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

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.

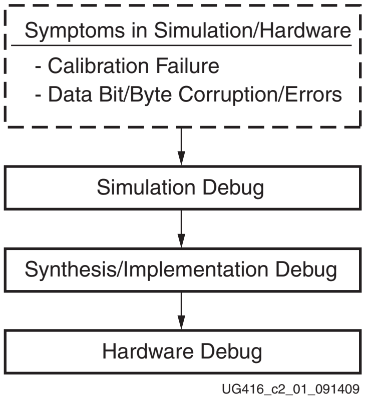

Symptoms in Simulation/Hardware
- Calibration Failure
- Data Bit/Byte Corruption/Errors
Simulation Debug
Synthesis/Implementation Debug
Hardware Debug
UG416_c2_01_091409

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

If this chapter does not help to resolve the issue, refer to Additional Resources, page 10 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 (CORE Generator Tool Native Interface Only), page 41 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.

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.

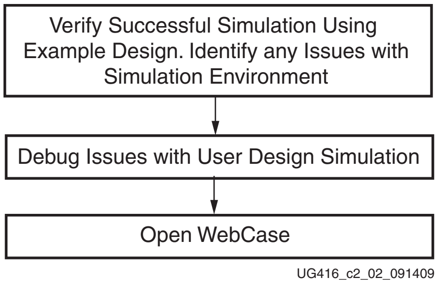
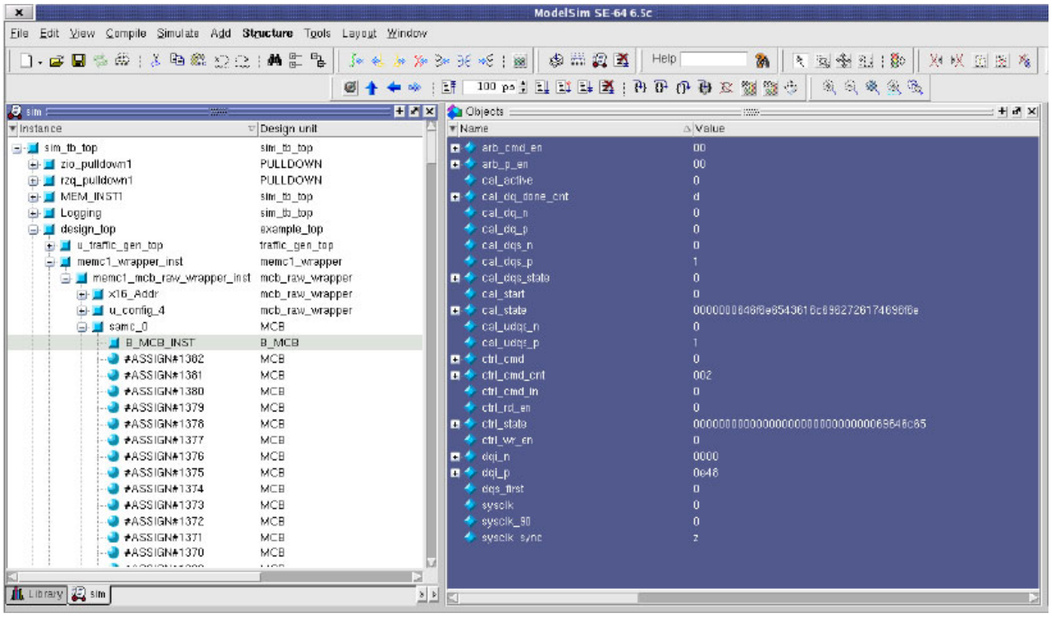

Verify Successful Simulation Using Example Design. Identify any Issues with Simulation Environment
Debug Issues with User Design Simulation
Open WebCase
UG416_c2_02_091409

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`.



Instance	Design unit
sim_tb_top	sim_tb_top
zio_pulldown1	PULLDOWN
rzq_pulldown1	PULLDOWN
MEM_INST1	sim_tb_top
Logging	sim_tb_top
design_top	example_top
u_traffic_gen_top	traffic_gen_top
memc1_wrapper_inst	memc1_wrapper
memc1_mcb_raw_wrapper_inst	mcb_raw_wrapper
x16_Addr	mcb_raw_wrapper
u_config_4	mcb_raw_wrapper
samc_0	MCB
B_MCB_INST	B_MCB
#ASSIGN#1382	MCB
#ASSIGN#1381	MCB
#ASSIGN#1380	MCB
#ASSIGN#1379	MCB
#ASSIGN#1378	MCB
#ASSIGN#1377	MCB
#ASSIGN#1376	MCB
#ASSIGN#1375	MCB
#ASSIGN#1374	MCB
#ASSIGN#1373	MCB
#ASSIGN#1372	MCB
#ASSIGN#1371	MCB
#ASSIGN#1370	MCB

Name	Value
arb_cmd_en	00
arb_p_en	00
cal_active	0
cal_dq_done_cnt	d
cal_dq_n	0
cal_dq_p	0
cal_dqs_n	0
cal_dqs_p	1
cal_dqs_state	0
cal_start	0
cal_state	00000000646f6e6543616c6962726174696f6e
cal_udqs_n	0
cal_udqs_p	1
ctrl_cmd	0
ctrl_cmd_cnt	002
ctrl_cmd_in	0
ctrl_rd_en	0
ctrl_state	000000000000000000000000000000069646c65
ctrl_wr_en	0
dqi_n	0000
dqi_p	0e46
dqs_first	0
sysclk	0
sysclk_90	0
sysclk_sync	z

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
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
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 (CORE Generator Tool Native Interface Only), page 41 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 (CORE Generator Tool Native Interface Only), page 41 for details on the available Traffic Generator data patterns and corresponding top-level parameters.