Spartan-6 FPGA Memory Interface Solutions

User Guide

UG416 (v1.2) March 3, 2010

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 1-29. Revised the text below the Calibrated Input Termination bullet on page 23. Added Xilinx ISim to first paragraph in Functional Simulation, page 39.
3/3/10	1.2	Added note about device migration to page 24.

Table of Contents

Chapter 2: Debugging MCB Designs

About This Guide

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 UG388, *Spartan-6 FPGA Memory Controller User Guide*.

Guide Contents

This manual contains the following chapters:

- Chapter 1, Getting Started, describes how to use the MIG tool available in the CORE Generator™ tool to implement a memory interface based on the MCB.
- Chapter 2, Debugging MCB Designs, 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.

References

The following Xilinx references provide additional information useful for this document:

1. UG626, *Synthesis and Simulation Design Guide*
2. ChipScope™ Pro Logic Analyzer tool
   http://www.xilinx.com/tools/cspro.htm
3. UG628, *Command Line Tools User Guide*, COMPXLIB
4. PlanAhead™ Design Analysis tool
   http://www.xilinx.com/tools/planahead.htm
5. UG612, *Xilinx Timing Constraints User Guide*
6. UG199, *Virtex®-5 FPGA ML561 Memory Interfaces Development Board User Guide*

Additional Resources

To find additional documentation, see the Xilinx website at:

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

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

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

XILINX®

Chapter 1

Getting Started

This chapter describes how to use the MIG tool available in the CORE Generator™ tool to implement a memory interface based on the MCB. It contains these sections:

- MIG Overview
- System Requirements
- MIG Tool: Step-by-Step Instructions
- MIG Directory Structure and File Descriptions
- MIG Example Design with Traffic Generator

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. 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 testbench to verify or demonstrate the MCB based memory interface in a simulation or hardware environment. 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 33 for more information.

System Requirements

The MIG tool requires this system configuration:

- Operating System
  - Windows XP Professional 32-bit/64-bit
  - Linux Red Hat Enterprise 4.0 32-bit/64-bit
  - Linux Red Hat Enterprise 5.0 Desktop 32-bit/64-bit
  - Windows Vista Business 32-bit/64-bit
  - SUSE 10 Enterprise 32-bit/64-bit
- Xilinx Software
  - ISE® tool, version 11.1 or greater with applicable Service Pack

Check the MIG release notes for the required service pack. ISE tool service packs can be downloaded from the Xilinx Download Center.

MIG Tool: Step-by-Step Instructions

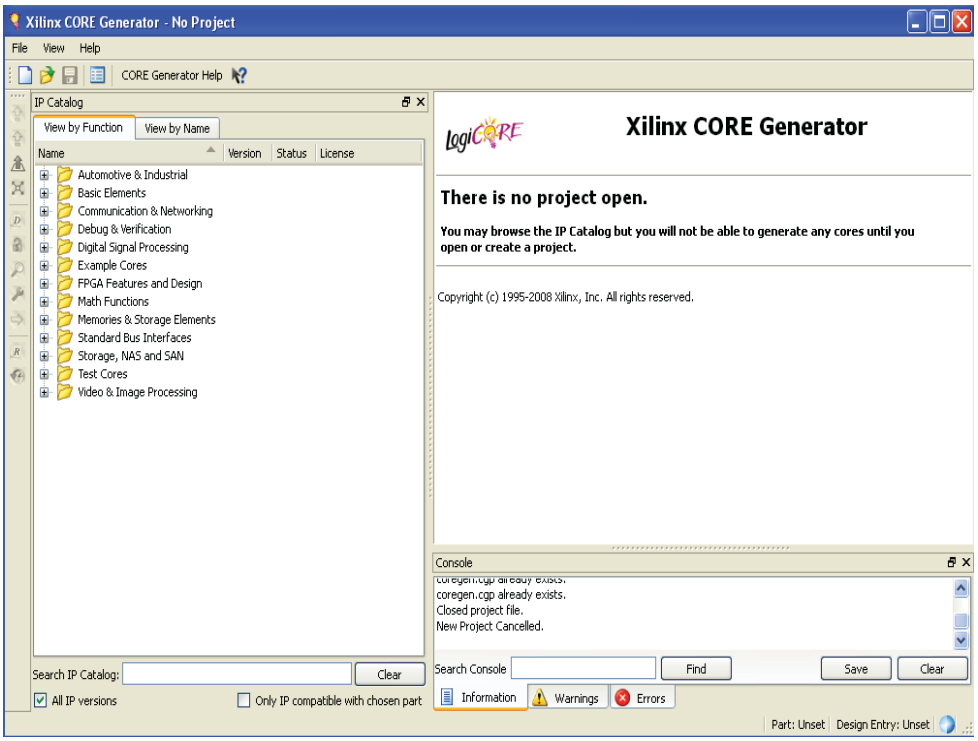
This section provides a walkthrough of the MIG GUI, detailing all necessary steps for implementing a Spartan-6 FPGA memory interface based on the MCB.

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.

Setting up a New Project

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

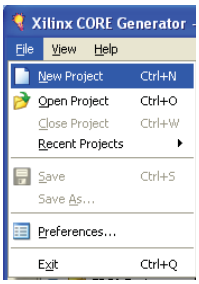
1. The CORE Generator system is launched by selecting **Start** → **Xilinx ISE Design Suite 11** → **ISE** → **Accessories** → **CORE Generator**.



UG416_c1_01_091409

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

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



UG416_c1_02_091409

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**.

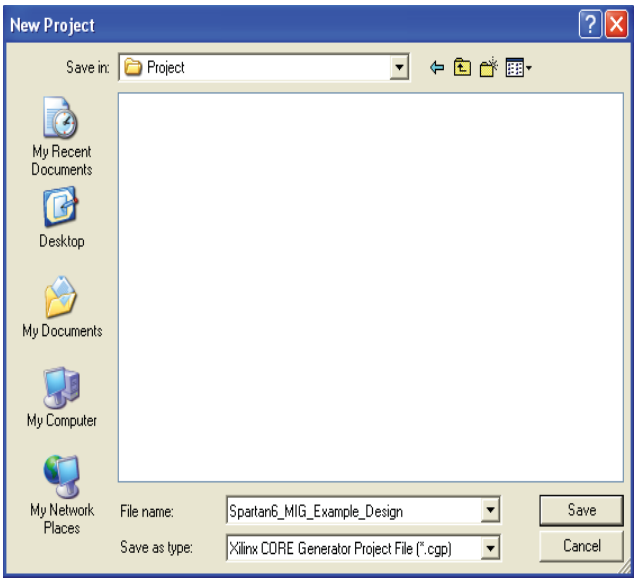


Figure 1-3: **Naming the New Project**

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.

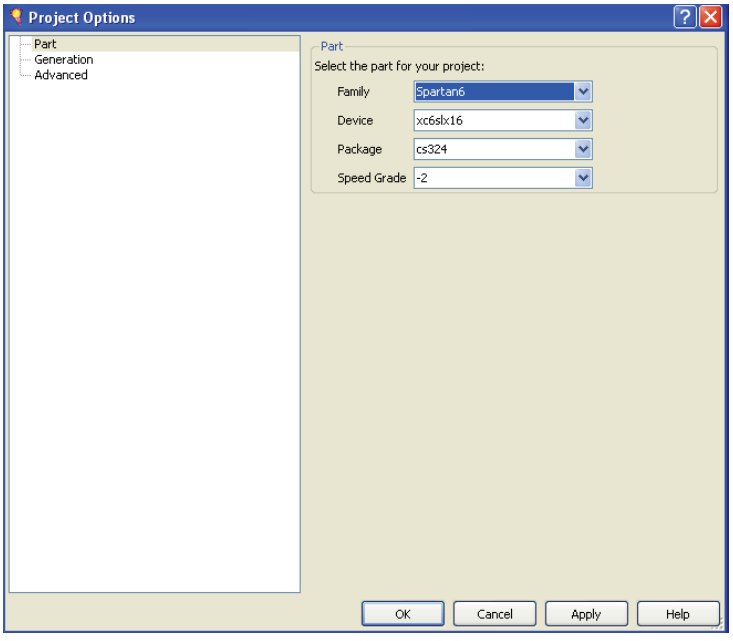


Figure 1-4: **Device Selection**

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

   Note: Currently, Verilog is the only supported design flow for the Spartan-6 FPGA Memory Controller Block.

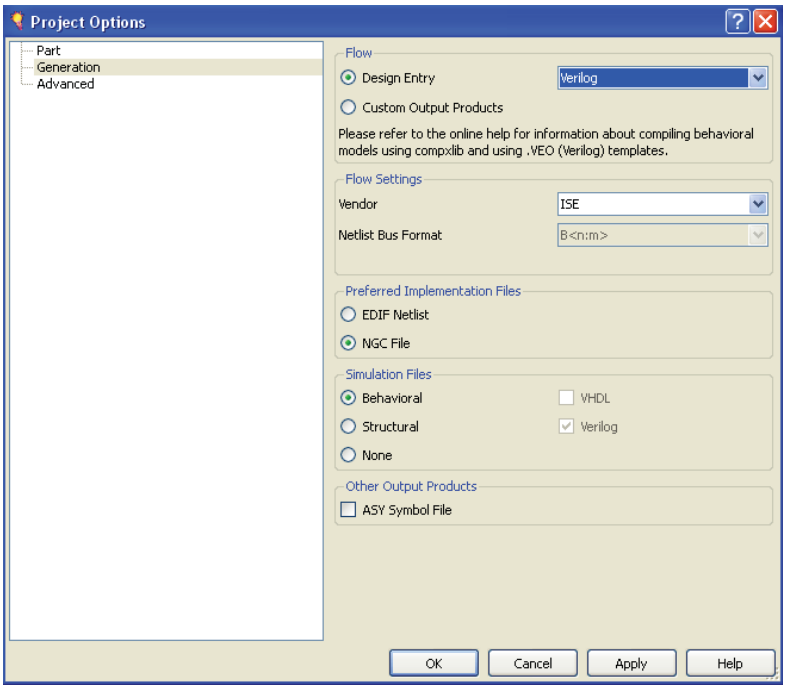


Figure 1-5: **Setting Up the Design Flow Options**

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.

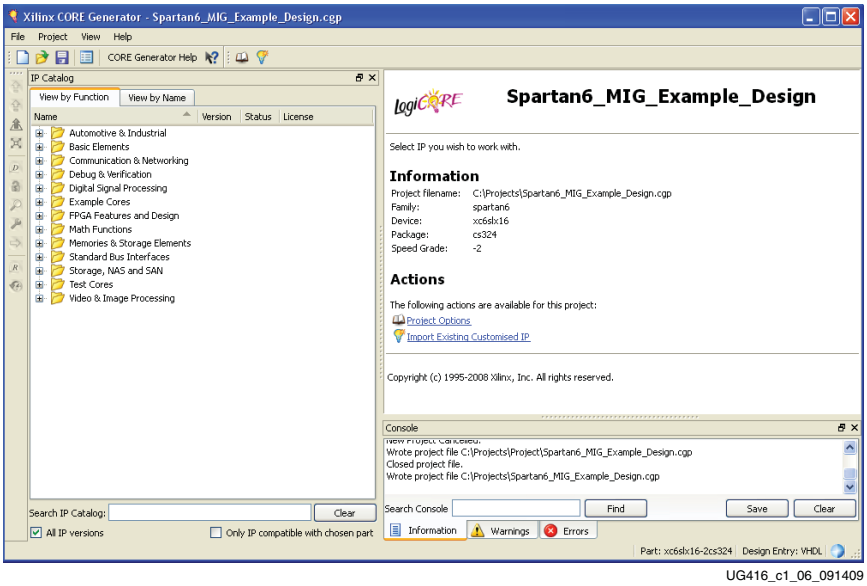
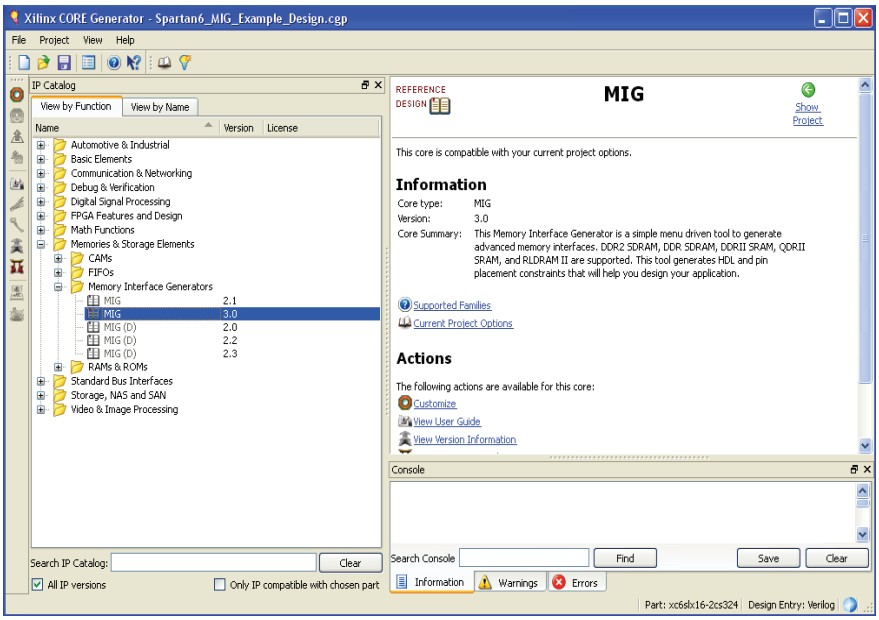


Figure 1-6: **Project Page**

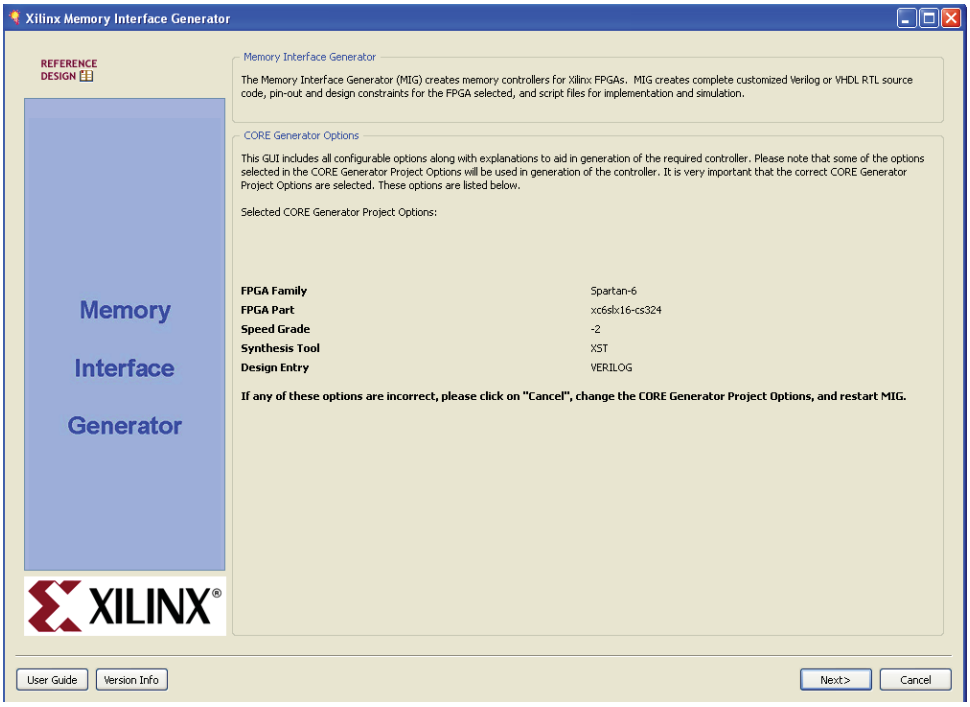
2. Expand the `Memories & Storage Elements` folder in the left window to access the MIG tool. Double-click the latest version under **Memories & Storage Elements** → **Memory Interface Generators** → **MIG** to launch the MIG tool.



UG416_c1_07_091409

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

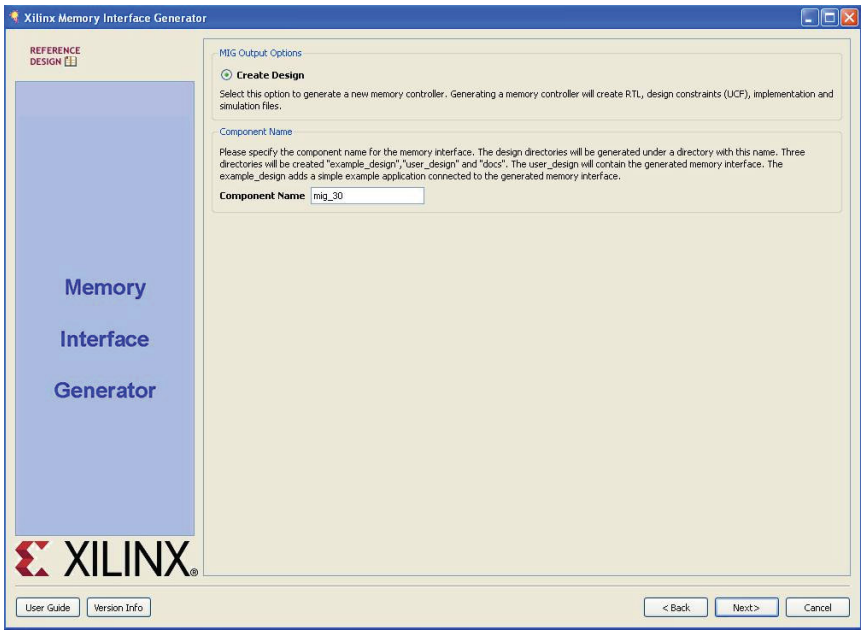
3. When the MIG tool startup page appears, click **Next** to advance to the MIG Output Options page.



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>`.

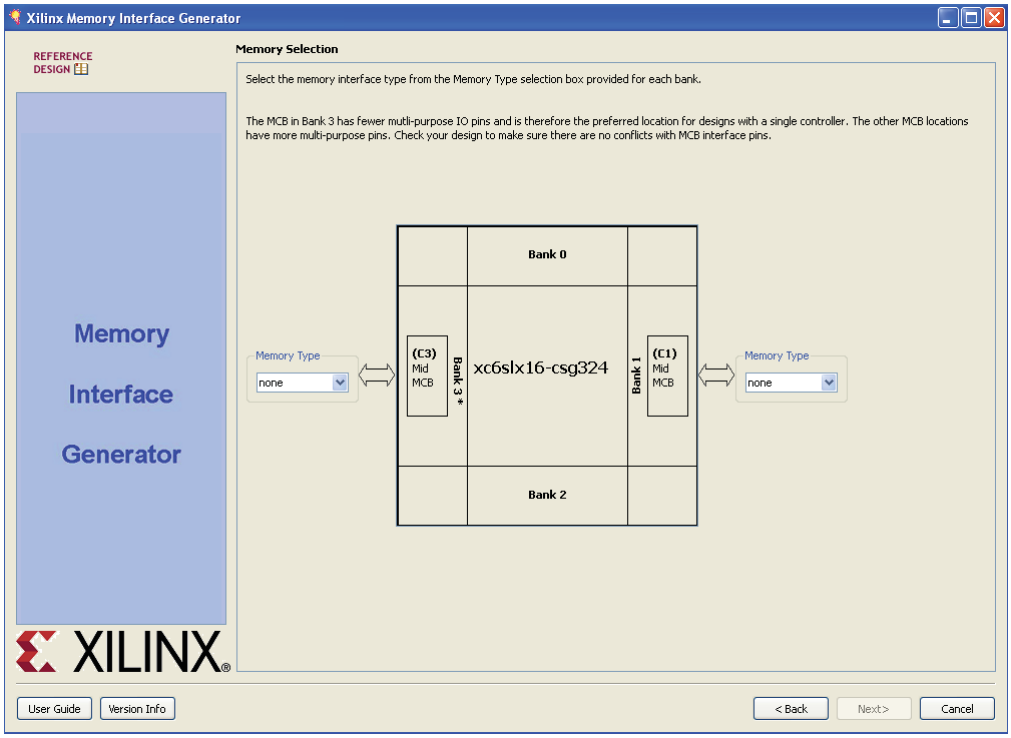
Note: <Component Name> cannot contain special characters. Only alphanumeric characters can be used to specify a component name, and the name should always start with a letter of the alphabet (a-z).

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.

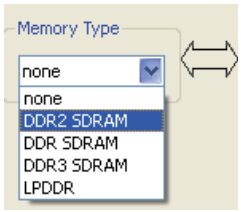


UG416_c1_10_102709

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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, for more details on MCB clocking requirements.

1. Select a memory standard from the Memory Type drop-down menu for each MCB that implements a memory interface. Click **Next**.



UG416_c1_11_091409

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

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:

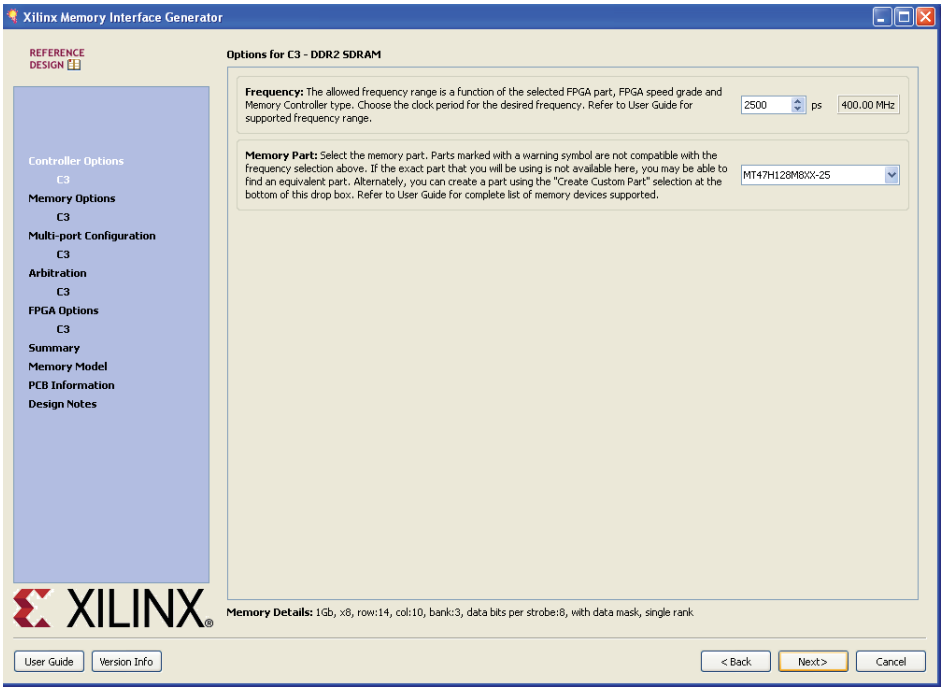
- Frequency

  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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, 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.

- Memory Part

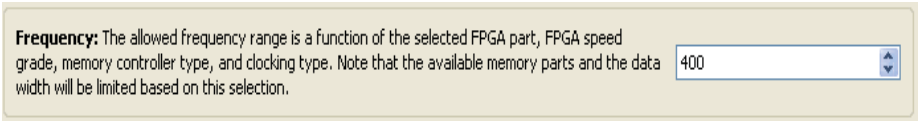
  This option selects a target memory device for the design. The selection can be made from the existing list of supported devices or a new custom device can be created.



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.

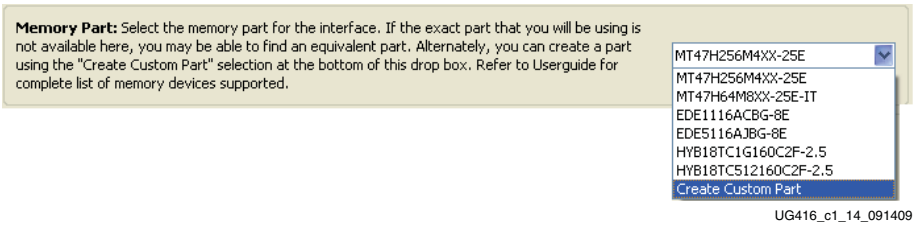


Figure 1-14: **Selecting a Memory Device**

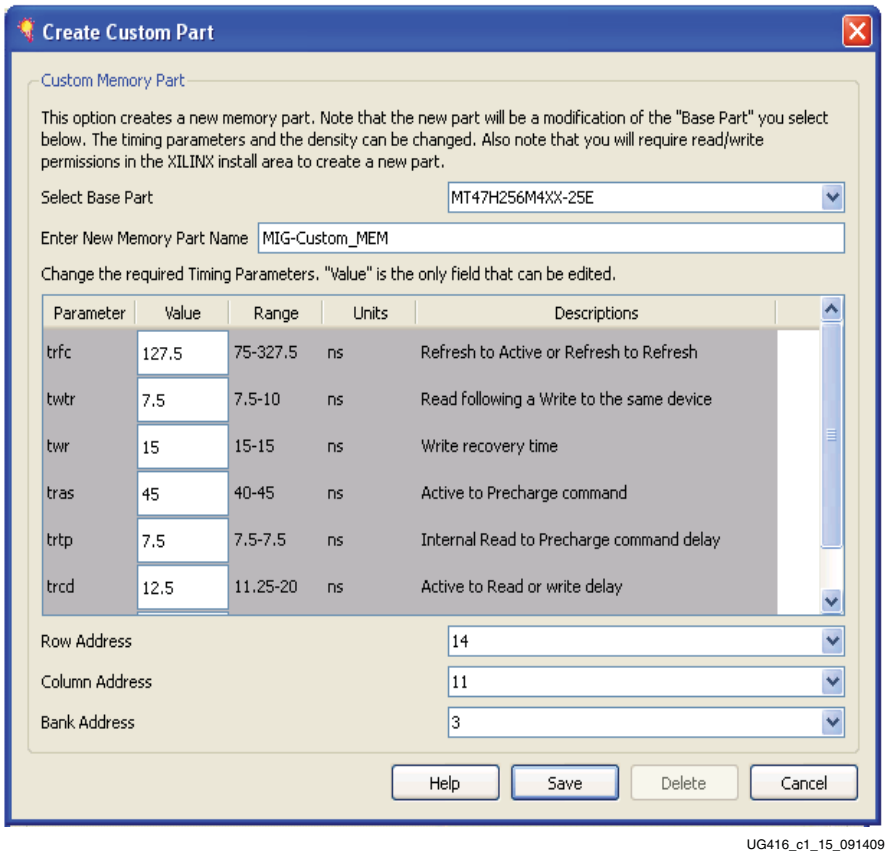


Figure 1-15: **Creating a Custom Memory Device**

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, first select a supported device in the Select Base Part drop-down menu that has similar parameters to the device being created. Enter a name for the new device in the Enter New Memory Part Name box (for example, **MIG-Custom_MEM**). Edit the timing parameter values and row/column/bank address bit counts as needed. Then 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.

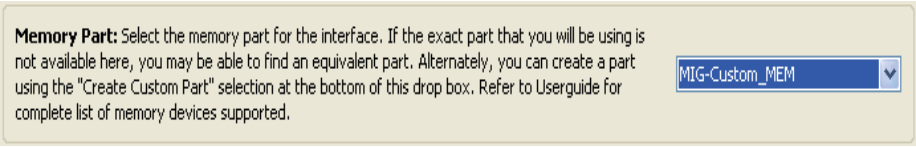


Figure 1-16: **Selecting a Custom Memory Device from the Memory Part List**

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.

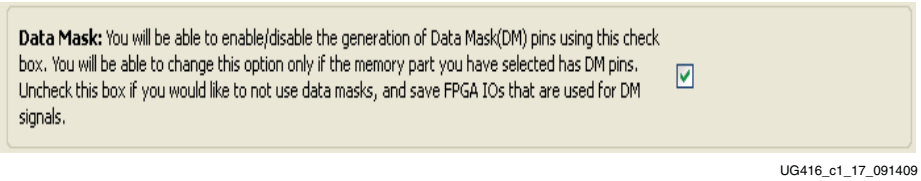


Figure 1-17: **Setting the Data Mask Option**

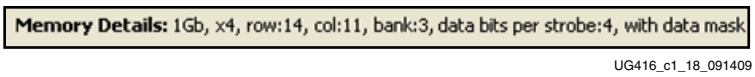
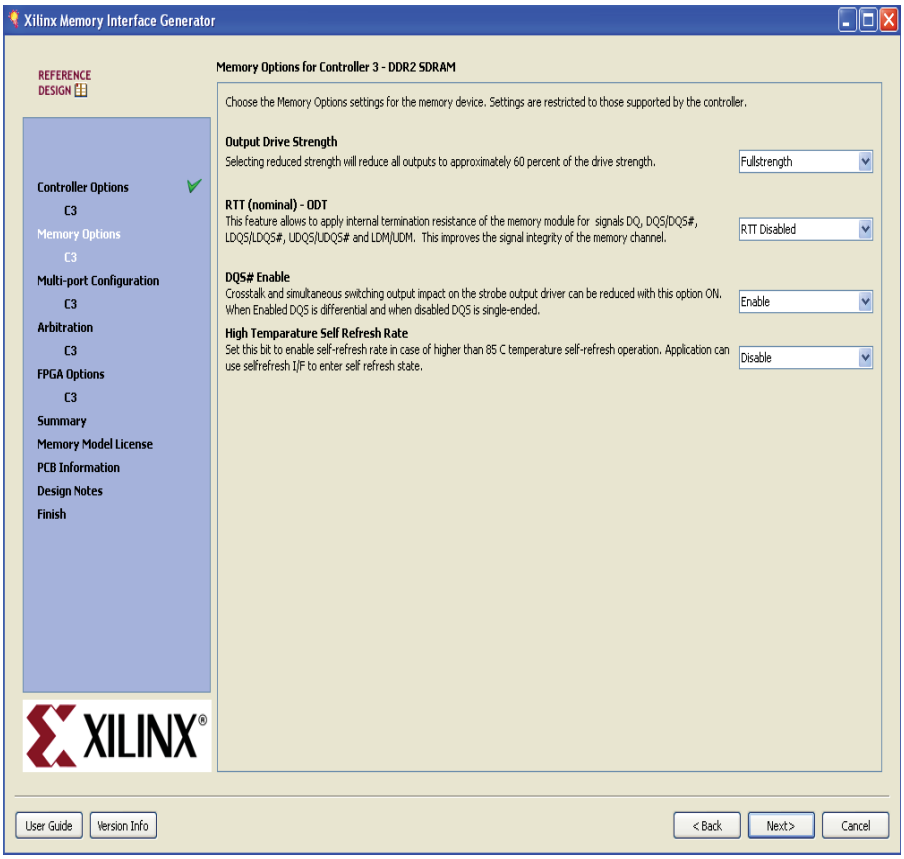


Figure 1-18: **Memory Configuration Details**

Setting Memory Device Options

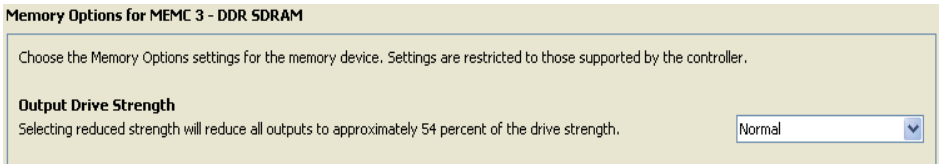
The Memory Options page is used to set up various mode register values that are loaded into the memory device during initialization. The list of available settings is determined by the memory standard selected for the interface. Burst length and CAS latency are automatically set by the MIG tool to offer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

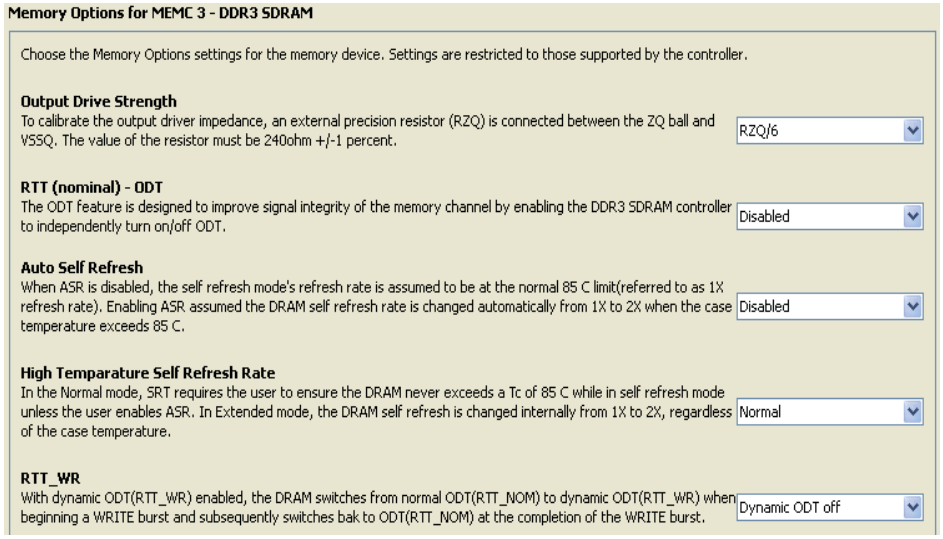


Figure 1-21: **Mode Register Settings for the DDR3 Controller**

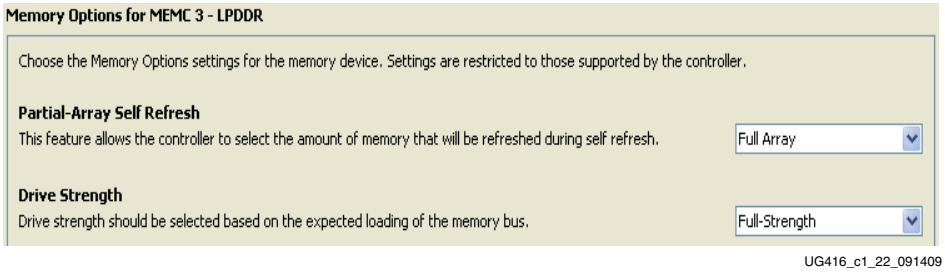
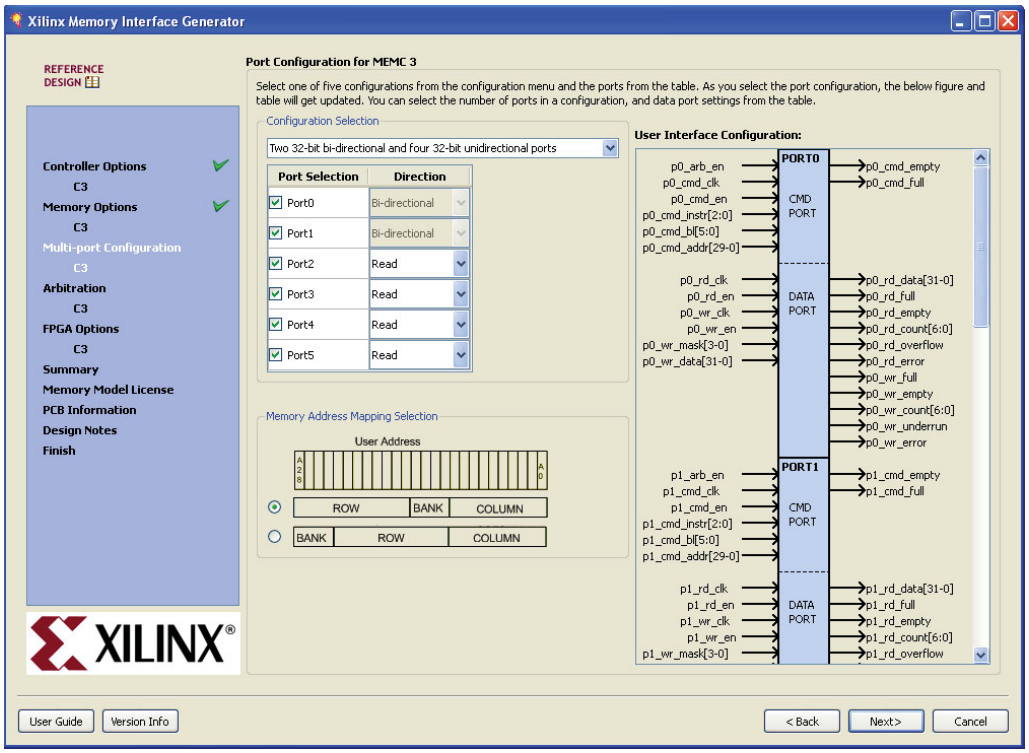


Figure 1-22: **Mode Register Settings for the LPDDR Controller**

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

Multi-Port Configuration

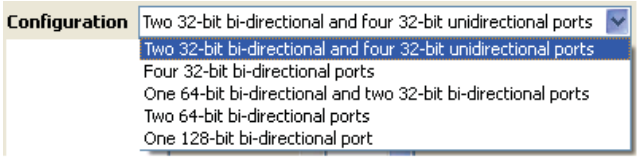
There are five possible port configurations for the MCB User Interface to the FPGA logic. For details on port configurations, see the "Port Configurations" section in UG388, *Spartan-6 FPGA Memory Controller User Guide*. Based on the selected configuration, the MIG tool generates the necessary user signal names and assignments in the top-level wrapper file.



UG416_c1_23_091409

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.

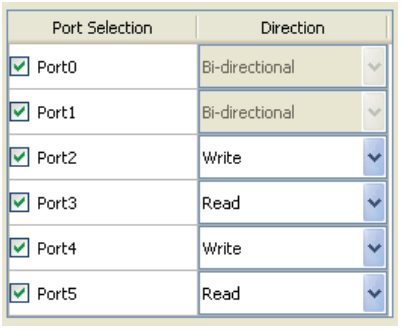


Figure 1-25: **Setting Port Selection and Direction**

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 now shows a summary of the completed multi-port configuration (see Figure 1-27). Click **Next** to proceed to arbitration setup.

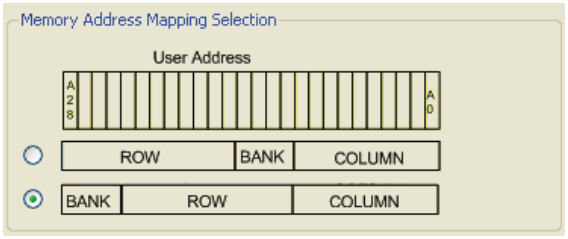


Figure 1-26: **Selecting the Memory Address Mapping Scheme**

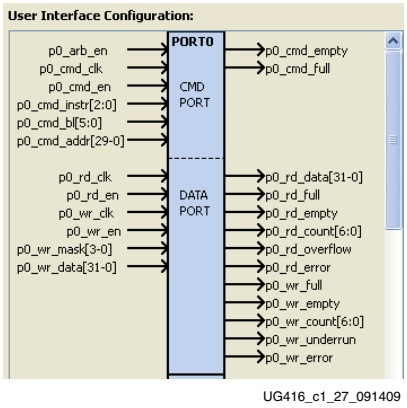


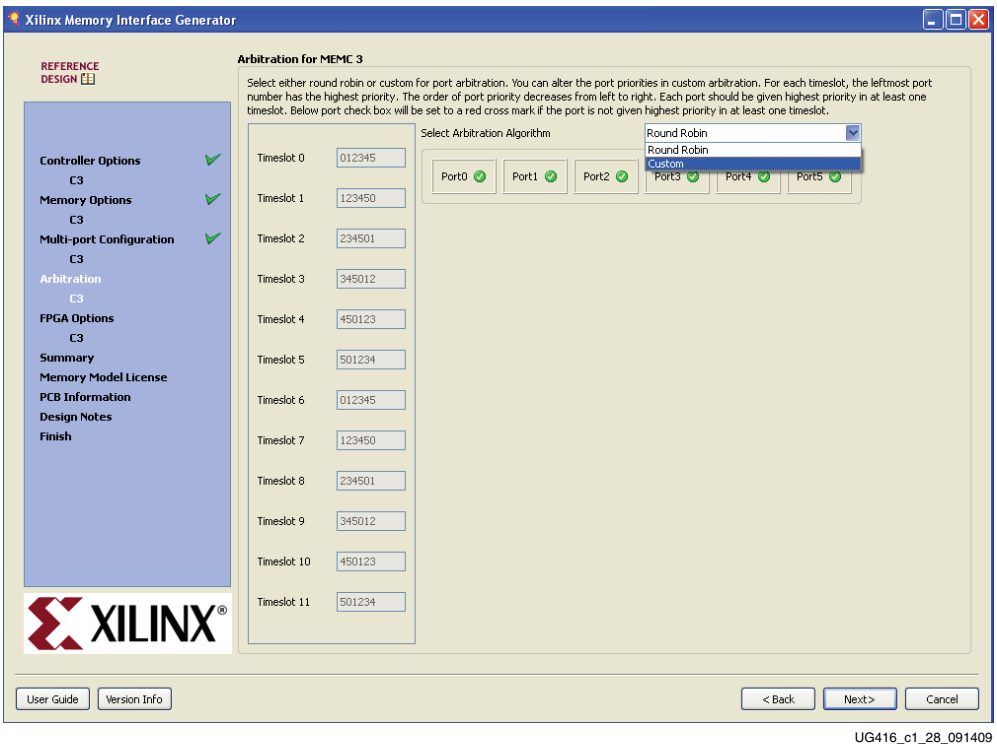
Figure 1-27: **User Interface Port Configuration**

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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, for more information on arbitration.

10. 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**.

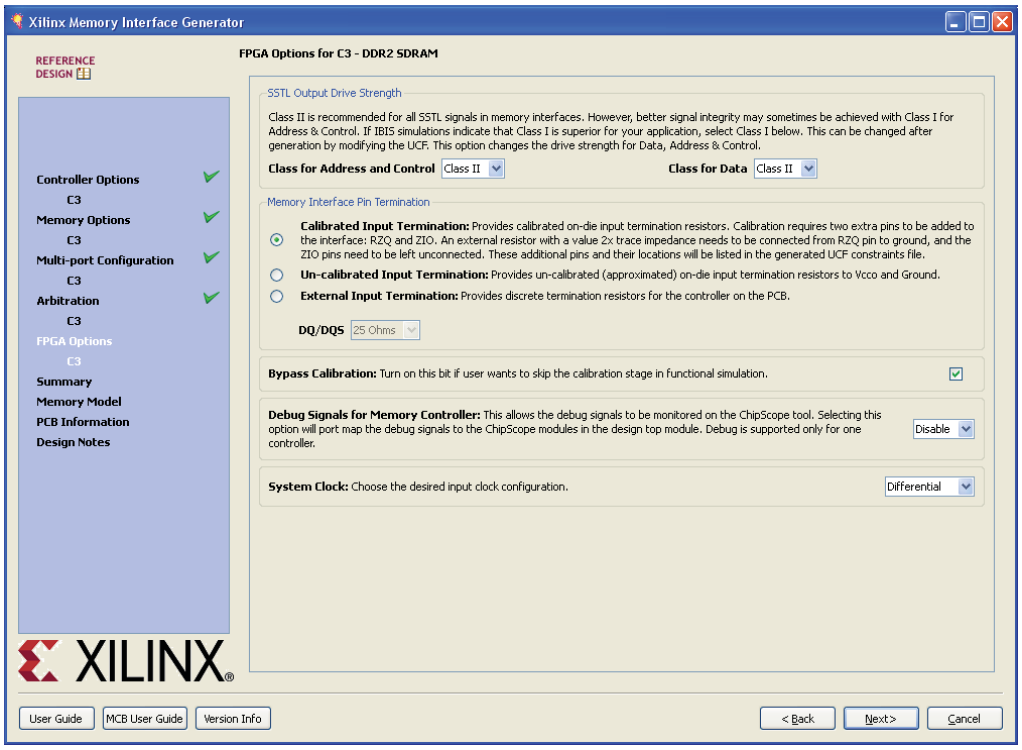


UG416_c1_28_091409

Figure 1-28: **Setting Up the Arbitration Scheme**

Setting FPGA Options

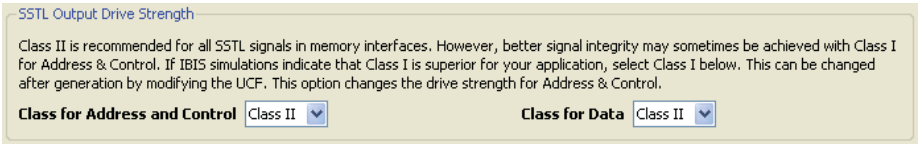
The FPGA Options page is used to configure some remaining aspects of the memory interface solution.



UG416_c1_29_021710

Figure 1-29: **FPGA Options Page**

11. 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 1-30: **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 (e.g., a 100Ω resistor to achieve a 50Ω effective input termination). RZQ should be left as a no-connect (NC) pin for designs not using calibrated input termination. In addition, the RZQ pin must be within the same I/O bank as the memory interface pins.

The ZIO pin is only required for designs using calibrated input termination and must be a no-connect pin (e.g., not connected to any PCB trace) assigned to a valid package pin (e.g., 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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, for more details on using calibrated input termination.

Note: If device migration is desired (i.e., 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 file.

- 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.

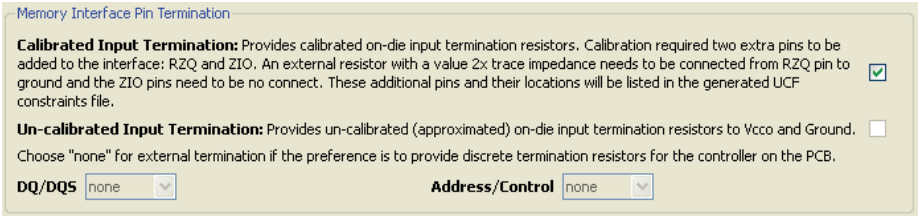


Figure 1-31: **Selecting the Method of Input Termination**

12. 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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, 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.

13. Choose a Calibration Memory Address to be used during any requested recalibration process.

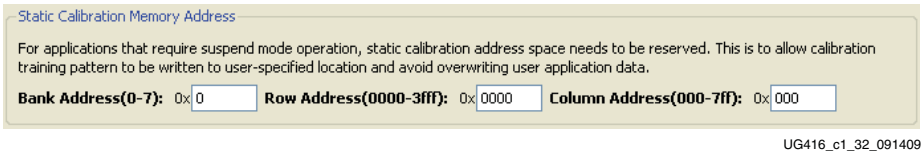


Figure 1-32: **Setting the Calibration Memory Address**

The MIG tool provides the option to bypass the initial calibration process during functional simulation to reduce simulation time.

14. Check the Bypass Calibration box to skip initial calibration during functional simulation.

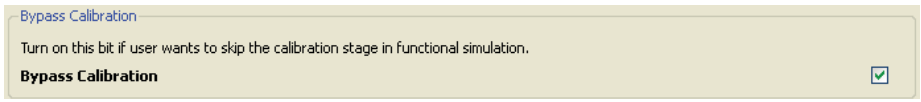


Figure 1-33: **Bypassing Calibration During Functional Simulation**

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, allowing the ChipScope tool to monitor traffic on the User Interface ports (see Debug Signals in Chapter 2 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 software 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 simply 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.

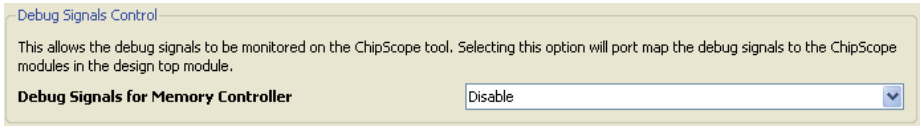
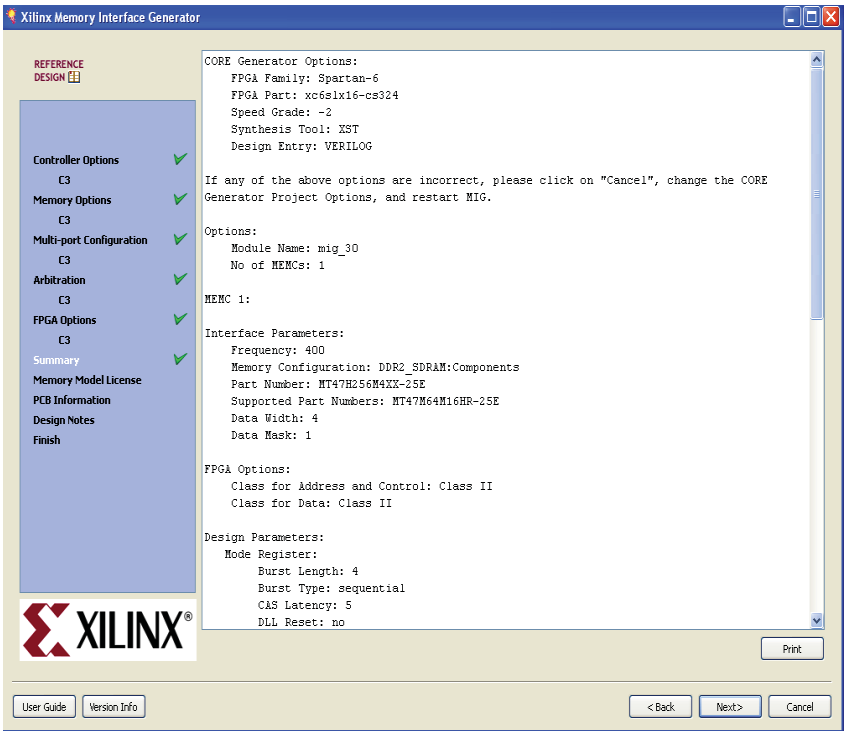


Figure 1-34: **Setting Up Debug Signal Control**

Design Summary

The Summary page 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

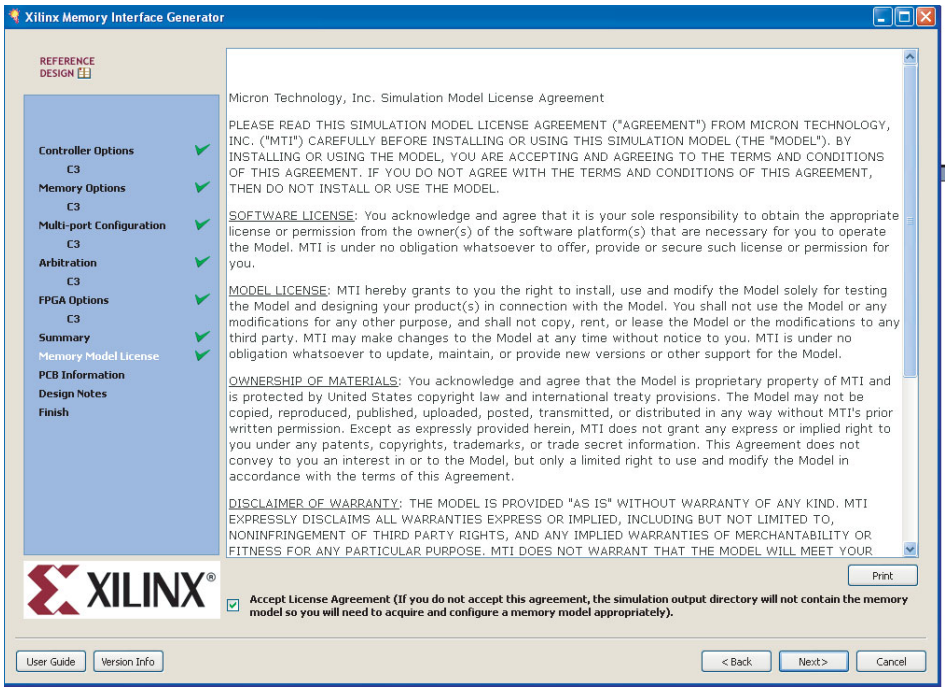
Figure 1-35: **Summary Page**

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. 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 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**

Directory and File Contents

<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 33. 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 `memc<x>_wrapper.v` file and soft IP.

<component name>/example_design/rtl/traffic_gen

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

Table 1-1: **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.
`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/par

The par folder contains the necessary constraint and script files for design implementation. Table 1-2 describes all files within this folder.

Table 1-2: **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 1-3 describes all files within this folder.

Table 1-3: **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 1-4 describes all files within this folder.

Table 1-4: **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 <component_name>.v file, described in Table 1-5.

Table 1-5: **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.

<component name>/user_design/par

The par folder contains the necessary constraint and script files for design implementation. Table 1-6 describes all files within this folder.

Table 1-6: **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 1-7 describes all files within this folder.

Table 1-7: **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 1-8 describes all files within this folder.

Table 1-8: **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.
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 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: testbench 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.

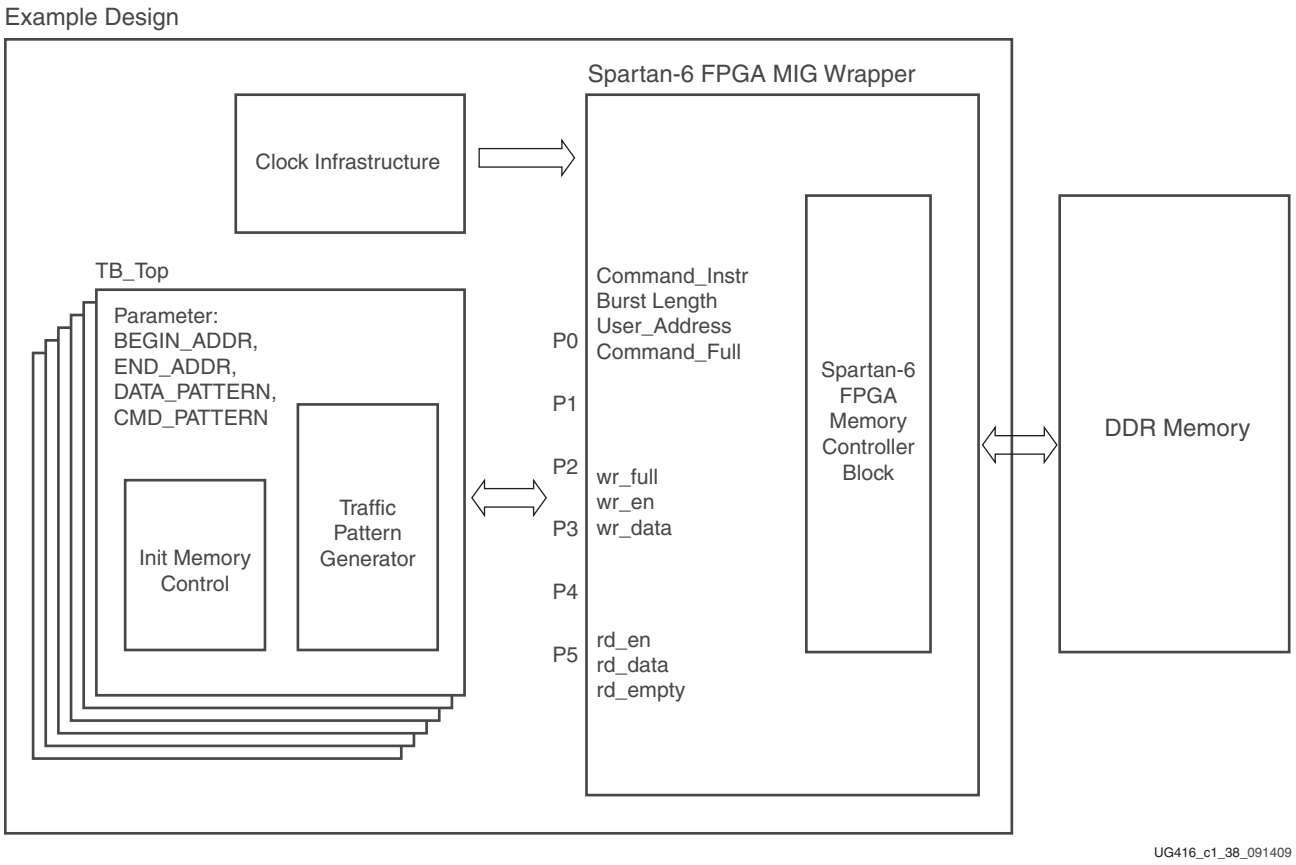


Figure 1-38: **MIG Example Design with Synthesizable Traffic Generator**

Traffic Generator Operation

The Traffic Generator module contained within the synthesizable testbench 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 testbench file (`tb_top.v`). See Modifying the Example Design, page 40 for information on using these parameters to change the port address space.

The testbench 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 testbench 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 testbench. As with the address space and data pattern, the default PRBS command pattern can be changed as described in Modifying the Example Design, page 40.

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 (i.e., 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).

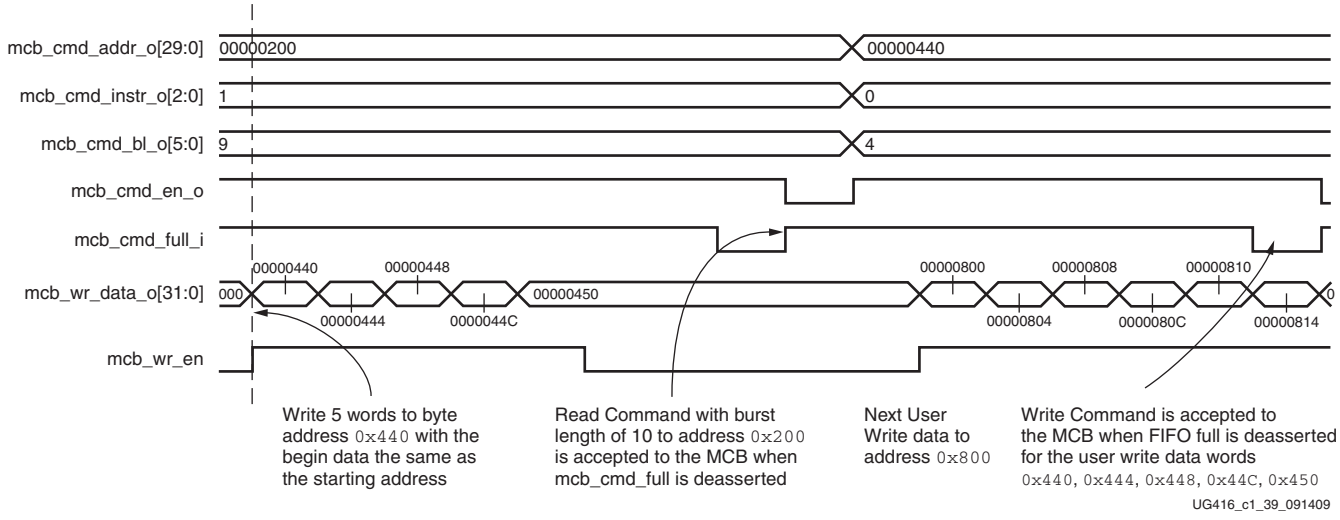


Figure 1-39: **Address as Data Pattern on DQ Bus**

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.

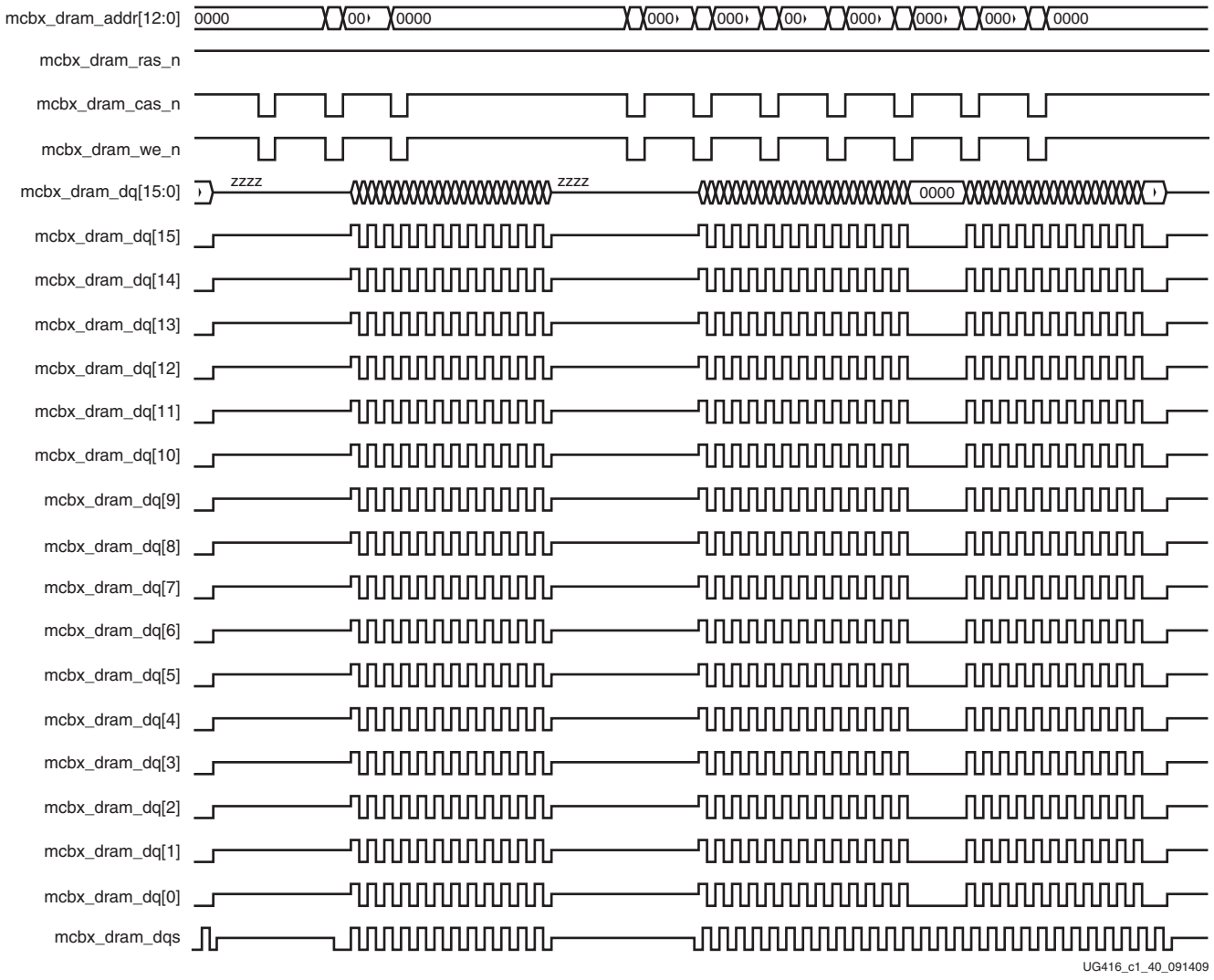


Figure 1-40: **Hammer Data Pattern on DQ Bus**

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.

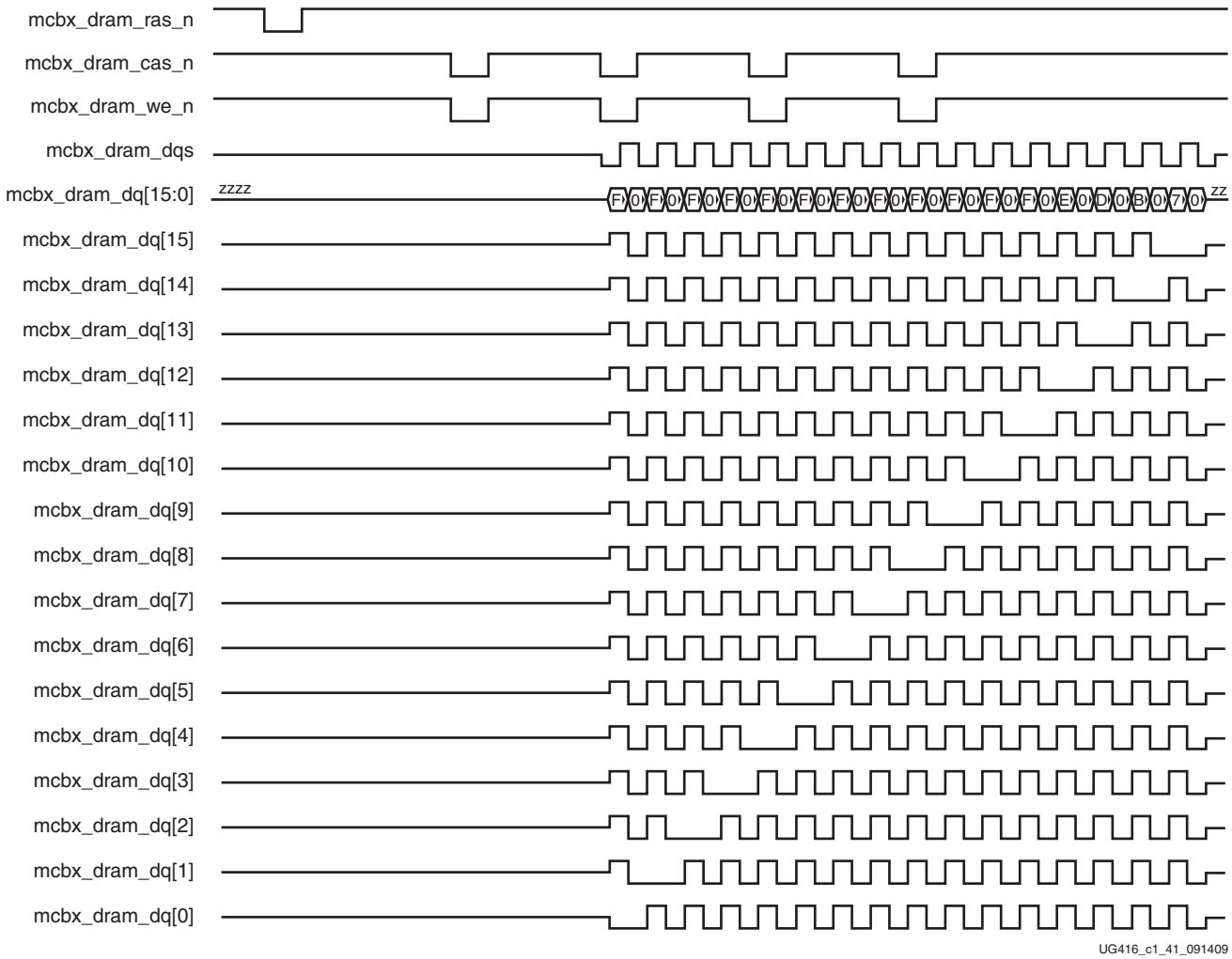


Figure 1-41: **Neighbor Data Pattern on DQ Bus**

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.

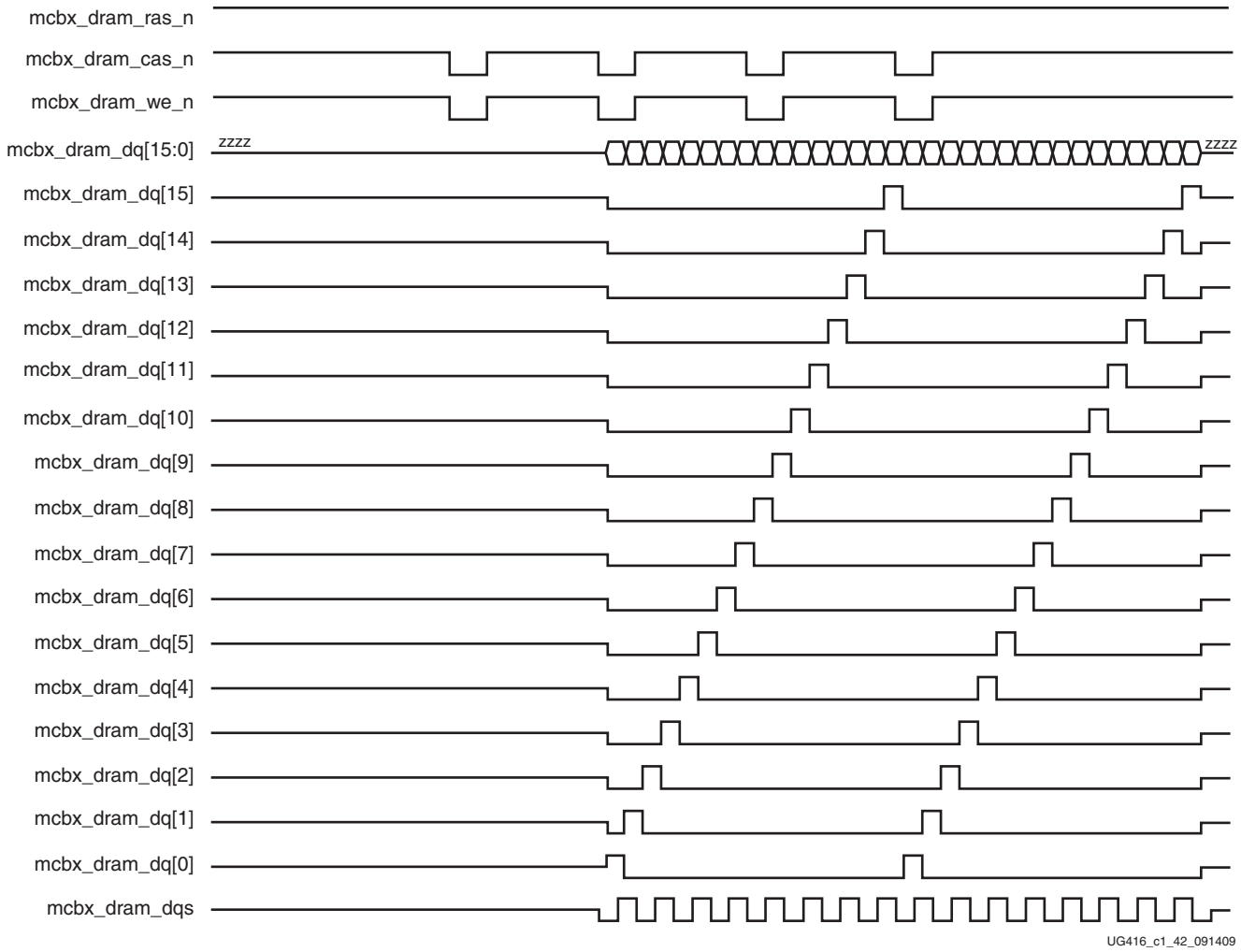


Figure 1-42: **Walking 1s Data Pattern on DQ Bus**

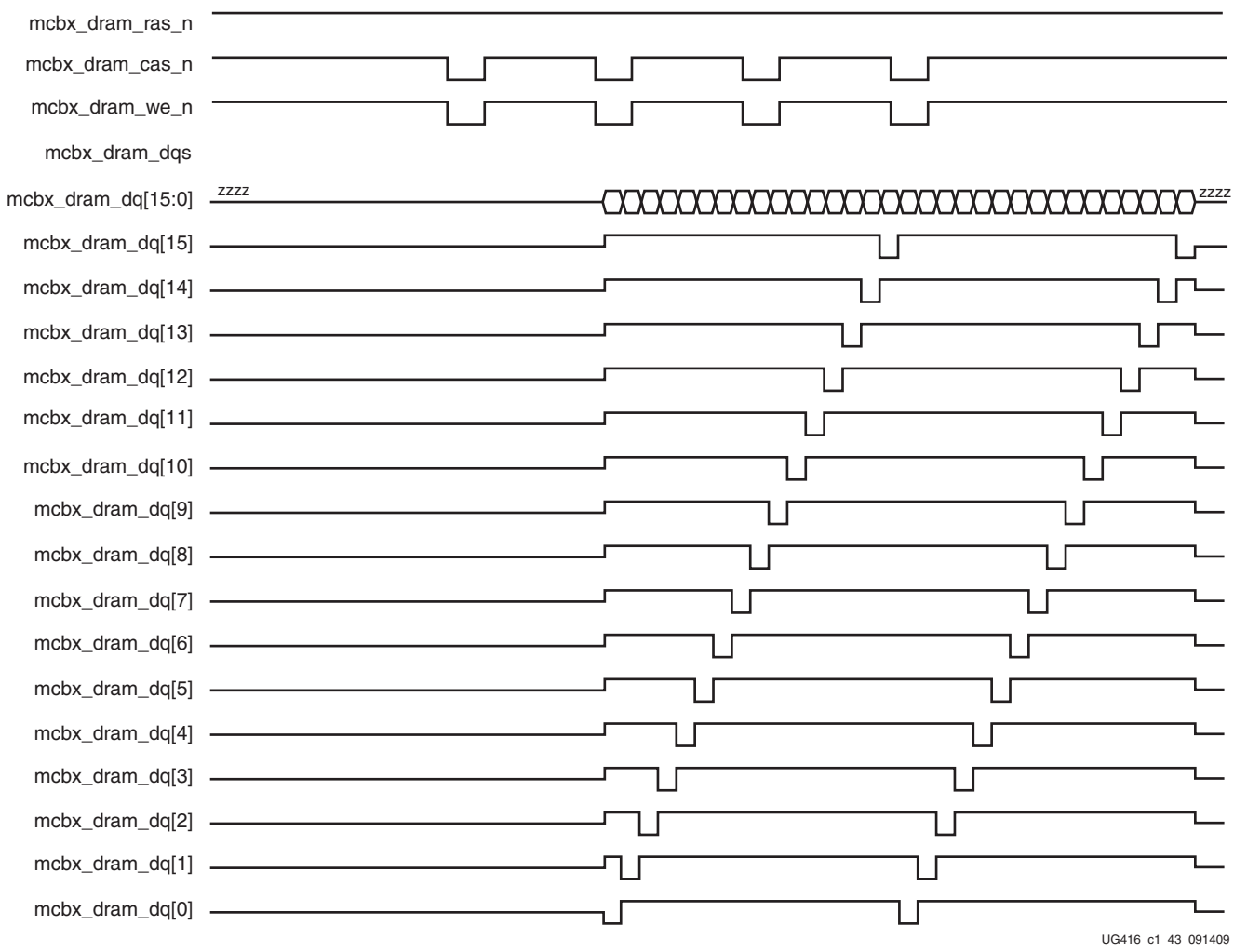


Figure 1-43: **Walking 0s Data Pattern on DQ Bus**

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.

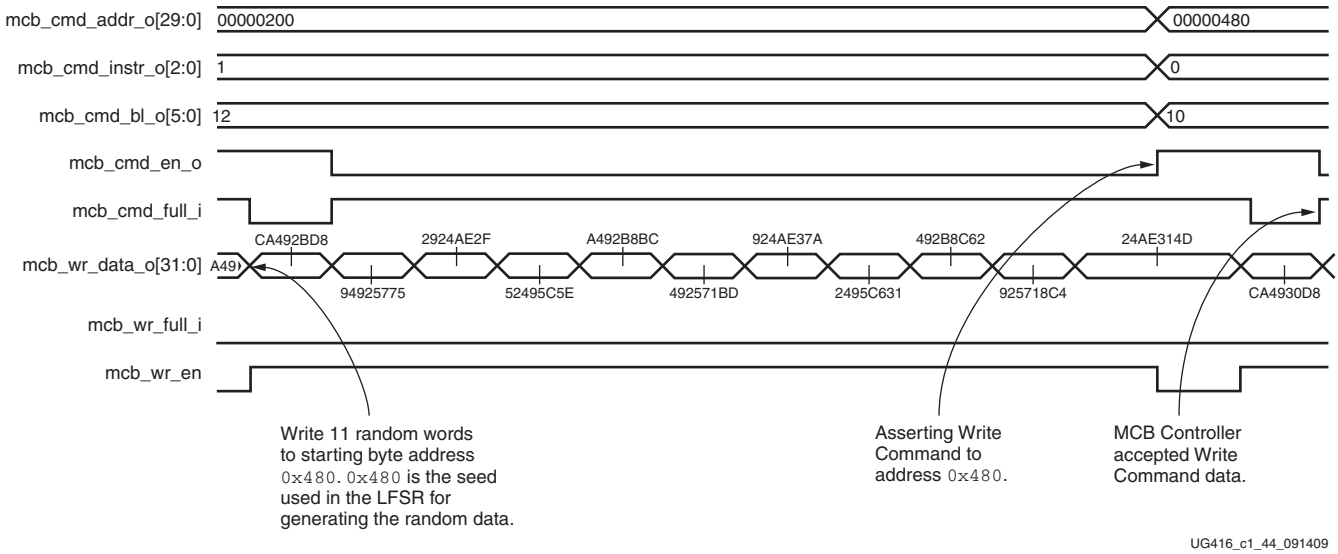


Figure 1-44: **PRBS Data Pattern on DQ Bus**

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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, for more details on simulating designs with the MCB.

Functional Simulation

To simulate the MIG example design or the user design, the Xilinx unisim library must be compiled and mapped to the simulator. Currently, the encrypted model of the Spartan-6 FPGA MCB is provided only for Xilinx ISim and ModelSim version 6.4b or above.

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

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

Top-Level Parameters

The top-level testbench file (`tb_top.v`) contains several parameters that can be modified to change the behavior of the traffic generator. Table 1-9 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 1-11, page 43.

Table 1-9: **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 1-10). This parameter is not brought to the top-level testbench 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 46).

Table 1-10: **Parameter for the Traffic Generator Module**

Parameter Name	Parameter Description	Parameter Value
CMD_PATTERN	Sets the command pattern to be generated	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 1-11 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 (i.e., 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 1-11: **Traffic Generator Signal Descriptions**

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

Table 1-11: **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 1-11: **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 testbench 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 1-12 shows some examples of setting the two mask parameters.

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

SADDR	EADDR	PRBS_SADDR_MASK_POS	PRBS_EADDR_MASK_POS
0x1000	0xFFFF	0x00001000	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).

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

When the memory space 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 (i.e., 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 1-11, page 43.

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.

XILINX®

Chapter 2

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 2-1 shows the overall flow for debugging problems associated with these two general types of issues.

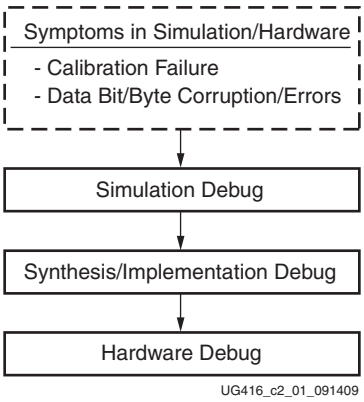


Figure 2-1: **Spartan-6 FPGA MCB Debug Flowchart**

If this chapter does not help to resolve the issue, refer to Additional Resources, page 5 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 testbench 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 33 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 UG388, *Spartan-6 FPGA Memory Controller User Guide*, 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.

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 2].

Simulation Debug

Figure 2-2 shows the debug flow for simulation.

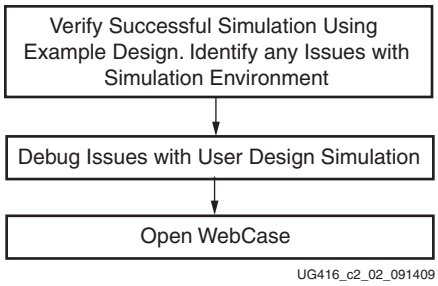


Figure 2-2: **Simulation Debug Flowchart**

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; they are not accessible in hardware. Table 2-1 lists the available simulation debug signals.

Table 2-1: **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[4:0]	Controller state (see Table 2-2).	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	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

Table 2-1: **Simulation Debug Signals *(Cont'd)***

Block Domain	Internal Signal Name	Description	Clock Domain
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
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[3:0]	Calibration state (see Table 2-3).	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 2-2: **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

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

State	Description
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
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 2-3: **FSM State Definitions for cal_state**

State	Description
0x00	Init
0x02	Reset DRP interface
0x16	Preamble Pulldown
0x17	Preamble Read

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

State	Description
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 testbench, 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 the MIG Example Design with Traffic Generator, page 33 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 3] and the *Synthesis and Simulation Design Guide* [Ref 1]. 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, page 33 for details on the available Traffic Generator data patterns and corresponding top-level parameters.

Synthesis and Implementation Debug

Figure 2-3 shows the debug flow for synthesis and implementation.

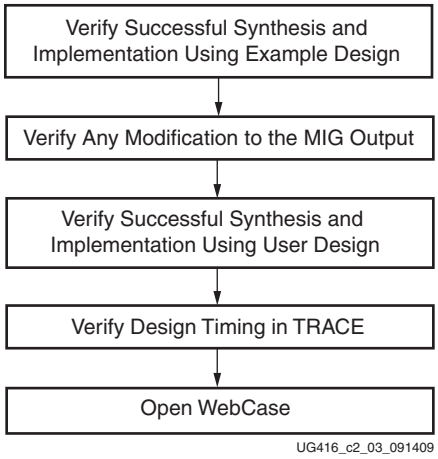


Figure 2-3: **Synthesis / Implementation Debug Flowchart**

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 (i.e., 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 4] improves performance and quality of the entire design. The *Xilinx Timing Constraints User Guide* [Ref 5] provides valuable information on all available Xilinx constraints.

Hardware Debug

Figure 2-4 shows the debug flow for hardware.

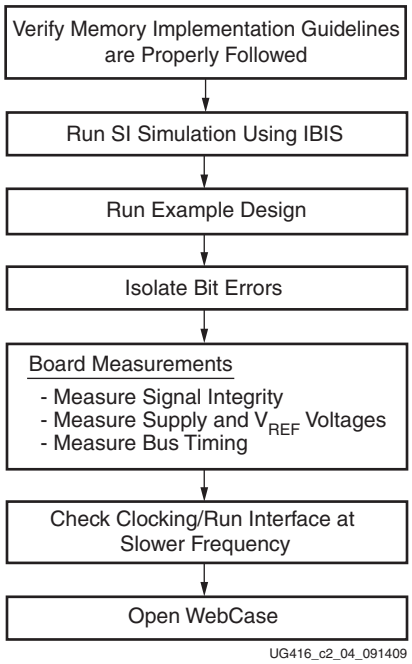


Figure 2-4: **Hardware Debug Flowchart**

Verify Memory Implementation Guidelines

See the "PCB Layout Guidelines" section in UG388, *Spartan-6 FPGA Memory Controller User Guide*, 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 UG388, *Spartan-6 FPGA Memory Controller User Guide*. 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 6] 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 33 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 UG388, *Spartan-6 FPGA Memory Controller User Guide*. 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 6] 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.