top-level file.

<component name>/user_design

The user_design directory structure contains all necessary RTL, constraints, and script files for simulation and implementation of a complete MCB based memory interface ready for integration into the overall FPGA application.

<component name>/user_design/rtl

The rtl folder in this directory contains the MIG generated top-level design files described in Table 1-6.

Table 1-6: **user_design/rtl Directory File Descriptions**

File Name	Description
<component_name>.v	This is the top-level file of the customized wrapper for the dedicated memory controller block.
infrastructure.v	This is the Spartan-6 FPGA PLL and clock network resources required for the memory design.
memc_wrapper.v	This is the top-level wrapper file containing an MCB and other FPGA resources necessary to create the desired memory interface.

Note: If the AXI4 interface is enabled, there is an additional directory <component name>/user_design/rtl/axi which contains the AXI RTL files. For more information on the AXI4 interface, refer to Chapter 2, EDK Flow Details.

<component name>/user_design/par

The par folder contains the necessary constraint and script files for design implementation. Table 1-7 describes all files within this folder.

Table 1-7: **user_design/par Directory File Descriptions**

File Name	Description
<component_name>.ucf	This file is the UCF for the core and the example design.
create_ise.bat	The user double-clicks this file to create an ISE tool project. The generated ISE tool project contains the recommended build options for the design. To run the project in GUI mode, the user double-clicks the ISE tool project file to open up the ISE tool in GUI mode with all project settings.
ise_flow.bat	This script file runs the design through synthesis, build, map, and par. It sets all the required options. Users should refer to this file for the recommended build options for the design.

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

<component name>/user_design/synth

The synth folder contains the necessary tool constraints and script files for synthesizing the user design. Table 1-8 describes all files within this folder.

Table 1-8: **user_design/synth Directory File Descriptions**

File Name	Description
mem_interface_top_synp.sdc	This SDC file has design constraints for the Synplify Pro synthesis tool.
script_synp.tcl	These script files set various tool options.

<component name>/user_design/sim

The sim folder contains the vendor memory model, top-level simulation module, and other files necessary for simulating the user design. Table 1-9 describes all files within this folder.

Table 1-9: **user_design/sim Directory File Descriptions**

File Name	Description
a_fifo.v	This file is the synchronous FIFO using LUT RAM.
cmd_gen.v	This module contains the command generator. It provides independent control for generating types of commands, addresses, and burst length.
cmd_prbs_gen.v	This module contains the PRBS generator. It generates PRBS commands, PRBS addresses, and PRBS burst lengths.

Table 1-9: **user_design/sim Directory File Descriptions *(Cont'd)***

File Name	Description
data_prbs_gen.v	This file is a 32-bit LFSR for generating a PRBS data pattern.
ddr<n>_model_c<x>.v	This file is the DDR SDRAM memory model.
ddr<n>_model_parameters_c<x>.vh	This file contains the DDR memory model parameter settings.
init_mem_pattern_ctr.v	This file generates flow control logic for the traffic generator.
mcb_flow_control.v	This module generates flow control logic between the memory controller core and the cmd_gen, read_data_path, and write_data_path modules.
mcb_traffic_gen.v	Top level of the traffic generator.
pipeline_inserter.v	This file is used to insert pipeline stages.
rd_data_gen.v	This module generates timing control for read and ready signals to mcb_flow_control.v.
read_data_path.v	This file 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 generates expected data for read data comparisons.
sim.do	This is the ModelSim simulator script file for the user design.
sim.exe	Double-click on this executable file to automatically simulate the design using the ModelSim simulator.
sim_tb_top.v	This file is the simulation top-level file.
sp6_data_gen.v	This data gen file generates different data patterns.
wr_data_gen.v	This module generates timing control for write and ready signals to mcb_flow_control.v.
write_data_path.v	This file is the top level for the write datapath.

MIG Example Design with Traffic Generator

This section describes an example design for the native interface. Refer to CORE Generator Tool with AXI4 Interface Only, page 53 for an example design using the AXI4 interface.

CORE Generator Tool Native Interface Only

This section explains how to simulate and implement the MIG generated example design. This design includes a traffic generator for demonstrating and testing the MCB based memory interface. The bitstream created from implementation of the example design can be targeted to a Spartan-6 FPGA SP601 or SP605 hardware evaluation board to demonstrate DDR2 or DDR3 interfaces, respectively.

The example design includes these modules as shown in Figure 1-38:

- Spartan-6 FPGA MIG Wrapper: top-level wrapper file produced by the MIG tool, containing an MCB and other FPGA resources necessary to create the desired memory interface.
- TB_top: test bench stimulus module with the Init Memory Control block and the Traffic Pattern Generator.
- Clock Infrastructure: Spartan-6 FPGA PLL and clock network resources required for the memory design.

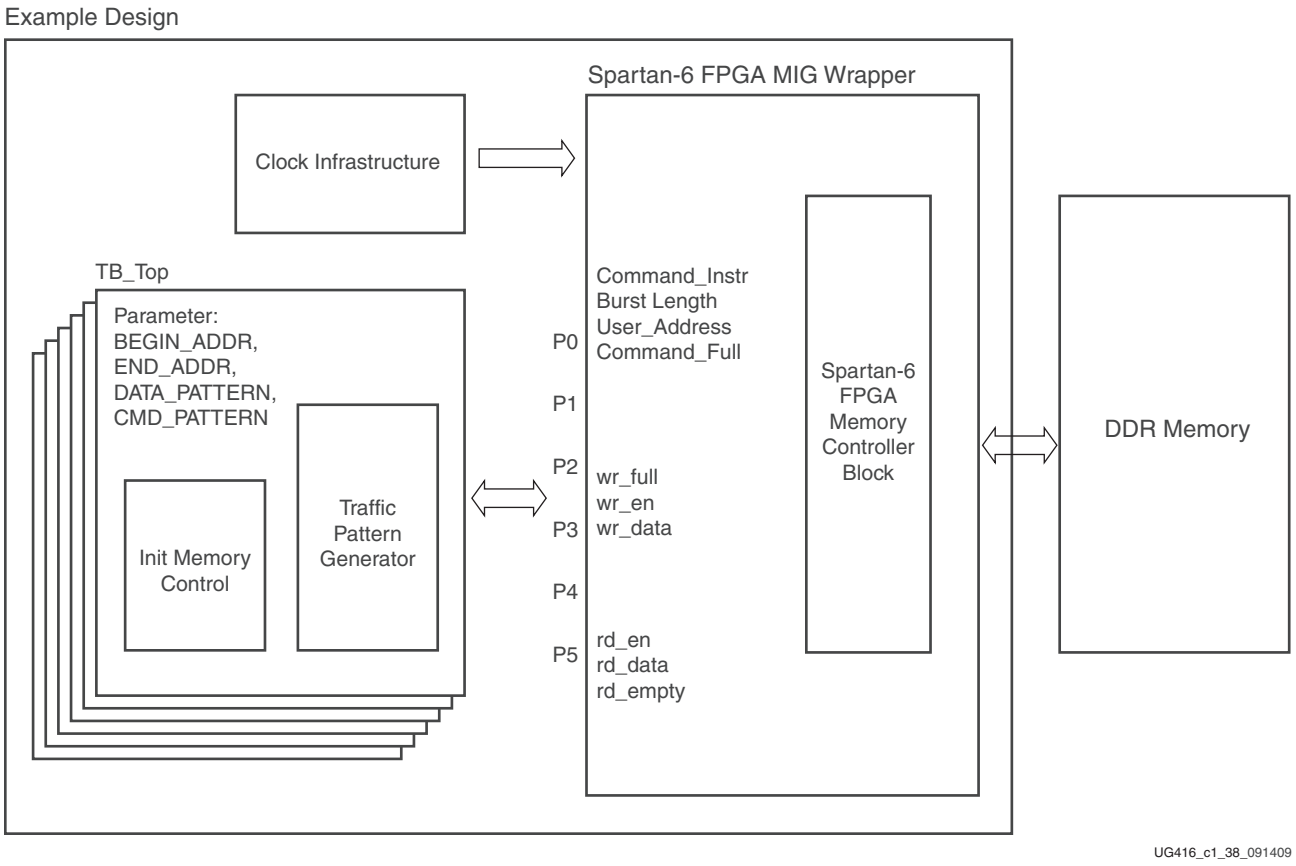


Figure 1-38: **MIG Example Design with Synthesizable Traffic Generator**

Traffic Generator Operation

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

The MIG tool creates a separate traffic generator for each enabled port of the User Interface. Each traffic generator can create traffic patterns for the entire address space of its associated port. A default address space for each port is assigned by the MIG tool using the BEGIN_ADDRESS and END_ADDRESS parameters found in the top-level test bench file (`tb_top.v`). See Modifying the Example Design, page 45 for information on using these parameters to change the port address space.

The test bench first initializes the entire address space of the port with the requested data pattern (data pattern options are discussed in the following subsections). The Init Memory Control block directs the traffic generator to step sequentially through all addresses in the port 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.

When the memory has been initialized, the traffic generator begins stimulating the User Interface ports to create traffic to and from the memory device. By default, the traffic generator sends pseudo-randomized commands to the port, meaning that the instruction sequences (R/W, R, W, etc.), addresses, and burst lengths are determined by pseudo-random bitstream (PRBS) generator logic in the test bench. As with the address space and data pattern, the default PRBS command pattern can be changed as described in Modifying the Example Design, page 45.

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 (for example, there is a mismatch between read data and expect data), an error signal is asserted and the readback address, readback data, and expect data are latched into the error_status outputs.

Each stimulus data pattern is described in the following subsections.

Address as Data Pattern (Default)

This pattern writes each memory location with its own address, a simple test for finding address bus related issues (see Figure 1-39).

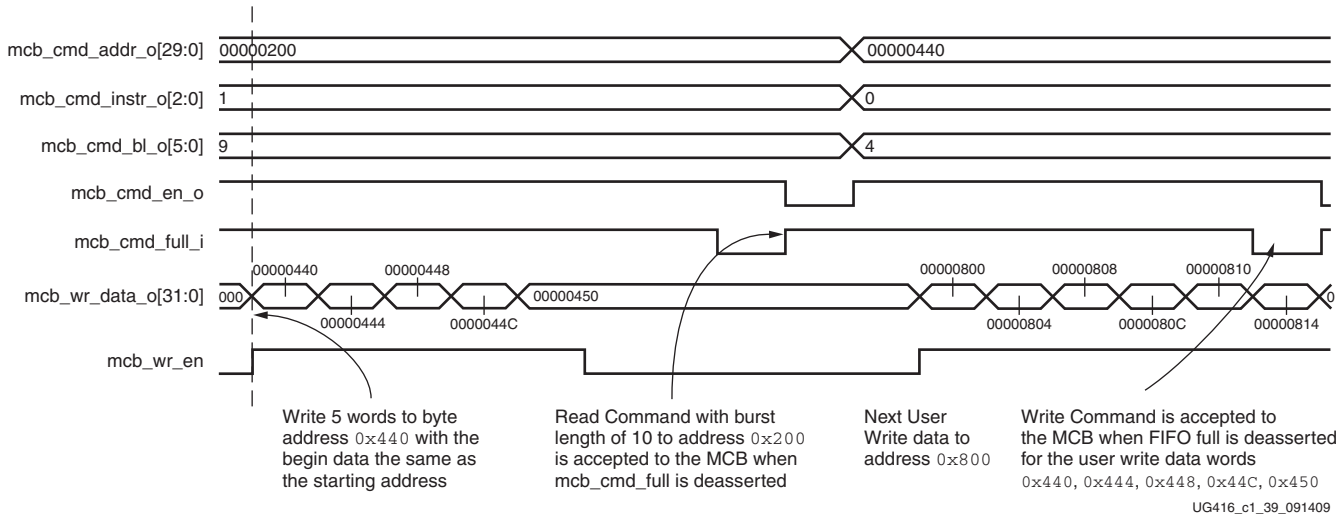


Figure 1-39: **Address as Data Pattern on DQ Bus**

Hammer Data Pattern

This pattern stresses the memory interface with simultaneous switching output (SSO) noise (see Figure 1-40). When multiple output drivers switch simultaneously, they can cause a voltage drop or ground bounce on the power planes of the PCB or inside the device package.

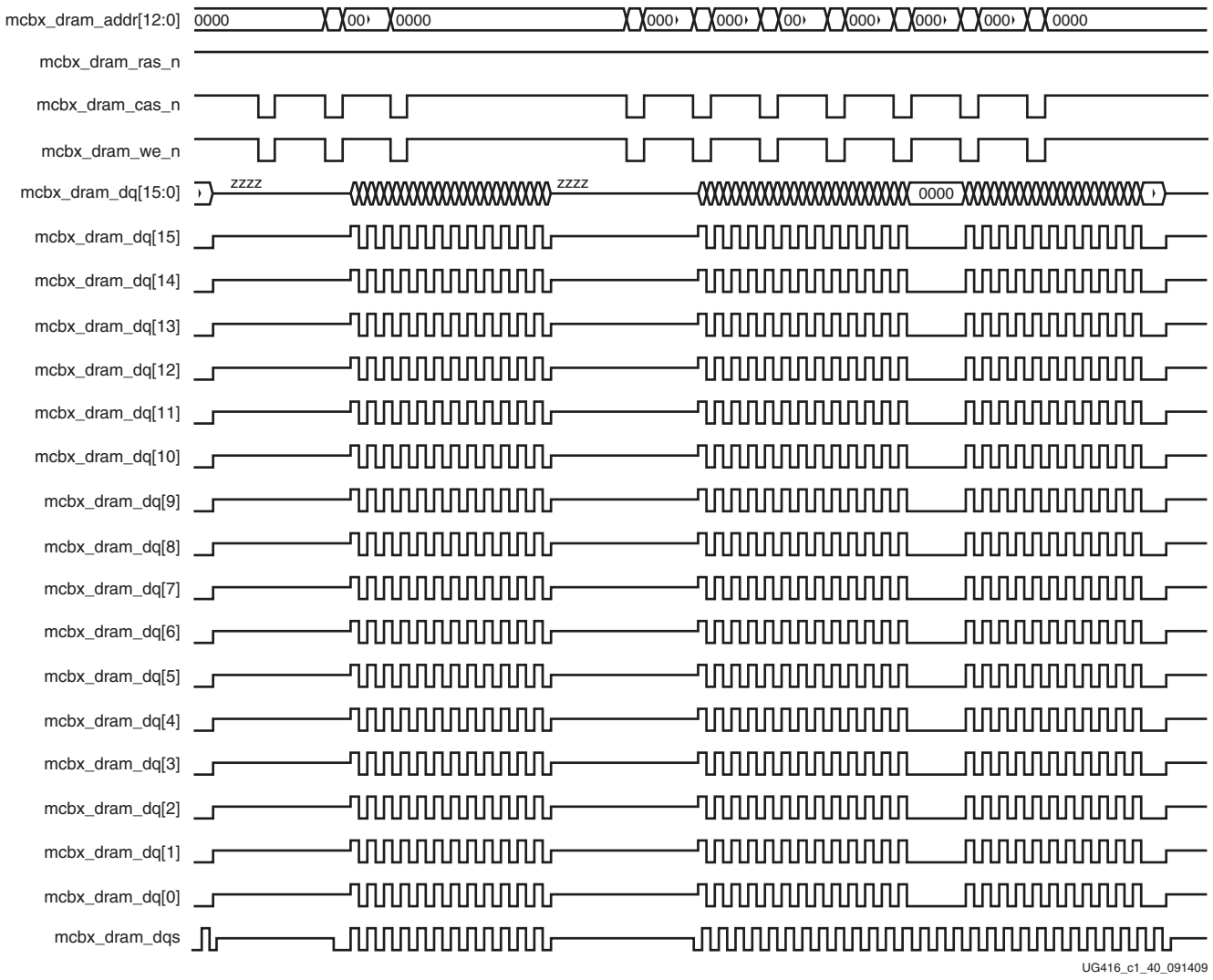


Figure 1-40: **Hammer Data Pattern on DQ Bus**

Neighbor Data Pattern

This pattern is similar to the Hammer pattern with the exception that one DQ pin remains Low on any given cycle (see Figure 1-41). This pattern can be used to measure the degree of noise coupling on a static I/O pin due to SSO noise created by other pins.

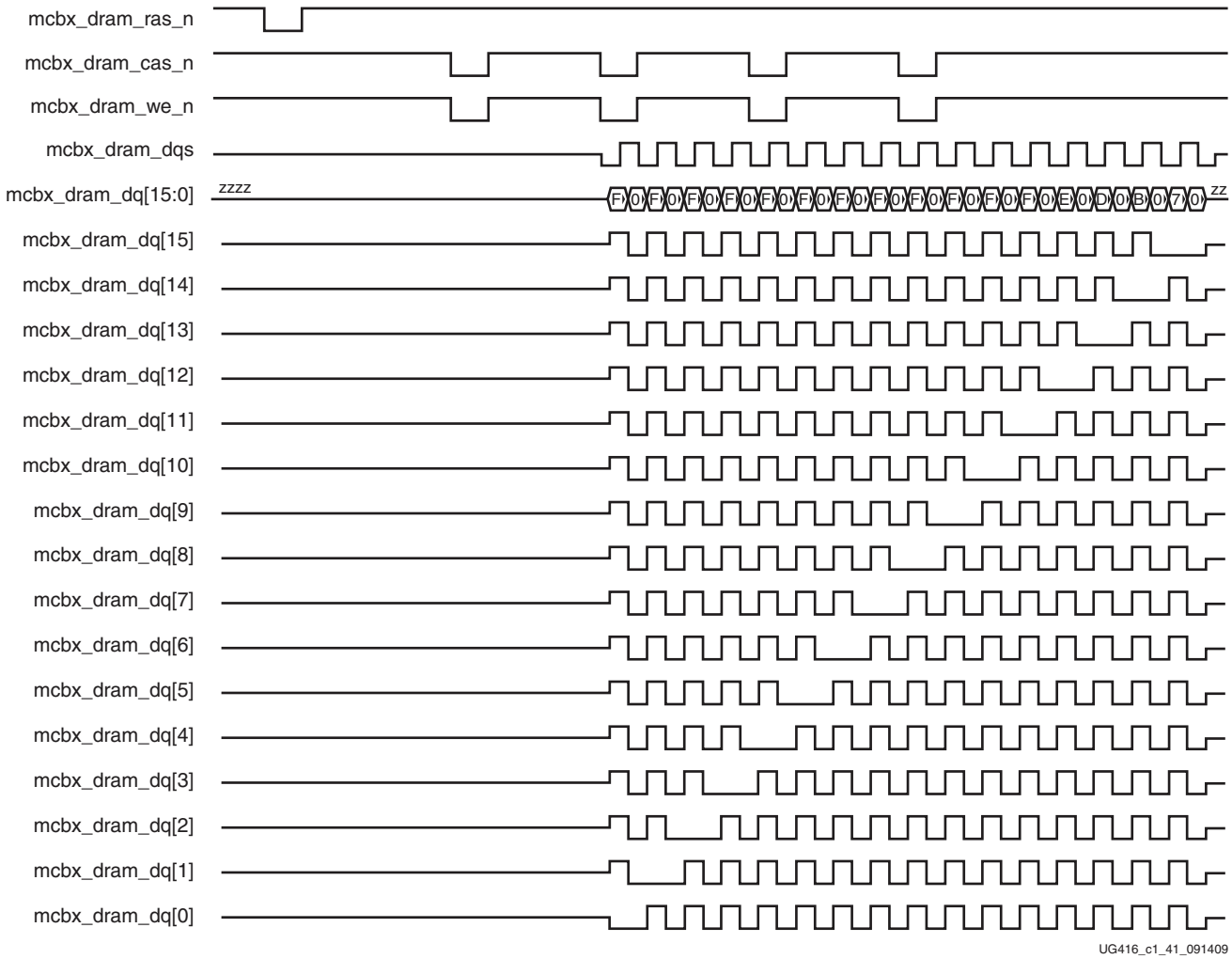


Figure 1-41: **Neighbor Data Pattern on DQ Bus**

Walking 1s and Walking 0s Data Pattern

The Walking 1s and Walking 0s patterns (see Figure 1-42 and Figure 1-43, respectively) ensure that each memory bit location can be set to both 1 and 0, independently from other bits. The DQ bus connectivity on the PCB can also be verified with these tests.

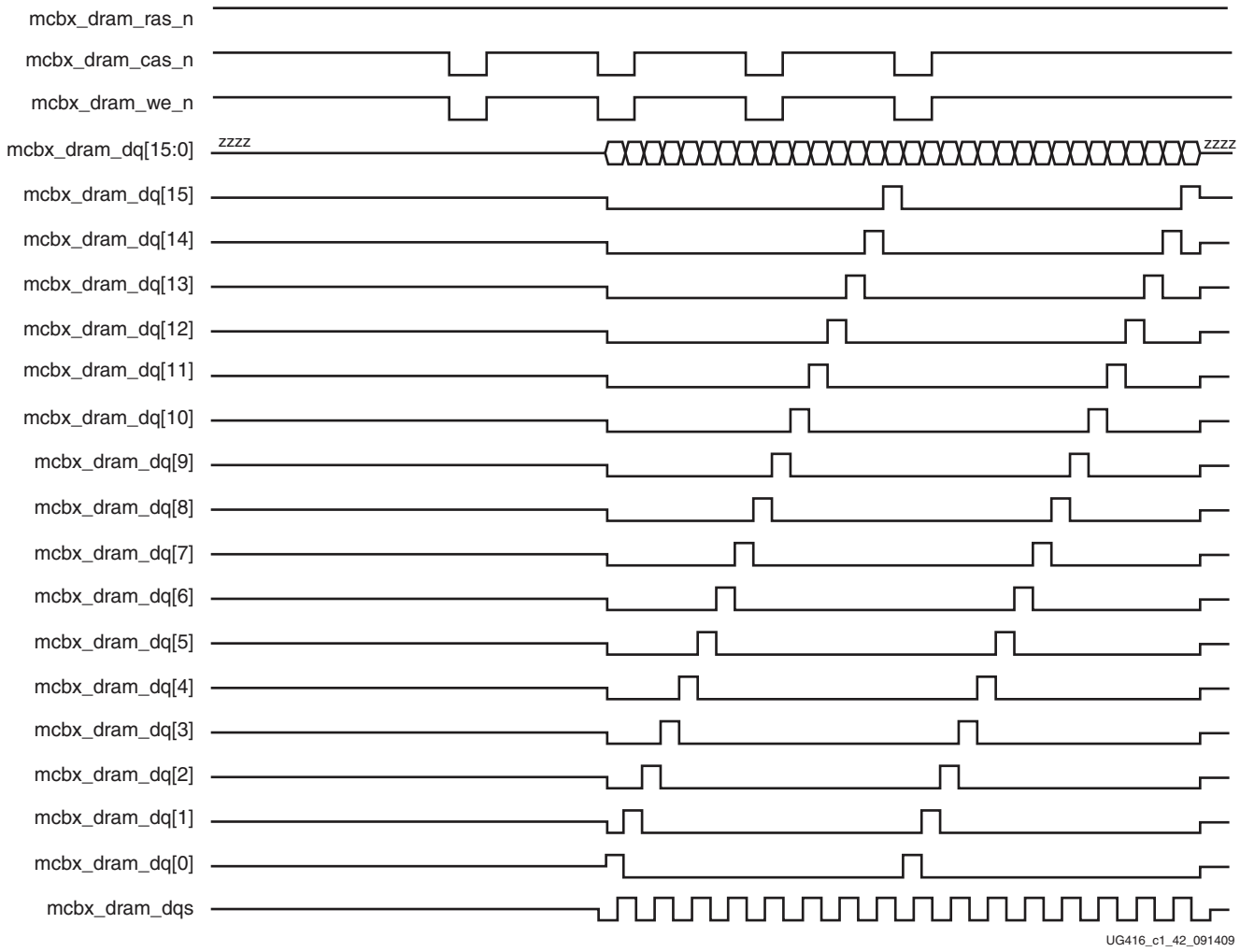


Figure 1-42: **Walking 1s Data Pattern on DQ Bus**

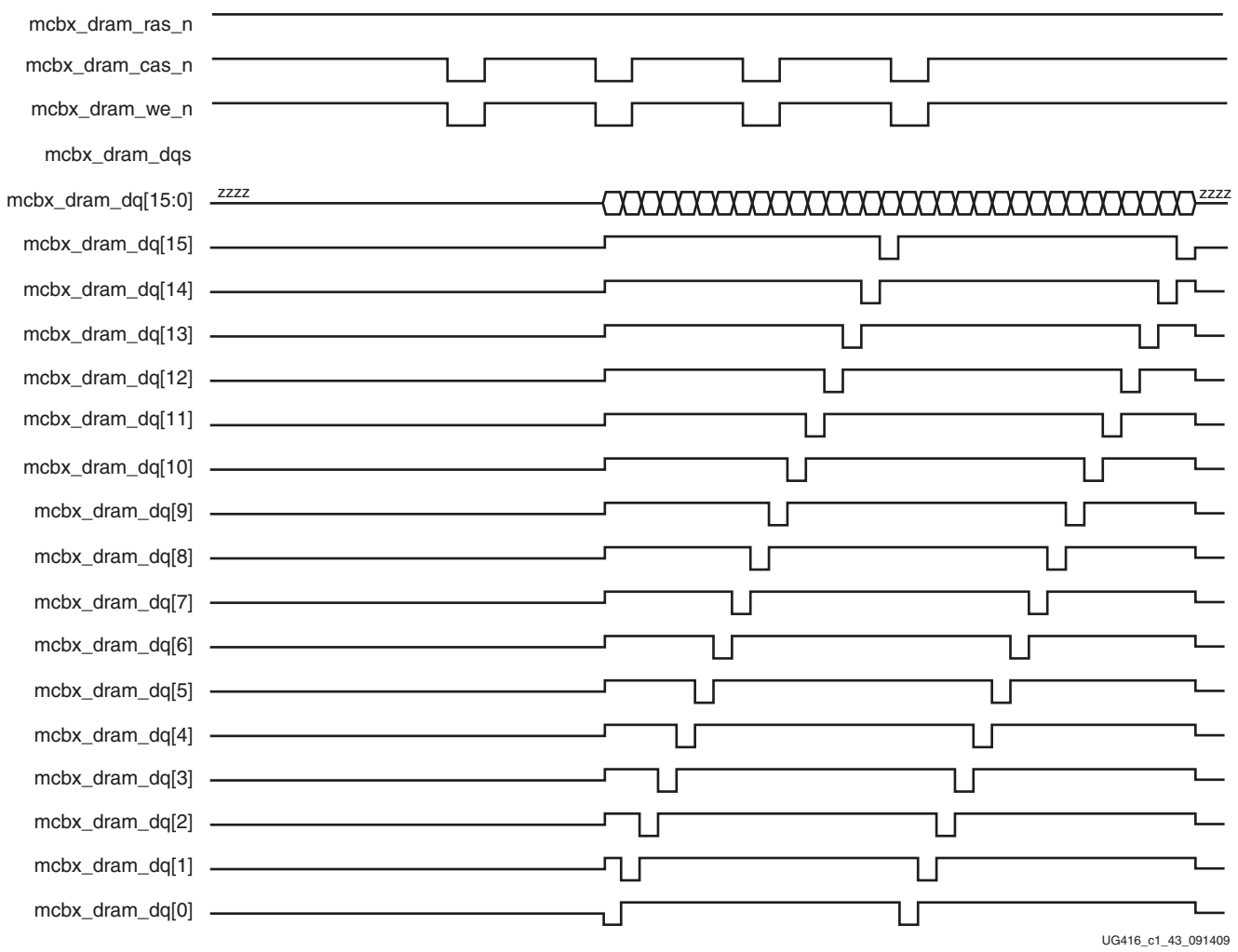


Figure 1-43: **Walking 0s Data Pattern on DQ Bus**

PRBS Data Pattern

This pattern creates PRBS data. The starting address of each data burst is used as a seed to a 32-bit LFSR circuit to generate bursts with randomized data, approximating a "real world" application test.

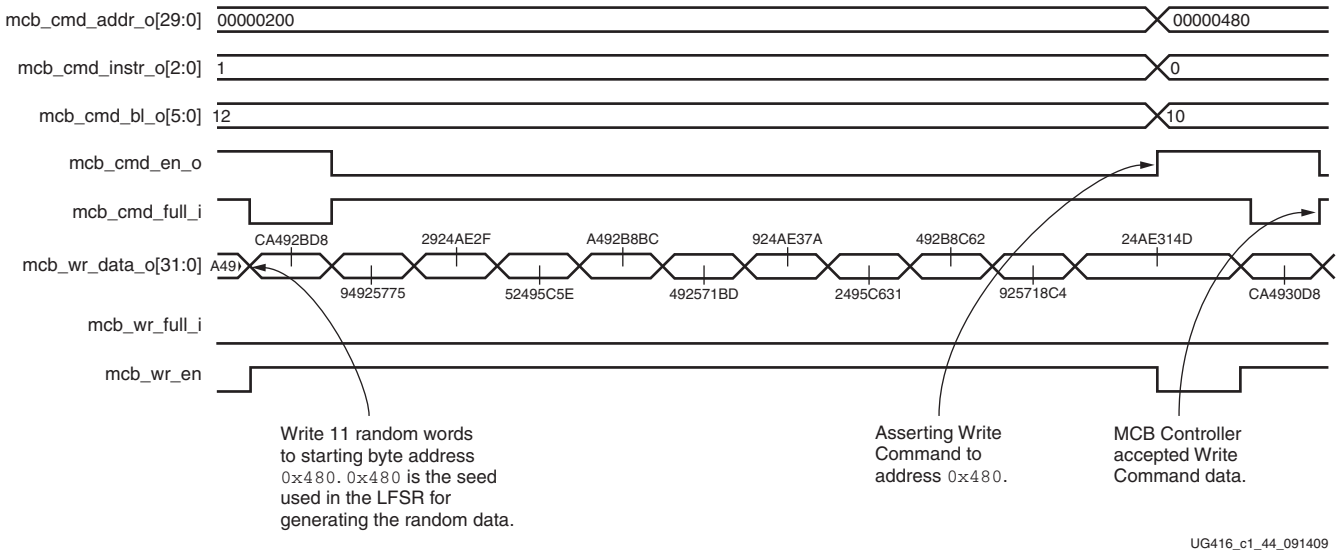


Figure 1-44: **PRBS Data Pattern on DQ Bus**

Setting Up for Simulation

In simulation, the user ports in the traffic generator are assigned with a small address range to avoid memory overflow if the system has limited physical memory installed. For hardware testing, the user can manually modify the HWTESTING parameter in `example_top` for a larger address space range.

See the "Simulation" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for more details on simulating designs with the MCB.

Functional Simulation

To simulate the MIG example design or the MIG user design, the Xilinx® UNISIM library must be compiled and mapped to the simulator. Currently, MIG generated designs are supported only for Xilinx ISim and ModelSim version 6.4b or above. However, the encrypted model of the Spartan-6 FPGA MCB is provided for ISim, ModelSim, and Cadence Incisive Enterprise Simulator (IES). EDK generated designs using the MCB are supported on all three of these simulators.

The Traffic Generator test bench provided with the example design allows pre-implementation functional simulations to be performed on the generated memory interface solution.

Memory Devices Supported for Functional Simulation

The MIG tool supports Micron DDR SDRAM, DDR2 SDRAM, DDR3 SDRAM, and LPDDR memory devices. It also supports Elpida DDR2 SDRAM memory devices for simulation. ModelSim and ISim are the simulation tools supported. ModelSim supports all of the listed memory devices, while ISim supports only the Micron devices.

To run the simulation:

1. Go to this directory:

   `<project_dir>/<component_name>/example_design/sim/functional`

2. Run the script command that corresponds to the chosen simulation tool and operating system:
   - Windows
     - For ModelSim, type at the prompt: **`sim.do`**
     - For ISim, type at the prompt: **`isim`**
   - Linux
     - For ModelSim, type at the prompt: **`source sim.do`**
     - For ISim, type at the prompt: **`source isim.do`**

Implementing the Example Design

The MIG tool automatically generates the `ise_flow.bat` script file found in the `par` folder of the example design. This script runs the design through the synthesis, translate, map, and par operations. Refer to this file to see all recommended build options for the design.

Modifying the Example Design

The test bench in the MIG generated example design can be modified to implement different data and command patterns. This section defines the test bench parameters and signal names that should be understood when making changes to the example design.

Top-Level Parameters

The top-level test bench file (`tb_top.v`) contains several parameters that can be modified to change the behavior of the traffic generator. Table 1-10 describes these parameters and identifies any default values. In general, the data pattern and address space parameters are the most likely to be modified, because the other parameters are normally fixed characteristics of the memory and MCB configuration.

The easiest way to change the data pattern implemented by the traffic generator is to open the `example_top.v` file in the `rtl` directory and edit the local parameter for Data Mode (for example, C3_p0_DATA_MODE). The four-bit code for this parameter can be changed using the binary values defined for the data_mode_i[3:0] signals in Table 1-12, page 48.

Table 1-10: **Parameters for the TB_TOP Module**

Parameter	Parameter Description	Parameter Value
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.
DATA_PATTERN	Sets the data pattern to be generated	Valid settings for this parameter are: ADDR (Default): The address is used as a data pattern. 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. WALKING1: Walking 1s are on the DQ pins and the starting position of 1 depends on the address value. 0: Walking 0s are on the DQ pins and the starting position of 1 depends on the address value. NEIGHBOR: The Hammer pattern is on all DQ pins except one. The address determines the exception pin location. PRBS: A 32-stage LFSR generates random data and is seeded by the starting address.
DWIDTH	The MIG tool sets the default based on the User Data port width	Valid settings for this parameter are 32, 64, and 128 bits.
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.
FAMILY	Indicates the Family type	The value of this parameter is “SPARTAN6”.
NUM_DQ_PINS	The MIG tool sets the default based on the number of data (DQ) pins for the selected memory	Valid settings for this parameter are “4”, “8”, and “16”.
PORT_MODE	The MIG tool sets the default based on the port configuration (bidirectional, W only, or R only)	Valid settings for this parameter are: BI_MODE: Generate a WRITE data pattern and monitor the READ data for comparison. WR_MODE: Generate only WRITE data patterns. No comparison logic is generated for the port. RD_MODE: Generate only READ control logic for the port.
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 BEGIN_ADDRESS value is ORed with the PRBS address for bit positions that have a “1” in this mask.

Traffic Generator Parameter

The CMD_PATTERN parameter can be modified within the Traffic Generator module (see Table 1-11). This parameter is not brought to the top-level test bench because it should not be modified under normal circumstances. However, certain situations might require a change to the default value, such as when address, burst length, and instruction values are provided from a block RAM (see Custom Command Sequences, page 51).

Table 1-11: **Parameter for the Traffic Generator Module**

Parameter Name	Parameter Description	Parameter Value
CMD_PATTERN	Parameter for setting command pattern circuits to be generated. For larger devices, the CMD_PATTERN can be set to "CGEN_ALL". This parameter enables all supported command pattern circuits to be generated. However, it is sometimes necessary to limit a specific command pattern because of limited resources in a smaller device.	Valid settings for this signal are: CGEN_FIXED: The address, burst length, and instruction are taken directly from the fixed_addr_i, fixed_bl_i, fixed_instr_i inputs. CGEN_SEQUENTIAL: The address is incremented sequentially, and the increment is determined by the data port size. CGEN_BRAM: The address, burst length, and instruction are taken directly from the bram_cmd_i input bus. CGEN_PRBS: A 32-stage LFSR generates pseudo-random addresses, burst lengths, and instruction sequences. The seed can be set from the 32-bit cmd_seed input. CGEN_ALL (Default): This option turns on all of the above options and allows addr_mode_i, instr_mode_i, and bl_mode_i to select the type of generation during run-time.

Traffic Generator Signal Descriptions

Table 1-12 describes all traffic generator signals. In the example design, the Init Memory Control block controls most of these signals to implement the default test flow (that is, initialize the memory with the data pattern, then start running traffic by generating pseudo-random command patterns). Any modification of the design to control these signals by other means should only be done with a thorough understanding of their behavior.

Table 1-12: **Traffic Generator Signal Descriptions**

Signal Name	Direction	Description
addr_mode_i[2:0]	Input	Valid settings for this signal are: 000: Block RAM address mode. The address comes from the bram_cmd_i input bus. 001: FIXED address mode. The address comes from the fixed_addr_i input bus. 010: 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. 011: SEQUENTIAL address mode. The address is generated from the internal address counter. The increment is determined by the User Interface port width.
bl_mode_i[1:0]	Input	Valid settings for this signal are: 00: Block RAM burst mode. The burst length comes from the bram_cmd_i input bus. 01: FIXED burst mode. The burst length comes from the fixed_instr_i input bus. 10: PRBS burst mode (Default). The burst length is generated from the internal 16-bit LFSR circuit. The seed can only be changed through the parameter section.
bram_cmd_i[38:0]	Input	This bus contains the block RAM interface ports: {BL, INSTR, ADDRESS}.
bram_rdy_o	Output	This block RAM interface output indicates when the traffic generator is ready to accept input from bram_cmd_i bus.
bram_valid_i	Input	For the block RAM interface, the bram_cmd_i bus is accepted when both bram_valid_i and bram_rdy_o are asserted.
clk_i	Input	This signal is the clock input.
cmd_seed_i[31:0]	Input	This bus is the seed for the command PRBS generator.
counts_rst	Input	When counts_rst is asserted, wr_data_counts and rd_data_counts are reset to zero.

Table 1-12: **Traffic Generator Signal Descriptions *(Cont'd)***

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

Table 1-12: **Traffic Generator Signal Descriptions *(Cont'd)***

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

Modifying Port Address Space

The address space for a port can be easily modified by changing the BEGIN_ADDRESS and END_ADDRESS parameters found in the top-level test bench file. These two values must be set to align to the port data width. The two additional parameters, PRBS_SADDR_MASK_POS and PRBS_EADDR_MASK_POS, are used in the default PRBS address mode to ensure that out-of-range addresses are not sent to the port.

The PRBS_SADDR_MASK_POS parameter creates an OR mask that shifts PRBS generated addresses with values below BEGIN_ADDRESS up into the valid address space of the port. It should be set to a 32-bit value equal to the BEGIN_ADDRESS parameter. The PRBS_EADDR_MASK_POS parameter creates an AND mask that shifts PRBS generated addresses with values above END_ADDRESS down into the valid address space of the port. It 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 1-13 shows some examples of setting the two mask parameters.

Table 1-13: **Example Settings for Address Space and PRBS Masks**

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

Custom Command Sequences

The traffic generator can send a custom command sequence to the User Interface port by reading address, instruction, and burst length values directly from a block RAM via the bram_cmd_i input bus. The CMD_PATTERN parameter in the Traffic Generator module must be set to “CGEN_ALL” (default) or “CGEN_BRAM” for this mode of operation. In the CGEN_ALL case, the addr_mode_i, instr_mode_i, and bl_mode_i inputs must be set to their respective block RAM mode values.

The bram_cmd_i input bus is a combination of the burst length, instruction, and address values as follows:

bram_cmd_i[38:0] = {BL[5:0], INSTR[2:0], ADDRESS[29:2]}

Address bits [1:0] and [31:30] are padded with 0s. The traffic generator accepts the bram_cmd_i value when both bram_valid_i and bram_rdy_o are asserted (High).

The command patterns instr_mode_i, addr_mode_i and bl_mode_i of the `traffic_gen` module can each be set independently. The provided `init_mem_pattern_ctr` module has interface signals to allow the command pattern to be modified in real time using the ChipScope tool's VIO. To change command pattern:

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

   0: bram address input.

   1: fixed_address.

   2: prbs address.

   3: sequential address.
3. Set vio_bl_mode_value to:

   0: bram bl input.

   1: fixed bl.

   2: prbs bl. If bl_mode value is set to 2, the addr_mode value is forced to 2.
4. Set vio_fixed_bl_value to: 1 — 64.

Memory Initialization and Traffic Test Flow

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

Memory Initialization

1. The data_mode_i input is set to select the data pattern (for example, data_mode_i[3:0] = `0010` for the address as the data pattern).
2. The start_addr_i input is set to define the lower address boundary.
3. The end_addr_i input is set to define the upper address boundary.
4. bl_mode_i is set to `01` to get the burst length from the fixed_bl_i input.
5. The fixed_bl_i input is set to either 16 or 32.
6. instr_mode_i is set to `0001` to get the instruction from the fixed_instr_i input.
7. The fixed_instr_i input is set to the "WR" command value of the memory device.
8. addr_mode_i is set to `011` for the sequential address mode to fill up the memory space.
9. mode_load_i is asserted for one clock cycle.

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

Traffic Test Flow

1. The addr_mode_i input is set to the desired mode (PRBS is the default).

2. The cmd_seed_i and data_seed_i input values are set for the internal PRBS generator. This step is not required for other patterns.
3. The instr_mode_i input is set to the desired mode (PRBS is the default).
4. The bl_mode_i input is set to the desired mode (PRBS is the default).
5. The data_mode_i input should have the same value as in the memory pattern initialization stage detailed in Memory Initialization.
6. The run_traffic_i input is asserted to start running traffic.
7. If an error occurs during testing (that is, the read data does not match the expected data), the error bit is set until reset is applied.
8. Upon an error, the error_status bus latches the values defined in Table 1-12, page 48.

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 the proper pattern is loaded into the memory space.

CORE Generator Tool with AXI4 Interface Only

This section describes an example design for the AXI4 interface. Refer to CORE Generator Tool Native Interface Only, page 37 for an example design using the native interface.

The MIG tool provides a synthesizable AXI4 test bench (per controller) to generate various traffic patterns to the memory controller with AXI4 user interface. This test bench consists of a controller, a traffic generator (traffic_generator) that generates traffic patterns through the AXI4 interface to a Spartan-6 FPGA MIG wrapper with AXI4 interface (memc_wrapper) core, and an infrastructure core that provides clock resources. A block diagram of the example design test bench is shown in Figure 1-45.

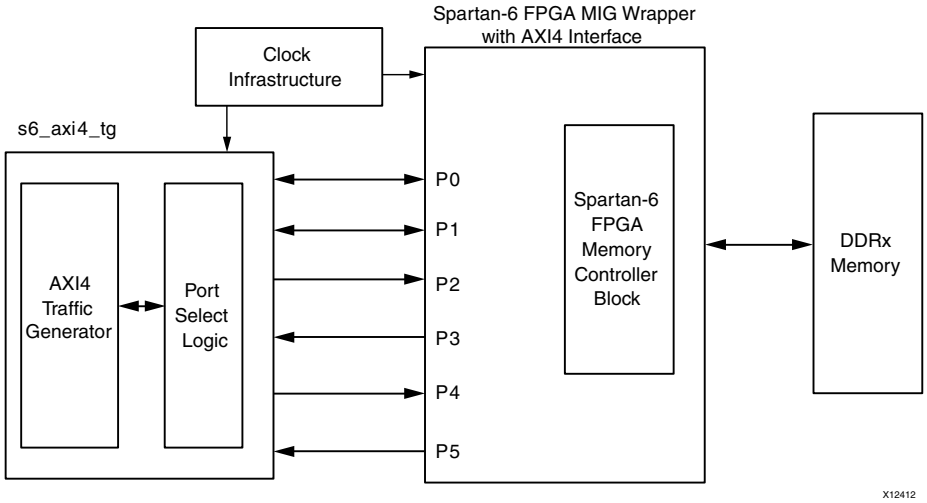


Figure 1-45: **Synthesizable Example Design Block for AXI4 Interface**

The Spartan-6 FPGA MIG wrapper can have up to six ports defined. Ports P0 and P1 are bidirectional, and ports P2 to P5 are unidirectional. Options for these configurable ports are located in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1]. In Figure 1-45, the ports P0 and P1 are 64-bit bidirectional ports, ports P2 and P4 are set as 32-bit write ports, whereas P3 and P5 are set as 32-bit read ports. The port select logic dynamically selects one of the ports to perform a write or a read transaction. It also takes care of matching the

widths for data integrity checks when transactions are performed on ports that have different widths. For example, a write transaction is performed on a 64-bit port (P0) and the same data is read with a 32-bit port (P5).

The AXI4 traffic generator (AXI4 TG) block generate AXI4 write and read transactions. Figure 1-46 shows the simple write transaction being performed on the AXI4 traffic generator interface. This consists of a command phase, a data phase, and a response phase, as shown in Figure 1-46. This follows the standard AXI4 protocol.

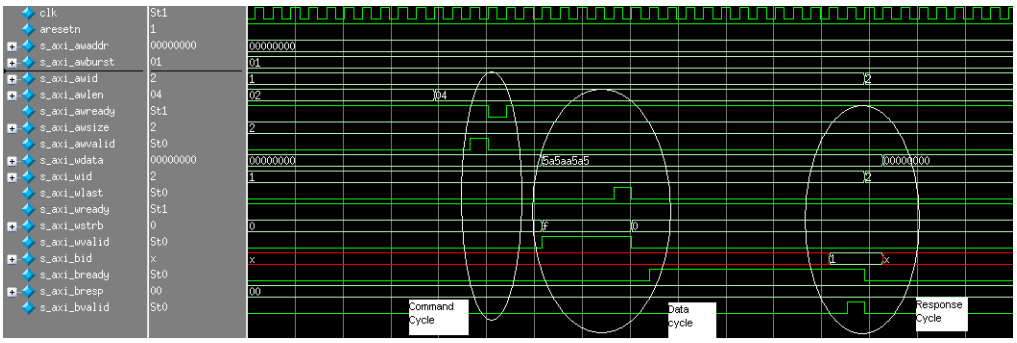


Figure 1-46: **AXI4 Interface Write Cycle**

Figure 1-47 shows the simple read transaction being performed on the AXI4 interface. This transaction consists of a command phase and a data phase, as shown in Figure 1-47. This follows the standard AXI4 protocol.

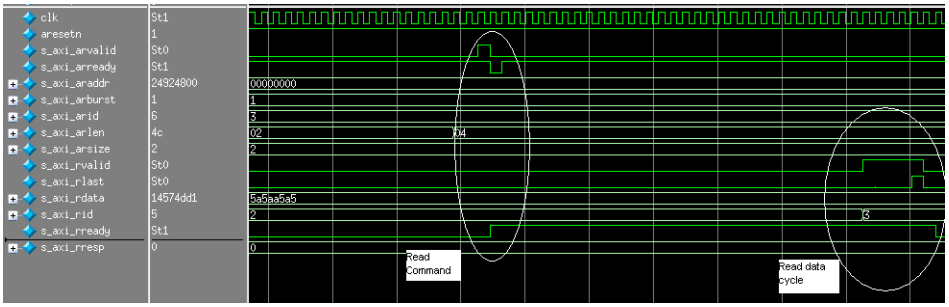


Figure 1-47: **AXI4 Interface Read Cycle**

The example design generated when the AXI4 interface is selected as the user interface is different compared to the standard user interface of the traffic generator. This synthesizable test bench verifies the basic AXI4 transactions as well as the memory controller transactions. However, this test bench does not verify all memory controller features It verifies the AXI4 Shim features. Table 1-14 shows the signals of interest during verification of the AXI4 test bench. These signals can be found in the example_top module.

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

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

Table 1-14: **Signals of Interest During Simulation for AXI4 Test Bench *(Cont'd)***

Signal	Description
cmd_err	When asserted, this signal indicates that the command phase of the AXI4 transaction (read or write) had an error.
write_err	When asserted, this signal indicates that the write transaction to memory resulted in 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 indicates the status of the write transaction. The status details are described in Table 1-15.
read_cmptd	This signal is asserted for one clock, indicating that the current read transaction completed.
read_err	When asserted, this signal indicates that the read transaction to the memory resulted in an error.
dbg_rd_sts_vld	When asserted, this signal indicates a valid status for the read transaction on the dbg_rd_sts bus. This signal is asserted even if the read transaction does not complete.
dbg_rd_sts	This signal indicates the status of the read transaction. The status details are described in Table 1-16.
cmptd_one_wr_rd	This signal indicates at least one write and one read transaction completed.

The initialization and the calibration sequence remain the same as that indicated in the previous section. The status generated for a write transaction can be found in Figure 1-48.

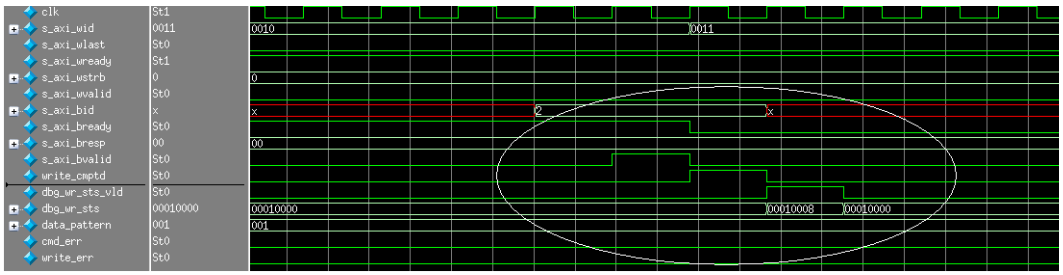


Figure 1-48: **Status for the Write Transaction**

Table 1-15 contains the information of the status signal dbg_wr_sts.

Table 1-15: **Debug Status for the Write Transaction**

Bits	Status Description
1:0	This is the write response received for the AXI4 interface.
5:2	These indicate the response ID for the write response.

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

Bits	Status Description
8:6	AXI wrapper write FSM state when time-out (watchdog timer should be enabled) occurred. • `3'b001`: Data write transaction • `3'b010`: Waiting for acknowledgment for written data • `3'b011`: Dummy data write transaction • `3'b100`: Waiting for response from the response channel
15:9	Reserved
16	Command error occurred during write transaction.
17	Write error occurred. The write transaction could not be completed.
20:18	Data pattern used for the current transaction: • `000`: `5A` and `A5` • `001`: PRBS pattern • `010`: Walking zeros • `011`: Walking ones • `100`: All ones • `101`: All zeros
31:21	Reserved

Figure 1-49 lists the status that is generated for a read transaction.

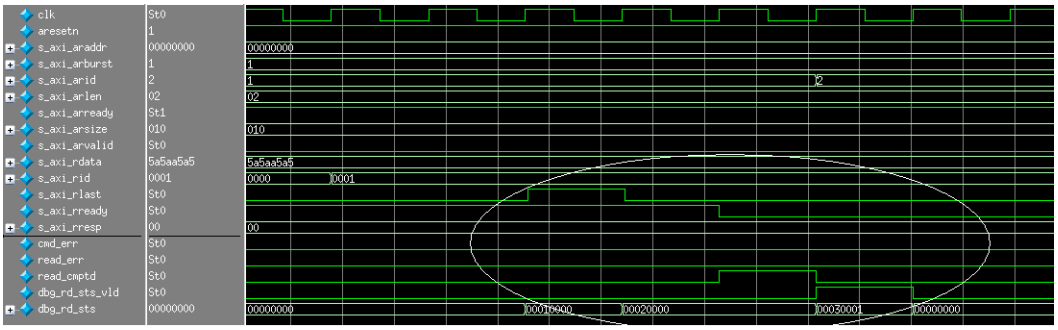


Figure 1-49: **Status for the Read Transaction**

Table 1-16 contains the information of the status signal dbg_rd_sts.

Table 1-16: **Debug Status for the Read Transaction**

Bits	Status Description
0	Read error response on the AXI4 interface.
1	Incorrect response ID presented by the AXI slave.

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

Bits	Status Description
3:2	AXI wrapper read FSM state when time-out (watchdog timer should be enabled) occurred. • `2'b01`: Read command transaction • `2'b10`: Data read transaction
15:4	Reserved
16	Command error occurred during read transaction.
17	Read error occurred. The read transaction could not be completed.
18	Data mismatch occurred between the written data and read data.
26:19	Pointer value for which the mismatch occurred.
29:27	Data pattern used for the current check: • `000`: `5A` and `A5` • `001`: PRBS pattern • `010`: Walking zeros • `011`: Walking ones • `100`: All ones
31:30	Reserved

The AXI4 write and read transactions are started only after the c*x*_calib_done signal is asserted, where *x* is the controller number.

XILINX®

Chapter 2

EDK Flow Details

This chapter describes how to use the MIG tool available in Xilinx® Platform Studio (XPS). It contains these sections:

- EDK Overview
- AXI Spartan-6 FPGA DDRx Memory Controller

EDK Overview

The Embedded Development Kit (EDK) provides an alternative package to the RTL than that of the MIG tool flow in the CORE Generator™ interface. The XPS IP Catalog contains the IP core `axi_s6_ddrx` with the same RTL that is provided by the MIG tool. The difference is that the RTL is packaged as an EDK pcore suitable for use in embedded processor based systems. The `axi_s6_ddrx` pcore only provides an AXI4 slave interface for each of the ports that are enabled. If a native MCB port is needed, refer to the Multi-Port Memory Controller (MPMC) IP provided by EDK as an alternative.

The `axi_s6_ddrx` IP is configured using the same MIG tool that is used in the CORE Generator tool. The GUI flow is the same as described in the MIG Overview in Chapter 1. However, instead of generating the UCF/RTL, the MIG tool sets the parameters for the RTL in the XPS MHS file. From the parameters, the pcore can generate the correct constraints for itself during platgen. Because the pcore is only a component in the system, the clock/reset structure must also be configured in XPS as it is not automatically generated as is done in the CORE Generator tool RTL. After the IP is configured and the ports are connected, the XPS tool is relied on to perform all other aspects of IP management including generating a bitstream and running simulations. For more information about EDK and XPS, see *EDK Concepts, Tools, and Techniques* [Ref 2] and *Embedded System Tools Reference Guide* [Ref 3].

The simplest way to get started with the `axi_s6_ddrx` memory controller is to use the base system builder (BSB) wizard in XPS. The BSB guides the user through a series of options to provide an entire embedded project with an optional `axi_s6_ddrx` memory controller. If the memory controller is selected, an already configured, connected, and tested `axi_s6_ddrx` controller is provided for a particular reference board, such as the SP601 and SP605 boards.

AXI Spartan-6 FPGA DDRx Memory Controller

The Advanced eXtensible Interface (AXI) Spartan®-6 FPGA DDRx Memory Controller core provides a high-performance multi-ported AXI4 slave front-end connection to LPDDR SDRAM/DDR/DDR2/DDR3 external memory. This core use the Memory Control Block (MCB) primitive and adapts the MCB native interface to use the AXI4 slave interface. This provides full functionality of all the features present on the Spartan-6 FPGA MCB core.

Feature Overview

In addition to the MCB feature set, the AXI features include:

- Supports read-only and write-only modes.
- Supports AXI4 INCR/WRAP transactions.
- Supports a mode to guarantee write coherency between ports.
- Does not reorder transactions.
- Round-Robin Read/Write arbitration.
- Little-endian AXI4 slave interface.
- One up to six AXI4 slave compliant memory interface(s).
- AXI4 slave interface running 1:1 clock rate to the Spartan-6 FPGA MCB controller port interface (can be asynchronous to memory).
- AXI4 slave interface data width of 32, 64, or 128 bits. AXI4 data width cannot be greater than the MCB native data width.
- Support for all MCB-supported memories (LPDDR, DDR, DDR2, and DDR3).
- Support for AXI4 long bursts up to 256 data beats.

Feature Description and AXI Protocol Support

This section describes how the AXI Spartan-6 FPGA DDRx Memory Controller interprets and supports the AXI4 specification. These interpretations of the AXI4 specification as it relates to a memory controller follow the Xilinx design conventions that balance performance, size, and complexity.

Interface Width

The AXI Read and Write data width can be 32, 64, or 128. It must be equal to the MCB data width. The MCB data width can be 32, 64, or 128 bits, depending on MCB configuration.

Interface Clock

Each AXI4 slave interface can run with a completely independent clock from each other and from the memory clock. All AXI channels and interface logic within a specific AXI4 slave interface use the same clock, with no additional clock conversion before passing into the associated MCB port.

Address Width

The address width must be parameterized to support the desired system address bus width. If the system address bus is defined wider than the memory size, it is acceptable to alias/wrap the memory across the address space. The MCB interface supports a maximum

of 30 bits for the address bus. The MSB of the AXI address is cut off, if necessary. A 32-bit constant address width is used for compatibility with EDK. The address also wraps if the address range specified by the base and high address is smaller than the memory size.

Read-Only or Write-Only AXI Ports

Each AXI4 interface can be configured as Read-only or Write-only even when connected to a bidirectional MCB port. This permits logic optimization when bidirectional data flow is not required. The Read-only or Write-only AXI port is required when connected to a unidirectional MCB port. When placed in Read-only or Write-only mode, unnecessary Read/Write arbitration logic and datapath logic are removed. If the MCB port is natively a bidirectional port, the MIG GUI and source RTL allow the user to choose a Read only or Write only AXI4 interface for FPGA resource savings.

Reset

The AXI4 interface has a single synchronous reset, active Low signaling, that resets the entire core and brings it to a known initialized state. A reset event causes a full reset including recalibration of the controller.

Bursts

These rules apply:

- The AXI Spartan-6 FPGA DDRx Memory Controller supports `INCR` and `WRAP` bursts including AXI4 extensions of `INCR` burst up to 256 data beats.
- Attempting `FIXED` bursts does not hang the AXI4 interface, but a `FIXED` burst does not have a logical meaning for a memory controller. For simplicity, `FIXED` burst commands result in an `INCR` command. No errors are flagged.
- Supports burst size down to 1 byte wide burst. Burst sizes below the native data width of the MCB port controller datapath is called a subsize burst or “narrow” transfer. Subsize burst is supported, but the AXI protocol defines a subsize burst to have data rotate through the correct byte lanes. Narrow burst support is conditional. If the system has no masters that produce narrow bursts, then significant logic can be reduced by removing support for the narrow bursts. This is controlled by the `C_S<`*Port_Num*`>_AXI_SUPPORTS_NARROW_BURST` parameter.
- The AXI Spartan-6 FPGA DDRx Memory Controller can assume that bursts do not cross a 4 KB address boundary as defined in the AXI4 specification. However, a burst that crosses a 4 KB boundary does not hang the interface, but it can cause that transaction to have undefined behavior on memory contents.

Cache Bits

These cache bit rules apply:

- The AXI Spartan-6 FPGA DDRx Memory Controller does not implement bridging, speculative pre-fetching, or L2 caching functions so it can ignore all CACHE bits and treat them as `00000`.
- The AXI Spartan-6 FPGA DDRx Memory Controller attempts to return B Responses as soon as possible without violating AXI ordering rules to reduce latency to master waiting for B Responses.
- Because the AXI Spartan-6 FPGA DDRx Memory Controller is connected to a multi-ported hard memory controller, it must not issue a B Response until the Write has completed to memory. The B response must guarantee that another Write or Read on

another MCB port that accesses the same memory location could not complete ahead of the current Write transaction. The parameter C_S<Port_Num>_AXI_STRICT_COHERENCY can be set to 0 to relax write coherency checking so that the B Response is returned earlier when the transaction is known to have completed relative to that port instead of being delayed to ensure the write completes across all ports.

Protection Bits

The AXI Spartan-6 FPGA DDRx Memory Controller ignores the AXI PROT bits and assume all transactions are normal, non-secure accesses.

Exclusive Access

This IP does not currently support exclusive access.

Response Signaling

The AXI Spartan-6 FPGA DDRx Memory Controller always generates an OKAY response.

IDs, Threads, and Reordering

The MCB interface is strictly linear; therefore no reordering or threads is implemented in the bridge. Transactions are returned in the exact order they are received.

Read/Write Acceptance Depth

The read acceptance depth is five outstanding transactions. The Write acceptance depth is four outstanding transactions.

Read/Write Arbitration

AXI has separate Read and Write channels. An external memory has only a single address bus. Therefore the AXI Spartan-6 FPGA DDRx Memory Controller must arbitrate between coincident Read and Write requests to determine which one to execute to memory. The arbitration algorithm for Read and Write requests is Round-Robin.

Endianess

The AXI Spartan-6 FPGA DDRx Memory Controller is little-endian only.

Region Bits

The AXI Spartan-6 FPGA DDRx Memory Controller does not have to make use of REGION bits and can ignore this signal.

Low Power Interface

The AXI Spartan-6 FPGA DDRx Memory Controller does not support low power interface.

Limitations

The AXI Spartan-6 FPGA DDRx Memory Controller does not support QoS.

Simulation Considerations

To simulate a design using `axi_s6_ddrx`, the user must create a test bench that connects a memory model to the `axi_s6_ddrx` I/O signals. This is generally performed by editing the `system_tb.v/.vhd` test bench template file created by the Simgen tool in XPS to add a memory model. Alternatively, users can transfer the simulator compile commands from Simgen into their own custom simulation/test bench environment.

Note: `axi_s6_ddrx` does not generally support structural simulation because it is not a supported flow for the underlying MIG PHYs. Thus structural simulation is not recommended.

An `axi_s6_ddrx` simulation should be performed in the behavioral/functional level and requires a simulator capable of mixed-mode Verilog and VHDL language support.

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

For behavioral simulation, the sysclk_2x, sysclk_2x_180, and ui_clk ports of `axi_s6_ddrx` must also be completely phase-aligned.

Top-Level Parameters

Table 2-1 lists the AXI parameters present on the AXI Spartan-6 FPGA DDRx Memory Controller. The <*Port_Num*> is 1 through 6. For details on the other parameters, refer to the *Spartan-6 FPGA Memory Controller User Guide* [Ref 1].

Table 2-1: **AXI Per-Port Top-Level Parameters**

Parameter Name	Default Value	Format (Range)	Description
C_S<*Port_Num*>_AXI_ENABLE	0	Integer (0, 1)	Enables the AXI/MCB port.
C_S<*Port_Num*>_AXI_ADDR_WIDTH	32	Integer (32)	Width of all ADDR signals.
C_S<*Port_Num*>_AXI_DATA_WIDTH	32	Integer (32,64,128)	Width of AXI WDATA, RDATA signals.
C_S<*Port_Num*>_AXI_ENABLE_AP	0	Integer (0, 1)	Enables Auto-Precharge on each transaction sent to the memory controller.
C_S<*Port_Num*>_AXI_ID_WIDTH	4	Integer (1-16)	Width of all ID signals for all channels.
C_S<*Port_Num*>_AXI_PROTOCOL	AXI4	String (AXI3, AXI4)	Specifies the AXI protocol.
C_S<*Port_Num*>_AXI_REG_EN0	0x00000	Hexadecimal	Reserved.
C_S<*Port_Num*>_AXI_REG_EN1	0x01000	Hexadecimal	Reserved.
C_S<*Port_Num*>_AXI_STRICT_COHERENCY	1	Integer (0, 1)	Delays B channel response until it can be guaranteed the write has been commited to memory. Required when accessing the same address between different ports.

Table 2-1: **AXI Per-Port Top-Level Parameters *(Cont'd)***

Parameter Name	Default Value	Format (Range)	Description
C_S<*Port_Num*>_AXI_SUPPORTS_NARROW_BURST	1	Integer (0, 1)	Enables logic to support narrow transfers over MCB. Required if the slave receives transactions smaller than the AXI/MCB native data width.
C_S<*Port_Num*>_AXI_SUPPORTS_READ	1	Integer (0, 1)	Indicates whether to include the AXI AR/R channels.
C_S<Port_Num>_AXI_SUPPORTS_WRITE	1	Integer (0, 1)	Indicates whether to include the AXI AW/W/B channels.

Ports and I/O Signals

Table 2-2 lists the available AXI Spartan-6 FPGA DDRx Memory Controller Port names, signal direction, and width.

Table 2-2: **Ports and I/O Signals**

Port Name	Direction	Width
System Signals		
sysclk_2x	Input	N/A
sysclk_2x_180	Input	N/A
pll_ce_0	Input	N/A
pll_ce_90	Input	N/A
pll_lock	Input	N/A
pll_lock_bufpll_o	Output	N/A
sysclk_2x_bufpll_o	Output	N/A
sysclk_2x_180_bufpll_o	Output	N/A
pll_ce_0_bufpll_o	Output	N/A
pll_ce_90_bufpll_o	Output	N/A
sys_rst	Input	N/A
ui_clk (same signal as mcb_drp_clk; see *Spartan-6 FPGA Memory Controller User Guide* [Ref 1])	Input	N/A
uo_done_cal	Output	N/A
AXI Signals (per port)		
s<*Port_Num*>_axi_aclk	Input	N/A
s<*Port_Num*>_axi_awid	Input	[C_s<*Port_Num*>_AXI_ID_WIDTH-1:0]
s<*Port_Num*>_axi_awaddr	Input	[C_s<*Port_Num*>_AXI_ADDR_WIDTH-1:0]
s<*Port_Num*>_axi_awlen	Input	[7:0]
s<*Port_Num*>_axi_awsize	Input	[2:0]

Table 2-2: **Ports and I/O Signals *(Cont'd)***

Port Name	Direction	Width
s<*Port_Num*>_axi_awburst	Input	[1:0]
s<*Port_Num*>_axi_awlock	Input	[1:0]
s<*Port_Num*>_axi_awcache	Input	[3:0]
s<*Port_Num*>_axi_awprot	Input	[2:0]
s<*Port_Num*>_axi_awqos	Input	[3:0]
s<*Port_Num*>_axi_awvalid	Input	N/A
s<*Port_Num*>_axi_awready	Output	N/A
s<*Port_Num*>_axi_wdata	Input	[C_S<*Port_Num*>_AXI_DATA_WIDTH-1:0]
s<*Port_Num*>_axi_wstrb	Input	[C_S<*Port_Num*>_AXI_DATA_WIDTH/8-1:0]
s<*Port_Num*>_axi_wlast	Input	N/A
s<*Port_Num*>_axi_wvalid	Input	N/A
s<*Port_Num*>_axi_wready	Output	N/A
s<*Port_Num*>_axi_bid	Output	[C_S<*Port_Num*>_AXI_ID_WIDTH-1:0]
s<*Port_Num*>_axi_bresp	Output	[1:0]
s<*Port_Num*>_axi_bvalid	Output	N/A
s<*Port_Num*>_axi_bready	Input	N/A
s<*Port_Num*>_axi_arid	Input	[C_S<*Port_Num*>_AXI_ID_WIDTH-1:0]
s<*Port_Num*>_axi_araddr	Input	[C_S<*Port_Num*>_AXI_ADDR_WIDTH-1:0]
s<*Port_Num*>_axi_arlen	Input	[7:0]
s<*Port_Num*>_axi_arsize	Input	[2:0]
s<*Port_Num*>_axi_arburst	Input	[1:0]
s<*Port_Num*>_axi_arlock	Input	[1:0]
s<*Port_Num*>_axi_arcache	Input	[3:0]
s<*Port_Num*>_axi_arprot	Input	[2:0]
s<*Port_Num*>_axi_arqos	Input	[3:0]
s<*Port_Num*>_axi_arvalid	Input	N/A
s<*Port_Num*>_axi_arready	Output	N/A
s<*Port_Num*>_axi_rid	Output	[C_s<*Port_Num*>_AXI_ID_WIDTH-1:0]
s<*Port_Num*>_axi_rdata	Output	[C_s<*Port_Num*>_AXI_DATA_WIDTH-1:0]
s<*Port_Num*>_axi_rresp	Output	[1:0]
s<*Port_Num*>_axi_rlast	Output	N/A
s<*Port_Num*>_axi_rvalid	Output	N/A
s<*Port_Num*>_axi_rready	Input	N/A
Memory Signals		
mcbx_dram_addr	Output	[C_MEM_ADDR_WIDTH-1:0]

Table 2-2: **Ports and I/O Signals *(Cont'd)***

Port Name	Direction	Width
mcbx_dram_ba	Output	[C_MEM_BANKADDR_WIDTH-1:0]
mcbx_dram_ras_n	Output	N/A
mcbx__dram_cas_n	Output	N/A
mcbx_dram_we_n	Output	N/A
mcbx_dram_cke	Output	N/A
mcbx_dram_clk	Output	N/A
mcbx_dram_clk_n	Output	N/A
mcbx_dram_dq	Input/Output	[C_NUM_DQ_PINS-1:0]
mcbx_dram_dqs	Input/Output	N/A
mcbx_dram_dqs_n	Input/Output	N/A
mcbx_dram_udqs	Input/Output	N/A
mcbx_dram_udqs_n	Input/Output	N/A
mcbx_dram_udm	Output	N/A
mcbx_dram_ldm	Output	N/A
mcbx_dram_odt	Output	N/A
mcbx_dram_ddr3_rst	Output	N/A
rzq	Input/Output	N/A
zio	Input/Output	N/A

XILINX®

Chapter 3

Debugging MCB Designs

This chapter 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. It contains these sections:

Introduction

The Spartan®-6 FPGA MCB simplifies the challenges associated with memory interface design. However, every application environment is unique and proper due diligence is still 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 chapter 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 MCB simulation model
- Design implementation verification
- Board layout verification
- Using the MCB 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

Issues might be seen in simulation and/or in hardware due to various root cause explanations. Figure 3-1 shows the overall flow for debugging problems associated with these two general types of issues.

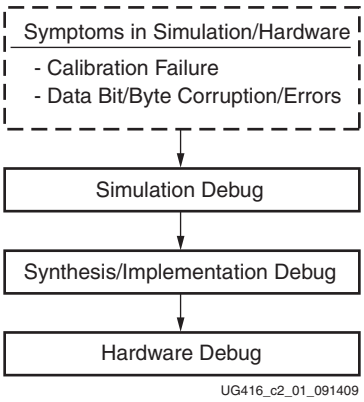


Figure 3-1: **Spartan-6 FPGA MCB Debug Flowchart**

If this chapter does not help to resolve the issue, refer to Appendix A, Additional Resources for support assistance.

Debug Tools

Many tools are available to debug memory interface design issues. This section indicates which resources are useful for debugging a given situation.

Example Design

Generation of an MCB design through the MIG tool produces an Example Design and a User Design. The Example Design includes a synthesizable test bench with a Traffic Generator that has been fully verified in simulation and hardware. This design can be used to observe the behavior of the MCB and can also aid in identifying board-related problems. Refer to MIG Example Design with Traffic Generator, page 37 for complete details on this design. This chapter further discusses using the Example Design to verify setup of a proper simulation environment and to perform hardware validation.

Debug Signals

The MIG tool includes a Debug Signals Control option on the FPGA Options screen. Enabling this feature allows all Command Path, Write Path, and Read Path signals documented in the "User (Fabric Side) Interface" section of *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] to be monitored using the ChipScope™ Analyzer. Selecting this option port maps the debug signals to the ChipScope ILA/ICON modules in the design top module. The ChipScope ILA module also sets up the default ChipScope tool trigger on the calib_done (end of calibration) and error signals (in the Example Design, the error flag from the traffic generator indicates a mismatch between actual and expected data). Chapter 1 provides details on enabling this debug feature.

Note: The default MIG design monitors all signals only in the first enabled user interface port irrespective of whether the port is bidirectional or unidirectional. Part of the debug bus is left unconnected so that users can connect and monitor the desired signals.

Reference Boards

SP601 and SP605 are Xilinx development boards that interface the MCB to external DDR2 and DDR3 memory devices, respectively. These boards are fully validated and can be used to test user designs and analyze board layout.

ChipScope Pro Tool

The ChipScope Pro tool inserts logic analyzer, bus analyzer, and virtual I/O software cores directly into the design. The ChipScope Pro tool allows the user to set trigger conditions to capture application and MCB port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool [Ref 5].

Simulation Debug

Figure 3-2 shows the debug flow for simulation.

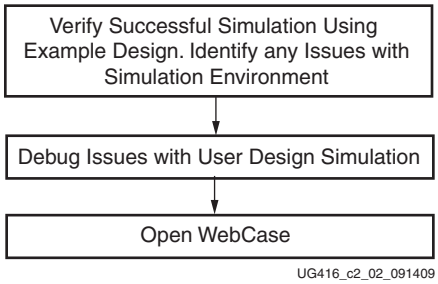


Figure 3-2: **Simulation Debug Flowchart**

Additional Debug Signals (Simulation Only)

The UNISIM model of the MCB primitive within the top-level MIG wrapper is encrypted, preventing access to internal nodes. However, some additional signals that might be useful in simulation debug have been made accessible by bringing them to the top level of the UNISIM model. These signals can only be viewed in simulation (see Figure 3-3); they are not accessible in hardware. The signals are located in the hierarchy path `*/memc*_mcb_raw_wrapper_inst/samc_0/B_MCB_INST`.

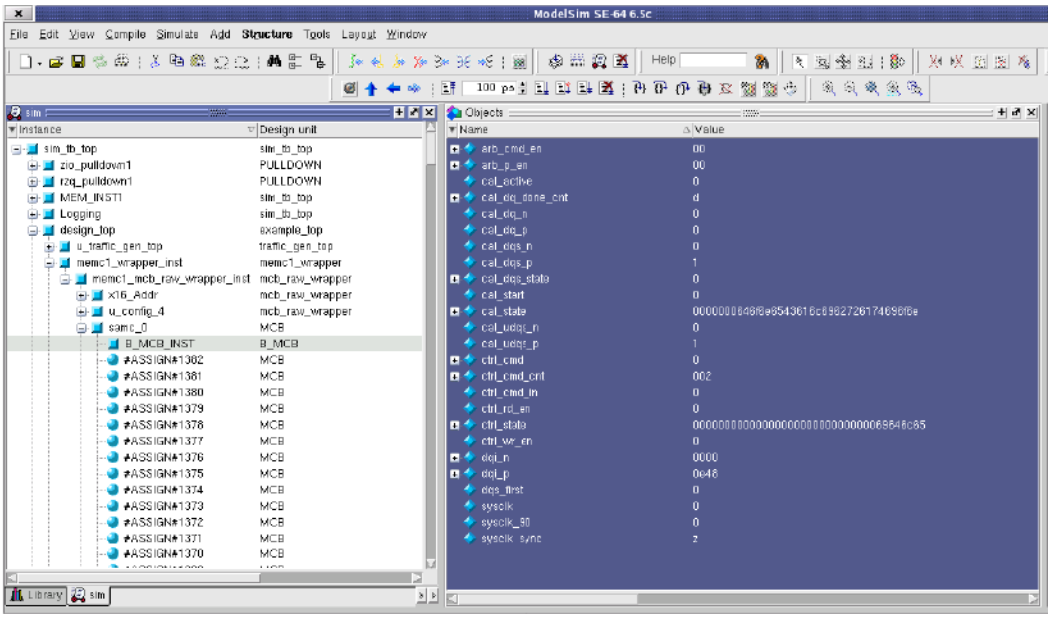


Figure 3-3: **Simulation Debug Signals Inside MCB in ModelSim Environment.**

Table 3-1 lists the available simulation debug signals.

Table 3-1: **Simulation Debug Signals**

Block Domain	Internal Signal Name	Description	Clock Domain
Clocks	sysclk	Internally generated clock from sysclk2x, 0° phase shift.	N/A
	sysclk_90	Internally generated clock from sysclk2x, 90° phase shift.	N/A
Controller	ctrl_state[144:0]	Controller state (see Table 3-2) ASCII radix.	sysclk90
	ctrl_rd_en	Controller read enable.	sysclk90
	ctrl_wr_en	Controller write enable.	~sysclk90
	ctrl_cmd_in	Controller input command flag from the arbiter or calibration logic.	~sysclk90
	ctrl_cmd[2:0]	Controller command received.	~sysclk90
	ctrl_cmd_cnt[9:0]	Controller current command count. This bus indicates the number of times to execute the current command.	~sysclk90
Arbiter and Data Capture	arb_cmd_en[5:0]	Arbiter enable to command FIFO.	~sysclk90
	arb_p_en[7:0]	Arbiter enable to data FIFO.	~sysclk90
	dqi_p[15:0]	Single data rate DQ bus between capture blocks and data FIFOs, rising edge.	sysclk90
	dqi_n[15:0]	Single data rate DQ bus between capture blocks and data FIFOs, falling edge.	sysclk90
	sysclk_sync	First valid data on DQ bus. It is registered on the next sysclk_90 edge.	N/A
	dqs_first	First edge of DQS occurred. This signal indicates start of read capture cycle.	N/A

Table 3-1: **Simulation Debug Signals *(Cont'd)***

Block Domain	Internal Signal Name	Description	Clock Domain
Calibration	cal_start	Start calibration. This pin forces the start of a calibration cycle.	ui_clk
	cal_active	Calibration currently running.	sysclk90
	cal_dq_done_cnt[3:0]	Current DQ signal calibrating.	sysclk90
	cal_state[144:0]	Calibration state (see Table 3-3) ASCII radix.	sysclk90
	cal_dqs_state[2:0]	DQS Calibration state. The states proceed from 0 to 7 in numerical order.	sysclk90
	cal_dqs_p	Single data rate DQSP. Should be all 1's during calibration.	sysclk90
	cal_dqs_n	Single data rate DQSN. Should be all 0's during calibration.	sysclk90
	cal_udqs_p	Single data rate UDQSP. Should be all 1's during calibration.	dqs_ioi_m
	cal_udqs_n	Single data rate UDQSN. Should be all 0's during calibration.	dqs_ioi_m
	cal_dq_p	Single data rate DQP selected by cal_dq_done_cnt. Should be all 1's during calibration.	sysclk90
	cal_dq_n	Single data rate DQN selected by cal_dq_done_cnt. Should be all 0's during calibration.	~sysclk90

Table 3-2: **FSM State Definitions for ctrl_state**

State	Description
0x00	Idle
0x01	Load Mode Register
0x02	Mode Register Wait
0x03	Precharge
0x04	Precharge Wait
0x05	Auto Refresh
0x06	Auto Refresh Wait
0x07	Active
0x08	Active Wait
0x09	First Read
0x0A	Burst Read
0x0B	Read Wait
0x0C	First Write
0x0D	Burst Write
0x0E	Write Wait
0x0F	Init Count 200

Table 3-2: **FSM State Definitions for ctrl_state *(Cont'd)***

State	Description
0x10	Init Count 200 Wait
0x11	ZQCL
0x12	Write Read
0x13	Read Write
0x14	Dummy First Read
0x15	Deep Memory State
0x16	Jump State
0x17	Init Done
0x18	Reset
0x19	Reset Wait
0x1A	Precharge All
0x1B	Precharge All Wait
0x1C	Self Refresh Enter
0x1D	Self Refresh Wait
0x1E	Self Refresh Exit
0x1F	Self Refresh Exit Wait

Table 3-3: **FSM State Definitions for cal_state**

State	Description
0x00	Init
0x02	Reset DRP interface
0x16	Preamble Pulldown
0x17	Preamble Read
0x18	Preamble Undo
0x01	Calibrate DRP/IOI
0x03	Issue Write Command
0x04	Wait for Write Command
0x05	Issue Read Command
0x06	Wait for Read Command
0x07	Wait for DRP Interface
0x08	DQS Calibration
0x09	Pre Done Calibration
0x0E	Done Calibration

Verify Simulation using the Example Design

The Example Design generated by the MIG tool includes a simulation test bench, appropriately set up the memory model and parameter file based on memory selection in the MIG tool, and a ModelSim `.do` script file. Refer to MIG Example Design with Traffic Generator, page 37 for detailed steps on running the Example Design simulation.

Successful completion of this Example Design simulation verifies a proper simulation environment. This shows that the simulation tool and Xilinx libraries are set up correctly. For detailed information on setting up Xilinx libraries, refer to COMPXLIB in the *Command Line Tools User Guide* [Ref 6] and the *Synthesis and Simulation Design Guide* [Ref 4]. For simulator support and detailed information on the MCB simulation model, refer to Simulation Debug.

A working Example Design simulation completes memory initialization and runs traffic in response to the Traffic Generator stimulus. Successful completion of memory initialization and calibration results in the assertion of the calib_done signal. When this signal is asserted, the Traffic Generator takes control and begins executing writes and reads according to its parameterization. Refer to MIG Example Design with Traffic Generator for details on the available Traffic Generator data patterns and corresponding top-level parameters.

Synthesis and Implementation Debug

Figure 3-4 shows the debug flow for synthesis and implementation.

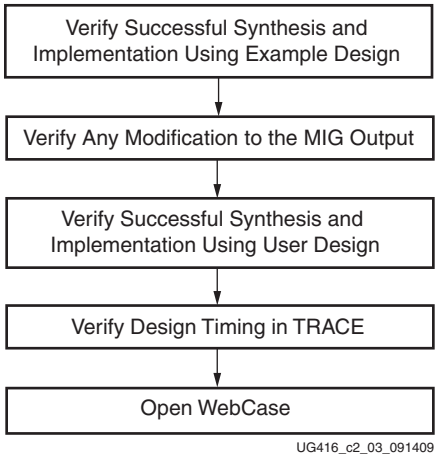


Figure 3-4: **Synthesis / Implementation Debug Flowchart**

Verify Successful Synthesis and Implementation

The Example Design and User Design generated by the MIG tool include synthesis/implementation script files and User Constraint Files (`.ucf`). These files should be used to properly synthesize and implement the targeted design and generate a working bitstream. The synthesis/implementation script file, called `ise_flow.bat`, is located in both `example_design/par` and `user_design/par` directories. Execution of this script runs either the Example Design or the User Design through Synthesis, Translate, MAP, PAR, TRACE, and BITGEN. The options set for each of these processes are the only options that have been tested with the MCB MIG design. A successfully implemented design completes all processes with no errors (including zero timing errors).

Verify Modifications to the MIG Output

The MIG tool allows the user to select which MCB to use for a particular memory interface. Based on the selected MCB, the MIG tool outputs a `.ucf` file with all required pin location constraints. This file is located in both `example_design/par` and `user_design/par` directories and should not be modified. The selected pins are required to properly interface to the MCB.

The MIG tool outputs an MCB wrapper file. This file should not be modified. Modifications are not supported and should be verified independently in behavioral simulation, synthesis, and implementation.

Identifying and Analyzing Timing Failures

The MCB design has been verified to meet timing. If timing violations are encountered, it is important to isolate the timing errors. The timing report output by TRACE (`.twx/.twr`) should be analyzed to determine if the failing paths exist in the MIG MCB design or the user interface to the MIG MCB design. If failures are encountered, the user must ensure the build (that is, XST, MAP, PAR) options specified in the `ise_flow.bat` file are used.

If failures still exist, Xilinx has many resources available to aid to closing timing. The PlanAhead™ tool [Ref 7] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 8] provides valuable information on all available Xilinx constraints.

Hardware Debug

Figure 3-5 shows the debug flow for hardware.

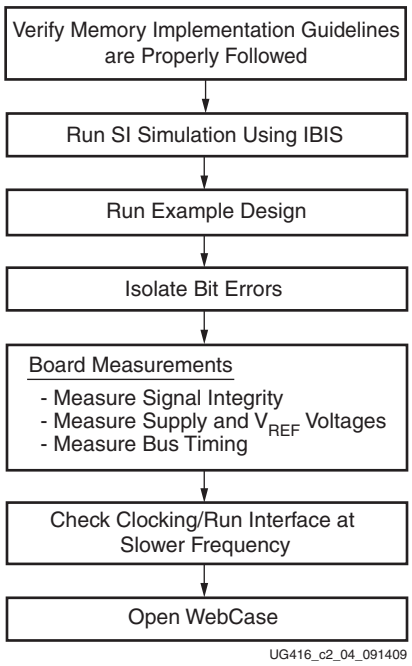


Figure 3-5: **Hardware Debug Flowchart**

Verify Memory Implementation Guidelines

See the "PCB Layout Guidelines" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1] for specifications on pinout guidelines, termination, I/O standards, and trace matching. The guidelines provided are specific to both memory technologies as well as MIG output designs. It is important to verify that these guidelines have been read and considered during board layout. Failure to follow these guidelines can result in problematic behavior in hardware as discussed in this chapter.

Clocking

The external clock source should be measured to ensure frequency, stability (jitter), and usage of the expected FPGA pin.

The designer must ensure that the design follows all clocking guidelines as outlined in the "Clocking" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1]. If clocking guidelines have been followed, the next step is to run the interface at a slower speed. Unfortunately not all designs/boards can accommodate this step. Lowering the frequency increases marginal setup time and/or hold time due to PCB trace mismatch, poor SI, or excessive loading.

Verify Board Pinout

The board schematic needs to be compared to the `<design_name>.pad` report generated by Place and Route. This step ensures the board pinout matches the pins assigned in the implemented design.

Note: The pin LOCs selected in the MIG output UCF are required to properly interface to the MCB and cannot be modified.

Run Signal Integrity Simulation with IBIS Models

To verify that board layout guidelines have been followed, signal integrity simulations must be run using IBIS. These simulations should always be run both pre-board and post-board layouts. The purpose for running these simulations is to confirm the signal integrity on the board.

The ML561 Hardware-Simulation Correlation chapter of the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 9] can be used as a guideline. This chapter provides a detailed look at signal integrity correlation results for the ML561 board and can be used as an example for what to look at and what is good to see. 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 can be applied to a Spartan-6 FPGA MCB design.

Run the Example Design

The MIG provided example design is a fully verified design that can be used to test the memory interface on the board. It rules out any issues with the user's backend logic interfacing with the MCB. In addition, the traffic generator provided by the MIG tool can be parameterized to send out different data patterns that test different board-level concerns. For example, a Hammer pattern stresses the memory interface for simultaneous switching outputs (SSO), while a "Walking 1s" or "Walking 0s" pattern tests if each memory DQ bit can be set to 1 and 0, independent of other bits. See MIG Example Design with Traffic Generator, page 37 for full details on the available data patterns.

Debugging Common Hardware Issues

When calibration failures and data errors are encountered in hardware, the ChipScope Analyzer should be used to analyze the behavior of datapath signals. The MIG tool provides the Debug Signals for Memory Controller feature to aid in this analysis. When this option is enabled in the MIG tool, the output example_design and user_design include ChipScope Generator ILA and ICON core instantiations. When the example_design is used in hardware, the `example_design/rtl/example_top.v` module should be referenced. When the user_design is used in hardware, the `user_design/rtl/<component_name>_debug_en.v` module should be referenced.

To analyze the signals mapped to the ILA core, first the designer should run the `ise_flow.bat` file located in the appropriate output par directory to generate a bitstream. This step properly generates the ChipScope tool cores and includes them in the output bitstream. Next the designer should open ChipScope Analyzer and configure the device.

Note: For detailed information on using ChipScope Analyzer, refer to the *ChipScope Pro Software and Cores User Guide*.

After configuration, the ChipScope Analyzer tool is loaded with Data and Trigger windows. The ports in these windows are listed as DataPort and TriggerPort signals. To

properly port map (name) these signals, the designer should open the appropriate rtl file noted in the above paragraph and view the assignments of the cx_dbg_data and cx_dbg_trig signals. Right-click the individual DataPort and TriggerPort signals to rename them to the appropriate data or trigger signal names.

The data signals assigned in the debug port include the command path, write datapath, and read datapath signals as defined in the "Interface Details" section in *Spartan-6 FPGA Memory Controller User Guide* [Ref 1]. The trigger signals assigned in the trigger port include the calib_done and error (example_design only) signals. The trigger can be asserted separately on calib_done to debug calibration failures and error to debug data errors after calibration. While analyzing data errors, refer to Isolating Bit Errors to determine where the error occurs.

A good starting point in hardware debug is to load the provided example_design with the Debug Signals for Memory Controller feature enabled onto the designer's board. This known working solution with a traffic generator design checks for data errors. This design should complete successfully with the assertion of calib_done and no assertions of error. Assertion of calib_done signifies calibration completion while no assertions of error signifies the data written to and read from the memory compare with no data errors. The designer should run the example_design on the board twice: once with the trigger in the trigger window set to calib_done and once set to error.

Isolating Bit Errors

An important hardware debug step is to try to isolate when and where the bit errors are occurring. Looking at the bit errors, these should be identified:

- Are errors seen on data bits belonging to certain DQS groups?
- Are the errors on accesses to certain addresses or banks?
- Do the errors only occur for certain data patterns or sequences?

  This case might indicate a shorted or open connection on the PCB. It can also indicate an SSO or crosstalk issue.

It might be necessary to isolate whether the data corruption is due to writes or reads. This case can be difficult to determine because if writes are the cause, read back of the data is bad as well. In addition, issues with control/address timing affect both writes and reads.

To try to isolate the issue:

- If the errors are intermittent, have the controller issue a small initial number of writes, followed by continuous reads from those locations. If the reads intermittently yield bad data, there is a read issue.
- If on-die termination is used, check that the correct value is enabled in the memory device, and that the timing on the ODT signal relative to the write burst is correct.

Board Measurements

The signal integrity of the board and bus timing must be analyzed. The ML561 Hardware-Simulation Correlation chapter in the *Virtex-5 FPGA ML561 Memory Interfaces Development Board User Guide* [Ref 9] describes expected bus signal integrity. While this guide is specific to Virtex-5 devices and the ML561 Development Board, the principles can be applied to a Spartan-6 FPGA MCB design.

Another important board measurement is the reference voltage levels. It is important that these voltage levels are measured when the bus is active. These levels can be correct when the bus is idle, but might droop when the bus is active.

XILINX®

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

The Solution Center specific to Virtex®-6 FPGA Memory Interface Solutions is located at Xilinx MIG Solution Center.

References

These references provide supplemental information useful for this document:

1. UG388, *Spartan-6 FPGA Memory Controller User Guide*
2. UG683, *EDK Concepts, Tools, and Techniques*
3. UG111, *Embedded System Tools Reference Manual*
4. UG626, *Synthesis and Simulation Design Guide*
5. ChipScope™ Pro Logic Analyzer tool //www.xilinx.com/tools/cspro.htm
6. UG628, *Command Line Tools User Guide*, COMPXLIB
7. PlanAhead™ Design Analysis tool //www.xilinx.com/tools/planahead.htm
8. UG612, *Xilinx Timing Constraints User Guide*
9. UG199, *Virtex®-5 FPGA ML561 Memory Interfaces Development Board User Guide*
10. ARM® AMBA® Specifications http://www.arm.com/products/system-ip/amba/amba-open-specifications.php