UltraScale Architecture-Based FPGAs Memory IP v1.4

LogiCORE IP Product Guide

Vivado Design Suite

PG150 April 20, 2022

This document supports the following memory core versions:

- DDR3 v1.4
- DDR4 v2.2
- LPDDR3 v1.0
- QDR II+ v1.4
- QDR-IV+ v2.0
- RLDRAM 3 v1.4

Table of Contents

SECTION I: SUMMARY

IP Facts

SECTION II: DDR3/DDR4

Chapter 1: Overview

Chapter 2: Product Specification

Chapter 3: Core Architecture

Chapter 4: Designing with the Core

Chapter 5: Design Flow Steps

Chapter 6: Example Design

Chapter 7: Test Bench

SECTION III: LPDDR3

Chapter 8: Overview

Chapter 9: Product Specification

Chapter 10: Core Architecture

Chapter 11: Designing with the Core

Chapter 12: Design Flow Steps

Chapter 13: Example Design

Chapter 14: Test Bench

SECTION IV: QDR II+ SRAM

Chapter 15: Overview

Chapter 16: Product Specification

Chapter 17: Core Architecture

Chapter 18: Designing with the Core

Chapter 19: Design Flow Steps

Chapter 20: Example Design

Chapter 21: Test Bench

SECTION V: QDR-IV SRAM

Chapter 22: Overview

Chapter 23: Product Specification

Chapter 24: Core Architecture

Chapter 25: Designing with the Core

Chapter 26: Design Flow Steps

Chapter 27: Example Design

Chapter 28: Test Bench

SECTION VI: RLDRAM 3

Chapter 29: Overview

Chapter 30: Product Specification

Chapter 31: Core Architecture

Chapter 32: Designing with the Core

Chapter 33: Design Flow Steps

Chapter 34: Example Design

Chapter 35: Test Bench

SECTION VII: TRAFFIC GENERATOR

Chapter 36: Traffic Generator

SECTION VIII: MULTIPLE IP CORES

Chapter 37: Multiple IP Cores

SECTION IX: DEBUGGING

Chapter 38: Debugging

SECTION X: APPENDICES

Appendix A: Upgrading

Appendix B: XCKU095/XCVU095 Recommended Memory Pinout Configurations

Appendix C: Additional Resources and Legal Notices

SECTION I: SUMMARY

IP Facts

Introduction

The Xilinx® UltraScale™ architecture-based FPGAs Memory IP core is a combined pre-engineered controller and physical layer (PHY) for interfacing UltraScale architecture FPGA user designs to DDR3 and DDR4 SDRAM, LPDDR3 SDRAM, QDR II+ SRAM, QDR-IV SRAM, and RLDRAM 3 devices.

This product guide provides information about using, customizing, and simulating a LogiCORE™ IP DDR3 or DDR4 SDRAM, LPDDR3 SDRAM, QDR II+ SRAM, QDR-IV SRAM, or a RLDRAM 3 interface core for UltraScale architecture-based FPGAs. It also describes the core architecture and provides details on customizing and interfacing to the core.

Features

For feature information on the DDR3/DDR4 SDRAM, LPDDR3 SDRAM, QDR II+ SRAM, QDR-IV SRAM, and RLDRAM 3 interfaces, see the following sections:

- Feature Summary in Chapter 1 for DDR3/ DDR4 SDRAM
- Feature Summary in Chapter 8 for LPDDR3 SDRAM
- Feature Summary in Chapter 15 for QDR II+ SRAM
- Feature Summary in Chapter 22 for QDR-IV SRAM
- Feature Summary in Chapter 29 for RLDRAM 3

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™, Virtex®, and Kintex® UltraScale
Supported User Interfaces	User
Resources	See Resource Utilization (DDR3/DDR4), Resource Utilization (LPDDR3), Resource Utilization (QDR II+), Resource Utilization (QDR-IV), Resource Utilization (RLDRAM 3).
Provided with Core	
Design Files	RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado Design Suite
Simulation(3)	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 58435
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.
3. Behavioral simulations are supported with Mixed Simulator Language. Netlist (post-synthesis and post-implementation) simulations are supported with Verilog Simulator Language and are not supported by Vivado Simulator.

SECTION II: DDR3/DDR4

Overview

☆ **IMPORTANT:** *This document supports DDR3 SDRAM core v1.4 and DDR4 SDRAM core v2.2.*

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Clocking
  - Resets
  - Protocol Description
  - Customizing and Generating the Core
  - Example Design

Core Overview

The Xilinx UltraScale™ architecture includes the DDR3/DDR4 SDRAM cores. These cores provide solutions for interfacing with these SDRAM memory types. Both a complete Memory Controller and a physical (PHY) layer only solution are supported. The UltraScale architecture for the DDR3/DDR4 cores are organized in the following high-level blocks:

- **Controller** – The controller accepts burst transactions from the user interface and generates transactions to and from the SDRAM. The controller takes care of the SDRAM timing parameters and refresh. It coalesces write and read transactions to reduce the number of dead cycles involved in turning the bus around. The controller also reorders commands to improve the utilization of the data bus to the SDRAM.

- **Physical Layer** – The physical layer provides a high-speed interface to the SDRAM. This layer includes the hard blocks inside the FPGA and the soft blocks calibration logic necessary to ensure optimal timing of the hard blocks interfacing to the SDRAM.

  The application logic is responsible for all SDRAM transactions, timing, and refresh.

  - These hard blocks include:
    - Data serialization and transmission
    - Data capture and deserialization
    - High-speed clock generation and synchronization
    - Coarse and fine delay elements per pin with voltage and temperature tracking
  - The soft blocks include:
    - **Memory Initialization** – The calibration modules provide a JEDEC® -compliant initialization routine for the particular memory type. The delays in the initialization process can be bypassed to speed up simulation time, if desired.
    - **Calibration** – The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance.

      Results of the calibration process are available through the Xilinx debug tools. After completion of calibration, the PHY layer presents raw interface to the SDRAM.

- **Application Interface** – The user interface layer provides a simple FIFO-like interface to the application. Data is buffered and read data is presented in request order.

  The above user interface is layered on top of the native interface to the controller. The native interface is not accessible by the user application and has no buffering and presents return data to the user interface as it is received from the SDRAM which is not necessarily in the original request order. The user interface then buffers the read and write data and reorders the data as needed.

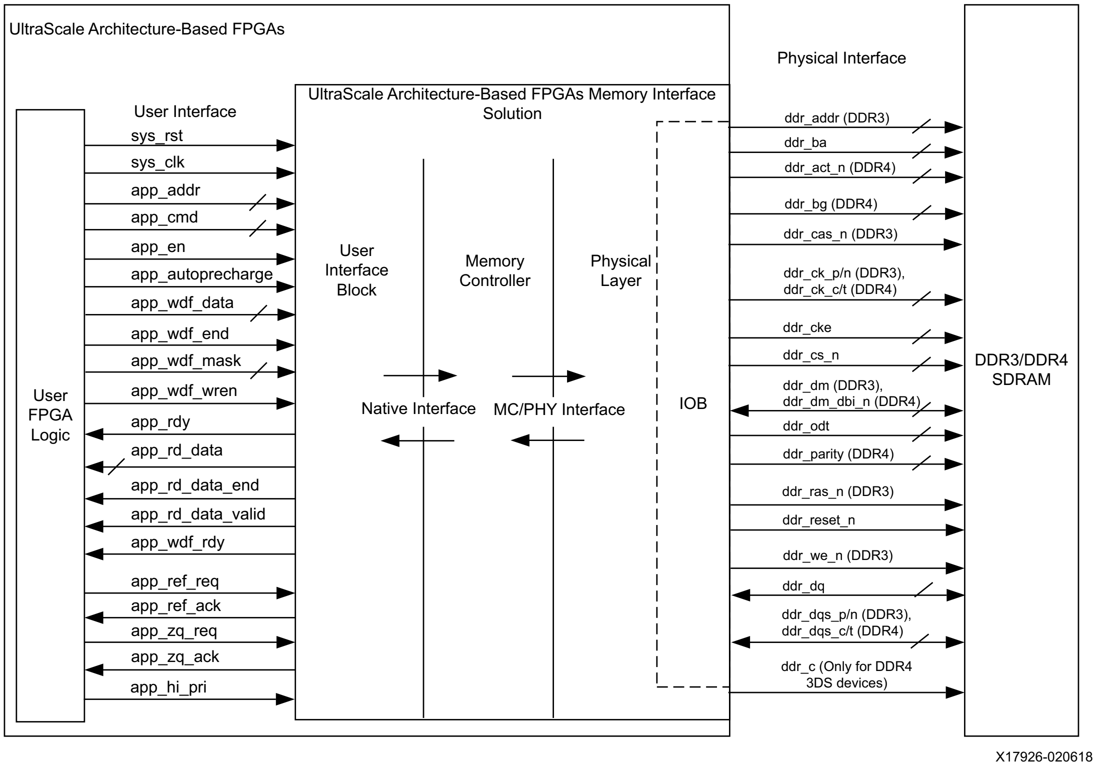


Figure 1-1: **UltraScale Architecture-Based FPGAs DDR3/DDR4 Memory Interface Solution**

Feature Summary

DDR3 SDRAM

- Component support for interface width of 8 to 80 bits (RDIMM, UDIMM, and SODIMM support)
    - Maximum component limit is 9 and this restriction is valid for components only and not for DIMMs
- DDR3 (1.5V) and DDR3L (1.35V)
- Dual slot support for RDIMMs, SODIMMs, and UDIMMs
- Quad-rank RDIMM support
- Density support
    - Support densities up to 8 GB for components, 32 GB for RDIMMs, 16 GB for SODIMMs, and 16 GB for UDIMMs
    - Other densities for memory device support is available through custom part selection
- 8-bank support
- x4 (x4 devices must be used in even multiples), x8, and x16 device support
- AXI4 Slave Interface

  Note: The x4-based component interfaces do not support AXI4, while x4-based RDIMM and LRDIMM does support AXI4.
- x4, x8, and x16 components are supported
- 8-word burst support
- Support for 5 to 14 cycles of column-address strobe (CAS) latency (CL)
- On-die termination (ODT) support
- Support for 5 to 10 cycles of CAS write latency
- Source code delivery in Verilog
- 4:1 memory to FPGA logic interface clock ratio
- Open, closed, and transaction based pre-charge controller policy
- Interface calibration and training information available through the Vivado hardware manager
- Optional Error Correcting Code (ECC) support for non-AXI4 72-bit interfaces

DDR4 SDRAM

- Component support for interface width of 8 to 80 bits (RDIMM, LRDIMM, UDIMM, and SODIMM support)
    - Maximum component limit is 9 and this restriction is valid for components only and not for DIMMs
- Density support
    - Support densities up to 32 GB for components, 64 GB for LRDIMMs, 128 GB for RDIMMs, 16 GB for SODIMMs, and 16 GB for UDIMMs
    - Other densities for memory device support is available through custom part selection
- AXI4 Slave Interface

  Note: The x4-based component interfaces do not support AXI4, while x4-based RDIMM and LRDIMM does support AXI4.
- x4, x8, and x16 components are supported
- Dual slot support for DDR4 RDIMMs, SODIMMs, LRDIMMs, and UDIMMs
- 8-word burst support
- Support for 9 to 24 cycles of column-address strobe (CAS) latency (CL)
- ODT support
- 3DS RDIMM and LRDIMM support
- 3DS component support
- Support for 9 to 18 cycles of CAS write latency
- Source code delivery in Verilog
- 4:1 memory to FPGA logic interface clock ratio
- Open, closed, and transaction based pre-charge controller policy
- Interface calibration and training information available through the Vivado hardware manager
- Optional Error Correcting Code (ECC) support for non-AXI4 72-bit interfaces
- CRC for write operations is not supported
- 2T timing for the address/command bus is not supported

RECOMMENDED: *Use x8 or x4-based interfaces for maximum efficiency. These devices have four bank groups and 16 banks which allow greater efficiency. Compared to the x16-based devices, which only have two bank groups and eight banks. The DDR4 devices have better access timing among the bank groups, so the larger number can increase the efficiency. Note that x16 DDP DDR4 DRAM is composed of two x8 devices that has the larger number of banks and groups. For more information, see AR: 71209.*

IMPORTANT: *DBI should be enabled with repeated single Burst Length = 8 (BL8) read access with all "0" on the DQ bus, followed by idle (NOP/DESELECT) inserted between each BL8 read burst as shown in Figure 1-2. Enabling the DBI feature effectively mitigates excessive power supply noise.*
If DBI is not an option, then encoding the data to remove all "0" bursts in application before it reaches the memory controller is an equally effective method for mitigating power supply noise. For x4-based RDIMM/LRDIMM interfaces which lack the DM/DBI pin, the power supply noise is mitigated by the ODT settings used for these topologies. For x4-based component interfaces wider than 16 bits, the data encoding method is recommended.

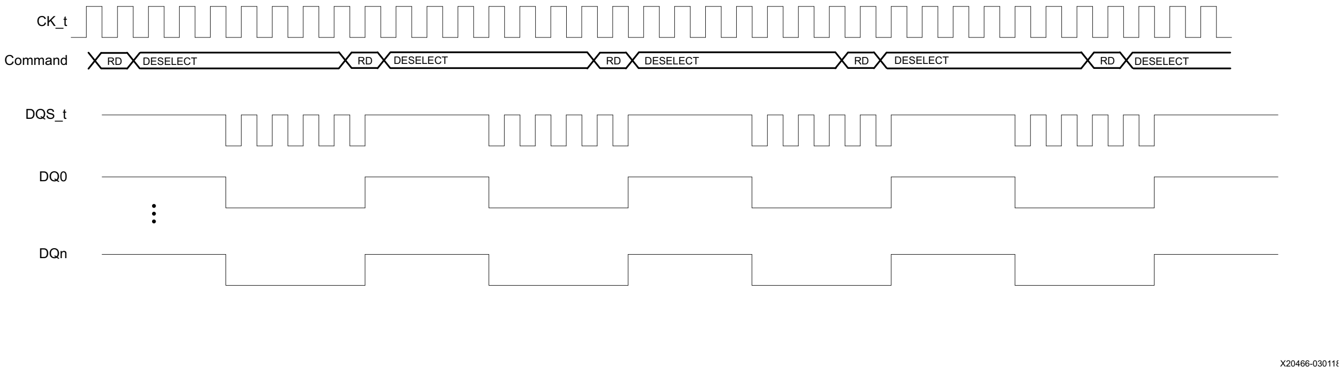


Figure 1-2: **DQ Pattern with BL8 Read Burst**

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Chapter 2

Product Specification

Standards

This core supports DRAMs that are compliant to the JESD79-3F, *DDR3 SDRAM Standard* and JESD79-4, *DDR4 SDRAM Standard,* JEDEC® Solid State Technology Association [Ref 1]. It also supports the DDR4 3DS Addendum.

For more information on UltraScale™ architecture documents, see References, page 789.

Performance

Maximum Frequencies

For more information on the maximum frequencies, see the following documentation:

- *Kintex UltraScale FPGAs Data Sheet, DC and AC Switching Characteristics* (DS892) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6]
- *UltraScale Maximum Memory Performance Utility* (XTP414) [Ref 21]

Efficiency and Latency Measurements

The performance of the Memory Controller is shown here for several typical workloads. Efficiency gives the data bus utilization in % for long traffic streams, and latency shows the round-trip command-to-read-data delay at the user interface. The results were taken from simulations and hardware testing. The definition of each workload is shown in the following sections. In each workload, the refresh interval is 7.8 µs and the periodic read interval is set to 1.0 µs. The address mapping option is ROW_COLUMN_BANK and ORDERING is NORMAL unless noted otherwise.

Efficiency Workloads

Sequential Read

- Simple address increment pattern
- 100% reads

Sequential Write

- Simple address increment pattern
- 100% writes (except for periodic reads generated by the controller for VT tracking)

Burst Read/Write Mix

- Repeating pattern of 64 sequential reads and 64 sequential writes
- 50/50 read/write mix

Short Burst Read/Write Mix

- Repeating pattern of four sequential reads and four sequential writes
- Full DRAM page accessed in bursts of four before changing the row address for high page hit rate
- 50/50 read/write mix

Random Address Read/Write Mix

- Repeating pattern of two random reads and two random writes
- Fully random address for a low page hit rate
- 50/50 read/write mix

Table 2-1: **DDR Bus Efficiency**

Workload	DDR3 x8 Efficiency [%]	DDR4 x8 Efficiency [%]
Sequential Read	94	94
Sequential Write	89	89
Burst Read/Write Mix	90	90
Short Burst Read/Write Mix	50	51
Random Address Read/Write Mix	23	24

Sequential write efficiency is lower than sequential read due to the injection of periodic reads in the write sequence. The burst workload achieves an efficiency just between sequential read and sequential write. The burst workload has read transactions frequently enough that periodic reads are not injected by the controller, but due to read/write bus turnaround the efficiency is still somewhat lower than a pure sequential read.

The short burst workload shows the effect of more frequent bus turnaround compared to the 64 transaction bursts. The random workload shows the effect of frequent bus turnaround and page misses. In all the cases in Table 2-1, efficiency is primarily limited by DRAM specifications and the Memory Controller is scheduling transactions as efficiently as possible.

The example DDR3/DDR4 "Idle" read latencies are shown in the following section for both the user interface and PHY only interfaces. Actual read latency in hardware might vary due to command and data bus flight times, package delays, CAS latency, etc.

The latency numbers are for an "Idle" case, where the Memory Controller starts off with no pending transactions, no pending refreshes or periodic reads, and any DRAM protocol or timing restrictions from previous commands have elapsed. When a new read transaction is received, there is nothing blocking progress and read data is returned with the minimum latency.

Idle Latency Categories

- **Page Miss** – DRAM bank open to row address that does not match the row address of the incoming read transaction and tRAS has elapsed
- **Closed Page** – All DRAM banks precharged and tRP has elapsed
- **Page Hit** – DRAM bank open to a row address that matches the Row address of the incoming transaction and tRCD has elapsed

Table 2-2 shows the user interface idle latency in DRAM clock cycles from assertion of `app_en` and `app_rdy` to assertion of `app_rd_data_valid`.

Table 2-2: **User Interface Idle Latency**

Latency Category	DDR3 1600 CL = 11 [tCK]	DDR4 2400 CL = 16 [tCK]
Page Miss	100	112
Closed Page	84	92
Page Hit	72	72

Table 2-3 shows the PHY Only interface read CAS command to read data latency. This is equivalent to page hit latency.

Table 2-3: **PHY Only Interface Read CAS Command to Read Data Latency**

Latency Category	DDR3 1600 CL = 11 [tCK]	DDR4 2400 CL = 16 [tCK]
Page Hit	40	44

Resource Utilization

For full details about performance and resource utilization, visit Performance and Resource Utilization (DDR3) and Performance and Resource Utilization (DDR4).

Port Descriptions

For a complete Memory Controller solution there are three port categories at the top-level of the memory interface core called the "user design."

- The first category is the memory interface signals that directly interfaces with the SDRAM. These are defined by the JEDEC specification.
- The second category is the application interface signals. These are described in the Protocol Description, page 118.
- The third category includes other signals necessary for proper operation of the core. These include the clocks, reset, and status signals from the core. The clocking and reset signals are described in their respective sections.

  The active-High **`init_calib_complete`** signal indicates that the initialization and calibration are complete and that the interface is now ready to accept commands for the interface.

For a PHY layer only solution, the top-level application interface signals are replaced with the PHY interface. These signals are described in the PHY Only Interface, page 157.

The signals that interface directly with the SDRAM and the clocking and reset signals are the same as for the Memory Controller solution.

Core Architecture

This chapter describes the UltraScale™ architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Overview

The UltraScale architecture-based FPGAs Memory Interface Solutions is shown in Figure 3-1.

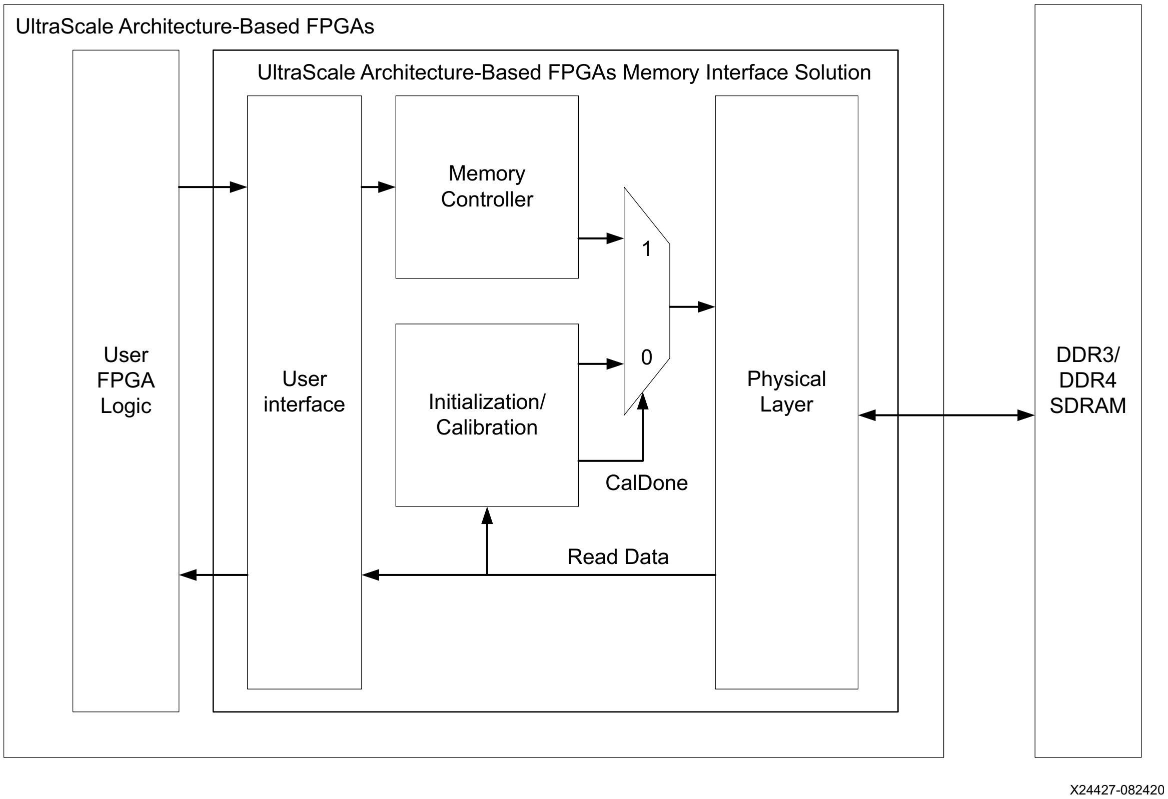


Figure 3-1: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core Architecture**

Memory Controller

The Memory Controller (MC) is designed to take Read, Write, and Read-Modify-Write transactions from the user interface (UI) block and issues them to memory efficiently with low latency, meeting all DRAM protocol and timing requirements, while using minimal FPGA resources. The controller operates with a DRAM to system clock ratio of 4:1 and can issue one Activate, one CAS, and one Precharge command on each system clock cycle.

The controller supports an open page policy and can achieve very high efficiencies with workloads with a high degree of spatial locality. The controller also supports a closed page policy and the ability to reorder transactions to efficiently schedule workloads with address patterns that are more random. The controller also allows a degree of control over low-level functions with a UI control signal for AutoPrecharge on a per transaction basis as well as signals that can be used to determine when DRAM refresh commands are issued.

The key blocks of the controller command path include:

1. The Group FSMs that queue up transactions, check DRAM timing, and decide when to request Precharge, Activate, and CAS DRAM commands.
2. The "Safe" logic and arbitration units that reorder transactions between Group FSMs based on additional DRAM timing checks while also ensuring forward progress for all DRAM command requests.
3. The Final Arbiter that makes the final decision about which commands are issued to the PHY and feeds the result back to the previous stages.

The maintenance blocks of the controller command path include:

1. Blocks that generate refresh and ZQCS commands
2. Commands needed for VT tracking
3. Optional block that implements a SECDED ECC for 72-bit wide data buses

Figure 3-2 shows the Memory Controller block diagram.

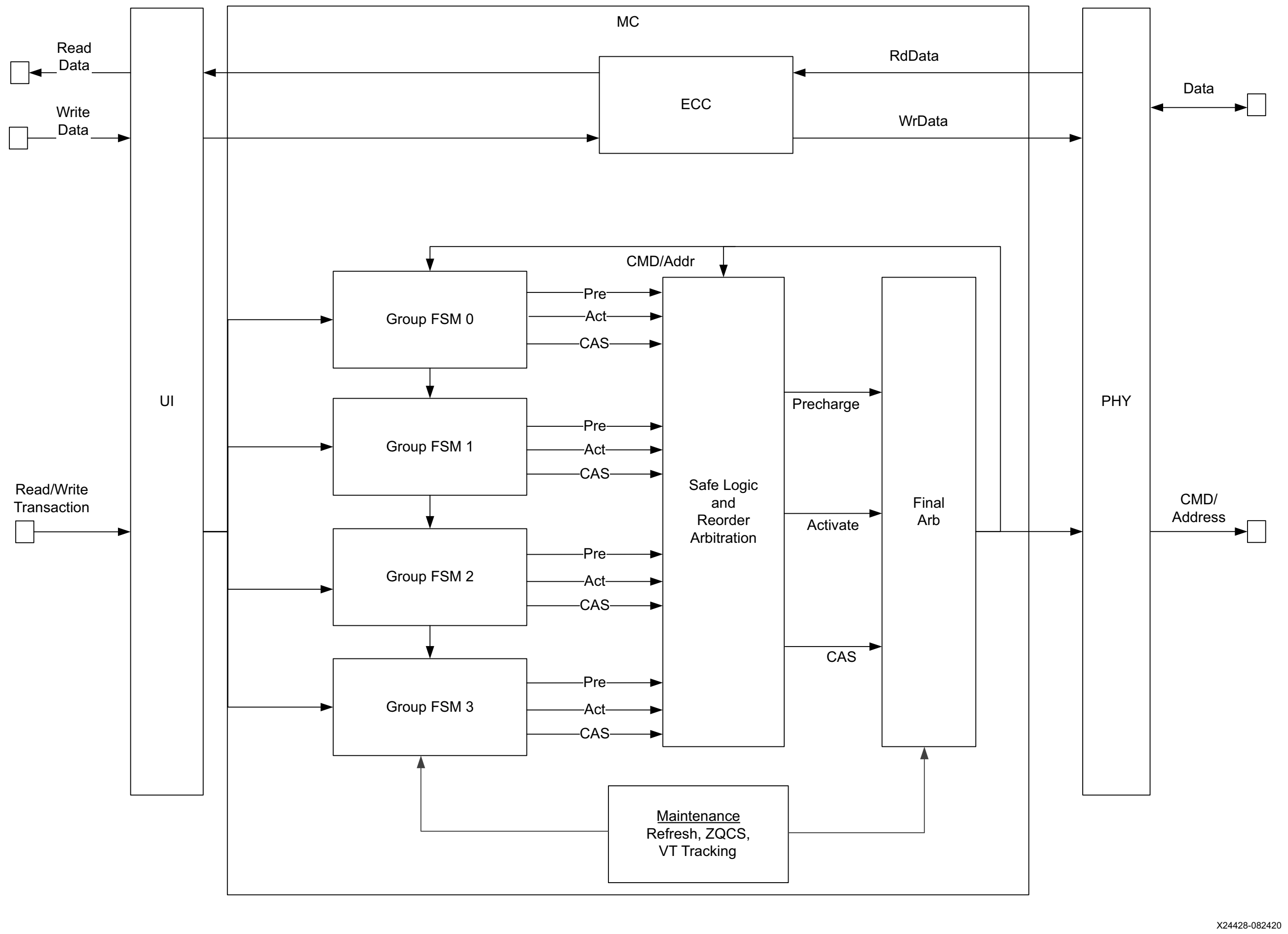


Figure 3-2: **Memory Controller Block Diagram**

Native Interface

The UI block is connected to the Memory Controller by the native interface, and provides the controller with address decode and read/write data buffering. On writes, data is requested by the controller one cycle before it is needed by presenting the data buffer address on the native interface. This data is expected to be supplied by the UI block on the next cycle. Hence there is no buffering of any kind for data (except due to the barrel shifting to place the data on a particular DDR clock).

On reads, the data is offered by the MC on the cycle it is available. Read data, along with a buffer address is presented on the native interface as soon as it is ready. The data has to be accepted by the UI block.

Read and write transactions are mapped to an mcGroup instance based on bank group and bank address bits of the decoded address from the UI block. Although there are no groups in DDR3, the name group represents either a real group in DDR4 x4 and x8 devices (which serves four banks of that group). For DDR3, each mcGroup module would service two banks.

In the case of DDR4 x16 interface, the mcGroup represents 1-bit of group (there are only one group bit in x16) and 1-bit of bank, whereby the mcGroup serves two banks.

The total number of outstanding requests depends on the number of mcGroup instances, as well as the round trip delay from the controller to memory and back. When the controller issues an SDRAM CAS command to memory, an mcGroup instance becomes available to take a new request, while the previous CAS commands, read return data, or write data might still be in flight.

Control and Datapaths

Control Path

The control path starts at the mcGroup instances. The mapping of SDRAM group and bank addresses to mcGroup instance ensures that transactions to the same full address map to the same mcGroup instance. Because each mcGroup instance processes the transactions it receives in order, read-after-write and write-after-write address hazards are prevented.

Datapath

Read and write data pass through the Memory Controller. If ECC is enabled, a SECDEC code word is generated on writes and checked on reads. For more information, see ECC, page 30. The MC generates the requisite control signals to the mcRead and mcWrite modules telling them the timing of read and write data. The two modules acquire or provide the data as required at the right time.

Read and Write Coalescing

The controller prioritizes reads over writes when reordering is enabled. If both read and write CAS commands are safe to issue on the SDRAM command bus, the controller selects only read CAS commands for arbitration. When a read CAS issues, write CAS commands are blocked for several SDRAM clocks specified by parameter tRTW. This extra time required for a write CAS to become safe after issuing a read CAS allows groups of reads to issue on the command bus without being interrupted by pending writes.

Reordering

Requests that map to the same mcGroup are never reordered. Reordering between the mcGroup instances is controlled with the ORDERING parameter. When set to "NORM," reordering is enabled and the arbiter implements a round-robin priority plan, selecting in priority order among the mcGroups with a command that is safe to issue to the SDRAM.

The timing of when it is safe to issue a command to the SDRAM can vary on the target bank or bank group and its page status. This often contributes to reordering.

When the ORDERING parameter is set to "STRICT," all requests have their CAS commands issued in the order in which the requests were accepted at the native interface. STRICT ordering overrides all other controller mechanisms, such as the tendency to coalesce read requests, and can therefore degrade data bandwidth utilization in some workloads.

Group Machines

In the Memory Controller, there are four group state machines. These state machines are allocated depending on technology (DDR3 or DDR4) and width (x4, x8, and x16). The following summarizes the allocation to each group machine. In this description, GM refers to the Group Machine (0 to 3), BG refers to group address, and BA refers to bank address. Note that group in the context of a group state machine denotes a notional group and does not necessarily refer to a real group (except in case of DDR4, part x4 and x8).

- DDR3, any part – Total of eight banks
    - GM 0: BA[2:1] == **`2'b00`**; services banks 0 and 1
    - GM 1: BA[2:1] == **`2'b01`**; services banks 2 and 3
    - GM 2: BA[2:1] == **`2'b10`**; services banks 4 and 5
    - GM 3: BA[2:1] == **`2'b11`**; services banks 6 and 7
- DDR4, x4 and x8 parts – Total of 16 banks
    - GM 0: services BG 0; four banks per group
    - GM 1: services BG 1; four banks per group
    - GM 2: services BG 2; four banks per group
    - GM 3: services BG 3; four banks per group
- DDR4, x16 parts – Total of eight banks
    - GM 0: services BG 0, BA[0] == 0; 2 banks per group
    - GM 1: services BG 0, BA[0] == 1; 2 banks per group
    - GM 2: services BG 1, BA[0] == 0; 2 banks per group
    - GM 3: services BG 1, BA[0] == 1; 2 banks per group

Figure 3-3 shows the Group FSM block diagram for one instance. There are two main sections to the Group FSM block, stage 1 and stage 2, each containing a FIFO and an FSM. Stage 1 interfaces to the UI, issues Precharge and Activate commands, and tracks the DRAM page status.

Stage 2 issues CAS commands and manages the RMW flow. There is also a set of DRAM timers for each rank and bank used by the FSMs to schedule DRAM commands at the earliest safe time. The Group FSM block is designed so that each instance queues up multiple transactions from the UI, interleaves DRAM commands from multiple transactions onto the DDR bus for efficiency, and executes CAS commands strictly in order.

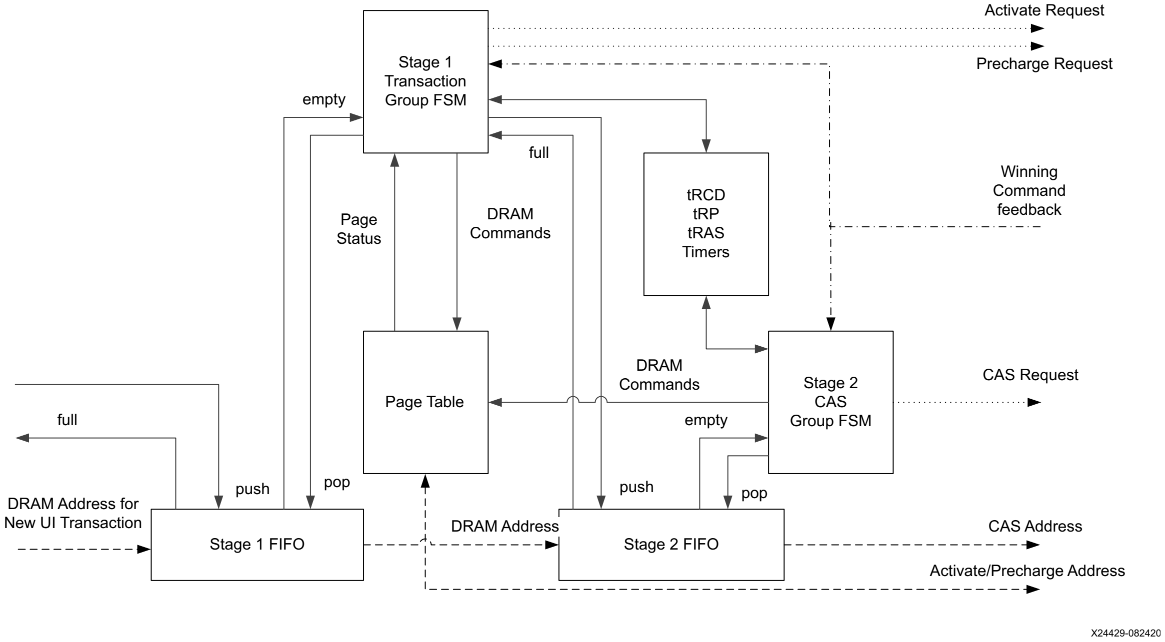


Figure 3-3: **Group FSM Block Diagram**

When a new transaction is accepted from the UI, it is pushed into the stage 1 transaction FIFO. The page status of the transaction at the head of the stage 1 FIFO is checked and provided to the stage 1 transaction FSM. The FSM decides if a Precharge or Activate command needs to be issued, and when it is safe to issue them based on the DRAM timers.

When the page is open and not already scheduled to be closed due to a pending RDA or WRA in the stage 2 FIFO, the transaction is transferred from the stage 1 FIFO to the stage 2 FIFO. At this point, the stage 1 FIFO is popped and the stage 1 FSM begins processing the next transaction. In parallel, the stage 2 FSM processes the CAS command phase of the transaction at the head of the stage 2 FIFO. The stage 2 FSM issues a CAS command request when it is safe based on the tRCD timers. The stage 2 FSM also issues both a read and write CAS request for RMW transactions.

ECC

The MC supports an optional SECDED ECC scheme that detects and corrects read data errors with 1-bit error per DQ bus burst and detects all 2-bit errors per burst. The 2-bit errors are not corrected. Three or more bit errors per burst might or might not be detected, but are never corrected. Enabling ECC adds four DRAM clock cycles of latency to all reads, whether errors are detected/corrected or not.

A Read-Modify-Write (RMW) scheme is also implemented to support Partial Writes when ECC is enabled. Partial Writes have one or more user interface write data mask bits set High. Partial Writes with ECC disabled are handled by sending the data mask bits to the DRAM Data Mask (DM) pins, so the RMW flow is used only when ECC is enabled. When ECC is enabled, Partial Writes require their own command, **`wr_bytes`** or **`0x3`**, so the MC knows when to use the RMW flow.

Note: When ECC is enabled, initialize (or write to) the memory space prior to performing partial writes (RMW).

Read-Modify-Write Flow

When a **`wr_bytes`** command is accepted at the user interface it is eventually assigned to a group state machine like other write or read transactions. The group machine breaks the Partial Write into a read phase and a write phase. The read phase performs the following:

1. First reads data from memory.
2. Checks for errors in the read data.
3. Corrects single bit errors.
4. Stores the result inside the Memory Controller.

Data from the read phase is not returned to the user interface. If errors are detected in the read data, an ECC error signal is asserted at the native interface. After read data is stored in the controller, the write phase begins as follows:

1. Write data is merged with the stored read data based on the write data mask bits.
2. New ECC check bits are generated for the merged data and check bits are written to memory.
3. Any multiple bit errors in the read phase results in the error being made undetectable in the write phase as new check bits are generated for the merged data. This is why the ECC error signal is generated on the read phase even though data is not returned to the user interface. This allows the system to know if an uncorrectable error has been turned into an undetectable error.

When the write phase completes, the group machine becomes available to process a new transaction. The RMW flow ties up a group machine for a longer time than a simple read or write, and therefore might impact performance.

ECC Module

The ECC module is instantiated inside the DDR3/DDR4 Memory Controller. It is made up of five submodules as shown in Figure 3-4.

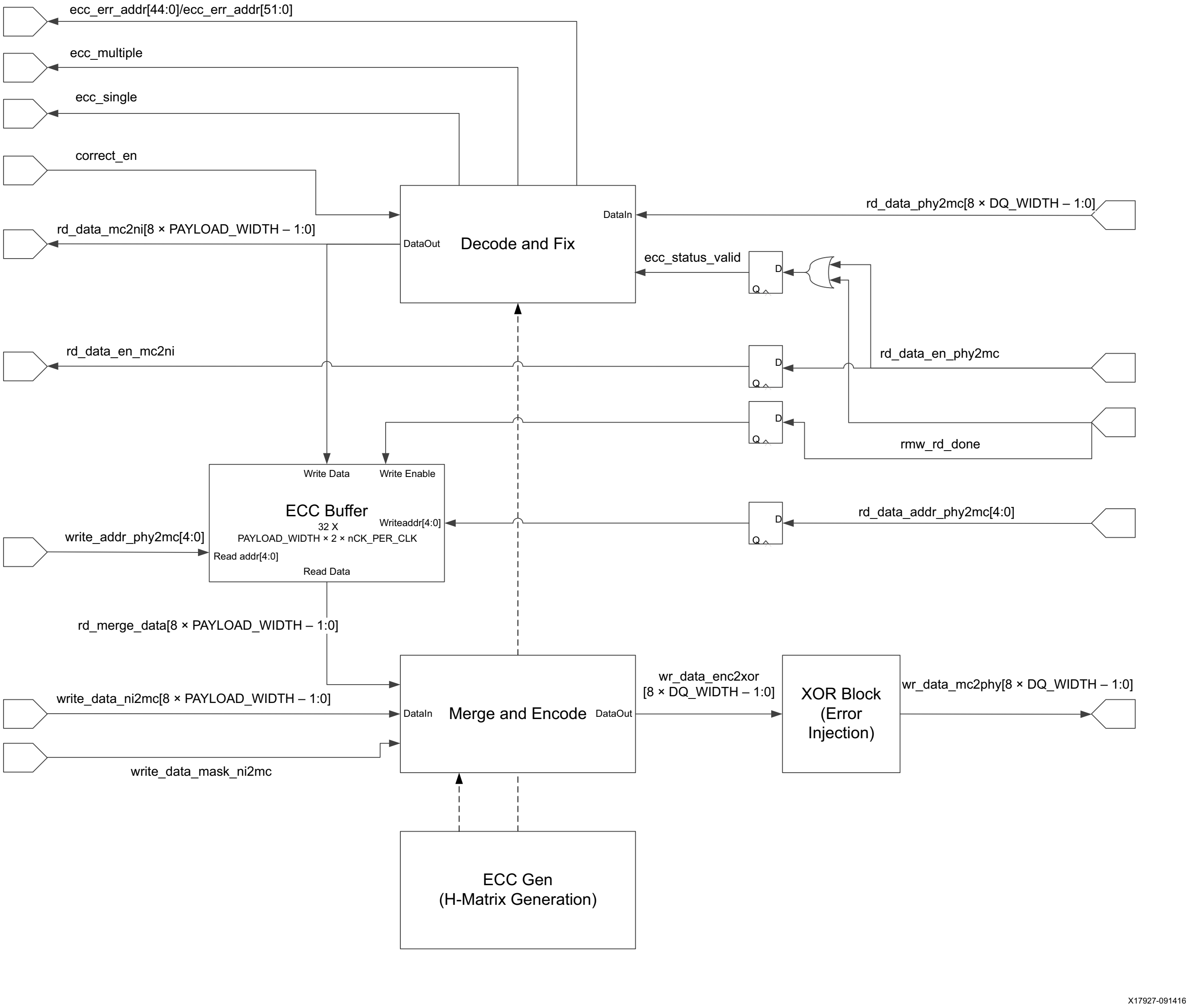


Figure 3-4: **ECC Block Diagram**

Read data and check bits from the PHY are sent to the Decode block, and on the next system clock cycle data and error indicators `ecc_single/ecc_multiple` are sent to the NI. `ecc_single` asserts when a correctable error is detected and the read data has been corrected. `ecc_multiple` asserts when an uncorrectable error is detected.

Read data is not modified by the ECC logic on an uncorrectable error. Error indicators are never asserted for "periodic reads," which are read transactions generated by the controller only for the purposes of VT tracking and are not returned to the user interface or written back to memory in an RMW flow.

Write data is merged in the Encode block with read data stored in the ECC Buffer. The merge is controlled on a per byte basis by the write data mask signal. All writes use this flow, so full writes are required to have all data mask bits deasserted to prevent unintended merging. After the Merge stage, the Encode block generates check bits for the write data. The data and check bits are output from the Encode block with a one system clock cycle delay.

The ECC Gen block implements an algorithm that generates an H-matrix for ECC check bit generation and error checking/correction. The generated code depends only on the PAYLOAD_WIDTH and DQ_WIDTH parameters, where DQ_WIDTH = PAYLOAD_WIDTH + ECC_WIDTH. Currently only DQ_WIDTH = 72 and ECC_WIDTH = 8 is supported.

Error Address

Each time a read CAS command is issued, the full DRAM address is stored in a FIFO in the decode block. When read data is returned and checked for errors, the DRAM address is popped from the FIFO and **`ecc_err_addr[51:0]`** is returned on the same cycle as signals **`ecc_single`** and **`ecc_multiple`** for the purposes of error logging or debug. Table 3-1 is a common definition of this address for DDR3 and DDR4.

Table 3-1: **ECC Error Address Definition**

ecc_err_addr [51:0]	**51**	**50:48**	**47:45**	**44**	**43:42**	**41:40**	**39:24**	**23:22**	**21:18**	**17:8**	**7:6**	**5:4**	**3**	**2**	**1:0**
DDR4 (x4/x8)	RSVD	3DS_CID	RSVD	RMW	RSVD	Row[17:0]		RSVD	RSVD	Col [9:0]	RSVD	Rank [1:0]	Group [1:0]		Bank [1:0]
DDR4 (x16)	RSVD	RSVD	RSVD	RMW	RSVD	Row[17:0]		RSVD	RSVD	Col [9:0]	RSVD	Rank [1:0]	RSVD	Group [0]	Bank [1:0]
DDR3	RSVD	RSVD	RSVD	RMW	RSVD		Row [15:0]	RSVD	Col[13:0]		RSVD	Rank [1:0]	RSVD	Bank[2:0]	

Latency

When the parameter ECC is ON, the ECC modules are instantiated and read and write data latency through the MC increases by one system clock cycle. When ECC is OFF, the data buses just pass through the MC and all ECC logic should be optimized out.

ECC Port Descriptions

Table 3-2 and Table 3-3 provide the ECC port descriptions at the User Interface.

Table 3-2: **DDR3 ECC Operation Signal Direction Description**

Signal	I/O	Description
ddr3_ecc_single[7:0]	O	The ddr3_ecc_single signal is non-zero if the read data from the external memory has a single bit error per beat of the read burst.
ddr3_ecc_multiple[7:0]	O	The ddr3_ecc_multiple signal is non-zero if the read data from the external memory has two bit errors per beat of the read burst. The SECDED algorithm does not correct the corresponding read data and puts a non-zero value on this signal to notify the corrupted read data at the User Interface.
ddr3_ecc_err_addr[44:0]	O	This bus contains the address of the current read command. The ddr3_ecc_err_addr signal is valid during the assertion of either ddr3_ecc_single or ddr3_ecc_multiple.

Table 3-3: **DDR4 ECC Operation Signal Direction Description**

Signal	I/O	Description
ddr4_ecc_single[7:0]	O	The ddr4_ecc_single signal is non-zero if the read data from the external memory has a single bit error per beat of the read burst.
ddr4_ecc_multiple[7:0]	O	The ddr4_ecc_multiple signal is non-zero if the read data from the external memory has two bit errors per beat of the read burst. The SECDED algorithm does not correct the corresponding read data and puts a non-zero value on this signal to notify the corrupted read data at the User Interface.
ddr4_ecc_err_addr[51:0]	O	This bus contains the address of the current read command. The ddr4_ecc_err_addr signal is valid during the assertion of either ddr4_ecc_single or ddr4_ecc_multiple.

Address Parity

The Memory Controller generates even command/address parity with a one DRAM clock delay after the chip select asserts Low. This signal is only used in DDR4 RDIMM configurations where parity is required by the DIMM RCD component.

Address parity is supported only for DDR4 RDIMM and LRDIMM configurations, which includes 3DS RDIMMs and LRDIMMs. The Memory Controller does not monitor the **`Alert_n`** parity error status output from the RDIMM/LRDIMM and it might return corrupted data to the User Interface after a parity error.

To detect this issue, you need to add a pin to your design to monitor the **`Alert_n`** signal. If an **`Alert_n`** event is detected, the memory contents should be considered corrupt. To recover from a parity error the Memory Controller must be reset, and all DRAM contents are lost.

PHY

The PHY is considered the low-level physical interface to an external DDR3 or DDR4 SDRAM device as well as all calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

The PHY is included in the complete Memory Interface Solution core, but can also be implemented as a standalone PHY only block. A PHY only solution can be selected if you plan to implement a custom Memory Controller. For details about interfacing to the PHY only block, see the PHY Only Interface, page 157.

IMPORTANT: *The PHY interface is not DFI-compliant.*

Overall PHY Architecture

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

The Memory Controller and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is either divided by four or divided by two. This depends on the DDR3 or DDR4 memory clock. A more detailed block diagram of the PHY design is shown in Figure 3-5.

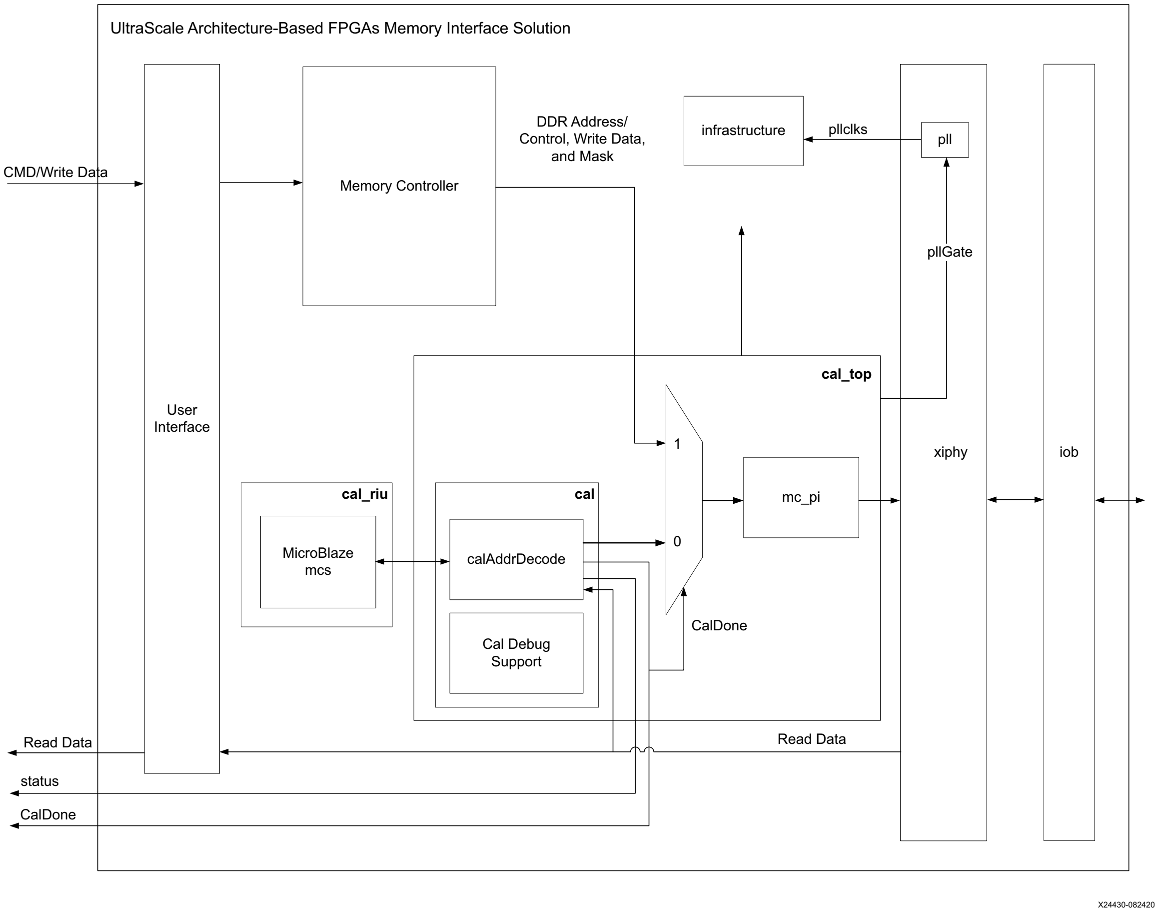


Figure 3-5: **PHY Block Diagram**

The Memory Controller is designed to separate out the command processing from the low-level PHY requirements to ensure a clean separation between the controller and physical layer. The command processing can be replaced with custom logic if desired, while the logic for interacting with the PHY stays the same and can still be used by the calibration logic.

Table 3-4: **PHY Modules**

Module Name	Description
<module>_...cal_top.sv	Contains <module>_...cal_top.sv, <module>_...mc_pi.sv, and MUXes between the calibration and the Memory Controller.
<module>_...cal_riu.sv	Contains the MicroBlaze processing system and associated logic.
<module>_...mc_pi.sv	Adjusts signal timing for the PHY for reads and writes.
<module>_...cal_addr_decode.sv	FPGA logic interface for the MicroBlaze processor.
<module>_...config_rom.sv	Configuration storage for calibration options.
microblaze_mcs_0.sv	MicroBlaze MCS module
<module>_...iob.sv	Instantiates all byte IOB modules.
<module>_...iob_byte.sv	Generates the I/O buffers for all the signals in a given byte lane.
<module>_...debug_microblaze.sv	Simulation-only file to parse debug statements from software running in MicroBlaze to indicate status and calibration results to the log.
<module>_...cal_cplx.sv	RTL state machine for complex pattern calibration.
<module>_...cal_cplx_data.sv	Data patterns used for complex pattern calibration.
<module>_...xiphy.sv	Top-level XIPHY module.
<module>_...phy.sv	Top-level of the PHY, contains pll and xiphy.sv modules.

The PHY architecture encompasses all of the logic contained in **`<module>_...phy.sv`**. The PHY contains wrappers around dedicated hard blocks to build up the memory interface from smaller components. A byte lane contains all of the clocks, resets, and datapaths for a given subset of I/O. Multiple byte lanes are grouped together, along with dedicated clocking resources, to make up a single bank memory interface. Each nibble in the PHY contains a Register Interface Unit (RIU), a dedicated integrated block in the XIPHY that provides an interface to the general interconnect logic for changing settings and delays for calibration. For more information on the hard silicon physical layer architecture, see the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7].

The memory initialization is executed in Verilog RTL. The calibration and training are implemented by an embedded MicroBlaze™ processor. The MicroBlaze Controller System (MCS) is configured with an I/O Module and a block RAM. The **`<module>_...cal_addr_decode.sv`** module provides the interface for the processor to the rest of the system and implements helper logic. The **`<module>_...config_rom.sv`** module stores settings that control the operation of initialization and calibration, providing run time options that can be adjusted without having to recompile the source code.

The address unit connects the MCS to the local register set and the PHY by performing address decode and control translation on the I/O module bus from spaces in the memory map and MUXing return data (**`<module>_...cal_addr_decode.sv`**). In addition, it provides address translation (also known as "mapping") from a logical conceptualization of the DRAM interface to the appropriate pinout-dependent location of the delay control in the PHY address space.

Although the calibration architecture presents a simple and organized address map for manipulating the delay elements for individual data, control and command bits, there is flexibility in how those I/O pins are placed. For a given I/O placement, the path to the FPGA logic is locked to a given pin. To enable a single binary software file to work with any memory interface pinout, a translation block converts the simplified RIU addressing into the pinout-specific RIU address for the target design (see Table 3-5).

The specific address translation is written by DDR3/DDR4 SDRAM after a pinout is selected and cannot be modified. The code shows an example of the RTL structure that supports this.

```
Casez(io_address)// MicroBlaze I/O module address
  // … static address decoding skipped
  //========================================//
  //===========DQ ODELAYS===================//
  //========================================//
  //Byte0
28`h0004100: begin //c0_ddr4_dq[0] IO_L20P_T3L_N2_AD1P_44
    riu_addr_cal = 6`hD;
    riu_nibble = 'h6;
  end
  // … additional dynamic addressing follows
```

In this example, `DQ0` is pinned out on Bit[0] of nibble 0 (nibble 0 according to instantiation order). The RIU address for the ODELAY for Bit[0] is `0x0D`. When `DQ0` is addressed — indicated by address `0x000_4100`), this snippet of code is active. It enables nibble 0 (decoded to one-hot downstream) and forwards the address `0x0D` to the RIU address bus.

The MicroBlaze I/O module interface is not always fast enough for implementing all of the functions required in calibration. A helper circuit implemented in `<module>_...cal_addr_decode.sv` is required to obtain commands from the registers and translate at least a portion into single-cycle accuracy for submission to the PHY. In addition, it supports command repetition to enable back-to-back read transactions and read data comparison.

Table 3-5: **XIPHY RIU Addressing and Description**

RIU Address	Name	Description
0x00	NIBBLE_CTRL0	Nibble Control 0. Control for enabling DQS gate in the XIPHY, GT_STATUS for gate feedback, and clear gate which resets gate circuit.
0x01	NIBBLE_CTRL1	Nibble Control 1. TX_DATA_PHASE control for every bit in the nibble.
0x02	CALIB_CTRL	Calibration Control. XIPHY control and status for BISC.
0x03	Reserved	Reserved
0x04	Reserved	Reserved
0x05	BS_CTRL	Bit slice reset. Resets the ISERDES and IFIFOs in a given nibble.
0x06	Reserved	Reserved
0x07	PQTR	Rising edge delay for DQS.
0x08	NQTR	Falling edge delay for DQS.

Table 3-5: **XIPHY RIU Addressing and Description *(Cont'd)***

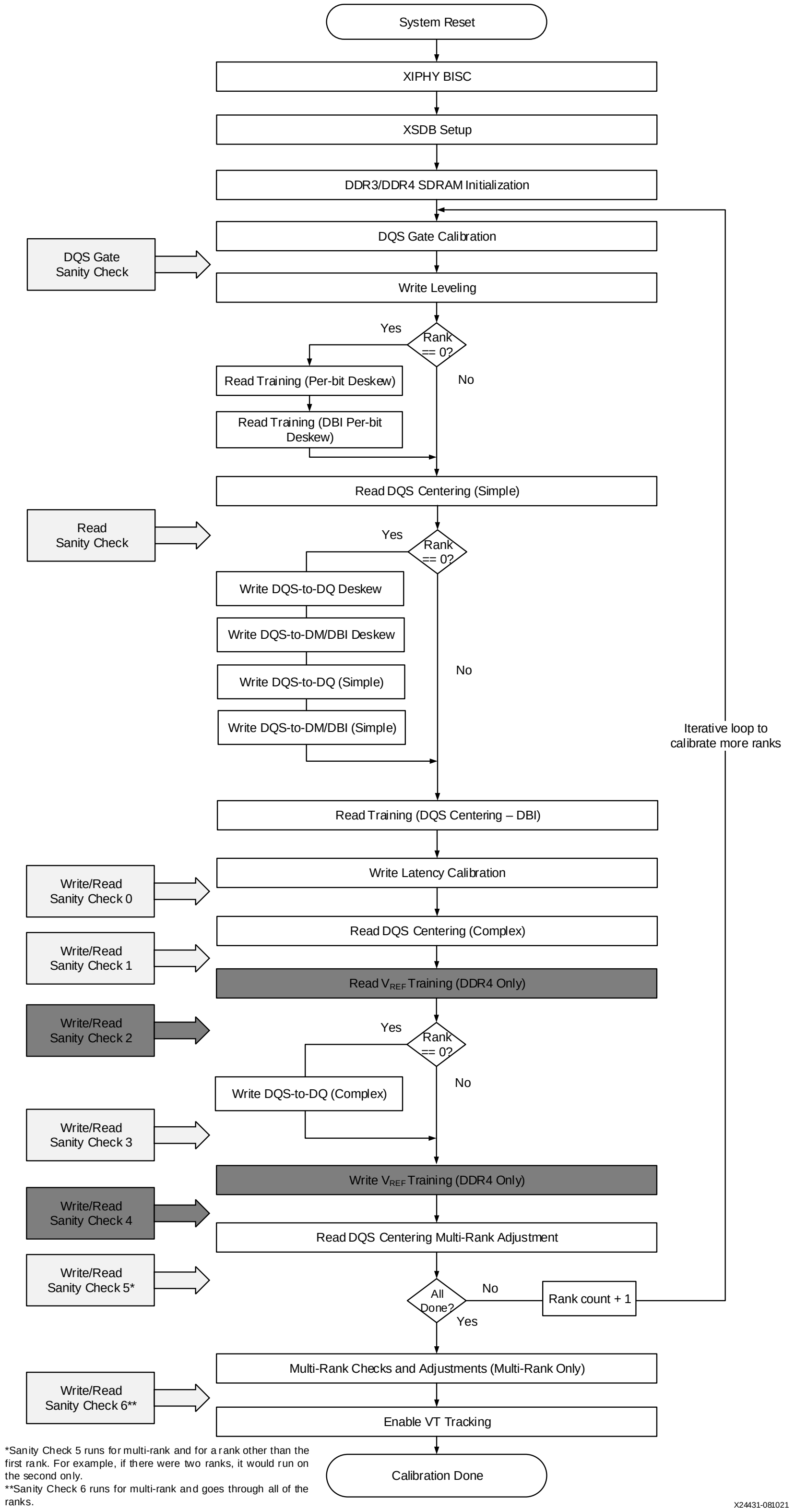
RIU Address	Name	Description
0x09	Reserved	Reserved
0x0A	TRISTATE_ODELAY	Output delay for 3-state.
0x0B	ODELAY0	Output delay for bit slice 0.
0x0C	ODELAY1	Output delay for bit slice 1.
0x0D	ODELAY2	Output delay for bit slice 2.
0x0E	ODELAY3	Output delay for bit slice 3.
0x0F	ODELAY4	Output delay for bit slice 4.
0x10	ODELAY5	Output delay for bit slice 5.
0x11	ODELAY6	Output delay for bit slice 6.
0x12	IDELAY0	Input delay for bit slice 0.
0x13	IDELAY1	Input delay for bit slice 1.
0x14	IDELAY2	Input delay for bit slice 2.
0x15	IDELAY3	Input delay for bit slice 3.
0x16	IDELAY4	Input delay for bit slice 4.
0x17	IDELAY5	Input delay for bit slice 5.
0x18	IDELAY6	Input delay for bit slice 6.
0x19	PQTR Align	BISC edge alignment computation for rising edge DQS.
0x1A	NQTR Align	BISC edge alignment computation for falling edge DQS.
0x1B to 0x2B	Reserved	Reserved
0x2C	WL_DLY_RNK0	Write Level register for Rank 0. Coarse and fine delay, WL_TRAIN.
0x2D	WL_DLY_RNK1	Write Level register for Rank 1. Coarse and fine delay.
0x2E	WL_DLY_RNK2	Write Level register for Rank 2. Coarse and fine delay.
0x2F	WL_DLY_RNK3	Write Level register for Rank 3. Coarse and fine delay.
0x30	RL_DLY_RNK0	DQS Gate register for Rank 0. Coarse and fine delay.
0x31	RL_DLY_RNK1	DQS Gate register for Rank 1. Coarse and fine delay.
0x32	RL_DLY_RNK2	DQS Gate register for Rank 2. Coarse and fine delay.
0x33	RL_DLY_RNK3	DQS Gate register for Rank 3. Coarse and fine delay.
0x34 to 0x3F	Reserved	Reserved

Memory Initialization and Calibration Sequence

After deassertion of the system reset, the PHY performs some required internal calibration steps first.

1. The built-in self-check of the PHY (BISC) is run. BISC is used in the PHY to compute internal skews for use in voltage and temperature tracking after calibration is completed.
2. After BISC is completed, calibration logic performs the required power-on initialization sequence for the memory.
3. This is followed by several stages of timing calibration for the write and read datapaths.
4. After calibration is completed, PHY calculates internal offsets to be used in voltage and temperature tracking.
5. PHY indicates calibration is finished and the controller begins issuing commands to the memory.

Figure 3-6 shows the overall flow of memory initialization and the different stages of calibration. The dark gray color is not available for this release.



*Sanity Check 5 runs for multi-rank and for a rank other than the first rank. For example, if there were two ranks, it would run on the second only.

**Sanity Check 6 runs for multi-rank and goes through all of the ranks.

Figure 3-6: **PHY Overall Initialization and Calibration Sequence**

When simulating a design out of DDR3/DDR4 SDRAM, the calibration it set to be bypassed to enable you to generate traffic to and from the DRAM as quickly as possible. When running in hardware or simulating with calibration, enabled signals are provided to indicate what step of calibration is running or, if an error occurs, where an error occurred.

The first step in determining calibration status is to check the CalDone port. After the CalDone port is checked, the status bits should be checked to indicate the steps that were ran and completed. Calibration halts on the very first error encountered, so the status bits indicate which step of calibration was last run. The status and error signals can be checked through either connecting the Vivado analyzer signals to these ports or through the XSDB tool (also through Vivado).

The calibration status is provided through the XSDB port, which stores useful information regarding calibration for display in the Vivado IDE. The calibration status and error signals are also provided as ports to allow for debug or triggering. Table 3-6 lists the pre-calibration status signal description.

Table 3-6: **Pre-Calibration XSDB Status Signal Description**

XSDB Status Register	XSDB Bits[8:0]	Description	Pre-Calibration Step
DDR_PRE_CAL_STATUS	0	Done	MicroBlaze has started up
	1	Done	Reserved
	2	Done	Reserved
	3	Done	Reserved
	4	Done	XSDB Setup Complete
	5	–	Reserved
	6	–	Reserved
	7	–	Reserved
	8	–	Reserved

Table 3-7 lists the status signals in the port as well as how they relate to the core XSDB data. In the status port, the mentioned bits are valid and the rest are reserved.

Table 3-7: **XSDB Status Signal Descriptions**

XSDB Status Register	XSDB Bits[8:0]	Status Port Bits[127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_RANKx_0	0	0	Start	DQS Gate	1
	1	1	Done	–	–
	2	2	Start	Check for DQS gate	2
	3	3	Done	–	–
	4	4	Start	Write leveling	3
	5	5	Done	–	–
	6	6	Start	Read Per-bit Deskew	4
	7	7	Done	–	–
	8	8	Start	Reserved	5
DDR_CAL_STATUS_RANKx_1	0	9	Done	–	–
	1	10	Start	Read DQS Centering (Simple)	6
	2	11	Done	–	–
	3	12	Start	Read Sanity Check	7
	4	13	Done	–	–
	5	14	Start	Write DQS-to-DQ Deskew	8
	6	15	Done	–	–
	7	16	Start	Write DQS-to-DM Deskew	9
	8	17	Done	–	–
DDR_CAL_STATUS_RANKx_2	0	18	Start	Write DQS-to-DQ (Simple)	10
	1	19	Done	–	–
	2	20	Start	Write DQS-to-DM (Simple)	11
	3	21	Done	–	–
	4	22	Start	Reserved	12
	5	23	Done	–	–
	6	24	Start	Write Latency Calibration	13
	7	25	Done	–	–
	8	26	Start	Write/Read Sanity Check 0	14

Table 3-7: **XSDB Status Signal Descriptions *(Cont'd)***

XSDB Status Register	XSDB Bits[8:0]	Status Port Bits[127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_RANKx_3	0	27	Done	–	–
	1	28	Start	Read DQS Centering (Complex)	15
	2	29	Done	–	–
	3	30	Start	Write/Read Sanity Check 1	16
	4	31	Done	–	–
	5	32	Start	Reserved	17
	6	33	Done	–	–
	7	34	Start	Write/Read Sanity Check 2	18
	8	35	Done	–	–
DDR_CAL_STATUS_RANKx_4	0	36	Start	Write DQS-to-DQ (Complex)	19
	1	37	Done	–	–
	2	38	Start	Write DQS-to-DM (Complex)	20
	3	39	Done	–	–
	4	40	Start	Write/Read Sanity Check 3	21
	5	41	Done	–	–
	6	42	Start	Reserved	22
	7	43	Done	–	–
	8	44	Start	Write/Read Sanity Check 4	23
DDR_CAL_STATUS_RANKx_5	0	45	Done	–	–
	1	46	Start	Read Level Multi-Rank Adjustment	24
	2	47	Done	–	–
	3	48	Start	Write/Read Sanity Check 5 (For More than 1 Rank)	25
	4	49	Done	–	–
	5	50	Start	Multi-Rank Adjustments and Checks	26
	6	51	Done	–	–
	7	52	Start	Write/Read Sanity Check 6 (All Ranks)	27
	8	53	Done	–	–

Table 3-8 lists the post-calibration XSDB status signal descriptions.

Table 3-8: **Post-Calibration XSDB Status Signal Description**

XSDB Status Register	XSDB Bits[8:0]	Description	Post-Calibration Step
DDR_POST_CAL_STATUS	0	Running	DQS Gate Tracking
	1	Idle	
	2	Fail	
	3	Running	Read Margin Check (Reserved)
	4	Running	Write Margin Check (Reserved)
	5	–	Reserved
	6	–	Reserved
	7	–	Reserved
	8	–	Reserved

Table 3-9 lists the error signals and a description of each error. To decode the error first look at the status to determine which calibration stage failed (the start bit would be asserted, the associated done bit deasserted) then look at the error code provided. The error asserts the first time an error is encountered.

Table 3-9: **Error Signal Descriptions**

STAGE_NAME	Stage	Code	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Error
DQS Gate	1	0x1	Byte	RIU Nibble	Calibration uses the calculated latency from the MPR register as a starting point and then backs off and begins sampling. If the sample occurs too late in the DQS burst and there are no taps left to decrement for the latency, then an error has occurred.
		0x2	Byte	RIU Nibble	Expected pattern was not found on GT_STATUS.
		0x3	Byte	RIU Nibble	CAS latency is too low. Calibration starts at a CAS latency (CL) – 3. For allowable CAS latencies, see EXTRA_CMD_DELAY Configuration Settings, page 173.
		0x4	Byte	RIU Nibble	Pattern not found on GT_STATUS, all 0s were sampled. Expecting to sample the preamble.
		0x5	Byte	RIU Nibble	Pattern not found on GT_STATUS, all 1s were sampled. Expecting to sample the preamble.
		0x6	Byte	RIU Nibble	Could not find the 0->1 transition with fine taps in at least 1 tck (estimated) of fine taps.
		0x7	Byte	RIU Nibble	Underflow of coarse taps when trying to limit maximum coarse tap setting.
		0x8	Byte	RIU Nibble	Violation of maximum read latency limit.
		0x9	Byte	RIU Nibble	Data check failed with DQS gate settings and read latency range has been exhausted.
DQS Gate Sanity Check	2	0xF	N/A	N/A	PHY fails to return same number of data bursts as expected
Write Leveling	3	0x1	Byte	N/A	Cannot find stable 0.
		0x2	Byte	N/A	Cannot find stable 1.
		0x3	Byte	N/A	Cannot find the left edge of noise region with fine taps.
		0x4	Byte	N/A	Could not find the 0->1 transition with fine taps in at least 1 tck (estimated) of ODELAY taps.
Read Per-Bit Deskew	4	0x1	Nibble	Bit	No valid data found for a given bit in the nibble when running the deskew pattern.
		0xF	Nibble	Bit	Timeout error waiting for read data bursts to return.

Table 3-9: **Error Signal Descriptions *(Cont'd)***

STAGE_NAME	Stage	Code	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Error
Read DQS Centering	6	0x1	Nibble	Bit	No valid data found for a given bit in the nibble.
		0x2	Nibble	Bit	Could not find the left edge of the data valid window to determine window size. All samples returned valid data.
		0xF	Nibble	Bit	Timeout error waiting for read data to return.
Read Sanity Check	7	0x1	Nibble	0	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for read data to return.
Write DQS-to-DQ Deskew	8	0x1	Byte	Bit	DQS deskew error. No valid data found; therefore, ran out of taps during search.
		0x2	Byte	Bit	DQ deskew error. Failure point not found.
		0xF	Byte	Bit	Timeout error waiting for all read data bursts to return.
Write DQS-to-DM/ DBI Deskew	9	0x1	Byte	Bit	DQS deskew error. No valid data found; therefore, ran out of taps during search.
		0x2	Byte	Bit	DM/DBI deskew error. Failure point not found.
		0xF	Byte	Bit	Timeout error waiting for all read data bursts to return.
Write DQS-to-DQ (Simple)	10	0x1	Byte	N/A	No valid data found; therefore, ran out of taps during search.
		0xF	Byte	N/A	Timeout error waiting for read data to return.
Write DQS-to-DM (Simple)	11	0x1	Byte	N/A	No valid data found; therefore, ran out of taps during search.
		0xF	Byte	N/A	Timeout error waiting for all read data bursts to return.
Write Latency Calibration	13	0x1	Byte	N/A	Could not find the data pattern within the allotted number of taps.
		0x2	Byte	N/A	Data pattern not found. Data late at the start, instead of F0A55A96, found 00F0A55A.
		0x3	Byte	N/A	Data pattern not found. Data too early, not enough movement to find pattern. Found pattern of A55A96FF, 5A96FFFF, or 96FFFFFF.
		0x4	Byte	N/A	Data pattern not found. Multiple reads to the same address resulted in a read mismatch.
		0xF	Byte	N/A	Timeout error waiting for read data to return.
Write Read Sanity Check	14	0x1	Nibble	0	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for read data to return.

Table 3-9: **Error Signal Descriptions *(Cont'd)***

STAGE_NAME	Stage	Code	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Error
Read_Leveling (Complex)	15	See Read DQS Centering error codes.			
Write Read Sanity Check	16	0x1	Nibble	N/A	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for all read data bursts to return.
Read V_{REF} Training	17	0x1	Byte	N/A	No valid window found for any V_{REF} value.
		0xF	Nibble	N/A	Timeout error waiting for read data to return.
Write Read Sanity Check	18	0x1	Nibble	0	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for read data to return.
Write DQS-to-DQ (Complex)	19	See Write DQS-to-DQ (Simple) error codes.			
Write Read Sanity Check	21	0x1	Nibble	N/A	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for all read data bursts to return.
Write V_{REF} Training	22	0x1	Byte	N/A	No valid window found for any V_{REF} value.
		0x2	Byte	N/A	Readback Write V_{REF} value from the DRAM does not match expected.
		0xF	Byte	N/A	Timeout error waiting for read data to return.
Write Read Sanity Check	23	0x1	Nibble	N/A	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for all read data bursts to return.
Write Read Sanity Check	25	0x1	Nibble	0	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for read data to return.
Multi-Rank Adjust and Checks	26	0x1	Byte	RIU Nibble	Could not find common setting across ranks for general interconnect read latency setting for given byte. Variance between ranks could not be compensated with coarse taps.
		0x2	Byte	RIU Nibble	DQS Gate skew between ranks for a given byte larger than 360°.
		0x3	Byte	RIU Nibble	Write skew between ranks for a given byte larger than 180°. Check Write Latency Coarse settings.
		0x4	Byte	N/A	Could not decrement coarse taps enough to limit coarse tap setting for all ranks.
		0x5	Byte	N/A	Violation of maximum read latency limit.
Write Read Sanity Check	27	0x1	Nibble	RIU Nibble	Read data comparison failure.
		0xF	N/A	N/A	Timeout error waiting for read data to return.

Table 3-9: **Error Signal Descriptions *(Cont'd)***

STAGE_NAME	Stage	Code	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Error
DQS Gate Tracking		0x1	Byte	Rank	Underflow of the coarse taps used for tracking.
		0x2	Byte	Rank	Overflow of the coarse taps used for tracking.

DQS Gate

During this stage of calibration, the read DQS preamble is detected and the gate to enable data capture within the FPGA is calibrated to be one clock cycle before the first valid data on DQ. The coarse and fine DQS gate taps (RL_DLY_COARSE and RL_DLY_FINE) are adjusted during this stage. Read commands are issued with gaps in between to continually search for the DQS preamble position. The DDR4 preamble training mode is enabled during this stage to increase the low preamble period and aid in detection. During this stage of calibration, only the read DQS signals are monitored and not the read DQ signals. DQS Preamble Detection is performed sequentially on a per byte basis.

During this stage of calibration, the coarse taps are first adjusted while searching for the low preamble position and the first rising DQS edge, in other words, a DQS pattern of 00X1.

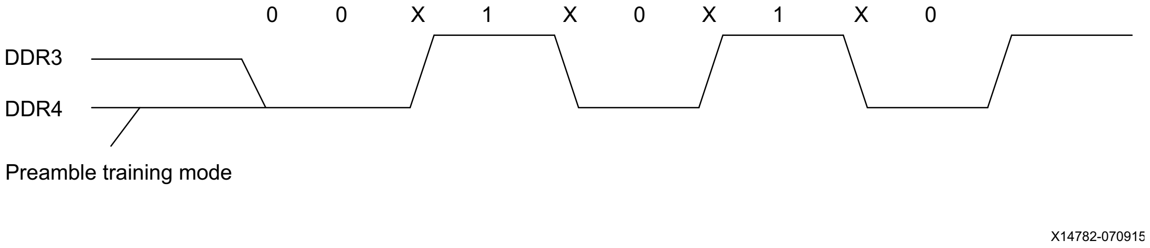


Figure 3-7: **DDR3 vs. DDR4 Preamble**

If the preamble is not found, the read latency is increased by one. The coarse taps are reset and then adjusted again while searching for the low preamble and first rising DQS edge. After the preamble position is properly detected, the fine taps are then adjusted to fine tune and edge align the position of the sample clock with the DQS.

DQS Gate Sanity Check

After completion of DQS gate calibration for all bytes in a given rank, read return timing is calculated and 10 read bursts with gaps between them are issued. Logic then checks that the FIFO is read 10 times. There is no data checking at this stage. This is just a basic functional check of the FIFO read port control logic, which is configured using the DQS gate calibration results. Read return timing is updated after DQS gate calibration for each rank. The final setting is determined by largest DQS gate delay out of all DQS lanes and all ranks.

Write Leveling

DDR3/DDR4 write leveling allows the controller to adjust each write DQS phase independently with respect to the CK forwarded to the DDR3/DDR4 SDRAM device. This compensates for the skew between DQS and CK and meets the tDQSS specification.

During write leveling, DQS is driven by the FPGA memory interface and DQ is driven by the DDR3/DDR4 SDRAM device to provide feedback. DQS is delayed until the 0 to 1 edge transition on DQ is detected. The DQS delay is achieved using both ODELAY and coarse tap delays.

After the edge transition is detected, the write leveling algorithm centers on the noise region around the transition to maximize margin. This second step is completed with only the use of ODELAY taps. Any reference to “FINE” is the ODELAY search.

Read DQS Deskew and Centering

Read Leveling is performed over multiple stages to maximize the data eye and center the internal read sampling clock in the read DQ window for robust sampling. To perform this, Read Leveling performs the following sequential steps:

1. Maximizes the DQ eye by removing skew and OCV effects using per bit read DQ deskew.
    - See Debugging Per-Bit Deskew Failures for details.
2. Sweeps DQS across all DQ bits and finds the center of the data eye using both easy (Multi-Purpose register data pattern) and complex data patterns. Centering of the data eye is completed for both the DQS and DQS#.
    - See Debugging Read MPR DQS Centering Failures for details.
    - See Debugging Complex Pattern Calibration Failures section for details.
3. Post calibration, continuously maintains the relative delay of DQS versus DQ across the VT range.

Read Per-Bit Deskew

Per-bit deskew is performed on a per-bit basis whereas Read Leveling DQS centering is performed on a per-nibble basis.

During per-bit deskew, Read Leveling Calibration, a pattern of `00000000_11111111` is written and read back while DQS adjustments (PQTR and NQTR individual fine taps on DQS) and DQ adjustments (IDELAY) are made.

At the end of this stage, the DQ bits are internally deskewed to the left edge of the incoming DQS.

Read DBI Per-Bit Deskew

If the Read DBI option is selected for DDR4, the DBI input pin is calibrated by powering on the read DBI functionality, and reading back the regular deskew pattern from the DRAM, `00000000_11111111`. The DRAM sends the data back as `11111111_11111111` but the DBI pin itself has the `00000000_11111111` pattern, that is used to calibrate the DBI input pin itself.

Debugging Read DQS Centering (Simple)

During DQS read centering (simple), the toggling `01010101` MPR pattern is continuously read back while DQS adjustments (PQTR and NQTR individual fine taps on DQS) and DQ adjustments (IDELAY) are made. This is to establish an initial DQS center point using an easy pattern that does not rely on writing a pattern to the DRAM.

Read Sanity Check

After read DQS centering but before Write DQS-to-DQ, a check of the data is made to ensure the previous stage of calibration did not inadvertently leave the alignment of the read path in a bad spot. A single MPR read command is sent to the DRAM, and the data is checked against the expected data across all bytes before continuing.

Write DQS-to-DQ

This stage of calibration is required to center align the write DQS in the write DQ window per bit. At the start of Write DQS Centering and Per-Bit Deskew, DQS is aligned to CK but no adjustments on the write window have been made. Write window adjustments are made in the following two sequential stages:

- Write Per-Bit Deskew
- Write DQS Centering

Write DQS-to-DQ Per-Bit Deskew

During write per-bit deskew, a toggling `10101010` pattern is continuously written and read back while making 90^{o} clock phase adjustments on the write DQ along with individual fine ODELAY adjustments on DQS and DQ. At the end of per-bit write DQ deskew, the write DQ bits are aligned as they are transmitted to the memory.

Write DQS-to-DQ Centering

During Write DQS Centering, the same toggling `10101010` pattern is continuously written and read back. ODELAY adjustments on DQS and DQ are also made but all of the DQ ODELAY adjustments for a given byte are made in step to maintain the previously deskewed alignment.

Write DQS-to-DM/DBI

When the write DBI option is selected for DDR4, the pin itself is calibrated as a DM and write DBI is enabled at the end of calibration.

In all previous stages of calibration, data mask signals are driven low before and after the required amount of time to ensure they have no impact on calibration. Now, both the read and the writes have been calibrated and data mask can reliably be adjusted. If DM signals are not used within the interface, this stage of calibration is skipped.

During DM Calibration, a data pattern of `55555555_55555555` is first written to address `0x000` followed by a write to the same address but with a data pattern of `BBBBBBBB_BBBBBBBB` with DM asserted during the rising edge of DQS. A read is then issued where the expected read back pattern is all "B" except for the data where DM was asserted. In these masked locations, a 5 is expected. The same series of steps completed during Write Per-Bit Deskew and Write DQS Centering is then completed but for the DM bits.

Read DQS Centering (DBI)

If the Read DBI option is selected for DDR4, the position of the DQS in the data valid window must also use the timing information of the DBI pin itself, because the DBI pin can be the limit to the data valid window.

The `0F0F0F0F` pattern is written to the DRAM and read back with read DBI enabled. The DRAM sends the data back as `FFFFFFFF` but the DBI pin has the clock pattern `01010101`, that is used to measure the data valid window of the DBI input pin itself. The final DQS location is determined based on the aggregate window for the DQ and DBI pins.

Write Latency Calibration

Write Latency Calibration is required to align DQS to the correct CK edge. During write leveling, DQS is aligned to the nearest rising edge of CK. However, this might not be the edge that captures the write command.

Depending on the interface type (UDIMM, RDIMM, LRDIMM, or component), the DQS could either be one CK cycle earlier than, two CK cycles earlier than, or aligned to the CK edge that captures the write command.

This is a pattern based calibration where coarse adjustments are made on a per byte basis until the expected on time write pattern is read back. The process is as follows:

1. Issue extended writes followed by a single read.
2. Check the pattern readback against the expected patterns.
3. If necessary add coarse adjustments.

4. Repeat until the on time write pattern is read back, signifying DQS is aligned to the correct CK cycle, or an incorrect pattern is received resulting in a Write Latency failure.

The following data is written at address `0x000`:

- Data pattern before (with extra DQS pulses): `0000000000000000`
- Data pattern written to address `0x000`: `FF00AA5555AA9966`
- Data pattern after (with extra DQS pulses): `FFFFFFFFFFFFFFFF`

Reads are then performed where the following patterns can be calibrated:

- On time write pattern read back: `FF00AA5555AA9966` (no adjustments needed)
- One DQS early write pattern read back: `AA5555AA9966FFFF`
- Two DQS early write pattern read back: `55AA9966FFFFFFFF`
- Three DQS early write pattern read back: `9966FFFFFFFFFFFF`

Write Latency Calibration can fail for the following cases and signify a board violation between DQS and CK trace matching:

- Four DQS early pattern `FFFFFFFFFFFFFFFF`
- One DQS late write pattern read back: `0000FF00AA5555AA`
- Two DQS late write pattern read back: `00000000FF00AA55`
- Three DQS late write pattern read back: `000000000000FF00`

Write/Read Sanity Check

After Write DQS-to-DQ, a check of the data is made to ensure the previous stage of calibration did not inadvertently leave the write or read path in a bad spot. A single write burst followed by a single read command to the same location is sent to the DRAM, and the data is checked against the expected data across all bytes before continuing. During this step, the expected data pattern as seen on a nibble is `937EC924`.

Read DQS Centering (Complex)

The final stage of DQS read centering that is completed before normal operation is repeating the steps performed during MPR DQS read centering but with a difficult/complex pattern. The purpose of using a complex pattern is to stress the system for SI effects such as ISI and noise while calculating the read DQS center position. This ensures that the read center position can reliably capture data with margin in a true system.

Write DQS-to-DQ Centering (Complex)

Note: The calibration step is only enabled for the first rank in a multi-rank system. Also, this is only enabled for data rates above 1,600 Mb/s.

For the same reasons as described in the Read DQS Centering (Complex), a complex data pattern is used on the write path to adjust the Write DQS-to-DQ alignment. The same steps as detailed in the Write DQS-to-DQ Centering are repeated just with a complex data pattern.

Read Leveling Multi-Rank Adjustment

For multi-rank systems the read DQS centering algorithm is ran on each rank, but the final delay setting must be common for all ranks. The results of training each rank separately are stored in XSDB, but the final delay setting is a computed average of the training results across all ranks. The final PQTR/NQTR delay is indicated by RDLVL_PQTR_CENTER_FINAL_NIBBLE/ RDLVL_NQTR_CENTER_FINAL_NIBBLE, while the DQ IDELAY is RDLVL_IDELAY_FINAL_BYTE_BIT.

Multi-Rank Adjustments and Checks

DQS Gate Multi-Rank Adjustment

During DQS gate calibration for multi-rank systems, each rank is allowed to calibrate independently given the algorithm as described in DQS Gate, page 48. After all ranks have been calibrated, an adjustment is required before normal operation to ensure fast rank-to-rank switching. The general interconnect signal **`clb2phy_rd_en`** (indicated by DQS_GATE_READ_LATENCY_RANK_BYTE in XSDB) that controls the gate timing on a DRAM-clock-cycle resolution is adjusted here to be the same for a given byte across all ranks.

The coarse taps are adjusted so the timing of the gate opening stays the same for any given rank, where four coarse taps are equal to a single read latency adjustment in the general interconnect. During this step, the algorithm tries to find a common **`clb2phy_rd_en`** setting where across all ranks for a given byte the coarse setting would not overflow or underflow, starting with the lowest read latency setting found for the byte during calibration. If the lowest setting does not work for all ranks, the **`clb2phy_rd_en`** increments by one and the check is repeated. The fine tap setting is < 90°, so it is not included in the adjustment.

If the check reaches the maximum **`clb2phy_rd_en`** setting initially found during calibration without finding a value that works between all ranks for a byte, an error is asserted. If after the adjustment is made and the coarse taps are larger than 360° (four coarse tap settings), a different error is asserted. For the error codes, see Table 3-9, "Error Signal Descriptions," on page 45.

For multi-rank systems, the coarse taps must be seven or less so additional delay is added using the general interconnect read latency to compensate for the coarse tap requirement.

Write Latency Multi-Rank Check

The write latency is allowed to fall wherever it can in multi-rank systems, each rank is allowed to calibrate independently given the algorithms in Write Leveling and Write Latency Calibration. After all ranks have been calibrated and before it finishes, a check is made to ensure certain XIPHY requirements are met on the write path. The difference in write latency between the ranks is allowed to be 180° (or two XIPHY coarse taps).

Enable VT Tracking

After the DQS gate multi-rank adjustment (if required), a signal is sent to the XIPHY to recalibrate internal delays to start voltage and temperature tracking. The XIPHY asserts a signal when complete, **`phy2clb_phy_rdy_upp`** for upper nibbles and **`phy2clb_phy_rdy_low`** for lower nibbles.

For multi-rank systems, when all nibbles are ready for normal operation there is a requirement of the XIPHY where two write-read bursts are required to be sent to the DRAM before starting normal traffic. A data pattern of F00FF00F is used for the first and 0FF00FF0 for the second. The data itself is not checked and is expected to fail.

Write Read Sanity Check (Multi-Rank Only)

For multi-rank systems, a check of the data for each rank is made to ensure the previous stages of calibration did not inadvertently leave the write or read path in a bad spot. A single write burst followed by a single read command to the same location is sent to each DRAM rank. The data is checked against the expected data across all bytes before continuing.

After all stages are completed across all ranks without any error, calDone gets asserted to indicate user traffic can begin. In XSDB, DBG_END contains **`0x1`** if calibration completes and 0x2 if there is a failure.

Read and Write V_{REF} Calibration

Starting with the release of Vivado 2016.1, both read and write V_{REF} calibration is disabled. Through characterization it has been determined that read and write V_{REF} calibration are not required. The eye sizes found with the default read/write V_{REF} settings are comparable to the eye sizes found with the calibrated read/write V_{REF} values. As these stages of calibration add calibration time, the stages do not have a positive effect on the eye sizes. Therefore, stages are disabled starting with the 2016.1 release.

If you would like to manually re-enable the stages, follow these steps:

1. Follow the steps in *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] for modifying IP in the "Editing IP Sources" section.
2. Open the `core_name/rtl/ip_top/core_name_ddr4.sv` in a text editor outside of the Vivado Integrated Design Environment.
3. Locate the following lines:

   ```
   parameter CAL_RD_VREF = "SKIP",
   parameter CAL_RD_VREF_PATTERN = "SIMPLE",
   parameter CAL_WR_VREF = "SKIP",
   parameter CAL_WR_VREF_PATTERN = "SIMPLE",
   ```

   Note: These lines occur twice. Once under `ifdef` SIMULATION and again under `else`. You need to modify the lines within the `else`.
4. Modify the SKIP setting to full:

   ```
   parameter CAL_RD_VREF = "FULL",
   parameter CAL_RD_VREF_PATTERN = "SIMPLE",
   parameter CAL_WR_VREF = "FULL",
   parameter CAL_WR_VREF_PATTERN = "SIMPLE",
   ```

DDR4 LRDIMM Memory Initialization and Calibration Sequence

Most of the LRDIMM calibration sequence details are in line with the DDR4 core calibration sequence details as described in the previous Memory Initialization and Calibration Sequence section, unless otherwise stated below.

Figure 3-8 shows the overall flow of memory initialization and the different stages of the LRDIMM calibration sequence.

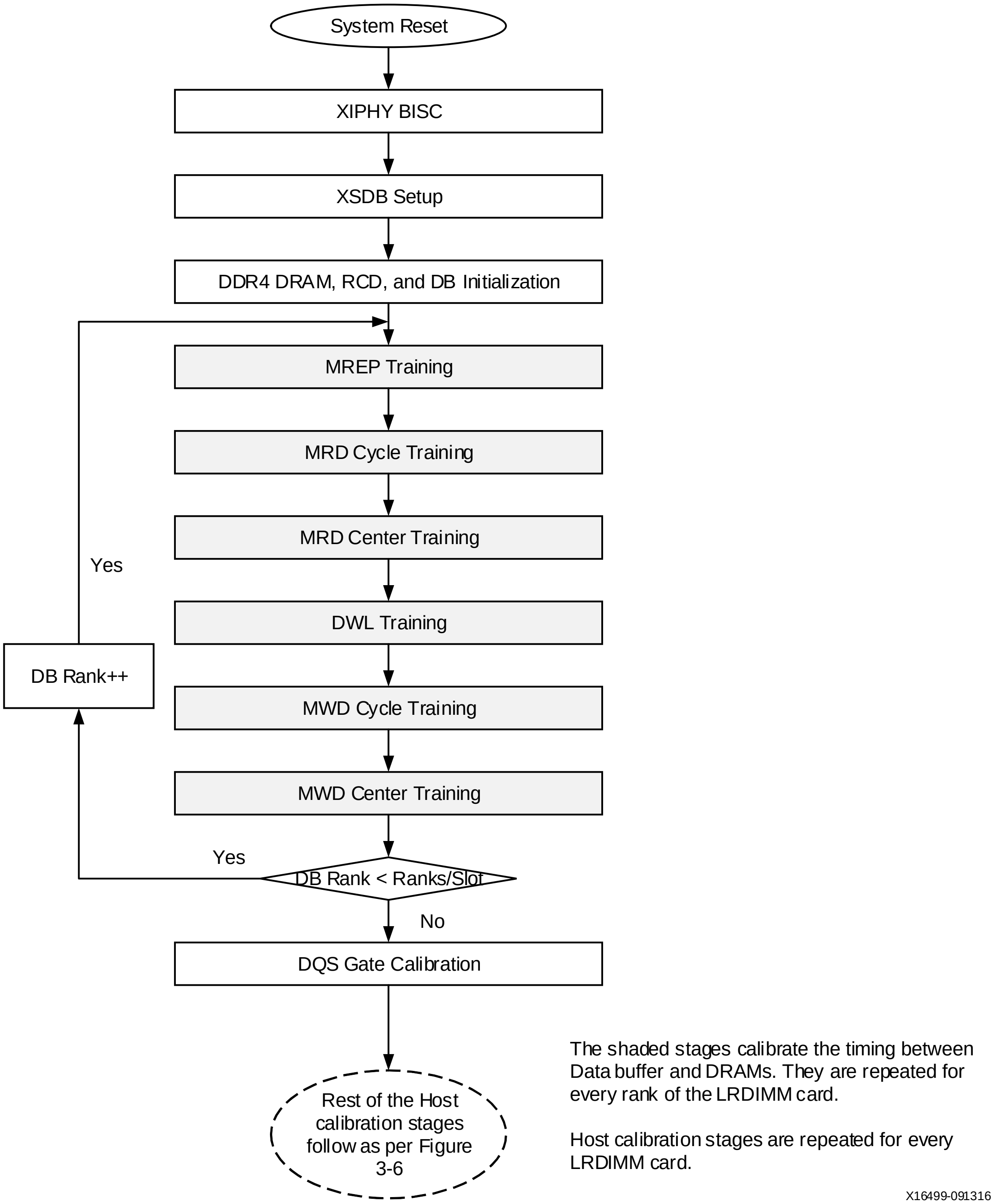


Figure 3-8: **LRDIMM Calibration Sequence**

The following data buffer calibration stages are added to meet the timing between the data buffer and DRAMs and these are repeated for each and every rank of the LRDIMM card/slot.

- MREP Training
- MRD Cycle Training
- MRD Center Training
- DWL Training
- MWD Cycle Training
- MWD Center Training

Whereas the host side calibration stages would exercise the timing between host and data buffer and they are performed once per every LRDIMM card/slot.

All the calibration stages between data buffer and DRAMs are exercised first and then the host side calibration stages are exercised.

At the end of each of the data buffer calibration stages, Per Buffer Addressing (PBA) mode is enabled to program the calibrated latency and the delay values into the data buffer registers.

The following sections describe the data buffer calibration stages.

MREP Training

This training is to align the Read MDQS phase with the data buffer clock. In this training mode, host drives the read commands, DRAM sends out the MDQS, data buffer samples the strobe with the clock, and feeds the result on DQ. Calibration continues to perform this training to find the 1 to 0 transition on Read MDQS sampled with the data buffer clock.

MRD Cycle Training

This training is to find out the correct cycle to maintain the set Read Latency value at the data buffer. In this training mode, host pre-programs the DB MPR registers with the expected pattern and issues the read commands. Data buffer compares the read data with the expected data and feeds the result on to the DQ bus. Calibration picks up the correct cycle based on the result of the comparison.

MRD Center Training

This training is to perform center alignment of the Read MDQS in the Read MDQ window at the data buffer. In this training mode, host pre-programs the DB MPR registers with the expected pattern and issues the read commands. Data buffer compares the read data with the expected data and feeds the result on to the DQ bus. Calibration finds the left and right edges of the valid window and centers it.

DWL Training

This training is to align the Write MDQS phase with the DRAM clock. In this training mode, DB drives the MDQS pulses, DRAM samples the clock with MDQS, and feeds the result on to MDQ. Data buffer forwards this result from MDQ to DQ. Calibration continues to perform this training to find 0 to 1 transition on the clock sampled with the Write Read at the DRAM.

MWD Cycle Training

This training is to find out the right cycle to maintain the set Write Latency value in the DRAM. In this training mode, host pre-programs the DB MPR registers with the expected pattern, issues the write commands to load the data into memory and issues the reads to the memory. Data buffer compares the read data with the expected data and feeds the result on to the DQ bus. Calibration picks up the correct cycle based on the result of the comparison.

MWD Center Training

This training is to center align the Write MDQS in the Write MDQ window at the DRAM. In this training mode, host pre-programs the DB MPR registers with the expected pattern, issues the write commands to load the data into memory, and issues the reads to the memory. Data buffer would compare the read data with the expected data and feeds the result on to the DQ bus. Calibration finds the left and right edges of the valid window and centers it.

CAL_STATUS

There are two types of LRDIMM devices available: dual-rank cards and quad-rank cards. Because the data buffer calibration stages are repeated for every rank of the card, the calibration sequence numbering is going to be different for dual-rank cards versus quad-rank cards.

The calibration status is provided through the XSDB port, which stores useful information regarding calibration for display in the Vivado IDE. The calibration status is provided as ports to allow for debug or triggering.

Table 3-10 lists the calibration status signals in the port as well as how they relate to the core XSDB data for dual-rank LRDIMM card.

Table 3-10: **XSDB Status Signal Description for Dual-Rank LRDIMM Card**

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_0	0	0	Start	Data Buffer Rank 0 MREP	1
	1	1	Done	–	–
	2	2	Start	Data Buffer Rank 0 MRD Cycle	2
	3	3	Done	–	–
	4	4	Start	Data Buffer Rank 0 MRD Center	3
	5	5	Done	–	–
	6	6	Start	Data Buffer Rank 0 DWL	4
	7	7	Done	–	–
	8	8	Start	Data Buffer Rank 0 MWD Cycle	5
DDR_CAL_STATUS_SLOTx_1	0	9	Done	–	–
	1	10	Start	Data Buffer Rank 0 MWD Center	6
	2	11	Done	–	–
	3	12	Start	Data Buffer Rank 1 MREP	7
	4	13	Done	–	–
	5	14	Start	Data Buffer Rank 1 MRD Cycle	8
	6	15	Done	–	–
	7	16	Start	Data Buffer Rank 1 MRD Center	9
	8	17	Done	–	–
DDR_CAL_STATUS_SLOTx_2	0	18	Start	Data Buffer Rank 1 DWL	10
	1	19	Done	–	–
	2	20	Start	Data Buffer Rank 1 MWD Cycle	11
	3	21	Done	–	–
	4	22	Start	Data Buffer Rank 1 MWD Center	12
	5	23	Done	–	–
	6	24	Start	DQS Gate	13
	7	25	Done	–	–
	8	26	Start	DQS Gate Sanity Check	14

Table 3-10: **XSDB Status Signal Description for Dual-Rank LRDIMM Card *(Cont'd)***

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_3	0	27	Done	–	–
	1	28	Start	Write Leveling	15
	2	29	Done	–	–
	3	30	Start	Read Per-Bit Deskew	16
	4	31	Done	–	–
	5	32	Start	Read Per-Bit DBI Deskew	17
	6	33	Done	–	–
	7	34	Start	Read DQS Centering (Simple)	18
	8	35	Done	–	–
DDR_CAL_STATUS_SLOTx_4	0	36	Start	Read Sanity Check	19
	1	37	Done	–	–
	2	38	Start	Write DQS to DQ Deskew	20
	3	39	Done	–	–
	4	40	Start	Write DQS to DM/DBI Deskew	21
	5	41	Done	–	–
	6	42	Start	Write DQS to DQ (Simple)	22
	7	43	Done	–	–
	8	44	Start	Write DQS to DM/DBI (Simple)	23
DDR_CAL_STATUS_SLOTx_5	0	45	Done	–	–
	1	46	Start	Read DQS Centering DBI (Simple)	24
	2	47	Done	–	–
	3	48	Start	Write Latency Calibration	25
	4	49	Done	–	–
	5	50	Start	Write Read Sanity Check 0	26
	6	51	Done	–	–
	7	52	Start	Read DQS Centering (Complex)	27
	8	53	Done	–	–

Table 3-10: **XSDB Status Signal Description for Dual-Rank LRDIMM Card *(Cont'd)***

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_6	0	54	Start	Write Read Sanity Check 1	28
	1	55	Done	–	–
	2	56	Start	Read V_{REF} Training	29
	3	57	Done	–	–
	4	58	Start	Write Read Sanity Check 2	30
	5	59	Done	–	–
	6	60	Start	Write DQS to DQ (Complex)	31
	7	61	Done	–	–
	8	62	Start	Write DQS to DM/DBI (Complex)	32
DDR_CAL_STATUS_SLOTx_7	0	63	Done	–	–
	1	64	Start	Write Read Sanity Check 3	33
	2	65	Done	–	–
	3	66	Start	Write V_{REF} Training	34
	4	67	Done	–	-
	5	68	Start	Write Read Sanity Check 4	35
	6	69	Done	–	–
	7	70	Start	Read DQS Centering Multi Rank Adjustment	36
	8	71	Done	–	–
DDR_CAL_STATUS_SLOTx_8	0	72	Start	Write Read Sanity Check 5	37
	1	73	Done	–	–
	2	74	Start	Multi Rank Adjustment and Checks	38
	3	75	Done	–	-
	4	76	Start	Write Read Sanity Check 6	39
	5	77	Done	–	–

Table 3-11 lists the calibration status signals in the port as well as how they relate to the core XSDB data for quad-rank LRDIMM card.

Table 3-11: **Status Signal Description for Quad-Rank LRDIMM Card**

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_0	0	0	Start	Data Buffer Rank 0 MREP	1
	1	1	Done	–	–
	2	2	Start	Data Buffer Rank 0 MRD Cycle	2
	3	3	Done	–	–
	4	4	Start	Data Buffer Rank 0 MRD Center	3
	5	5	Done	–	–
	6	6	Start	Data Buffer Rank 0 DWL	4
	7	7	Done	–	–
	8	8	Start	Data Buffer Rank 0 MWD Cycle	5
DDR_CAL_STATUS_SLOTx_1	0	9	Done	–	–
	1	10	Start	Data Buffer Rank 0 MWD Center	6
	2	11	Done	–	–
	3	12	Start	Data Buffer Rank 1 MREP	7
	4	13	Done	–	–
	5	14	Start	Data Buffer Rank 1 MRD Cycle	8
	6	15	Done	–	–
	7	16	Start	Data Buffer Rank 1 MRD Center	9
	8	17	Done	–	–
DDR_CAL_STATUS_SLOTx_2	0	18	Start	Data Buffer Rank 1 DWL	10
	1	19	Done	–	–
	2	20	Start	Data Buffer Rank 1 MWD Cycle	11
	3	21	Done	–	–
	4	22	Start	Data Buffer Rank 1 MWD Center	12
	5	23	Done	–	–
	6	24	Start	Data Buffer Rank 2 MREP	13
	7	25	Done	–	–
	8	26	Start	Data Buffer Rank 2 MRD Cycle	14

Table 3-11: **Status Signal Description for Quad-Rank LRDIMM Card *(Cont'd)***

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_3	0	27	Done	–	–
	1	28	Start	Data Buffer Rank 2 MRD Center	15
	2	29	Done	–	–
	3	30	Start	Data Buffer Rank 2 DWL	16
	4	31	Done	–	–
	5	32	Start	Data Buffer Rank 2 MWD Cycle	17
	6	33	Done	–	–
	7	34	Start	Data Buffer Rank 2 MWD Center	18
	8	35	Done	–	–
DDR_CAL_STATUS_SLOTx_4	0	36	Start	Data Buffer Rank 3 MREP	19
	1	37	Done	–	–
	2	38	Start	Data Buffer Rank 3 MRD Cycle	20
	3	39	Done	–	–
	4	40	Start	Data Buffer Rank 3 MRD Center	21
	5	41	Done	–	–
	6	42	Start	Data Buffer Rank 3 DWL	22
	7	43	Done	–	–
	8	44	Start	Data Buffer Rank 3 MWD Cycle	23
DDR_CAL_STATUS_SLOTx_5	0	45	Done	–	–
	1	46	Start	Data Buffer Rank 3 MWD Center	24
	2	47	Done	–	–
	3	48	Start	DQS Gate	25
	4	49	Done	–	–
	5	50	Start	DQS Gate Sanity Check	26
	6	51	Done	–	–
	7	52	Start	Write Leveling	27
	8	53	Done	–	–

Table 3-11: **Status Signal Description for Quad-Rank LRDIMM Card *(Cont'd)***

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_6	0	54	Start	Read Per-Bit Deskew	28
	1	55	Done	–	–
	2	56	Start	Read Per-Bit DBI Deskew	29
	3	57	Done	–	–
	4	58	Start	Read DQS Centering (Simple)	30
	5	59	Done	–	–
	6	60	Start	Read Sanity Check	31
	7	61	Done	–	–
	8	62	Start	Write DQS to DQ Deskew	32
DDR_CAL_STATUS_SLOTx_7	0	63	Done	–	–
	1	64	Start	Write DQS to DM/DBI Deskew	33
	2	65	Done	–	–
	3	66	Start	Write DQS to DQ (Simple)	34
	4	67	Done	–	–
	5	68	Start	Write DQS to DM/DBI (Simple)	35
	6	69	Done	–	–
	7	70	Start	Read DQS Centering DBI (Simple)	36
	8	71	Done	–	–
DDR_CAL_STATUS_SLOTx_8	0	72	Start	Write Latency Calibration	37
	1	73	Done	–	–
	2	74	Start	Write Read Sanity Check 0	38
	3	75	Done	–	–
	4	76	Start	Read DQS Centering (Complex)	39
	5	77	Done	–	–
	6	78	Start	Write Read Sanity Check 1	40
	7	79	Done	–	–
	8	80	Start	Read V_{REF} Training	41

Table 3-11: **Status Signal Description for Quad-Rank LRDIMM Card *(Cont'd)***

XSDB Status Register	XSDB Bits [8:0]	Status Port Bits [127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_SLOTx_9	0	81	Done	–	–
	1	82	Start	Write Read Sanity Check 2	42
	2	83	Done	–	–
	3	84	Start	Write DQS to DQ (Complex)	43
	4	85	Done	–	–
	5	86	Start	Write DQS to DM/DBI (Complex)	44
	6	87	Done	–	–
	7	88	Start	Write Read Sanity Check 3	45
	8	89	Done	–	–
DDR_CAL_STATUS_SLOTx_10	0	90	Start	Write V_{REF} Training	46
	1	91	Done	–	–
	2	92	Start	Write Read Sanity Check 4	47
	3	93	Done	–	–
	4	94	Start	Read DQS Centering Multi Rank Adjustment	48
	5	95	Done	–	–
	6	96	Start	Write Read Sanity Check 5	49
	7	97	Done	–	–
	8	98	Start	Multi Rank Adjustment and Checks	50
DDR_CAL_STATUS_SLOTx_11	0	99	Done	–	–
	1	100	Start	Write Read Sanity Check 6	51
	2	101	Done	-	-

ERROR STATUS

The Error signal descriptions of host calibration stages in Table 3-9 holds good for LRDIMM host calibration stages, except that the stage numbering is as per LRDIMM dual-rank or quad-rank configuration.

Table 3-12 lists the error signals of the dual-rank LRDIMM data buffer calibration stages and their description.

Table 3-12: **Error Signal Description of Dual-Rank LRDIMM Data Buffer Calibration Stages**

STAGE_NAME	Stage	Code	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Error
Data Buffer Rank 0 MREP	1	1	Nibble	–	Edge 1 to 0 transition is not found for Rank 0
Data Buffer Rank 0 MRD Cycle	2	1	Nibble	–	Pattern did not match for any of the Read latencies of Rank 0
Data Buffer Rank 0 MRD Center	3	1	Nibble	–	Found very short read valid window for Rank 0
Data Buffer Rank 0 DWL	4	1	Nibble	–	Edge 0 to 1 transition is not found for Rank 0
Data Buffer Rank 0 MWD Cycle	5	1	Nibble	–	Pattern did not match for any of the Write latencies of Rank 0
Data Buffer Rank 0 MWD Center	6	1	Nibble	–	Found very short write valid window for Rank 0
Data Buffer Rank 1 MREP	7	1	Nibble	–	Edge 1 to 0 transition is not found for Rank 1
Data Buffer Rank 1 MRD Cycle	8	1	Nibble	–	Pattern did not match for any of the Read latencies of Rank 1
Data Buffer Rank 1 MRD Center	9	1	Nibble	–	Found very short read valid window for Rank 1
Data Buffer Rank 1 DWL	10	1	Nibble	–	Edge 0 to 1 transition is not found for Rank 1
Data Buffer Rank 1 MWD Cycle	11	1	Nibble	–	Pattern did not match for any of the Write latencies of Rank 1
Data Buffer Rank 1 MWD Center	12	1	Nibble	–	Found very short write valid window for Rank 1

Table 3-13 lists the error signals of the quad-rank LRDIMM data buffer calibration stages and their description.

Table 3-13: **Error Signal Description Of Quad-Rank LRDIMM Data Buffer Calibration Stages**

STAGE_NAME	Stage	Code	DDR_CAL _ERROR_1	DDR_CAL _ERROR_0	Error
Data Buffer Rank 0 MREP	1	1	Nibble	–	Edge 1 to 0 transition is not found for Rank 0
Data Buffer Rank 0 MRD Cycle	2	1	Nibble	–	Pattern did not match for any of the Read latencies of Rank 0
Data Buffer Rank 0 MRD Center	3	1	Nibble	–	Found very short read valid window for Rank 0
Data Buffer Rank 0 DWL	4	1	Nibble	–	Edge 0 to 1 transition is not found for Rank 0

Table 3-13: **Error Signal Description Of Quad-Rank LRDIMM Data Buffer Calibration Stages *(Cont'd)***

STAGE_NAME	Stage	Code	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Error
Data Buffer Rank 0 MWD Cycle	5	1	Nibble	–	Pattern did not match for any of the Write latencies of Rank 0
Data Buffer Rank 0 MWD Center	6	1	Nibble	–	Found very short write valid window for Rank 0
Data Buffer Rank 1 MREP	7	1	Nibble	–	Edge 1 to 0 transition is not found for Rank 1
Data Buffer Rank 1 MRD Cycle	8	1	Nibble	–	Pattern did not match for any of the Read latencies of Rank 1
Data Buffer Rank 1 MRD Center	9	1	Nibble	–	Found very short read valid window for Rank 1
Data Buffer Rank 1 DWL	10	1	Nibble	–	Edge 0 to 1 transition is not found for Rank 1
Data Buffer Rank 1 MWD Cycle	11	1	Nibble	–	Pattern did not match for any of the Write latencies of Rank 1
Data Buffer Rank 1 MWD Center	12	1	Nibble	–	Found very short write valid window for Rank 1
Data Buffer Rank 2 MREP	13	1	Nibble	–	Edge 1 to 0 transition is not found for Rank 2
Data Buffer Rank 2 MRD Cycle	14	1	Nibble	–	Pattern did not match for any of the Read latencies of Rank 2
Data Buffer Rank 2 MRD Center	15	1	Nibble	–	Found very short read valid window for Rank 2
Data Buffer Rank 2 DWL	16	1	Nibble	–	Edge 0 to 1 transition is not found for Rank 2
Data Buffer Rank 2 MWD Cycle	17	1	Nibble	–	Pattern did not match for any of the Write latencies of Rank 2
Data Buffer Rank 2 MWD Center	18	1	Nibble	–	Found very short write valid window for Rank 2
Data Buffer Rank 3 MREP	19	1	Nibble	–	Edge 1 to 0 transition is not found for Rank 3
Data Buffer Rank 3 MRD Cycle	20	1	Nibble	–	Pattern did not match for any of the Read latencies of Rank 3
Data Buffer Rank 3 MRD Center	21	1	Nibble	–	Found very short read valid window for Rank 3
Data Buffer Rank 3 DWL	22	1	Nibble	–	Edge 0 to 1 transition is not found for Rank 3
Data Buffer Rank 3 MWD Cycle	23	1	Nibble	–	Pattern did not match for any of the Write latencies of Rank 3
Data Buffer Rank 3 MWD Center	24	1	Nibble	–	Found very short write valid window for Rank 3

Save Restore

The feature saves the calibration data into an external memory and restores the same information at a later point of time for a quick calibration completion. The IP provides a set of XSDB ports in the user interface through which, you can save and restore the memory controller calibration data.

When the FPGA is programmed and asked to calibrate in a regular mode, all required calibration stages are executed. You can start talking to the DRAM when the calibration completes and issues a save request at any point of time to save the calibration data. This is called save cycle. The FPGA can be reprogrammed or turned off after the save cycle.

At a later point of time, the same design can be reprogrammed and asked to calibrate in restore mode in which, the calibration completes in a very quick time. This is called restore cycle. The placeholder that keeps the calibration data inside the memory controller while the FPGA is powered is called XSDB block RAM.

It is required to save and restore the entire XSDB block RAM. Its end address can be obtained from the END_ADDR0/1 locations of the XSDB debug information. An example to calculate the end address is available in step 2, page 604.

If Match_Cycle is set to no wait, to minimize stage 1 configuration time, DCI calibration needs to be reset by instantiating the DCIRESET primitive. Also, the RST input to the primitive needs to be toggled, such that the DCI state machine is reset and the calibration process restarts.

Figure 3-13 describes the save/restore sequence briefly.

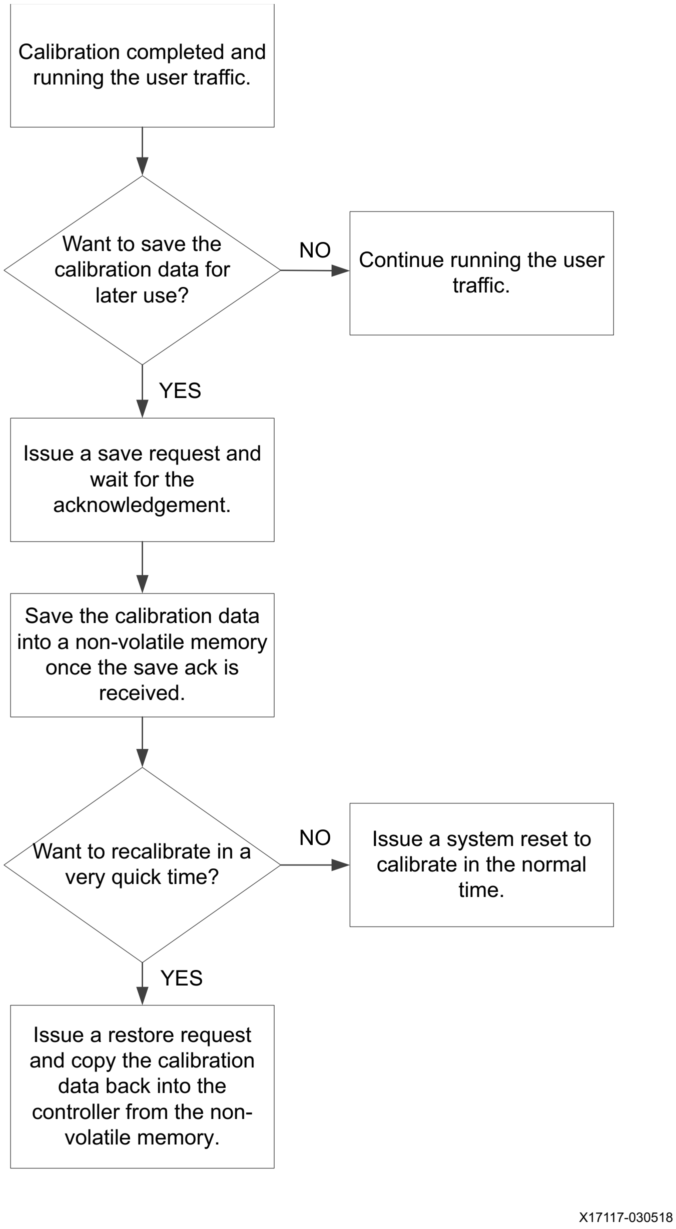


Figure 3-9: **Save and Restore Sequence of the Calibration Data**

Table 3-14: **User Interface Ports Description for Save and Restore**

Signal Name	I/O	Width	Description
app_save_req	I	1	Request for saving the calibration data. No further memory requests are accepted when it is asserted. Must be asserted from Low to High only after calibration completion.
app_save_ack	O	1	Save request acknowledgment. The signal stays High after it is asserted until a system reset is issued.

Table 3-14: **User Interface Ports Description for Save and Restore *(Cont'd)***

Signal Name	I/O	Width	Description
app_restore_en	I	1	XSDB block RAM restore enable. It must be asserted High within 50 general interconnect cycles after ui_clk_sync_rst is deasserted in the restore cycle until calibration completes. Assert this to notify MicroBlaze to wait for XSDB block RAM restoration completion. After the XSDB block RAM is restored, assert app_restore_complete to notify MicroBlaze to continue calibration. When asserted, • MicroBlaze waits for app_restore_complete before proceeding to calibration • Disables all calibration stages except DQS gating
app_restore_complete	I	1	XSDB block RAM restore complete. It should be asserted High after the entire XSDB block RAM is restored until calibration completes.
app_dbg_out	O		Debug Output Do not connect any signals to app_dbg_out and keep the port open during instantiation
app_xsdb_select	I	1	Save restore XSDB ports Select. Assert for the XSDB block RAM read or write access. It should be asserted as long as the access is required.
app_xsdb_rd_en	I	1	XSDB block RAM read enable. Asserting this for one cycle issues one read command.
app_xsdb_wr_en	I	1	XSDB block RAM write enable. Asserting this for one cycle issues one write command. The corresponding write address (app_xsdb_addr) and write data (app_xsdb_wr_data) are taken in the same cycle.
app_xsdb_addr	I	16	XSDB block RAM address. This address is used for both read and write commands. app_xsdb_addr is taken in the same cycle when app_xsdb_rd_en or app_xsdb_wr_en is valid.
app_xsdb_wr_data	I	9	XSDB block RAM write data. app_xsdb_wr_data is taken in the same cycle when app_xsdb_wr_en is valid.
app_xsdb_rd_data	O	9	XSDB block RAM read data. app_xsdb_rd_data is valid when app_xsdb_rdy is asserted.
app_xsdb_rdy	O	1	Acknowledge for the previous command. Acts as a read data valid for read commands. Any new command must be sent only after receiving the app_xsdb_rdy response for the current command.

The following is a save restore flow description:

1. Save cycle
    a. Memory Controller boots up in a normal manner, completes calibration, and runs the user traffic.
    b. Issue a save request to the Memory Controller by asserting **`app_save_req`**. Any read or write request that comes along with or after the save request is dropped and the controller behavior is not guaranteed. Thus, the traffic must be stopped before requesting the calibration data save.

c. Memory Controller asserts the **`app_save_ack`** after finishing all pending DRAM commands. Figure 3-10 shows the save request and acknowledge assertions.

d. When the **`app_save_ack`** is asserted, save the XSDB block RAM content into an external memory through the XSDB ports provided in the user interface as shown in Figure 3-11. The saved data can be used to restore the calibration in a shorter time at a later point of time.

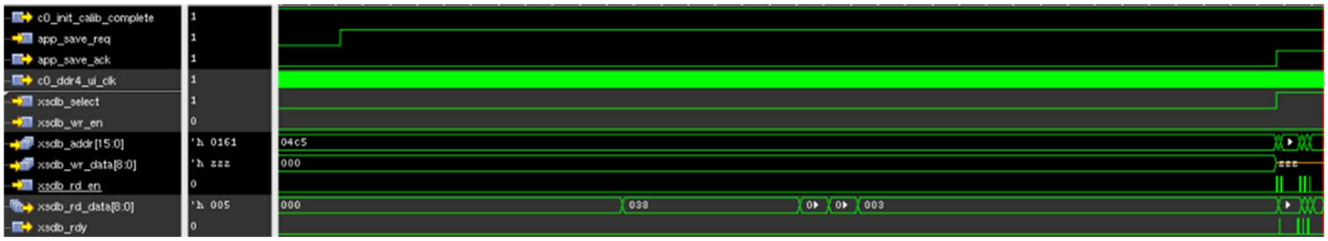


Figure 3-10: **Save Request and Acknowledge Assertions**

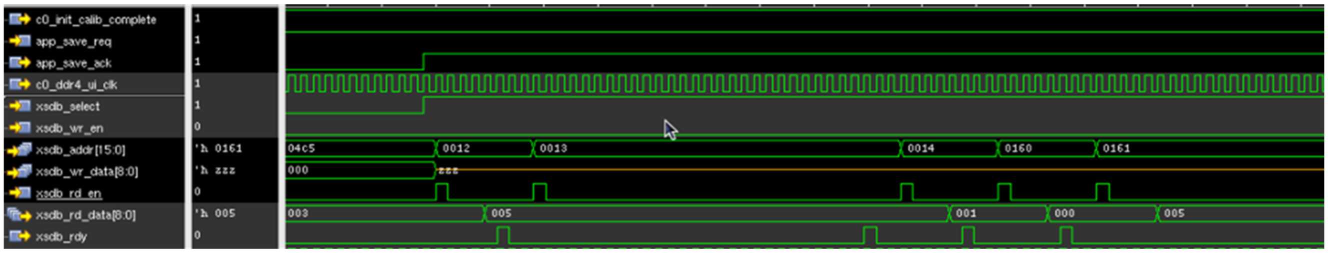


Figure 3-11: **XSDB Interface Timing for Reading XSDB Block RAM Content**

2. Restore cycle

a. Assert the **`app_restore_en`** signal within 50 general interconnect cycles after the user interface reset (**`ui_clk_sync_rst`**) is deasserted in the restore cycle. It should stay asserted until the calibration completes.

b. Restore the XSDB block RAM content from the external saved space into the Memory Controller through the XSDB ports provided in the user interface. The XSDB write timing is shown in Figure 3-12. Assert the **`app_restore_complete`** after the entire XSDB block RAM is restored as shown in Figure 3-13.

c. Memory Controller recognizes this as a restore boot up when **`app_restore_en`** is asserted. The calibration sequence is going to be shortened in the restore mode. When **`app_restore_complete`** is asserted, the entire calibration data from XSDB block RAM is restored into PHY with minimal calculations.

d. Memory Controller skips all calibration stages except the DQS gating stage and finishes calibration. User traffic starts after the calibration as usual.

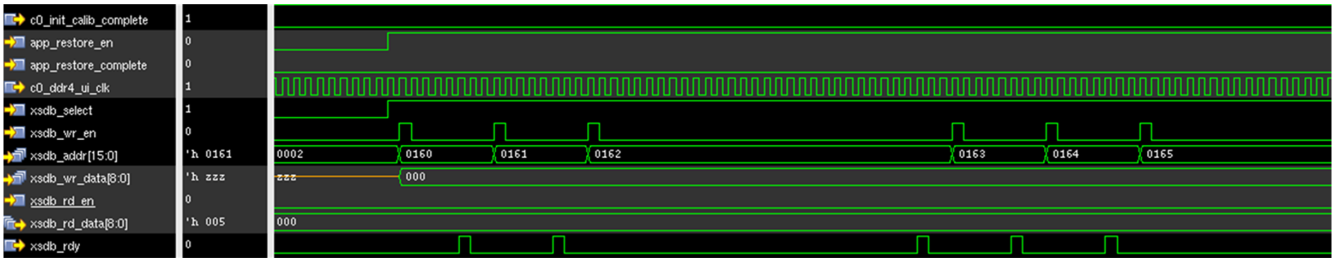


Figure 3-12: **XSDB Interface Timing for Writing XSDB Block RAM Content**

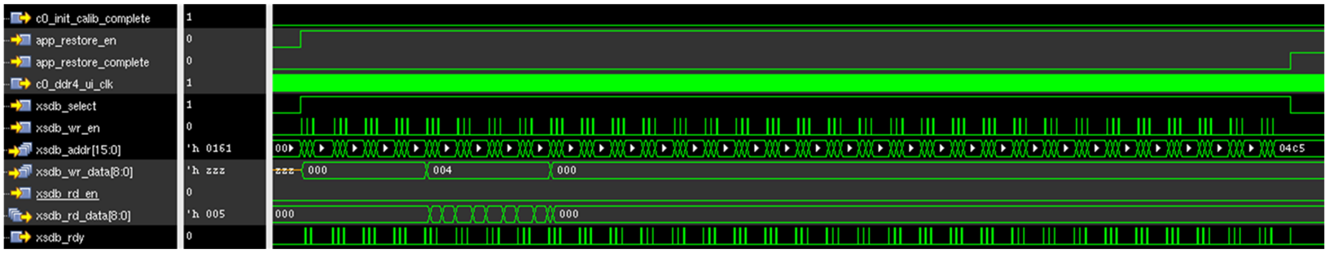


Figure 3-13: **Asserting app_restore_complete After Writing Entire Block RAM Content**

Reset Sequence

The **`sys_rst`** signal resets the entire memory design which includes general interconnect (fabric) logic which is driven by the MMCM clock (clkout0) and RIU logic. MicroBlaze™ and calibration logic are driven by the MMCM clock (clkout6). The **`sys_rst`** input signal is synchronized internally to create the **`ui_clk_sync_rst`** signal. The **`ui_clk_sync_rst`** reset signal is synchronously asserted and synchronously deasserted.

Figure 3-14 shows the **`ui_clk_sync_rst`** (fabric reset) is synchronously asserted with a few clock delays after **`sys_rst`** is asserted. When **`ui_clk_sync_rst`** is asserted, there are a few clocks before the clocks are shut off.

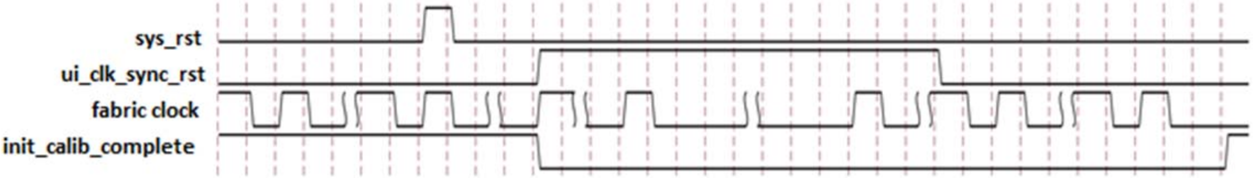


Figure 3-14: **Reset Sequence Waveform**

The following are the reset sequencing steps:

1. Reset to design is initiated after **`ui_clk_sync_rst`** goes High.

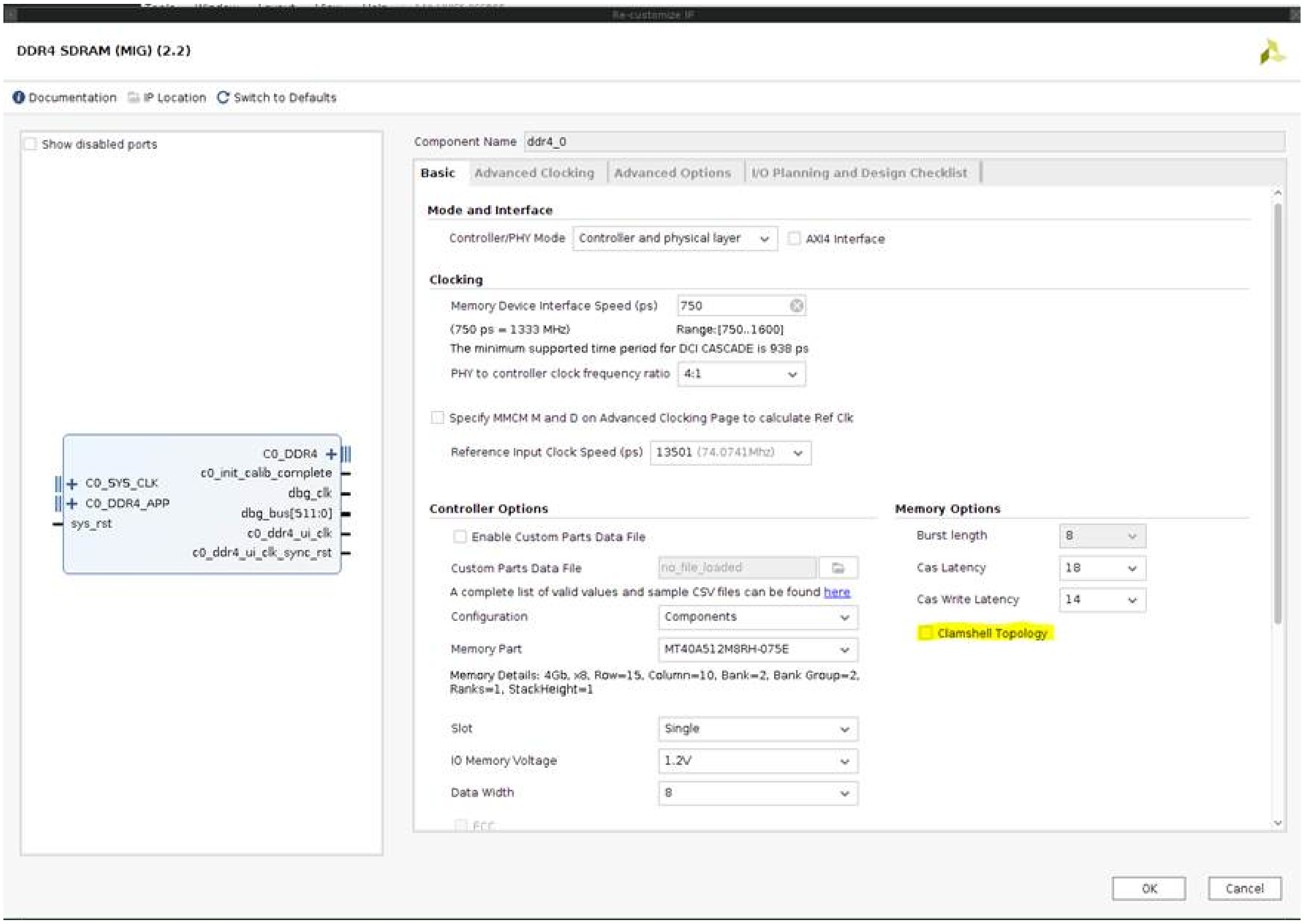
2. **`init_calib_complete`** signal goes Low when **`ui_clk_sync_rst`** is High.
3. Reset to design is deactivated after **`ui_clk_sync_rst`** is Low.
4. After **`ui_clk_sync_rst`** is deactivated, the **`init_calib_complete`** is asserted after calibration is completed.

Clamshell Topology

This feature is supported for DDR4 **Controller/PHY Mode** option in the **Controller and physical layer** pull-down for User Interface, AXI interfaces, and Physical Layer Only interface. Clamshell topology supports the Physical Layer Ping Pong interface.

Note: Only DDR4 single Rank components are supported with this feature.

The clamshell topology saves the component area by placing them on both sides (top and bottom) of the board to mimic the address mirroring concept of the multi-rank RDIMMs. Address mirroring improves the signal integrity of the address and control ports and makes the PCB routing easier. The clamshell feature is available in the **Basic** tab as shown in Figure 3-15.



DDR4 SDRAM (MIG) (2.2)

Component Name: ddr4_0

Mode and Interface

Controller/PHY Mode: Controller and physical layer

☐ AXI4 Interface

Clocking

Memory Device Interface Speed (ps): 750

(750 ps = 1333 MHz) Range:[750..1600]

The minimum supported time period for DCI CASCADE is 938 ps

PHY to controller clock frequency ratio: 4:1

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps): 13501 (74.0741Mhz)

Controller Options

☐ Enable Custom Parts Data File

Custom Parts Data File: no_file_loaded

A complete list of valid values and sample CSV files can be found here

Configuration: Components

Memory Part: MT40A512M8RH-075E

Memory Details: 4Gb, x8, Row=15, Column=10, Bank=2, Bank Group=2, Ranks=1, StackHeight=1

Slot: Single

IO Memory Voltage: 1.2V

Data Width: 8

☐ ECC

Memory Options

Burst length: 8

Cas Latency: 18

Cas Write Latency: 14

☐ Clamshell Topology

Figure 3-15: **Vivado Customize IP Dialog Box – Clamshell Topology**

The components are split into two categories called non-mirrored and mirrored. One additional chip select signal is added to the design for the mirrored components. Figure 3-16 shows the difference between the regular component topology and the clamshell topology.

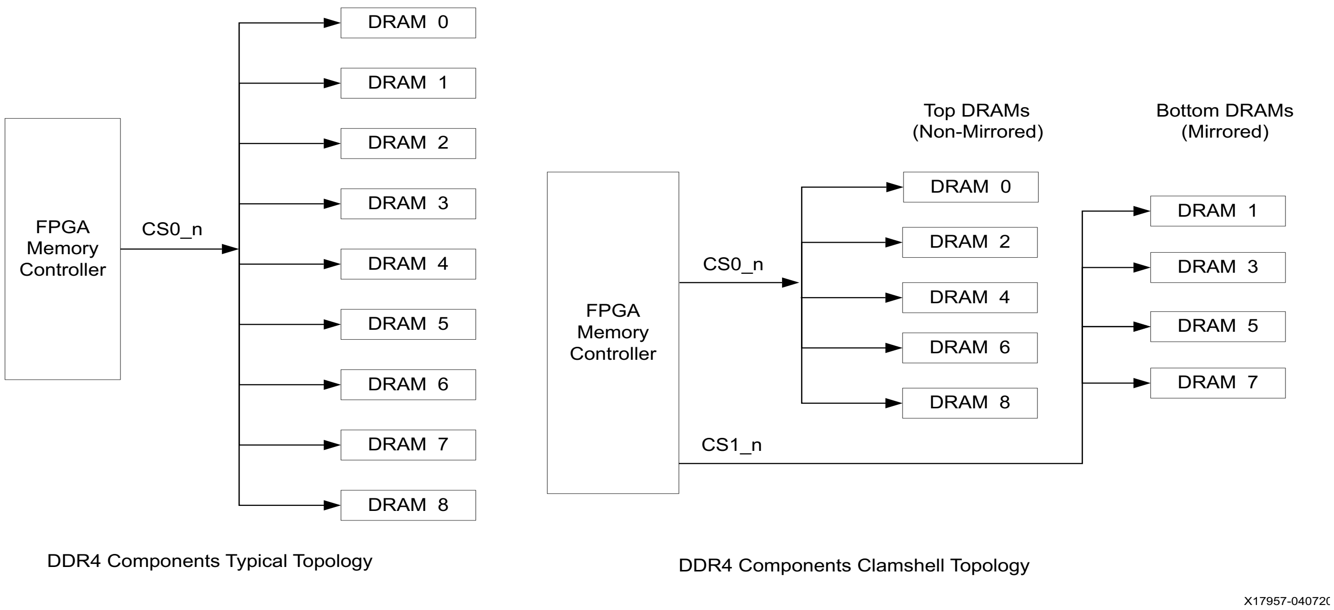


Figure 3-16: **Regular Component Topology vs. Clamshell Topology**

As mentioned in Figure 3-15, `CS0_n` of a clamshell design drives the non-mirrored components while `CS1_n` driving the mirrored components. For more information on the PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

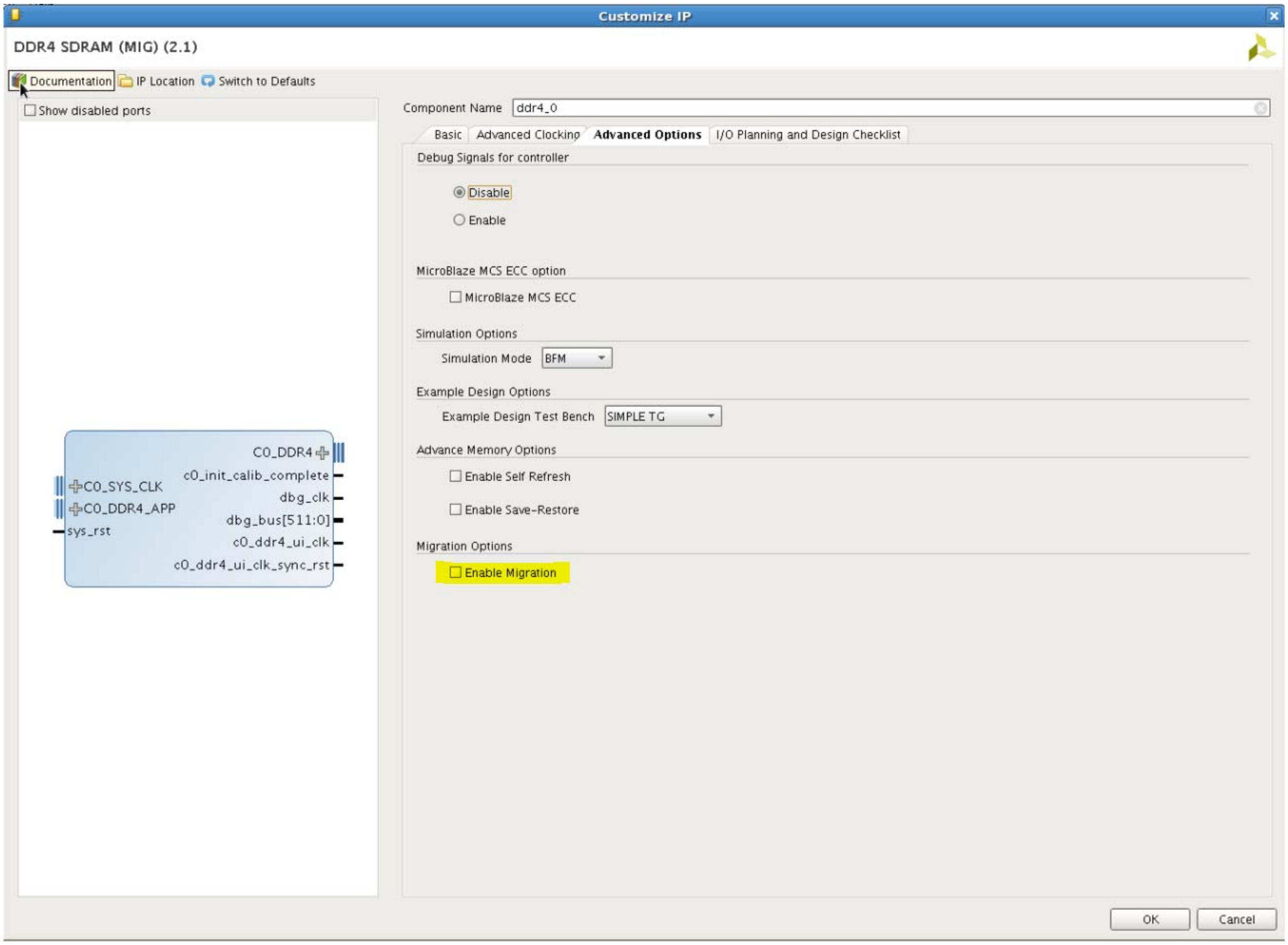
Migration Feature

This feature is supported for DDR4 **Controller/PHY Mode** option in the **Controller and physical layer** for User Interface, AXI interfaces, and Physical Layer Only interface. Migration does not support the Physical Layer Ping Pong interface. This feature is helpful when migrating a design from the existing FPGA package to another compatible package. It also supports pin compatible packages within and across UltraScale and UltraScale+ families. For more information on the on pin compatible FPGAs, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

The migration option compensates the package skews of all address/command signals on the targeted device to keep the phase relationship of the source device intact. It is required only for the address/command bus as there is no calibration for these signals.

The data bus (`DQ` and `DQS`) skews are not required to compensate because it is completed during the regular calibration sequence. The tool supports a skew difference of 0 to 75 ps only.

Figure 3-17 shows the **Advanced Options** tab to enable the migration feature.



Component Name: ddr4_0

Debug Signals for controller

● Disable
○ Enable

MicroBlaze MCS ECC option

☐ MicroBlaze MCS ECC

Simulation Options

Simulation Mode: BFM

Example Design Options

Example Design Test Bench: SIMPLE TG

Advance Memory Options

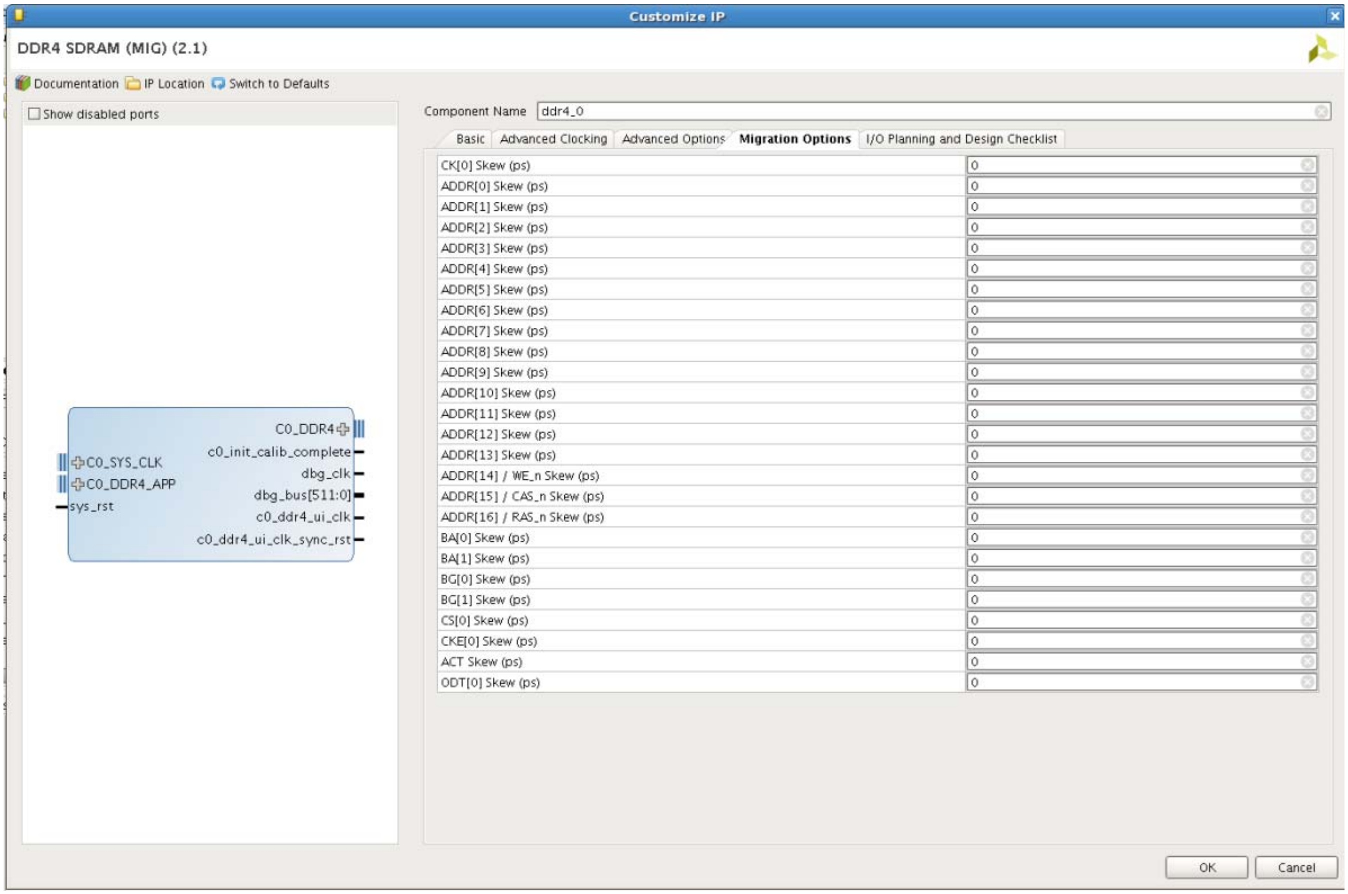
☐ Enable Self Refresh
☐ Enable Save-Restore

Migration Options

☐ Enable Migration

Figure 3-17: **Vivado Customize IP Dialog Box – Enable Migration**

When **Enable Migration** is selected, a **Migration Options** tab is displayed as shown in Figure 3-18. It has entries for all address and command signals to enter the skew values on the corresponding pins. All entries are in picoseconds (ps).



Component Name ddr4_0

CK[0] Skew (ps)	0
ADDR[0] Skew (ps)	0
ADDR[1] Skew (ps)	0
ADDR[2] Skew (ps)	0
ADDR[3] Skew (ps)	0
ADDR[4] Skew (ps)	0
ADDR[5] Skew (ps)	0
ADDR[6] Skew (ps)	0
ADDR[7] Skew (ps)	0
ADDR[8] Skew (ps)	0
ADDR[9] Skew (ps)	0
ADDR[10] Skew (ps)	0
ADDR[11] Skew (ps)	0
ADDR[12] Skew (ps)	0
ADDR[13] Skew (ps)	0
ADDR[14] / WE_n Skew (ps)	0
ADDR[15] / CAS_n Skew (ps)	0
ADDR[16] / RAS_n Skew (ps)	0
BA[0] Skew (ps)	0
BA[1] Skew (ps)	0
BG[0] Skew (ps)	0
BG[1] Skew (ps)	0
CS[0] Skew (ps)	0
CKE[0] Skew (ps)	0
ACT Skew (ps)	0
ODT[0] Skew (ps)	0

Figure 3-18: **Vivado Customize IP Dialog Box – Migration Options**

Table 3-15 to Table 3-17 show examples on the skew calculations that need to be entered in Figure 3-18 while migrating the FPGA device. The procedure to retrieve the delay values for the source and target devices is available in the Migration chapter in the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

These delay values for all used pins are listed in columns 2 and 3 for the source and target devices, respectively. The difference in the delay of the target device from the source is mentioned in column 4. Note that the skew can be positive or negative. Because the GUI expects only the positive skew values, the column 4 values are adjusted in the column 5 such that the lowest skew difference becomes zero. The calculated values in column 5 are to be entered in Vivado as shown in Figure 3-18.

Table 3-15: **Calculation for All Positive Skews**

Port Name	Source Device Delay (in ps)	Target Device Delay (in ps)	Skew (Target Source)	Skew (Entered in GUI)
ADDR[0]	159	190	31	20
ADDR[1]	162	185	23	12
CK	154	165	11	0
CKE	160	182	22	11
CS	150	195	45	34

The lowest skew among all entries of column 4 (Table 3-15) is +11 ps. Therefore, column 5 gets formed by subtracting this lowest skew value (+11 ps) from column 4.

Table 3-16: **Calculation for All Negative Skews**

Port Name	Source Device Delay (in ps)	Target Device Delay (in ps)	Skew (Target Source)	Skew (Entered in GUI)
ADDR[0]	189	150	-39	0
ADDR[1]	172	155	-17	22
CK	184	165	-19	20
CKE	170	162	-8	31
CS	180	175	-5	34

The lowest skew among all entries of column 4 (Table 3-16) is -39 ps. Then, column 5 gets formed by subtracting this lowest skew value (-39 ps) from column 4.

Table 3-17: **Calculation for Mix of Positive and Negative Skews**

Port Name	Source Device Delay (in ps)	Target Device Delay (in ps)	Skew (Target Source)	Skew (Entered in GUI)
ADDR[0]	169	190	21	39
ADDR[1]	172	185	13	31
CK	154	165	11	29
CKE	170	152	-18	0
CS	180	175	-5	13

The lowest skew among all entries of column 4 (Table 3-17) is -18 ps. Hence, column 5 gets formed by subtracting this lowest skew value (-18 ps) from column 4.

MicroBlaze MCS ECC

The MicroBlaze MCS local memory provides an option to enable Error Correcting Code (ECC). Error correction corrects single bit errors and detects double bit errors. Two additional ports are added to indicate single bit errors (LMB_CE) and double bit errors (LMB_UE).

The MicroBlaze MCS ECC can be selected from the **MicroBlaze MCS ECC option** section in the **Advanced Options** tab. The block RAM size increases if the ECC option for MicroBlaze MCS is selected.

Memory Settings

This section captures the settings of memory components and DIMMs.

DDR3 Register Module

DDR3 register module settings are captured in Table 3-18. The register contents are programmed to default value of 0s, unless otherwise specified in the table.

Table 3-18: **DDR3 Register Module Settings**

Register	Field	Possible Values and Description
RC3	DBA[1:0], DA[4:3]	Value based on DRAM loads on the card.
RC4	DBA[1:0], DA[4:3]	Value based on DRAM loads on the card.
RC5	DBA[1:0], DA[4:3]	Value based on DRAM loads on the card.
RC10	DBA[0], DA[4:3]	Value based on the targeted speed.
RC11	DA[4:3]	Value based on the targeted voltage.

DDR4 Register Module

DDR4 register module settings are captured in Table 3-19. The register contents are programmed to default value of 0s, unless otherwise specified in the table.

Table 3-19: **DDR4 Register Module Settings**

Register	Field	Possible Values and Description
RC03	DA[3:0]	Value based on DRAM loads on the card.
RC04	DA[3:0]	Value based on DRAM loads on the card.
RC05	DA[3:0]	Value based on DRAM loads on the card.

Table 3-19: **DDR4 Register Module Settings *(Cont'd)***

Register	Field	Possible Values and Description
RC08	DA[1:0]	For non-3DS configurations: 01 = Number of physical ranks per slot is 4 (LRDIMM Quad rank) 11 = Number of physical ranks per slot is 2 or 1 For 3DS configurations: 11 = 1 height 10 = 2 height 01 = 4 height
	DA[3]	0 = If address pins are 18 1 = If address pins are 17
RC0A	DA[2:0]	Value based on the targeted speed.
RC0B	DA[3]	1 = Input receiver Vref source is External VrefCA input
RC0D	DA[1:0]	01 = Direct Quad CS mode is when the number of ranks per slot is 4 (LRDIMM Quad rank) 00 = Direct Dual CS mode is when the number of ranks per slot is < 4
	DA[2]	0 = LRDIMM configuration 1 = RDIMM configuration
RC2X	DA[0]	1 = I2C bus interface is disabled
RC3X	DA[7:0]	Value based on the targeted speed.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The memory interface requires one mixed-mode clock manager (MMCM), one TXPLL per I/O bank used by the memory interface, and two BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used.

Note: DDR3/DDR4 SDRAM generates the appropriate clocking structure and no modifications to the RTL are supported.

The DDR3/DDR4 SDRAM tool generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to MMCM (located in center bank of memory interface)
- MMCM to BUFG (located at center bank of memory interface) driving FPGA logic and all TXPLLs
- MMCM to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate FPGA logic
- Clocking pair of the interface must be in the same SLR of memory interface for the stacked silicon interconnect (SSI) technology devices

Requirements

GCIO

- Must use a differential I/O standard
- Must be in the same I/O column as the memory interface
- Must be in the same SLR of memory interface for the SSI technology devices
- The I/O standard and termination scheme are system dependent. For more information, consult the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 7].

MMCM

- MMCM is used to generate the FPGA logic system clock (1/4 of the memory clock)
- Must be located in the center bank of memory interface
- Must use internal feedback
- Input clock frequency divided by input divider must be ≥ 70 MHz (CLKINx / D ≥ 70 MHz)
- Must use integer multiply and output divide values

Input Clock Requirement

- The clock generator driving the GCIO should have jitter < 3 ps RMS.
- The input clock should always be clean and stable. The IP functionality is not guaranteed if this input system clock has a glitch, discontinuous, etc.
- No spread spectrum clock is allowed.

BUFGs and Clock Roots

- One BUFG is used to generate the system clock to FPGA logic and another BUFG is used to divide the system clock by two.
- BUFGs and clock roots must be located in center most bank of the memory interface.
    - For two bank systems, the bank with the higher number of bytes selected is chosen as the center bank. If the same number of bytes is selected in two banks, then the top bank is chosen as the center bank.
    - For four bank systems, either of the center banks can be chosen. DDR3/DDR4 SDRAM refers to the second bank from the top-most selected bank as the center bank.
    - Both the BUFGs must be in the same bank.

TXPLL

- CLKOUTPHY from TXPLL drives XIPHY within its bank
- TXPLL must be set to use a CLKFBOUT phase shift of 90°
- TXPLL must be held in reset until the MMCM lock output goes High
- Must use internal feedback

Figure 4-1 shows an example of the clocking structure for a three bank memory interface. The GCIO drives the MMCM located at the center bank of the memory interface. MMCM drives both the BUFGs located in the same bank. The BUFG (which is used to generate system clock to FPGA logic) output drives the TXPLLs used in each bank of the interface.

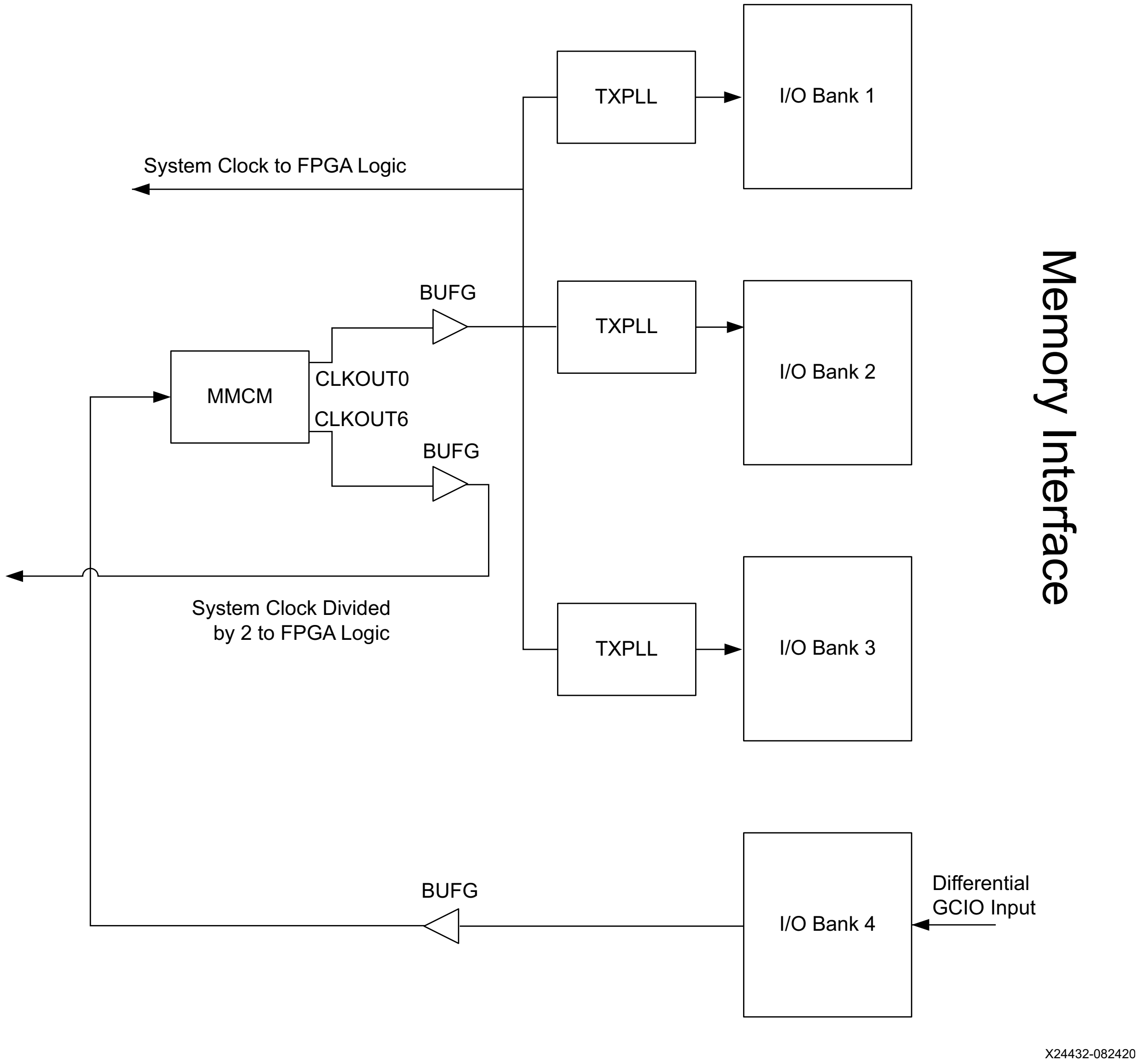


Figure 4-1: **Clocking Structure for Three Bank Memory Interface**

The MMCM is placed in the center bank of the memory interface.

- For two bank systems, MMCM is placed in a bank with the most number of bytes selected. If they both have the same number of bytes selected in two banks, then MMCM is placed in the top bank.
- For four bank systems, MMCM is placed in a second bank from the top.

For designs generated with System Clock configuration of **No Buffer**, MMCM must not be driven by another MMCM/PLL. Cascading clocking structures MMCM → BUFG → MMCM and PLL → BUFG → MMCM are not allowed.

If the MMCM is driven by the GCIO pin of the other bank, then the CLOCK_DEDICATED_ROUTE constraint with value "BACKBONE" must be set on the net that is driving MMCM or on the MMCM input. Setting up the CLOCK_DEDICATED_ROUTE constraint on the net is preferred. But when the same net is driving two MMCMs, the CLOCK_DEDICATED_ROUTE constraint must be managed by considering which MMCM needs the BACKBONE route.

In such cases, the CLOCK_DEDICATED_ROUTE constraint can be set on the MMCM input. To use the "BACKBONE" route, any clock buffer that exists in the same CMT tile as the GCIO must exist between the GCIO and MMCM input. The clock buffers that exists in the I/O CMT are BUFG, BUFGCE, BUFGCTRL, and BUFGCE_DIV. So DDR3/DDR4 SDRAM instantiates BUFG between the GCIO and MMCM when the GCIO pins and MMCM are not in the same bank (see Figure 4-1).

If the GCIO pin and MMCM are allocated in different banks, DDR3/DDR4 SDRAM generates CLOCK_DEDICATED_ROUTE constraints with value as "BACKBONE." If the GCIO pin and MMCM are allocated in the same bank, there is no need to set any constraints on the MMCM input.

Similarly when designs are generated with System Clock Configuration as a **No Buffer** option, you must take care of the "BACKBONE" constraint and the BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM if GCIO pin and MMCM are allocated in different banks. DDR3/DDR4 SDRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations. For more information on clocking, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

XDC syntax for CLOCK_DEDICATED_ROUTE constraint is given here:

```
For DDR3:
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hier -filter {NAME =~ */
u_ddr3_infrastructure/gen_mmcme*.u_mmcme_adv_inst/CLKIN1}]

For DDR4:
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hier -filter {NAME =~ */
u_ddr4_infrastructure/gen_mmcme*.u_mmcme_adv_inst/CLKIN1}]
```

For more information on the CLOCK_DEDICATED_ROUTE constraints, see the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 9].

Note: If two different GCIO pins are used for two DDR3/DDR4 SDRAM IP cores in the same bank, center bank of the memory interface is different for each IP. DDR3/DDR4 SDRAM generates MMCM LOC and CLOCK_DEDICATED_ROUTE constraints accordingly.

Sharing of Input Clock Source (sys_clk_p)

If the same GCIO pin must be used for two IP cores, generate the two IP cores with the same frequency value selected for option **Reference Input Clock Period (ps)** and **System Clock Configuration** option as **No Buffer**. Perform the following changes in the wrapper file in which both IPs are instantiated:

1. DDR3/DDR4 SDRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) of both the IP cores.
2. System clock pins must be allocated within the same I/O column of the memory interface pins allocated. Add the pin LOC constraints for system clock pins and clock constraints in your top-level XDC.
3. You must add a "BACKBONE" constraint on the net that is driving the MMCM or on the MMCM input if GCIO pin and MMCM are not allocated in the same bank. Apart from this, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM to use the "BACKBONE" route.

Note:

- The UltraScale™ architecture includes an independent XIPHY power supply and TXPLL for each XIPHY. This results in clean, low jitter clocks for the memory system.
- Skew spanning across multiple BUFGs is not a concern because single point of contact exists between BUFG → TXPLL and the same BUFG → System Clock Logic.
- System input clock cannot span I/O columns because the longer the clock lines span, the more jitter is picked up.

TXPLL Usage

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used. One PLL per bank is used if a bank is used by a single memory interface. You can use a second PLL for other usage. To use a second PLL, you can perform the following steps:

1. Generate the design for the **System Clock Configuration** option as **No Buffer**.
2. DDR3/DDR4 SDRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock

inputs (`sys_clk_i`) and also to the input of PLL (PLL instance that you have in your design).

3. You can use the PLL output clocks.

Additional Clocks

You can produce up to four additional clocks which are created from the same MMCM that generates `ui_clk`. Additional clocks can be selected from the **Clock Options** section in the **Advanced** tab. The GUI lists the possible clock frequencies from MMCM and the frequencies for additional clocks vary based on selected memory frequency (**Memory Device Interface Speed (ps)** value in the **Basic** tab), selected FPGA, and FPGA speed grade.

Reduce System Noise During Calibration

The system design should be as quiet as possible during the calibration process. In particular, the Soft Error Mitigation (SEM) IP, if used, should be disabled during calibration. For calibration that occurs immediately after the configuration or reconfiguration of the FPGA, use the ICAP arbitration interface to hold off the SEM IP in the boot stage. For more information on the ICAP Arbitration Interface, see “ICAP Arbitration Interface” section in Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10].

For situations where the memory interface is reset and recalibrated without a reconfiguration of the FPGA, the SEM IP must be set into IDLE state to disable the memory scan and to send the SEM IP back into the scanning (Observation or Detect only) states afterwards. This can be done in two methods, through the “Command Interface” or “UART interface.” See Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10] for more information.

Resets

An asynchronous reset (`sys_rst`) input is provided. This is an active-High reset and the `sys_rst` must assert for a minimum pulse width of 5 ns. The `sys_rst` can be an internal or external pin.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must have a common reset input.*

For more information on reset, see the Reset Sequence in Chapter 3, Core Architecture.

Note: The best possible calibration results are achieved when the FPGA activity is minimized from the release of this reset input until the memory interface is fully calibrated as indicated by the `init_calib_complete` port (see the User Interface section of this document).

PCB Guidelines for DDR3

Strict adherence to all documented DDR3 PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

PCB Guidelines for DDR4

Strict adherence to all documented DDR4 PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

Pin and Bank Rules

DDR3 Pin Rules

IMPORTANT: *Xilinx advises Tandem Configuration users to avoid using bank 65 for design applications, especially when using Tandem PROM, to avoid complications because the programming bitstream is split into two stages. Specifically, IP cores built by the Memory IP or Memory Interface Generator (MIG) must not use bank 65 I/O. This ensures that IP can remain completely within stage 2, and avoid complications with its embedded I/O and demanding timing constraints.*

The rules are for single and multi-rank memory interfaces.

- Address/control means `cs_n`, `ras_n`, `cas_n`, `we_n`, `ba`, `ck`, `cke`, `a`, `parity` (valid for RDIMMs only), and `odt`. Multi-rank systems have one `cs_n`, `cke`, `odt`, and one `ck` pair per rank.
- Pins in a byte lane are numbered N0 to N12.
- Byte lanes in a bank are designed by T0, T1, T2, or T3. Nibbles within a byte lane are distinguished by a “U” or “L” designator added to the byte lane designator (T0, T1, T2, or T3). Thus they are T0L, T0U, T1L, T1U, T2L, T2U, T3L, and T3U.

Note: There are two PLLs per bank and a controller uses one PLL in every bank that is being used by the interface.

1. `dqs`, `dq`, and `dm` location.
   a. Designs using x8 or x16 components – `dqs` must be located on a dedicated byte clock pair in the upper nibble designated with “U” (N6 and N7). `dq` associated with a `dqs` must be in same byte lane on any of the other pins except pins N1 and N12.

b. Designs using x4 components – **`dqs`** must be located on the dedicated **`dqs`** pair in the nibble (N0 and N1 in the lower nibble, N6 and N7 in the upper nibble). **`dq`**'s associated with a **`dqs`** must be in the same nibble on any of the other pins except pin N12 (upper nibble).

c. **`dm`** (if used) must be located on pin N0 in the byte lane with the corresponding **`dqs`**. When **`dm`** is disabled, pin N0 can be used for **`dq`** and pin N0 must not be used for address/control signal. Pin N0 must not be used for Address/Control when **`dm`** is not used (exception **`reset#`** pin).

Note: **`dm`** is not supported with x4 devices.

d. **`dm`**, if not used, must be pulled low on the PCB. Typical values used for this are equal to the DQ trace impedance such as 40 or 50Ω. Consult with the memory vendor for their specific recommendation. Unpredictable failures occur if this is not pulled low appropriately.

IMPORTANT: *Also, ensure that the interface is configured in the GUI to not use the data mask. Otherwise, the calibration logic attempts to train this pin which results in a calibration failure.*

---

2. The x4 components must be used in pairs. Odd numbers of x4 components are not permitted. Both the upper and lower nibbles of a data byte must be occupied by a x4 **`dq`**/**`dqs`** group.
3. Byte lanes with a **`dqs`** are considered to be data byte lanes. Pins N1 and N12 can be used for address/control in a data byte lane. If the data byte is in the same bank as the remaining address/control pins, see step #4.
4. Address/control can be on any of the 13 pins in the address/control byte lanes. Address/control must be contained within the same bank.
5. For dual slot configurations of RDIMMs and UDIMMs: **`cs`**, **`odt`**, **`cke`**, and **`ck`** port widths are doubled. For exact mapping of the signals, see the DIMM Configurations.
6. One **`vrp`** pin per bank is used and DCI is required for the interfaces. A **`vrp`** pin is required in I/O banks containing inputs as well as in output only banks. It is required in output only banks because address/control signals use SSTL15_DCI/SSTL135_DCI to enable usage of controlled output impedance. DCI cascade is allowed. When DCI cascade is selected, **`vrp`** pin can be used as a normal I/O. All rules for the DCI in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7] must be followed.

RECOMMENDED: *Xilinx strongly recommends that the DCIUpdateMode option is kept with the default value of ASREQUIRED so that the DCI circuitry is allowed to operate normally.*

---

7. **`ck`** pair(s) must be on any PN pair(s) in the Address/Control byte lanes.
8. **`reset_n`** can be on any pin as long as general interconnect timing is met and I/O standard must be SSTL15. Reset to DRAM should be pulled down so it is held low during

power up. When `dm` is disabled, the `reset` pin can be allocated to N0th pin of data byte lane or any other free pin of that byte lane as long as other rules are not violated.

RECOMMENDED: *The recommended resistor should be a 4.7 kΩ pull-down.*

9. Banks can be shared between two controllers.
   a. Each byte lane is dedicated to a specific controller (except for `reset_n`).
   b. Byte lanes from one controller cannot be placed inside the other. For example, with controllers A and B, "AABB" is allowed, while "ABAB" is not.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must share a common reset input.*

10. All I/O banks used by the memory interface must be in the same column.
11. All I/O banks used by the memory interface must be in the same SLR of the column for the SSI technology devices.
12. Maximum height of interface is five contiguous banks. The maximum supported interface is 80-bit wide.

    Maximum component limit is nine and this restriction is valid for components only and not for DIMMs.

13. Bank skipping is not allowed.
14. Input clock for the MMCM in the interface must come from a GCIO pair in the I/O column used for the memory interface. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements and MMCM requirements should be considered). For more information, see Clocking, page 81.
15. There are dedicated V_{REF} pins (not included in the rules above). Either internal or external V_{REF} is permitted. If an external V_{REF} is not used, the V_{REF} pins must be pulled to ground by a resistor value specified in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7]. These pins must be connected appropriately for the standard in use.
16. The interface must be contained within the same I/O bank type (High Range or High Performance). Mixing bank types is not permitted with the exceptions of the `reset_n` in step 7 and the input clock mentioned in step 12.
17. The `par` pin is required for DDR3 RDIMMs. For more information on parity errors, see the Address Parity, page 34.
18. The system reset pin (`sys_rst_n`) must not be allocated to Pins N0 and N6 if the byte is used for the memory I/Os.

Note: If PCB compatibility between x4 and x8 based DIMMs is desired, additional restrictions apply. The upper x4 DQS group must be placed within the lower byte nibble (N0 to N5). This allows DM to be placed on N0 for the x8 pinout, pin compatibility for all DQ bits, and the added DQS pair for x4 be placed on N0/N1.

For example, a typical DDR3 x4 based RDIMM data sheet shows the DQS9 associated with DQ4, DQ5, DQ6, and DQ7. This `DQS9_p` is used for the DM in an x8 configuration. This nibble must be connected to the lower nibble of the byte lane. The Vivado® generated XDC labels this DQS9 as DSQ1 (for more information, see the Pin Mapping for x4 RDIMMs/LRDIMMs). Table 4-1 and Table 4-2 include an example for one of the configurations of x4/x8/x16.

Table 4-1: **Byte Lane View of Bank on FPGA Die for x8 and x16 Support**

I/O Type	Byte Lane	Pin Number	Signal Name
–	T0U	N12	–
N	T0U	N11	DQ[7:0]
P	T0U	N10	DQ[7:0]
N	T0U	N9	DQ[7:0]
P	T0U	N8	DQ[7:0]
DQSCC-N	T0U	N7	DQS0_N
DQSCC-P	T0U	N6	DQS0_P
N	T0L	N5	DQ[7:0]
P	T0L	N4	DQ[7:0]
N	T0L	N3	DQ[7:0]
P	T0L	N2	DQ[7:0]
DQSCC-N	T0L	N1	–
DQSCC-P	T0L	N0	DM0

Table 4-2: **Byte Lane View of Bank on FPGA Die for x4, x8, and x16 Support**

I/O Type	Byte Lane	Pin Number	Signal Name
–	T0U	N12	–
N	T0U	N11	DQ[3:0]
P	T0U	N10	DQ[3:0]
N	T0U	N9	DQ[3:0]
P	T0U	N8	DQ[3:0]
DQSCC-N	T0U	N7	DQS0_N
DQSCC-P	T0U	N6	DQS0_P
N	T0L	N5	DQ[7:4]
P	T0L	N4	DQ[7:4]
N	T0L	N3	DQ[7:4]
P	T0L	N2	DQ[7:4]

Table 4-2: **Byte Lane View of Bank on FPGA Die for x4, x8, and x16 Support *(Cont'd)***

I/O Type	Byte Lane	Pin Number	Signal Name
DQSCC-N	T0L	N1	–/DQS9_N
DQSCC-P	T0L	N0	DM0/DQS9_P

Pin Swapping

- Pins can swap freely within each byte group (data and address/control), except for the DQS pair which must be on the dedicated `dqs` pair in the nibble (for more information, see the `dqs`, `dq`, and `dm` location., page 87).
- Byte groups (data and address/control) can swap easily with each other.
- Pins in the address/control byte groups can swap freely within and between their byte groups.
- No other pin swapping is permitted.

DDR3 Pinout Examples

IMPORTANT: *Due to the calibration stage, there is no need for `set_input_delay/ set_output_delay` on the DDR3 SDRAM. Ignore the unconstrained inputs and outputs for DDR3 SDRAM and the signals which are calibrated.*

Table 4-3 shows an example of a 16-bit DDR3 interface contained within one bank. This example is for a component interface using two x8 DDR3 components.

Table 4-3: **16-Bit DDR3 (x8/x16 Part) Interface Contained in One Bank**

Bank	Signal Name	Byte Group	I/O Type
1	a0	T3U_12	–
1	a1	T3U_11	N
1	a2	T3U_10	P
1	a3	T3U_9	N
1	a4	T3U_8	P
1	a5	T3U_7	N
1	a6	T3U_6	P
1	a7	T3L_5	N
1	a8	T3L_4	P
1	a9	T3L_3	N
1	a10	T3L_2	P
1	a11	T3L_1	N

Table 4-3: **16-Bit DDR3 (x8/x16 Part) Interface Contained in One Bank *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	a12	T3L_0	P
1	a13	T2U_12	–
1	a14	T2U_11	N
1	we_n	T2U_10	P
1	cas_n	T2U_9	N
1	ras_n	T2U_8	P
1	ck_n	T2U_7	N
1	ck_p	T2U_6	P
1	cs_n	T2L_5	N
1	ba0	T2L_4	P
1	ba1	T2L_3	N
1	ba2	T2L_2	P
1	sys_clk_n	T2L_1	N
1	sys_clk_p	T2L_0	P
1	cke	T1U_12	–
1	dq15	T1U_11	N
1	dq14	T1U_10	P
1	dq13	T1U_9	N
1	dq12	T1U_8	P
1	dqs1_n	T1U_7	N
1	dqs1_p	T1U_6	P
1	dq11	T1L_5	N
1	dq10	T1L_4	P
1	dq9	T1L_3	N
1	dq8	T1L_2	P
1	odt	T1L_1	N
1	dm1	T1L_0	P
1	vrp	T0U_12	–
1	dq7	T0U_11	N
1	dq6	T0U_10	P

Table 4-3: **16-Bit DDR3 (x8/x16 Part) Interface Contained in One Bank *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	dq5	T0U_9	N
1	dq4	T0U_8	P
1	dqs0_n	T0U_7	N
1	dqs0_p	T0U_6	P
1	dq3	T0L_5	N
1	dq2	T0L_4	P
1	dq1	T0L_3	N
1	dq0	T0L_2	P
1	reset_n	T0L_1	N
1	dm0	T0L_0	P

Table 4-4 shows an example of a 16-bit DDR3 interface contained within one bank. This example is for a component interface using four x4 DDR3 components.

Table 4-4: **16-Bit DDR3 Interface (x4 Part) Contained in One Bank**

Bank	Signal Name	Byte Group	I/O Type
1	a0	T3U_12	–
1	a1	T3U_11	N
1	a2	T3U_10	P
1	a3	T3U_9	N
1	a4	T3U_8	P
1	a5	T3U_7	N
1	a6	T3U_6	P
1	a7	T3L_5	N
1	a8	T3L_4	P
1	a9	T3L_3	N
1	a10	T3L_2	P
1	a11	T3L_1	N
1	a12	T3L_0	P
1	a13	T2U_12	–
1	a14	T2U_11	N
1	we_n	T2U_10	P
1	cas_n	T2U_9	N
1	ras_n	T2U_8	P

Table 4-4: **16-Bit DDR3 Interface (x4 Part) Contained in One Bank *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	ck_n	T2U_7	N
1	ck_p	T2U_6	P
1	cs_n	T2L_5	N
1	ba0	T2L_4	P
1	ba1	T2L_3	N
1	ba2	T2L_2	P
1	sys_clk_n	T2L_1	N
1	sys_clk_p	T2L_0	P
1	cke	T1U_12	–
1	dq15	T1U_11	N
1	dq14	T1U_10	P
1	dq13	T1U_9	N
1	dq12	T1U_8	P
1	dqs3_n	T1U_7	N
1	dqs3_p	T1U_6	P
1	dq11	T1L_5	N
1	dq10	T1L_4	P
1	dq9	T1L_3	N
1	dq8	T1L_2	P
1	dqs2_n	T1L_1	N
1	dqs2_p	T1L_0	P
1	vrp	T0U_12	–
1	dq7	T0U_11	N
1	dq6	T0U_10	P
1	dq5	T0U_9	N
1	dq4	T0U_8	P
1	dqs1_n	T0U_7	N
1	dqs1_p	T0U_6	P
1	dq3	T0L_5	N
1	dq2	T0L_4	P
1	dq1	T0L_3	N
1	dq0	T0L_2	P

Table 4-4: **16-Bit DDR3 Interface (x4 Part) Contained in One Bank *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	dqs0_n	T0L_1	N
1	dqs0_p	T0L_0	P

Two DDR3 32-bit interfaces can fit in three banks by using all of the pins in the banks. To fit the configuration in three banks for various scenarios, different Vivado IDE options can be selected (based on requirement). Various Vivado IDE options that lead to pin savings are listed as follows:

- In data byte group, pins 1 and 12 are unused. Unused pins of the data byte group can be used for Address/Control pins if all Address/Control pins are allocated in the same bank.

  For example, if T3 byte group of Bank #2 is selected for data. Pins T3L_1 and T3U_12 are not used by data and these pins can be used for Address/Control if all Address/Control pins are allocated in Bank #2.
- If DCI cascade is selected, the `vrp` pin can be used as normal a I/O.
- Memory reset pin (`reset_n` pin) can be allocated anywhere as long as timing is met.
- System clock pins can be allocated in different banks and must be within the same column of the memory interface banks selected.
- By disabling the **Enabling Chip Select Pin** option in the Vivado IDE, it frees up a pin and the `cs#` ports are not generated.
- By disabling the **Data Mask** option in Vivado IDE, it frees up a pin and the data mask (`dm`) port is not generated.

One of the configurations with two 32-bit DDR3 interfaces in three banks is given in Table 4-5 (it is valid for memory part of x8/x16). Two interface signals are separated by name `c0_` and `c1_`. Example is given with interface-0 (c0) selected in banks 0 and 1 and interface-1 (c1) selected in banks 1 and 2.

Table 4-5: **Two 32-Bit DDR3 Interfaces Contained in Three Banks**

Bank	Signal Name	Byte Group	I/O Type
2	c1_ddr3_we_n	T3U_12	–
2	c1_ddr3_ck_c[0]	T3U_11	N
2	c1_ddr3_ck_t[0]	T3U_10	P
2	c1_ddr3_cas_n	T3U_9	N
2	c1_ddr3_ras_n	T3U_8	P
2	c1_ddr3_ba[2]	T3U_7	N
2	c1_ddr3_ba[1]	T3U_6	P
2	c1_ddr3_ba[0]	T3L_5	N

Table 4-5: **Two 32-Bit DDR3 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
2	c1_ddr3_adr[15]	T3L_4	P
2	c1_ddr3_adr[14]	T3L_3	N
2	c1_ddr3_adr[13]	T3L_2	P
2	c1_ddr3_adr[12]	T3L_1	N
2	c1_ddr3_adr[11]	T3L_0	P
2	c1_ddr3_adr[10]	T2U_12	–
2	c1_ddr3_adr[9]	T2U_11	N
2	c1_ddr3_adr[8]	T2U_10	P
2	c1_ddr3_adr[7]	T2U_9	N
2	c1_ddr3_adr[6]	T2U_8	P
2	c1_ddr3_adr[5]	T2U_7	N
2	c1_ddr3_adr[4]	T2U_6	P
2	c1_ddr3_adr[3]	T2L_5	N
2	c1_ddr3_adr[2]	T2L_4	P
2	c1_ddr3_adr[1]	T2L_3	N
2	c1_ddr3_adr[0]	T2L_2	P
2	c1_sys_clk_n	T2L_1	N
2	c1_sys_clk_p	T2L_0	P
2	c1_ddr3_cke[0]	T1U_12	–
2	c1_ddr3_dq[31]	T1U_11	N
2	c1_ddr3_dq[30]	T1U_10	P
2	c1_ddr3_dq[29]	T1U_9	N
2	c1_ddr3_dq[28]	T1U_8	P
2	c1_ddr3_dqs_n[3]	T1U_7	N
2	c1_ddr3_dqs_p[3]	T1U_6	P
2	c1_ddr3_dq[27]	T1L_5	N
2	c1_ddr3_dq[26]	T1L_4	P
2	c1_ddr3_dq[25]	T1L_3	N
2	c1_ddr3_dq[24]	T1L_2	P
2	c1_ddr3_odt[0]	T1L_1	N
2	c1_ddr3_dm[3]	T1L_0	P
2	vrp	T0U_12	–

Table 4-5: **Two 32-Bit DDR3 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
2	c1_ddr3_dq[23]	T0U_11	N
2	c1_ddr3_dq[22]	T0U_10	P
2	c1_ddr3_dq[21]	T0U_9	N
2	c1_ddr3_dq[20]	T0U_8	P
2	c1_ddr3_dqs_n[2]	T0U_7	N
2	c1_ddr3_dqs_p[2]	T0U_6	P
2	c1_ddr3_dq[19]	T0L_5	N
2	c1_ddr3_dq[18]	T0L_4	P
2	c1_ddr3_dq[17]	T0L_3	N
2	c1_ddr3_dq[16]	T0L_2	P
2	c1_ddr3_cs_n[0]	T0L_1	N
2	c1_ddr3_dm[2]	T0L_0	P
1	c1_ddr3_reset_n	T3U_12	–
1	c1_ddr3_dq[15]	T3U_11	N
1	c1_ddr3_dq[14]	T3U_10	P
1	c1_ddr3_dq[13]	T3U_9	N
1	c1_ddr3_dq[12]	T3U_8	P
1	c1_ddr3_dqs_n[1]	T3U_7	N
1	c1_ddr3_dqs_p[1]	T3U_6	P
1	c1_ddr3_dq[11]	T3L_5	N
1	c1_ddr3_dq[10]	T3L_4	P
1	c1_ddr3_dq[9]	T3L_3	N
1	c1_ddr3_dq[8]	T3L_2	P
1	–	T3L_1	N
1	c1_ddr3_dm[1]	T3L_0	P
1	–	T2U_12	–
1	c1_ddr3_dq[7]	T2U_11	N
1	c1_ddr3_dq[6]	T2U_10	P
1	c1_ddr3_dq[5]	T2U_9	N
1	c1_ddr3_dq[4]	T2U_8	P
1	c1_ddr3_dqs_n[0]	T2U_7	N
1	c1_ddr3_dqs_p[0]	T2U_6	P
1	c1_ddr3_dq[3]	T2L_5	N

Table 4-5: **Two 32-Bit DDR3 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	c1_ddr3_dq[2]	T2L_4	P
1	c1_ddr3_dq[1]	T2L_3	N
1	c1_ddr3_dq[0]	T2L_2	P
1	–	T2L_1	N
1	c1_ddr3_dm[0]	T2L_0	P
1	–	T1U_12	–
1	c0_ddr3_dq[31]	T1U_11	N
1	c0_ddr3_dq[30]	T1U_10	P
1	c0_ddr3_dq[29]	T1U_9	N
1	c0_ddr3_dq[28]	T1U_8	P
1	c0_ddr3_dqs_n[3]	T1U_7	N
1	c0_ddr3_dqs_p[3]	T1U_6	P
1	c0_ddr3_dq[27]	T1L_5	N
1	c0_ddr3_dq[26]	T1L_4	P
1	c0_ddr3_dq[25]	T1L_3	N
1	c0_ddr3_dq[24]	T1L_2	P
1	–	T1L_1	N
1	c0_ddr3_dm[3]	T1L_0	P
1	–	T0U_12	–
1	c0_ddr3_dq[23]	T0U_11	N
1	c0_ddr3_dq[22]	T0U_10	P
1	c0_ddr3_dq[21]	T0U_9	N
1	c0_ddr3_dq[20]	T0U_8	P
1	c0_ddr3_dqs_n[2]	T0U_7	N
1	c0_ddr3_dqs_p[2]	T0U_6	P
1	c0_ddr3_dq[19]	T0L_5	N
1	c0_ddr3_dq[18]	T0L_4	P
1	c0_ddr3_dq[17]	T0L_3	N
1	c0_ddr3_dq[16]	T0L_2	P
1	c0_ddr3_reset_n	T0L_1	N
1	c0_ddr3_dm[2]	T0L_0	P
0	c0_ddr3_cs_n[0]	T3U_12	–

Table 4-5: **Two 32-Bit DDR3 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
0	c0_ddr3_dq[15]	T3U_11	N
0	c0_ddr3_dq[14]	T3U_10	P
0	c0_ddr3_dq[13]	T3U_9	N
0	c0_ddr3_dq[12]	T3U_8	P
0	c0_ddr3_dqs_n[1]	T3U_7	N
0	c0_ddr3_dqs_p[1]	T3U_6	P
0	c0_ddr3_dq[11]	T3L_5	N
0	c0_ddr3_dq[10]	T3L_4	P
0	c0_ddr3_dq[9]	T3L_3	N
0	c0_ddr3_dq[8]	T3L_2	P
0	c0_ddr3_cke[0]	T3L_1	N
0	c0_ddr3_dm[1]	T3L_0	P
0	c0_ddr3_odt[0]	T2U_12	–
0	c0_ddr3_dq[7]	T2U_11	N
0	c0_ddr3_dq[6]	T2U_10	P
0	c0_ddr3_dq[5]	T2U_9	N
0	c0_ddr3_dq[4]	T2U_8	P
0	c0_ddr3_dqs_n[0]	T2U_7	N
0	c0_ddr3_dqs_p[0]	T2U_6	P
0	c0_ddr3_dq[3]	T2L_5	N
0	c0_ddr3_dq[2]	T2L_4	P
0	c0_ddr3_dq[1]	T2L_3	N
0	c0_ddr3_dq[0]	T2L_2	P
0	c0_ddr3_we_n	T2L_1	N
0	c0_ddr3_dm[0]	T2L_0	P
0	c0_ddr3_cas_n	T1U_12	–
0	c0_ddr3_ck_c[0]	T1U_11	N
0	c0_ddr3_ck_t[0]	T1U_10	P
0	c0_sys_clk_n	T1U_9	N
0	c0_sys_clk_p	T1U_8	P
0	c0_ddr3_ras_n	T1U_7	N
0	c0_ddr3_ba[2]	T1U_6	P
0	c0_ddr3_ba[1]	T1L_5	N

Table 4-5: **Two 32-Bit DDR3 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
0	c0_ddr3_ba[0]	T1L_4	P
0	c0_ddr3_addr[15]	T1L_3	N
0	c0_ddr3_addr[14]	T1L_2	P
0	c0_ddr3_addr[13]	T1L_1	N
0	c0_ddr3_addr[12]	T1L_0	P
0	vrp	T0U_12	–
0	c0_ddr3_addr[11]	T0U_11	N
0	c0_ddr3_addr[10]	T0U_10	P
0	c0_ddr3_addr[9]	T0U_9	N
0	c0_ddr3_addr[8]	T0U_8	P
0	c0_ddr3_addr[7]	T0U_7	N
0	c0_ddr3_addr[6]	T0U_6	P
0	c0_ddr3_addr[5]	T0L_5	N
0	c0_ddr3_addr[4]	T0L_4	P
0	c0_ddr3_addr[3]	T0L_3	N
0	c0_ddr3_addr[2]	T0L_2	P
0	c0_ddr3_addr[1]	T0L_1	N
0	c0_ddr3_addr[0]	T0L_0	P

DDR4 Pin Rules

IMPORTANT: *Xilinx advises Tandem Configuration users to avoid using bank 65 for design applications, especially when using Tandem PROM, to avoid complications because the programming bitstream is split into two stages. Specifically, IP cores built by the Memory IP or Memory Interface Generator (MIG) must not use bank 65 I/O. This ensures that IP can remain completely within stage 2, and avoid complications with its embedded I/O and demanding timing constraints.*

The rules are for single and multi-rank memory interfaces.

- Address/control means **`cs_n`**, **`ras_n`** (a16), **`cas_n`** (a15), **`we_n`** (a14), **`ba`**, **`bg`**, **`ck`**, **`cke`**, **`a`**, **`odt`**, **`act_n`**, and **`parity`** (valid for RDIMMs and LRDIMMs only). Multi-rank systems have one **`cs_n`**, **`cke`**, **`odt`**, and one **`ck`** pair per rank.
- Pins in a byte lane are numbered N0 to N12.
- Byte lanes in a bank are designed by T0, T1, T2, or T3. Nibbles within a byte lane are distinguished by a "U" or "L" designator added to the byte lane designator (T0, T1, T2, or T3). Thus they are T0L, T0U, T1L, T1U, T2L, T2U, T3L, and T3U.

Note: There are two PLLs per bank and a controller uses one PLL in every bank that is being used by the interface.

1. **`dqs,`** **`dq`**, and **`dm/dbi`** location.
    a. Designs using x8 or x16 components – **`dqs`** must be located on a dedicated byte clock pair in the upper nibble designated with "U" (N6 and N7). **`dq`** associated with a **`dqs`** must be in same byte lane on any of the other pins except pins N1 and N12.
    b. Designs using x4 components – **`dqs`** must be located on a dedicated byte clock pair in the nibble (N0 and N1 in the lower nibble, N6 and N7 in the upper nibble). **`dq`** associated with a **`dqs`** must be in same nibble on any of the other pins except pin N12 (upper nibble). The lower nibble **`dq`** and upper nibble **`dq`** must be allocated in the same byte lane.

    Note: The **`dm/dbi`** port is not supported in x4 DDR4 devices.

    c. **`dm/dbi`** must be on pin N0 in the byte lane with the associated **`dqs`**.

    Note: When the IP is configured as NO_DM_NO_DBI, the IP will always generate the ports and they cannot be disabled, but they will not be used during calibration or operation. On the FPGA side, these pins can be left floating. At the memory device, follow the manufacturer's recommendation which is leave floating or a weak pull-up to VDDQ. If the ports are not routed to the memory device and the IP is not set to NO_DM_NO_DBI, then the design will fail calibration.

    d. The x16 components must have the **`ldqs`** connected to the even **`dqs`** and the **`udqs`** must be connected to the **`ldqs`** + 1. The first x16 component has **`ldqs`** connected to **`dqs0`** and **`udqs`** connected to **`dqs1`** in the XDC file. The second x16 component has **`ldqs`** connected to **`dqs2`** and **`udqs`** connected to **`dqs3`**. This pattern continues as needed for the interface. This does not restrict the physical location of the byte lanes. The byte lanes associated with the **`dqs`s`** might be moved as desired in the Vivado IDE to achieve optimal PCB routing.

    Consider x16 part with data width of 32 and all data bytes are allocated in a single bank. In such cases, DQS needs to be mapped as given in Table 4-6.

    In Table 4-6, the Bank-Byte and Selected Memory Data Bytes indicate byte allocation in the I/O pin planner. The following example is given for one of the generated configuration in the I/O pin planner. Based on pin allocation, DQ byte allocation might vary.

    DQS Allocated (in IP on the FPGA) indicates DQS that is allocated on the FPGA end. Memory device mapping indicates how DQS needs to be mapped on the memory end.

Table 4-6: **DQS Mapping for x16 Component**

Bank-Byte	Selected Memory Data Bytes	DQS Allocated (in IP on FPGA)	Memory Device Mapping
BankX_BYTE3	DQ[0-7]	DQS0	Memory Device 0 – LDQS
BankX_BYTE2	DQ[8-15]	DQS1	Memory Device 0 – UDQS

Table 4-6: **DQS Mapping for x16 Component *(Cont'd)***

Bank-Byte	Selected Memory Data Bytes	DQS Allocated (in IP on FPGA)	Memory Device Mapping
BankX_BYTE1	DQ[16-23]	DQS2	Memory Device 1 – LDQS
BankX_BYTE0	DQ[24-31]	DQS3	Memory Device 1 – UDQS

2. The x4 components must be used in pairs. Odd numbers of x4 components are not permitted. Both the upper and lower nibbles of a data byte must be occupied by a x4 **`dq/dqs`** group. Each byte lane containing two x4 nibbles must have sequential nibbles with the even nibble being the lower number. For example, a byte lane can have nibbles 0 and 1, or 2 and 3, but must not have 1 and 2. The ordering of the nibbles within a byte lane is not important.

3. Byte lanes with a **`dqs`** are considered to be data byte lanes. Pins N1 and N12 can be used for address/control in a data byte lane. If the data byte is in the same bank as the remaining address/control pins, see step #4.

4. Address/control can be on any of the 13 pins in the address/control byte lanes. Address/control must be contained within the same bank.

5. One **`vrp`** pin per bank is used and DCI is required for the interfaces. A **`vrp`** pin is required in I/O banks containing inputs as well as in output only banks. It is required in output only banks because address/control signals use SSTL12_DCI to enable usage of controlled output impedance. DCI cascade is allowed for data rates of 2,133 Mb/s and lower. When DCI cascade is used, **`vrp`** pin can be used as a normal I/O. All rules for the DCI in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7] must be followed.

RECOMMENDED: *Xilinx strongly recommends that the DCIUpdateMode option is kept with the default value of ASREQUIRED so that the DCI circuitry is allowed to operate normally.*

---

6. **`ck`** pair(s) must be on any PN pair(s) in the Address/Control byte lanes.

7. **`reset_n`** can be on any pin as long as general interconnect timing is met and I/O standard must be LVCMOS12. Reset to DRAM should be pulled down so it is held low during power up.

RECOMMENDED: *The recommended resistor should be a 4.7 kΩ pull-down.*

---

8. Banks can be shared between two controllers.
   a. Each byte lane is dedicated to a specific controller (except for **`reset_n`**).
   b. Byte lanes from one controller cannot be placed inside the other. For example, with controllers A and B, "AABB" is allowed, while "ABAB" is not.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must share a common reset input.*

---

9. All I/O banks used by the memory interface must be in the same column.
10. All I/O banks used by the memory interface must be in the same SLR of the column for the SSI technology devices.
11. For dual slot configurations of RDIMMs, LRDIMMs, and UDIMMs: **`cs`**, **`odt`**, **`cke`**, and **`ck`** port widths are doubled. For exact mapping of the signals, see the DIMM Configurations.
12. Maximum height of interface is five contiguous banks. The maximum supported interface is 80-bit wide.

    Maximum component limit is nine and this restriction is valid for components only and not for DIMMs.
13. Bank skipping is not allowed.
14. Input clock for the MMCM in the interface must come from the a GCIO pair in the I/O column used for the memory interface. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements and MMCM requirements should be considered). For more information, see Clocking, page 81.
15. The dedicated V_{REF} pins in the banks used for DDR4 must be tied to ground with a resistor value specified in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7]. Internal V_{REF} is required for DDR4.
16. The interface must be contained within the same I/O bank type (High Performance). Mixing bank types is not permitted with the exceptions of the **`reset_n`** in step #7 and the input clock mentioned in step #14.
17. The **`par`** input for command and address parity, **`alert_n`** input/output, and the TEN input for Connectivity Test Mode are not supported by this interface. Consult *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11] on how to connect these signals when not used. For more information on parity errors, see the Address Parity, page 34.
18. For all other DRAM/DIMM pins that are not mentioned in this section, for example, SAx, SCL, SDA, contact the memory vendor for proper connectivity.
19. The system reset pin (**`sys_rst_n`**) must not be allocated to Pins N0 and N6 if the byte is used for the memory I/Os.

IMPORTANT: *Component interfaces should be created with the same component for all components in the interface. x16 components have a different number of bank groups than the x8 components. For example, a 72-bit wide component interface should be created by using nine x8 components or five x16 components where half of one component is not used. Four x16 components and one x8 component is not permissible.*

Note: Pins N0 and N6 within the byte lane used by a memory interface can be utilized for other purposes when not needed for the memory interface. However, the functionality of these pins is not available until VTC_RDY asserts on the BITSLICE_CONTROL. For more information, see the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7].

If PCB compatibility between x4 and x8 based DIMMs is desired, additional restrictions apply. The upper x4 DQS group must be placed within the lower byte nibble (N0 to N5). This allows DM to be placed on N0 for the x8 pinout, pin compatibility for all DQ bits, and the added DQS pair for x4 be placed on N0/N1.

For example, a typical DDR4 x4 based RDIMM/LRDIMM data sheet shows the DQS9 associated with DQ4, DQ5, DQ6, and DQ7. This `DQS9_t` is used for the DM/DBI in an x8 configuration. This nibble must be connected to the lower nibble of the byte lane. The Vivado generated XDC labels this DQS9 as DSQ1 (for more information, see the Pin Mapping for x4 RDIMMs/LRDIMMs). Table 4-7 and Table 4-8 include an example for one of the configurations of x4/x8/x16.

Table 4-7: **Byte Lane View of Bank on FPGA Die for x8 and x16 Support**

I/O Type	Byte Lane	Pin Number	Signal Name
–	T0U	N12	–
N	T0U	N11	DQ[7:0]
P	T0U	N10	DQ[7:0]
N	T0U	N9	DQ[7:0]
P	T0U	N8	DQ[7:0]
DQSCC-N	T0U	N7	DQS0_c
DQSCC-P	T0U	N6	DQS0_t
N	T0L	N5	DQ[7:0]
P	T0L	N4	DQ[7:0]
N	T0L	N3	DQ[7:0]
P	T0L	N2	DQ[7:0]
DQSCC-N	T0L	N1	–
DQSCC-P	T0L	N0	DM0/DBI0

Table 4-8: **Byte Lane View of Bank on FPGA Die for x4, x8, and x16 Support**

I/O Type	Byte Lane	Pin Number	Signal Name
–	T0U	N12	–
N	T0U	N11	DQ[3:0]
P	T0U	N10	DQ[3:0]
N	T0U	N9	DQ[3:0]
P	T0U	N8	DQ[3:0]
DQSCC-N	T0U	N7	DQS0_c
DQSCC-P	T0U	N6	DQS0_t
N	T0L	N5	DQ[7:4]

Table 4-8: **Byte Lane View of Bank on FPGA Die for x4, x8, and x16 Support *(Cont'd)***

I/O Type	Byte Lane	Pin Number	Signal Name
P	T0L	N4	DQ[7:4]
N	T0L	N3	DQ[7:4]
P	T0L	N2	DQ[7:4]
DQSCC-N	T0L	N1	–/DQS9_c
DQSCC-P	T0L	N0	DM0/DBI0/DQS9_t

Pin Swapping

- Pins can swap freely within each byte group (data and address/control), except for the DQS pair which must be on the dedicated **`dqs`** pair in the nibble (for more information, see the **`dqs,`** **`dq`**, and **`dm/dbi`** location., page 101).
- Byte groups (data and address/control) can swap easily with each other.
- Pins in the address/control byte groups can swap freely within and between their byte groups.
- No other pin swapping is permitted.

DDR4 Pinout Examples

IMPORTANT: *Due to the calibration stage, there is no need for* `set_input_delay/` `set_output_delay` *on the DDR4 SDRAM. Ignore the unconstrained inputs and outputs for DDR4 SDRAM and the signals which are calibrated.*

Table 4-9 shows an example of a 32-bit DDR4 interface contained within two banks. This example is for a component interface using four x8 DDR4 components.

Table 4-9: **32-Bit DDR4 Interface Contained in Two Banks**

Bank	Signal Name	Byte Group	I/O Type
Bank 1			
1	–	T3U_12	–
1	–	T3U_11	N
1	–	T3U_10	P
1	–	T3U_9	N
1	–	T3U_8	P
1	–	T3U_7	N
1	–	T3U_6	P
1	–	T3L_5	N

Table 4-9: **32-Bit DDR4 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	–	T3L_4	P
1	–	T3L_3	N
1	–	T3L_2	P
1	–	T3L_1	N
1	–	T3L_0	P
1	–	T2U_12	–
1	–	T2U_11	N
1	–	T2U_10	P
1	–	T2U_9	N
1	–	T2U_8	P
1	–	T2U_7	N
1	–	T2U_6	P
1	–	T2L_5	N
1	–	T2L_4	P
1	–	T2L_3	N
1	–	T2L_2	P
1	–	T2L_1	N
1	–	T2L_0	P
1	reset_n	T1U_12	–
1	dq31	T1U_11	N
1	dq30	T1U_10	P
1	dq29	T1U_9	N
1	dq28	T1U_8	P
1	dqs3_c	T1U_7	N
1	dqs3_t	T1U_6	P
1	dq27	T1L_5	N
1	dq26	T1L_4	P
1	dq25	T1L_3	N
1	dq24	T1L_2	P
1	unused	T1L_1	N
1	dm3/dbi3	T1L_0	P

Table 4-9: **32-Bit DDR4 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	vrp	T0U_12	–
1	dq23	T0U_11	N
1	dq22	T0U_10	P
1	dq21	T0U_9	N
1	dq20	T0U_8	P
1	dqs2_c	T0U_7	N
1	dqs2_t	T0U_6	P
1	dq19	T0L_5	N
1	dq18	T0L_4	P
1	dq17	T0L_3	N
1	dq16	T0L_2	P
1	–	T0L_1	N
1	dm2/dbi2	T0L_0	P
Bank 2			
2	a0	T3U_12	–
2	a1	T3U_11	N
2	a2	T3U_10	P
2	a3	T3U_9	N
2	a4	T3U_8	P
2	a5	T3U_7	N
2	a6	T3U_6	P
2	a7	T3L_5	N
2	a8	T3L_4	P
2	a9	T3L_3	N
2	a10	T3L_2	P
2	a11	T3L_1	N
2	a12	T3L_0	P
2	a13	T2U_12	–
2	we_n/a14	T2U_11	N
2	cas_n/a15	T2U_10	P
2	ras_n/a16	T2U_9	N

Table 4-9: **32-Bit DDR4 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
2	act_n	T2U_8	P
2	ck_c	T2U_7	N
2	ck_t	T2U_6	P
2	ba0	T2L_5	N
2	ba1	T2L_4	P
2	bg0	T2L_3	N
2	bg1	T2L_2	P
2	sys_clk_n	T2L_1	N
2	sys_clk_p	T2L_0	P
2	cs_n	T1U_12	–
2	dq15	T1U_11	N
2	dq14	T1U_10	P
2	dq13	T1U_9	N
2	dq12	T1U_8	P
2	dqs1_c	T1U_7	N
2	dqs1_t	T1U_6	P
2	dq11	T1L_5	N
2	dq10	T1L_4	P
2	dq9	T1L_3	N
2	dq8	T1L_2	P
2	odt	T1L_1	N
2	dm1/dbi1	T1L_0	P
2	vrp	T0U_12	–
2	dq7	T0U_11	N
2	dq6	T0U_10	P
2	dq5	T0U_9	N
2	dq4	T0U_8	P
2	dqs0_c	T0U_7	N
2	dqs0_t	T0U_6	P
2	dq3	T0L_5	N
2	dq2	T0L_4	P

Table 4-9: **32-Bit DDR4 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
2	dq1	T0L_3	N
2	dq0	T0L_2	P
2	cke	T0L_1	N
2	dm0/dbi0	T0L_0	P

Table 4-10 shows an example of a 16-bit DDR4 interface contained within a single bank. This example is for a component interface using four x4 DDR4 components.

Table 4-10: **16-Bit DDR4 Interface (x4 Part) Contained in One Bank**

Bank	Signal Name	Byte Group	I/O Type
1	a0	T3U_12	–
1	a1	T3U_11	N
1	a2	T3U_10	P
1	a3	T3U_9	N
1	a4	T3U_8	P
1	a5	T3U_7	N
1	a6	T3U_6	P
1	a7	T3L_5	N
1	a8	T3L_4	P
1	a9	T3L_3	N
1	a10	T3L_2	P
1	a11	T3L_1	N
1	a12	T3L_0	P
1	a13	T2U_12	–
1	we_n/a14	T2U_11	N
1	cas_n/a15	T2U_10	P
1	ras_n/a16	T2U_9	N
1	act_n	T2U_8	P
1	ck_c	T2U_7	N
1	ck_t	T2U_6	P
1	ba0	T2L_5	N
1	ba1	T2L_4	P
1	bg0	T2L_3	N
1	bg1	T2L_2	P

Table 4-10: **16-Bit DDR4 Interface (x4 Part) Contained in One Bank *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	odt	T2L_1	N
1	cke	T2L_0	P
1	cs_n	T1U_12	–
1	dq15	T1U_11	N
1	dq14	T1U_10	P
1	dq13	T1U_9	N
1	dq12	T1U_8	P
1	dqs3_c	T1U_7	N
1	dqs3_t	T1U_6	P
1	dq11	T1L_5	N
1	dq10	T1L_4	P
1	dq9	T1L_3	N
1	dq8	T1L_2	P
1	dqs2_c	T1L_1	N
1	dqs2_t	T1L_0	P
1	vrp	T0U_12	–
1	dq7	T0U_11	N
1	dq6	T0U_10	P
1	dq5	T0U_9	N
1	dq4	T0U_8	P
1	dqs1_c	T0U_7	N
1	dqs1_t	T0U_6	P
1	dq3	T0L_5	N
1	dq2	T0L_4	P
1	dq1	T0L_3	N
1	dq0	T0L_2	P
1	dqs0_c	T0L_1	N
1	dqs0_t	T0L_0	P

Note: System clock pins (`sys_clk_p` and `sys_clk_n`) are allocated in different banks.

Two DDR4 32-bit interfaces can fit in three banks by using all of the pins in the banks. To fit the configuration in three banks for various scenarios, different Vivado IDE options can be selected (based on requirement). Various Vivado IDE options that lead to pin savings are listed as follows:

- In data byte group, pins 1 and 12 are unused. Unused pins of the data byte group can be used for Address/Control pins if all Address/Control pins are allocated in the same bank.

  For example, if T3 byte group of Bank #2 is selected for data. Pins T3L_1 and T3U_12 are not used by data and these pins can be used for Address/Control if all Address/Control pins are allocated in Bank #2.
- If DCI cascade is selected, the `vrp` pin can be used as normal a I/O. DCI cascade is allowed for data rates of 2,133 Mb/s and lower.
- Memory reset pin (`reset_n` pin) can be allocated anywhere as long as timing is met.
- System clock pins can be allocated in different banks and must be within the same column of the memory interface banks selected.

One of the configurations with two 32-bit DDR4 interfaces in three banks is given in Table 4-11 (it is valid for memory part of x8/x16). Two interface signals are separated by name `c0_` and `c1_`. Example is given with interface-0 (c0) selected in banks 0 and 1 and interface-1 (c1) selected in banks 1 and 2.

Table 4-11: **Two 32-Bit DDR4 Interfaces Contained in Three Banks**

Bank	Signal Name	Byte Group	I/O Type
2	c1_ddr4_cke[0]	T3U_12	–
2	c1_ddr4_ck_c[0]	T3U_11	N
2	c1_ddr4_ck_t[0]	T3U_10	P
2	c1_ddr4_bg[1]	T3U_9	N
2	c1_ddr4_bg[0]	T3U_8	P
2	c1_ddr4_ba[1]	T3U_7	N
2	c1_ddr4_ba[0]	T3U_6	P
2	c1_ddr4_adr[16]	T3L_5	N
2	c1_ddr4_adr[15]	T3L_4	P
2	c1_ddr4_adr[14]	T3L_3	N
2	c1_ddr4_adr[13]	T3L_2	P
2	c1_ddr4_adr[12]	T3L_1	N
2	c1_ddr4_adr[11]	T3L_0	P
2	c1_ddr4_adr[10]	T2U_12	–
2	c1_ddr4_adr[9]	T2U_11	N
2	c1_ddr4_adr[8]	T2U_10	P
2	c1_ddr4_adr[7]	T2U_9	N
2	c1_ddr4_adr[6]	T2U_8	P
2	c1_ddr4_adr[5]	T2U_7	N

Table 4-11: **Two 32-Bit DDR4 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
2	c1_ddr4_adr[4]	T2U_6	P
2	c1_ddr4_adr[3]	T2L_5	N
2	c1_ddr4_adr[2]	T2L_4	P
2	c1_ddr4_adr[1]	T2L_3	N
2	c1_ddr4_adr[0]	T2L_2	P
2	c1_sys_clk_n	T2L_1	N
2	c1_sys_clk_p	T2L_0	P
2	c1_ddr4_act_n	T1U_12	–
2	c1_ddr4_dq[31]	T1U_11	N
2	c1_ddr4_dq[30]	T1U_10	P
2	c1_ddr4_dq[29]	T1U_9	N
2	c1_ddr4_dq[28]	T1U_8	P
2	c1_ddr4_dqs_c[3]	T1U_7	N
2	c1_ddr4_dqs_t[3]	T1U_6	P
2	c1_ddr4_dq[27]	T1L_5	N
2	c1_ddr4_dq[26]	T1L_4	P
2	c1_ddr4_dq[25]	T1L_3	N
2	c1_ddr4_dq[24]	T1L_2	P
2	c1_ddr4_odt[0]	T1L_1	N
2	c1_ddr4_dm_dbi[3]	T1L_0	P
2	vrp	T0U_12	–
2	c1_ddr4_dq[23]	T0U_11	N
2	c1_ddr4_dq[22]	T0U_10	P
2	c1_ddr4_dq[21]	T0U_9	N
2	c1_ddr4_dq[20]	T0U_8	P
2	c1_ddr4_dqs_c[2]	T0U_7	N
2	c1_ddr4_dqs_t[2]	T0U_6	P
2	c1_ddr4_dq[19]	T0L_5	N
2	c1_ddr4_dq[18]	T0L_4	P
2	c1_ddr4_dq[17]	T0L_3	N
2	c1_ddr4_dq[16]	T0L_2	P
2	c1_ddr4_cs_n[0]	T0L_1	N
2	c1_ddr4_dm_dbi[2]	T0L_0	P

Table 4-11: **Two 32-Bit DDR4 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	c1_ddr4_reset_n	T3U_12	–
1	c1_ddr4_dq[15]	T3U_11	N
1	c1_ddr4_dq[14]	T3U_10	P
1	c1_ddr4_dq[13]	T3U_9	N
1	c1_ddr4_dq[12]	T3U_8	P
1	c1_ddr4_dqs_c[1]	T3U_7	N
1	c1_ddr4_dqs_t[1]	T3U_6	P
1	c1_ddr4_dq[11]	T3L_5	N
1	c1_ddr4_dq[10]	T3L_4	P
1	c1_ddr4_dq[9]	T3L_3	N
1	c1_ddr4_dq[8]	T3L_2	P
1	–	T3L_1	N
1	c1_ddr4_dm_dbi[1]	T3L_0	P
1	–	T2U_12	–
1	c1_ddr4_dq[7]	T2U_11	N
1	c1_ddr4_dq[6]	T2U_10	P
1	c1_ddr4_dq[5]	T2U_9	N
1	c1_ddr4_dq[4]	T2U_8	P
1	c1_ddr4_dqs_c[0]	T2U_7	N
1	c1_ddr4_dqs_t[0]	T2U_6	P
1	c1_ddr4_dq[3]	T2L_5	N
1	c1_ddr4_dq[2]	T2L_4	P
1	c1_ddr4_dq[1]	T2L_3	N
1	c1_ddr4_dq[0]	T2L_2	P
1	–	T2L_1	N
1	c1_ddr4_dm_dbi[0]	T2L_0	P
1	–	T1U_12	–
1	c0_ddr4_dq[31]	T1U_11	N
1	c0_ddr4_dq[30]	T1U_10	P
1	c0_ddr4_dq[29]	T1U_9	N
1	c0_ddr4_dq[28]	T1U_8	P
1	c0_ddr4_dqs_c[3]	T1U_7	N

Table 4-11: **Two 32-Bit DDR4 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	c0_ddr4_dqs_t[3]	T1U_6	P
1	c0_ddr4_dq[27]	T1L_5	N
1	c0_ddr4_dq[26]	T1L_4	P
1	c0_ddr4_dq[25]	T1L_3	N
1	c0_ddr4_dq[24]	T1L_2	P
1	–	T1L_1	N
1	c0_ddr4_dm_dbi[3]	T1L_0	P
1	–	T0U_12	–
1	c0_ddr4_dq[23]	T0U_11	N
1	c0_ddr4_dq[22]	T0U_10	P
1	c0_ddr4_dq[21]	T0U_9	N
1	c0_ddr4_dq[20]	T0U_8	P
1	c0_ddr4_dqs_c[2]	T0U_7	N
1	c0_ddr4_dqs_t[2]	T0U_6	P
1	c0_ddr4_dq[19]	T0L_5	N
1	c0_ddr4_dq[18]	T0L_4	P
1	c0_ddr4_dq[17]	T0L_3	N
1	c0_ddr4_dq[16]	T0L_2	P
1	c0_ddr4_reset_n	T0L_1	N
1	c0_ddr4_dm_dbi[2]	T0L_0	P
0	c0_ddr4_bg[1]	T3U_12	–
0	c0_ddr4_dq[15]	T3U_11	N
0	c0_ddr4_dq[14]	T3U_10	P
0	c0_ddr4_dq[13]	T3U_9	N
0	c0_ddr4_dq[12]	T3U_8	P
0	c0_ddr4_dqs_c[1]	T3U_7	N
0	c0_ddr4_dqs_t[1]	T3U_6	P
0	c0_ddr4_dq[11]	T3L_5	N
0	c0_ddr4_dq[10]	T3L_4	P
0	c0_ddr4_dq[9]	T3L_3	N
0	c0_ddr4_dq[8]	T3L_2	P
0	c0_ddr4_cke[0]	T3L_1	N
0	c0_ddr4_dm_dbi[1]	T3L_0	P

Table 4-11: **Two 32-Bit DDR4 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
0	c0_ddr4_act_n	T2U_12	–
0	c0_ddr4_dq[7]	T2U_11	N
0	c0_ddr4_dq[6]	T2U_10	P
0	c0_ddr4_dq[5]	T2U_9	N
0	c0_ddr4_dq[4]	T2U_8	P
0	c0_ddr4_dqs_c[0]	T2U_7	N
0	c0_ddr4_dqs_t[0]	T2U_6	P
0	c0_ddr4_dq[3]	T2L_5	N
0	c0_ddr4_dq[2]	T2L_4	P
0	c0_ddr4_dq[1]	T2L_3	N
0	c0_ddr4_dq[0]	T2L_2	P
0	c0_ddr4_cs_n[0]	T2L_1	N
0	c0_ddr4_dm_dbi[0]	T2L_0	P
0	c0_ddr4_odt[0]	T1U_12	–
0	c0_ddr4_ck_c[0]	T1U_11	N
0	c0_ddr4_ck_t[0]	T1U_10	P
0	c0_sys_clk_n	T1U_9	N
0	c0_sys_clk_p	T1U_8	P
0	c0_ddr4_bg[0]	T1U_7	N
0	c0_ddr4_ba[1]	T1U_6	P
0	c0_ddr4_ba[0]	T1L_5	N
0	c0_ddr4_adr[16]	T1L_4	P
0	c0_ddr4_adr[15]	T1L_3	N
0	c0_ddr4_adr[14]	T1L_2	P
0	c0_ddr4_adr[13]	T1L_1	N
0	c0_ddr4_adr[12]	T1L_0	P
0	vrp	T0U_12	–
0	c0_ddr4_adr[11]	T0U_11	N
0	c0_ddr4_adr[10]	T0U_10	P
0	c0_ddr4_adr[9]	T0U_9	N
0	c0_ddr4_adr[8]	T0U_8	P
0	c0_ddr4_adr[7]	T0U_7	N

Table 4-11: **Two 32-Bit DDR4 Interfaces Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
0	c0_ddr4_adr[6]	T0U_6	P
0	c0_ddr4_adr[5]	T0L_5	N
0	c0_ddr4_adr[4]	T0L_4	P
0	c0_ddr4_adr[3]	T0L_3	N
0	c0_ddr4_adr[2]	T0L_2	P
0	c0_ddr4_adr[1]	T0L_1	N
0	c0_ddr4_adr[0]	T0L_0	P

Pin Mapping for x4 RDIMMs/LRDIMMs

Table 4-12 is an example showing the pin mapping for x4 DDR3 registered DIMMs between the memory data sheet and the XDC.

Table 4-12: **Pin Mapping for x4 DDR3 DIMMs**

Memory Data Sheet	DDR3 SDRAM XDC
DQ[63:0]	DQ[63:0]
CB3 to CB0	DQ[67:64]
CB7 to CB4	DQ[71:68]
DQS0, $\overline{\text{DQS0}}$	DQS[0], DQS_N[0]
DQS1, $\overline{\text{DQS1}}$	DQS[2], DQS_N[2]
DQS2, $\overline{\text{DQS2}}$	DQS[4], DQS_N[4]
DQS3, $\overline{\text{DQS3}}$	DQS[6], DQS_N[6]
DQS4, $\overline{\text{DQS4}}$	DQS[8], DQS_N[8]
DQS5, $\overline{\text{DQS5}}$	DQS[10], DQS_N[10]
DQS6, $\overline{\text{DQS6}}$	DQS[12], DQS_N[12]
DQS7, $\overline{\text{DQS7}}$	DQS[14], DQS_N[14]
DQS8, $\overline{\text{DQS8}}$	DQS[16], DQS_N[16]
DQS9, $\overline{\text{DQS9}}$	DQS[1], DQS_N[1]
DQS10, $\overline{\text{DQS10}}$	DQS[3], DQS_N[3]
DQS11, $\overline{\text{DQS11}}$	DQS[5], DQS_N[5]
DQS12, $\overline{\text{DQS12}}$	DQS[7], DQS_N[7]
DQS13, $\overline{\text{DQS13}}$	DQS[9], DQS_N[9]
DQS14, $\overline{\text{DQS14}}$	DQS[11], DQS_N[11]
DQS15, $\overline{\text{DQS15}}$	DQS[13], DQS_N[13]

Table 4-12: **Pin Mapping for x4 DDR3 DIMMs *(Cont'd)***

Memory Data Sheet	DDR3 SDRAM XDC
DQS16, $\overline{\text{DQS16}}$	DQS[15], DQS_N[15]
DQS17, $\overline{\text{DQS17}}$	DQS[17], DQS_N[17]

Table 4-13 is an example showing the pin mapping for x4 DDR4 registered DIMMs between the memory data sheet and the XDC.

Table 4-13: **Pin Mapping for x4 DDR4 DIMMs**

Memory Data Sheet	DDR4 SDRAM XDC
DQ[63:0]	DQ[63:0]
CB3 to CB0	DQ[67:64]
CB7 to CB4	DQ[71:68]
DQS0	DQS[0]
DQS1	DQS[2]
DQS2	DQS[4]
DQS3	DQS[6]
DQS4	DQS[8]
DQS5	DQS[10]
DQS6	DQS[12]
DQS7	DQS[14]
DQS8	DQS[16]
DQS9	DQS[1]
DQS10	DQS[3]
DQS11	DQS[5]
DQS12	DQS[7]
DQS13	DQS[9]
DQS14	DQS[11]
DQS15	DQS[13]
DQS16	DQS[15]
DQS17	DQS[17]

Protocol Description

This core has the following interfaces:

- User Interface
- AXI4 Slave Interface
- PHY Only Interface

User Interface

The user interface signals are described in Table 4-14 and connects to an FPGA user design to allow access to an external memory device. The user interface is layered on top of the native interface which is described earlier in the controller description.

Table 4-14: **User Interface**

Signal	I/O	Description
app_addr[APP_ADDR_WIDTH – 1:0]	I	This input indicates the address for the current request.
app_cmd[2:0]	I	This input selects the command for the current request.
app_autoprecharge(1)	I	This input instructs the controller to set the A10 autoprecharge bit on the DRAM CAS command for the current request.
app_en	I	This is the active-High strobe for the app_addr[], app_cmd[2:0], and app_hi_pri inputs.
app_rdy	O	This output indicates that the user interface is ready to accept commands. If the signal is deasserted when app_en is enabled, the current app_cmd, app_autoprecharge, and app_addr must be retried until app_rdy is asserted.
app_hi_pri	I	This input is reserved and should be tied to 0.
app_rd_data [APP_DATA_WIDTH – 1:0]	O	This provides the output data from read commands.
app_rd_data_end	O	This active-High output indicates that the current clock cycle is the last cycle of output data on app_rd_data[].
app_rd_data_valid	O	This active-High output indicates that app_rd_data[] is valid.
app_wdf_data [APP_DATA_WIDTH – 1:0]	I	This provides the data for write commands.
app_wdf_end	I	This active-High input indicates that the current clock cycle is the last cycle of input data on app_wdf_data[].
app_wdf_mask [APP_MASK_WIDTH – 1:0]	I	This provides the mask for app_wdf_data[]. For DDR3 interface, app_wdf_mask port appears in the Data Mask enabled option in Vivado IDE. For DDR4 interface, app_wdf_mask port appears in the "ECC" Vivado IDE option values of TRUE. For "ECC" Vivado IDE option values of FALSE, the port appears for DM_NO_DBI and DM_DBI_RD.

Table 4-14: **User Interface *(Cont'd)***

Signal	I/O	Description
app_wdf_rdy	O	This output indicates that the write data FIFO is ready to receive data. Write data is accepted when app_wdf_rdy = 1'b1 and app_wdf_wren = 1'b1.
app_wdf_wren	I	This is the active-High strobe for app_wdf_data[].
app_ref_req(2)	I	User refresh request.
app_ref_ack(2)	O	User refresh request completed.
app_zq_req(2)	I	User ZQCS command request.
app_zq_ack(2)	O	User ZQCS command request completed.
ui_clk	O	This user interface clock must be one quarter of the DRAM clock.
init_calib_complete	O	PHY asserts init_calib_complete when calibration is finished.
ui_clk_sync_rst	O	This is the active-High user interface reset.
addn_ui_clkout1	O	Additional clock outputs provided based on user requirement.
addn_ui_clkout2	O	Additional clock outputs provided based on user requirement.
addn_ui_clkout3	O	Additional clock outputs provided based on user requirement.
addn_ui_clkout4	O	Additional clock outputs provided based on user requirement.
dbg_clk	O	Debug Clock. Do not connect any signals to dbg_clk and keep the port open during instantiation.
sl_iport0	I [36:0]	Input Port 0 (* KEEP = "true" *)
sl_oport0	O [16:0]	Output Port 0 (* KEEP = "true" *)
c0_ddr4_app_correct_en_i	I	DDR4 Correct Enable Input

Notes:

1. This port appears when "Enable Precharge Input" option is enabled in the Vivado IDE.
2. These ports appear upon enabling "Enable User Refresh and ZQCS Input" option in the Vivado IDE.

app_addr[APP_ADDR_WIDTH – 1:0]

This input indicates the address for the request currently being submitted to the user interface. The user interface aggregates all the address fields of the external SDRAM and presents a flat address space.

The MEM_ADDR_ORDER parameter determines how **`app_addr`** is mapped to the SDRAM address bus and chip select pins. This mapping can have a significant impact on memory bandwidth utilization. "ROW_COLUMN_BANK" is the recommended MEM_ADDR_ORDER setting. Table 4-15 through Table 4-24 show the "ROW_COLUMN_BANK" mapping for DDR3 and DDR4 with examples. Note that the three LSBs of **`app_addr`** map to the column address LSBs which correspond to SDRAM burst ordering.

The controller does not support burst ordering so these low order bits are ignored, making the effective minimum `app_addr` step size hex 8.

Table 4-15: **DDR3 "ROW_COLUMN_BANK" Mapping**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[BANK_WIDTH + COL_WIDTH + ROW_WIDTH +: RANK_WIDTH]
Row	app_addr[BANK_WIDTH + COL_WIDTH +: ROW_WIDTH]
Column	app_addr[3 + BANK_WIDTH +: COL_WIDTH – 3], app_addr[2:0]
Bank	app_addr[3 +: BANK_WIDTH – 1], app_addr[2 + BANK_WIDTH +: 1]

Table 4-16: **DDR3 4 GB (512 MB x8) Single-Rank Mapping Example**

SDRAM Bus	Row[15:0]	Column[9:0]	Bank[2:0]
app_addr Bits	28 through 13	12 through 6, and 2, 1, 0	4, 3, 5

Table 4-17: **DDR4 "ROW_COLUMN_BANK" Mapping**

SDRAM	app_addr Mapping
Rank	(RANKS == 1) ? 1'b0: (S_HEIGHT == 1) ? app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH]: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH + LR_WIDTH +: RANK_WIDTH]
Logical Rank (3DS)	(S_HEIGHT==1) ? 1'b0: app_addr[BANK_GROUP_WIDTH + BANK_WIDTH + COL_WIDTH + ROW_WIDTH +: LR_WIDTH]
Row	app_addr[BANK_GROUP_WIDTH + BANK_WIDTH + COL_WIDTH +: ROW_WIDTH
Column	app_addr[3 + BANK_GROUP_WIDTH + BANK_WIDTH +: COL_WIDTH – 3], app_addr[2:0]
Bank	app_addr[3 + BANK_GROUP_WIDTH +: BANK_WIDTH
Bank Group	app_addr[3 +: BANK_GROUP_WIDTH]

Table 4-18: **DDR3 "BANK_ROW_COLUMN" Mapping**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[BANK_WIDTH + COL_WIDTH + ROW_WIDTH +: RANK_WIDTH]
Row	app_addr[COL_WIDTH +: ROW_WIDTH]
Column	app_addr[0 +: COL_WIDTH]
Bank	app_addr[COL_WIDTH + ROW_WIDTH +: BANK_WIDTH]

Table 4-19: **DDR3 4 GB (512 MB x8) Single-Rank Mapping Example**

SDRAM Bus	Row[15:0]	Column[9:0]	Bank[2:0]
app_addr Bits	25 through 10	9 through 0	28 through 26

Note: The "+:" notation indicates indexed vector part select. For example, for the given input signal input [511:0] app_addr;
app_addr[255+:64] means bits 255 to 319 of app_addr are considered.

Table 4-20: **DDR4 "BANK_ROW_COLUMN" Mapping**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: (S_HEIGHT == 1) ? app_addr[COL_WIDTH + ROW_WIDTH+ BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH]: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH + LR_WIDTH +: RANK_WIDTH]
Logical Rank (3DS)	(S_HEIGHT == 1) ? 1'b0: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: LR_WIDTH]
Row	app_addr[COL_WIDTH +: ROW_WIDTH]
Column	app_addr[0 +: COL_WIDTH]
Group	app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH +: BANK_GROUP_WIDTH]
Bank	app_addr[COL_WIDTH + ROW_WIDTH +: BANK_WIDTH]

Table 4-21: **DDR3 "ROW_BANK_COLUMN" Mapping**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[COL_WIDTH +BANK_WIDTH + ROW_WIDTH +: RANK_WIDTH]
Row	app_addr[COL_WIDTH + BANK_WIDTH +: ROW_WIDTH]
Column	app_addr[0 +: COL_WIDTH]
Bank	app_addr[COL_WIDTH +: BANK_WIDTH]

Table 4-22: **DDR3 4 GB (512 MB x8) Single-Rank Mapping Example**

SDRAM Bus	Row[15:0]	Column[9:0]	Bank[2:0]
app_addr Bits	28 through 13	9 through 0	12 through 10

Table 4-23: **DDR4 "ROW_BANK_COLUMN" Mapping**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: (S_HEIGHT == 1) ? app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH + LR_WIDTH +: RANK_WIDTH]
Logical Rank (3DS)	(S_HEIGHT == 1) ? 1'b0: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: LR_WIDTH]
Row	app_addr[COL_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: ROW_WIDTH]

Table 4-23: **DDR4 "ROW_BANK_COLUMN" Mapping *(Cont'd)***

SDRAM	app_addr Mapping
Column	app_addr[0 +: COL_WIDTH]
Group	app_addr[COL_WIDTH + BANK_WIDTH +: BANK_GROUP_WIDTH]
Bank	app_addr[COL_WIDTH +: BANK_WIDTH]

Table 4-24: **DDR4 4 GB (512 MB x8) Single-Rank Mapping Example**

SDRAM Bus	Row[14:0]	Column[9:0]	Bank[1:0]	Bank Group[1:0]
app_addr Bits	28 through 14	13 through 7, and 2, 1, 0	6, 5	4, 3

The "ROW_COLUMN_BANK" setting maps **`app_addr[4:3]`** to the DDR4 bank group bits or DDR3 bank bits used by the controller to interleave between its group FSMs. The lower order address bits equal to **`app_addr[5]`** and above map to the remaining SDRAM bank and column address bits. The highest order address bits map to the SDRAM row. This mapping is ideal for workloads that have address streams that increment linearly by a constant step size of hex 8 for long periods. With this configuration and workload, transactions sent to the user interface are evenly interleaved across the controller group FSMs, making the best use of the controller resources.

In addition, this arrangement tends to generate hits to open pages in the SDRAM. The combination of group FSM interleaving and SDRAM page hits results in very high SDRAM data bus utilization.

Address streams other than the simple increment pattern tend to have lower SDRAM bus utilization. You can recover this performance loss by tuning the mapping of your design flat address space to the **`app_addr`** input port of the user interface. If you have knowledge of your address sequence, you can add logic to map your address bits with the highest toggle rate to the lowest **`app_addr`** bits, starting with **`app_addr[3]`** and working up from there.

For example, if you know that your workload address Bits[4:3] toggle much less than Bits[10:9], which toggle at the highest rate, you could add logic to swap these bits so that your address Bits[10:9] map to **`app_addr[4:3]`**. The result is an improvement in how the address stream interleaves across the controller group FSMs, resulting in better controller throughput and higher SDRAM data bus utilization.

Table 4-25 through Table 4-26 show the "ROW_COLUMN_LRANK_BANK" and "ROW_LRANK_COLUMN_BANK" mappings for DDR4 with 3DS examples.

Table 4-25: **DDR4 ROW_COLUMN_LRANK_BANK**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[ROW_WIDTH + COL_WIDTH + LR_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH]
Logical_rank	app_addr[3 + BANK_WIDTH + BANK_GROUP_WIDTH +: LR_WIDTH]

Table 4-25: **DDR4 ROW_COLUMN_LRANK_BANK *(Cont'd)***

SDRAM	app_addr Mapping
Row	app_addr[COL_WIDTH + LR_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: ROW_WIDTH]
Column	app_addr[3 + LR_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: COL_WIDTH – 3], app_addr[2:0]
Bank	app_addr[3 + BANK_GROUP_WIDTH +: BANK_WIDTH]
Group	app_addr[3 +: BANK_GROUP_WIDTH]

Table 4-26: **DDR4 ROW_LRANK_COLUMN_BANK**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[ROW_WIDTH + LR_WIDTH + COL_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH]
Logical_rank	app_addr[COL_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: LR_WIDTH]
Row	app_addr[LR_WIDTH + COL_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: ROW_WIDTH]
Column	app_addr[3 + BANK_WIDTH + BANK_GROUP_WIDTH +: COL_WIDTH – 3], app_addr[2:0]
Bank	app_addr[3 + BANK_GROUP_WIDTH +: BANK_WIDTH]
Group	app_addr[3 +: BANK_GROUP_WIDTH]

The ROW_COLUMN_BANK_INTLV is a mapping option that swaps a column and bank bit. With this option, a sequential address stream maps the first eight transactions across four banks instead of eight banks. Then the next eight transactions map to the next four banks, and so on. This helps with the performance when there are short bursts of sequential addresses instead of very long bursts.

Table 4-27 through Table 4-32 show the "ROW_COLUMN_BANK_INTLV" mapping for DDR3 and DDR4 with examples.

Table 4-27: **DDR3 ROW_COLUMN_BANK_INTLV**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH +: RANK_WIDTH]
Row	app_addr[BANK_WIDTH + COL_WIDTH +: ROW_WIDTH]
Column	app_addr[3 + BANK_WIDTH + 1 +: COL_WIDTH – 4], app_addr[3 + BANK_WIDTH – 1 +: 1], app_addr[2:0]
Bank	app_addr[3 +: BANK_WIDTH – 1], app_addr[3 + BANK_WIDTH +: 1]

Table 4-28: **DDR3 ROW_COLUMN_BANK_INTLV**

SDRAM Bus	Row[15:0]	Column[9:0]	Bank[2:0]
app_addr Bits	28 through 13	12 through 7, 5, and 2, 1, 0	4, 3, 6

Table 4-29: **DDR4 (x16) ROW_COLUMN_BANK_INTLV**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH
Row	app_addr[BANK_GROUP_WIDTH + BANK_WIDTH + COL_WIDTH +: ROW_WIDTH
Column	app_addr[3 + BANK_GROUP_WIDTH + BANK_WIDTH + 1 +: COL_WIDTH – 4], app_addr[3 + BANK_GROUP_WIDTH + 1 +: 1, app_addr[2:0]
Bank	app_addr[3 + BANK_GROUP_WIDTH + 2 +: BANK_WIDTH – 1], app_addr[3 + BANK_GROUP_WIDTH +: BANK_WIDTH – 1]
Bank Group	app_addr[3 +: BANK_GROUP_WIDTH]

Table 4-30: **DDR4 4 GB (256 MB x16) Single-Rank Mapping Example for ROW_COLUMN_BANK_INTLV**

SDRAM Bus	Row[14:0]	Column[9:0]	Bank[1:0]	Bank Group
app_addr Bits	27 through 13	12 through 7, 5, and 2, 1, 0	6, 4	3

Table 4-31: **DDR4 (x4, x8) ROW_COLUMN_BANK_INTLV**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0: app_addr[COL_WIDTH + ROW_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH +: RANK_WIDTH
Row	app_addr[BANK_GROUP_WIDTH + BANK_WIDTH + COL_WIDTH +: ROW_WIDTH
Column	app_addr[3 + BANK_GROUP_WIDTH + BANK_WIDTH + 1 +: COL_WIDTH – 4], app_addr[3 + BANK_GROUP_WIDTH +: 1, app_addr[2:0]
Bank	app_addr[3 + BANK_GROUP_WIDTH + 1 +: BANK_WIDTH]
Bank Group	app_addr[3 +: BANK_GROUP_WIDTH]

Table 4-32: **DDR4 4 GB (512 MB x8) Single-Rank Mapping Example for ROW_COLUMN_BANK_INTLV**

SDRAM Bus	Row[14:0]	Column[9:0]	Bank[1:0]	Bank Group[1:0]
app_addr Bits	28 through 14	13 through 8, 5, and 2, 1, 0	7, 6	4, 3

app_cmd[2:0]

This input specifies the command for the request currently being submitted to the user interface. The available commands are shown in Table 4-33. With ECC enabled, the **`wr_bytes`** operation is required for writes with any non-zero **`app_wdf_mask`** bits. The

`wr_bytes` triggers a read-modify-write flow in the controller, which is needed only for writes with masked data in ECC mode.

Table 4-33: **Commands for app_cmd[2:0]**

Operation	app_cmd[2:0] Code
Write	000
Read	001
wr_bytes	011

app_autoprecharge

This input specifies the state of the A10 **`autoprecharge`** bit for the DRAM CAS command for the request currently being submitted to the user interface. When this input is Low, the Memory Controller issues a DRAM RD or WR CAS command. When this input is High, the controller issues a DRAM RDA or WRA CAS command. This input provides per request control, but can also be tied off to configure the controller statically for open or closed page mode operation. The Memory Controller also has an option to automatically determine when to issue an AutoPrecharge. This option disables the **`app_autoprecharge`** input. For more information on the automatic mode, see Performance, page 189.

app_en

This input strobes in a request. Apply the desired values to **`app_addr`**[], **`app_cmd`**[2:0], and **`app_hi_pri`**, and then assert **`app_en`** to submit the request to the user interface. This initiates a handshake that the user interface acknowledges by asserting **`app_rdy`**.

app_wdf_data[APP_DATA_WIDTH – 1:0]

This bus provides the data currently being written to the external memory. APP_DATA_WIDTH is 2 × nCK_PER_CLK × DQ_WIDTH when ECC is disabled (ECC parameter value is OFF) and 2 × nCK_PER_CLK × (DQ_WIDTH – ECC_WIDTH) when ECC is enabled (ECC parameter is ON).

PAYLOAD_WIDTH indicates the effective DQ_WIDTH on which the user interface data has been transfered.

PAYLOAD_WIDTH is DQ_WIDTH when ECC is disabled (ECC parameter value is OFF).

PAYLOAD_WIDTH is (DQ_WIDTH – ECC_WIDTH) when ECC is enabled (ECC parameter is ON).

app_wdf_end

This input indicates that the data on the **`app_wdf_data[]`** bus in the current cycle is the last data for the current request.

app_wdf_mask[APP_MASK_WIDTH – 1:0]

This bus indicates which bits of **`app_wdf_data`**[] are written to the external memory and which bits remain in their current state. APP_MASK_WIDTH is APP_DATA_WIDTH/8.

app_wdf_wren

This input indicates that the data on the **`app_wdf_data`**[] bus is valid.

app_rdy

This output indicates whether the request currently being submitted to the user interface is accepted. If the user interface does not assert this signal after **`app_en`** is asserted, the current request must be retried. The **`app_rdy`** output is not asserted if:

- PHY/Memory initialization is not yet completed.
- All the controller Group FSMs are occupied (can be viewed as the command buffer being full).
  - A read is requested and the read buffer is full.
  - A write is requested and no write buffer pointers are available.
- A periodic read is being inserted.

app_rd_data[APP_DATA_WIDTH – 1:0]

This output contains the data read from the external memory.

app_rd_data_end

This output indicates that the data on the **`app_rd_data`**[] bus in the current cycle is the last data for the current request.

app_rd_data_valid

This output indicates that the data on the **`app_rd_data`**[] bus is valid.

app_wdf_rdy

This output indicates that the write data FIFO is ready to receive data. Write data is accepted when both **`app_wdf_rdy`** and **`app_wdf_wren`** are asserted.

app_ref_req

When asserted, this active-High input requests that the Memory Controller send a refresh command to the DRAM. It must be pulsed for a single cycle to make the request and then deasserted at least until the **`app_ref_ack`** signal is asserted to acknowledge the request and indicate that it has been sent.

app_ref_ack

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

app_zq_req

When asserted, this active-High input requests that the Memory Controller send a ZQ calibration command to the DRAM. It must be pulsed for a single cycle to make the request and then deasserted at least until the **`app_zq_ack`** signal is asserted to acknowledge the request and indicate that it has been sent.

app_zq_ack

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

ui_clk_sync_rst

This is the reset from the user interface which is in synchronous with **`ui_clk`**.

ui_clk

This is the output clock from the user interface. It must be a quarter the frequency of the clock going out to the external SDRAM, which depends on 4:1 mode selected in Vivado IDE.

init_calib_complete

PHY asserts **`init_calib_complete`** when calibration is finished. The application has no need to wait for **`init_calib_complete`** before sending commands to the Memory Controller.

Command Path

When the user logic **`app_en`** signal is asserted and the **`app_rdy`** signal is asserted from the user interface, a command is accepted and written to the FIFO by the user interface. The command is ignored by the user interface whenever **`app_rdy`** is deasserted. The user logic needs to hold **`app_en`** High along with the valid command, autoprecharge, and address values until **`app_rdy`** is asserted as shown for the "write with autoprecharge" transaction in Figure 4-2.

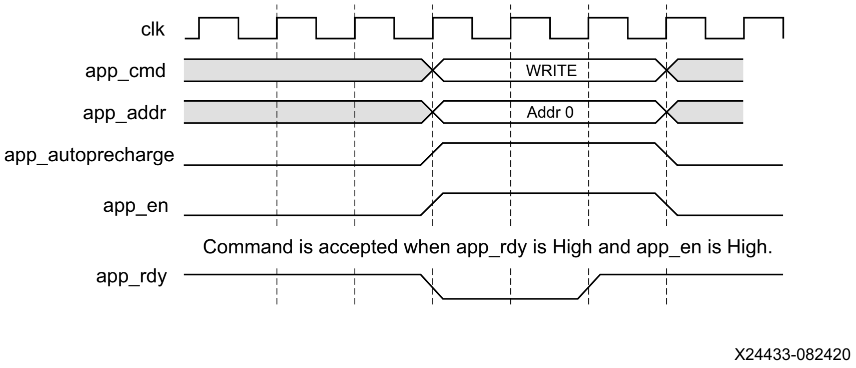


Figure 4-2: **User Interface Command Timing Diagram with app_rdy Asserted**

A non back-to-back write command can be issued as shown in Figure 4-3. This figure depicts three scenarios for the **`app_wdf_data`**, **`app_wdf_wren`**, and **`app_wdf_end`** signals as follows:

1. Write data is presented along with the corresponding write command.
2. Write data is presented before the corresponding write command.
3. Write data is presented after the corresponding write command, but should not exceed the limitation of two clock cycles.

For write data that is output after the write command has been registered, as shown in Note 3 (Figure 4-3), the maximum delay is two clock cycles.

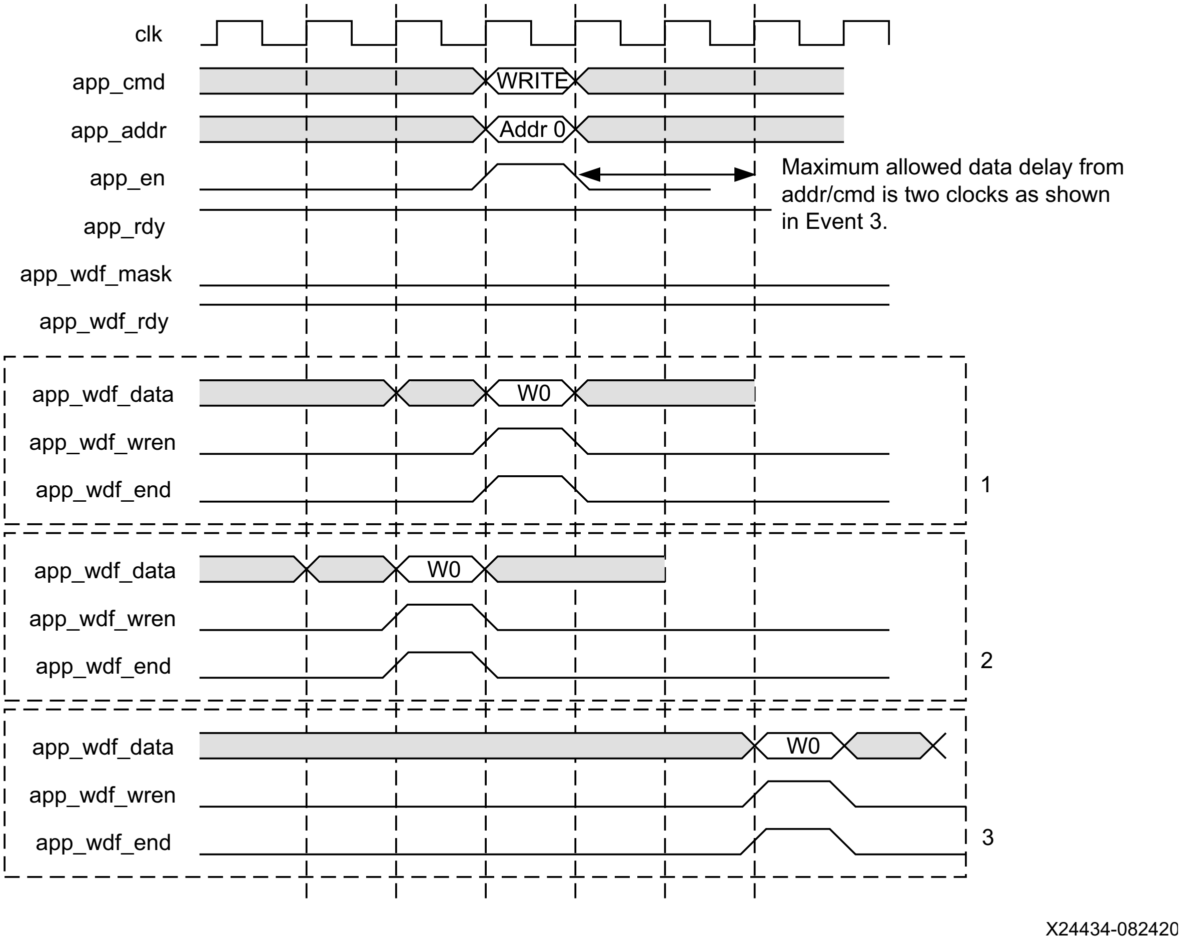


Figure 4-3: **4:1 Mode User Interface Write Timing Diagram (Memory Burst Type = BL8)**

Write Path

The write data is registered in the write FIFO when **`app_wdf_wren`** is asserted and **`app_wdf_rdy`** is High (Figure 4-4). If **`app_wdf_rdy`** is deasserted, the user logic needs to hold **`app_wdf_wren`** and **`app_wdf_end`** High along with the valid **`app_wdf_data`** value until **`app_wdf_rdy`** is asserted. The **`app_wdf_mask`** signal can be used to mask out the bytes to write to external memory.

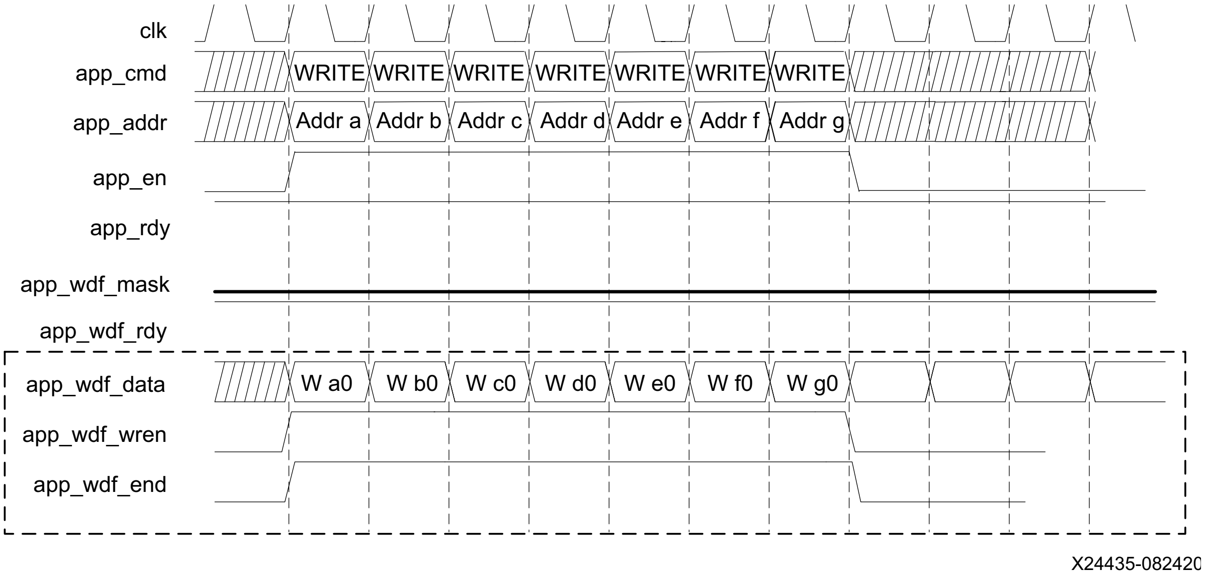


Figure 4-4: **4:1 Mode User Interface Back-to-Back Write Commands Timing Diagram (Memory Burst Type = BL8)**

The timing requirement for **`app_wdf_data`**, **`app_wdf_wren`**, and **`app_wdf_end`** relative to their associated write command is the same for back-to-back writes as it is for single writes, as shown in Figure 4-3.

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

For a 4:1 Memory Controller to DRAM clock ratio with an 8-bit memory, at the application interface, if the 64-bit data driven is **`0000_0806_0000_0805`** (Hex), the data at the DRAM interface is as shown in Figure 4-5. This is for a BL8 (Burst Length 8) transaction.

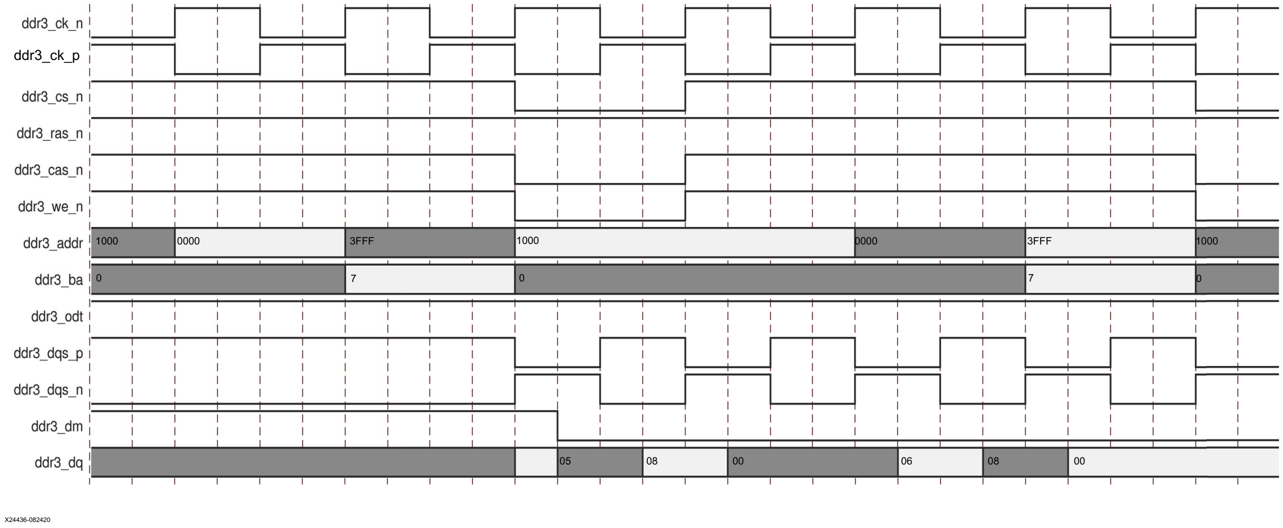


Figure 4-5: **Data at the DRAM Interface for 4:1 Mode**

The data values at different clock edges are as shown in Table 4-34.

Table 4-34: **Data Values at Different Clock Edges**

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

Table 4-35 shows a generalized representation of how DRAM DQ bus data is concatenated to form application interface data signals. `app_wdf_data` is shown in Table 4-35, but the table applies equally to `app_rd_data`. Each byte of the DQ bus has eight bursts, Rise0 (burst 0) through Fall3 (burst 7) as shown previously in Table 4-34, for a total of 64 data bits. When concatenated with Rise0 in the LSB position and Fall3 in the MSB position, a 64-bit chunk of the `app_wdf_data` signal is formed.

For example, the eight bursts of `ddr3_dq[7:0]` corresponds to DQ bus byte 0, and when concatenated as described here, they map to `app_wdf_data[63:0]`. To be clear on the concatenation order, `ddr3_dq[0]` from Rise0 (burst 0) maps to `app_wdf_data[0]`, and `ddr3_dq[7]` from Fall3 (burst 7) maps to `app_wdf_data[63]`. The table shows a second example, mapping DQ byte 1 to `app_wdf_data[127:64]`, as well as the formula for DQ byte N.

Table 4-35: **DRAM DQ Bus Data Map**

DQ Bus Byte	App Interface Signal	DDR Bus Signal at Each BL8 Burst Position				
		Fall3	...	Rise1	Fall0	Rise0
N	app_wdf_data[(N + 1) × 64 – 1: N × 64]	ddr3_dq[(N + 1) × 8 – 1:N × 8]	...	ddr3_dq[(N + 1) × 8 – 1:N × 8]	ddr3_dq[(N + 1) × 8 – 1:N × 8]	ddr3_dq[(N + 1) × 8 – 1:N × 8]
1	app_wdf_data[127:64]	ddr3_dq[15:8]	...	ddr3_dq[15:8]	ddr3_dq[15:8]	ddr3_dq[15:8]
0	app_wdf_data[63:0]	ddr3_dq[7:0]	...	ddr3_dq[7:0]	ddr3_dq[7:0]	ddr3_dq[7:0]

In a similar manner to the DQ bus mapping, the DM bus maps to `app_wdf_mask` by concatenating the DM bits in the same burst order. Example for the first two bytes of the DRAM bus are shown in Table 4-36, and the formula for mapping DM for byte N is also given.

Table 4-36: **DRAM DM Bus Data Map**

DM Bus Byte	App Interface Signal	DDR Bus Signal at Each BL8 Burst Position				
		Fall3	...	Rise1	Fall0	Rise0
N	app_wdf_mask[(N + 1) × 8 – 1:N × 8]	ddr3_dm[N]	...	ddr3_dm[N]	ddr3_dm[N]	ddr3_dm[N]
1	app_wdf_mask[15:0]	ddr3_dm[1]	...	ddr3_dm[1]	ddr3_dm[1]	ddr3_dm[1]
0	app_wdf_mask[7:0]	ddr3_dm[0]	...	ddr3_dm[0]	ddr3_dm[0]	ddr3_dm[0]

Read Path

The read data is returned by the user interface in the requested order and is valid when **`app_rd_data_valid`** is asserted (Figure 4-6 and Figure 4-7). The **`app_rd_data_end`** signal indicates the end of each read command burst and is not needed in user logic.

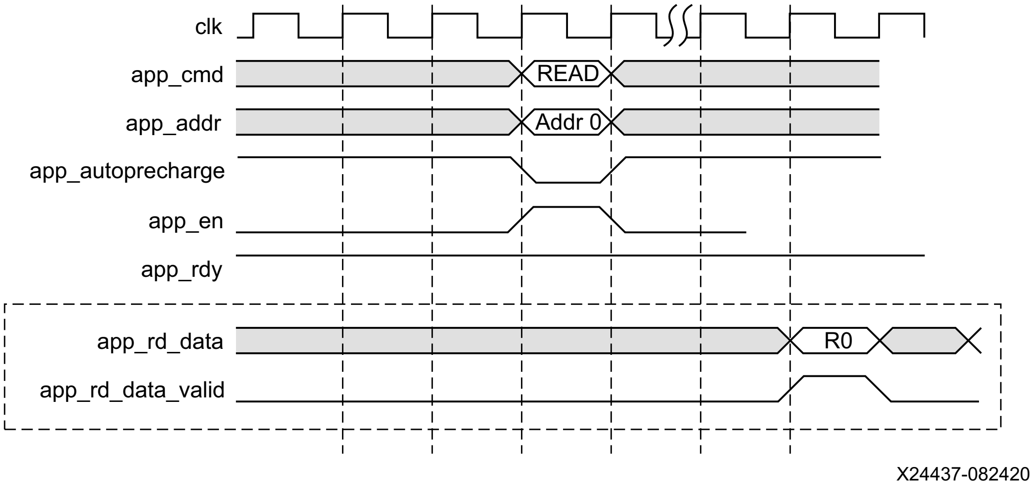


Figure 4-6: **4:1 Mode User Interface Read Timing Diagram (Memory Burst Type = BL8) #1**

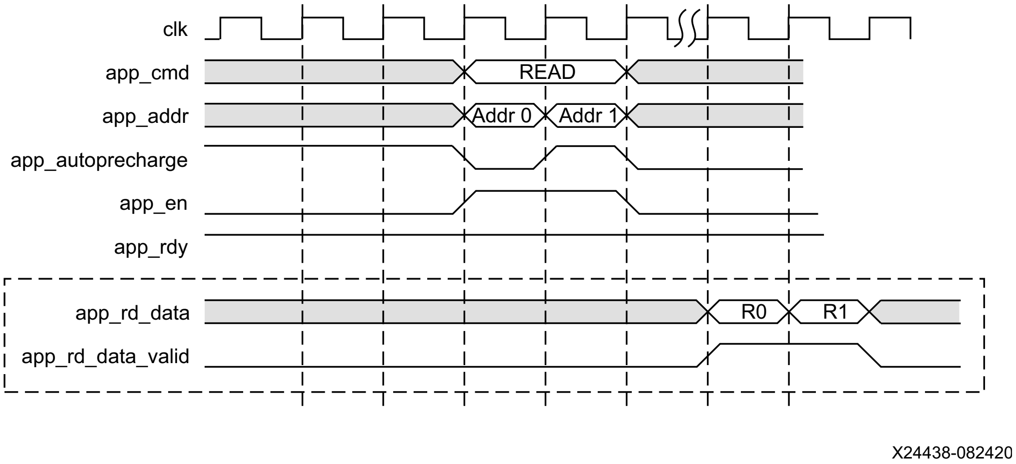


Figure 4-7: **4:1 Mode User Interface Read Timing Diagram (Memory Burst Type = BL8) #2**

In Figure 4-7, the read data returned is always in the same order as the requests made on the address/control bus.

Maintenance Commands

The UI can be configured by the Vivado IDE to enable two DRAM Refresh modes. The default mode configures the UI and the Memory Controller to automatically generate DRAM Refresh and ZQCS commands, meeting all DRAM protocol and timing requirements. The controller interrupts normal system traffic on a regular basis to issue these maintenance commands on the DRAM bus.

The User mode is enabled by checking the **Enable User Refresh and ZQCS Input** option in the Vivado IDE. In this mode, you are responsible for issuing Refresh and ZQCS commands at the rate required by the DRAM component specification after **`init_calib_complete`** asserts High. You use the **`app_ref_req`** and **`app_zq_req`** signals on the UI to request Refresh and ZQCS commands, and monitor **`app_ref_ack`** and **`app_zq_ack`** to know when the commands have completed. The controller manages all DRAM timing and protocol for these commands, other than the overall Refresh or ZQCS rate, just as it does for the default DRAM Refresh mode. These **`request/ack`** ports operate independently of the other UI command ports, like **`app_cmd`** and **`app_en`**.

The controller might not preserve the exact ordering of maintenance transactions presented to the UI on relative to regular read and write transactions. When you request a Refresh or ZQCS, the controller interrupts system traffic, just as in the default mode, and inserts the maintenance commands. To take the best advantage of this mode, you should request maintenance commands when the controller is idle or at least not very busy, keeping in mind that the DRAM Refresh rate and ZQCS rate requirements cannot be violated.

Figure 4-8 shows how the User mode ports are used and how they affect the DRAM command bus. This diagram shows the general idea about this mode of operation and is not timing accurate. Assuming the DRAM is idle with all banks closed, a short time after **`app_ref_req`** or **`app_zq_req`** are asserted High for one system clock cycle, the controller issues the requested commands on the DRAM command bus. The **`app_ref_req`** and **`app_zq_req`** can be asserted on the same cycle or different cycles, and they do not have to be asserted at the same rate. After a request signal is asserted High for one system clock, you must keep it deasserted until the acknowledge signal asserts.

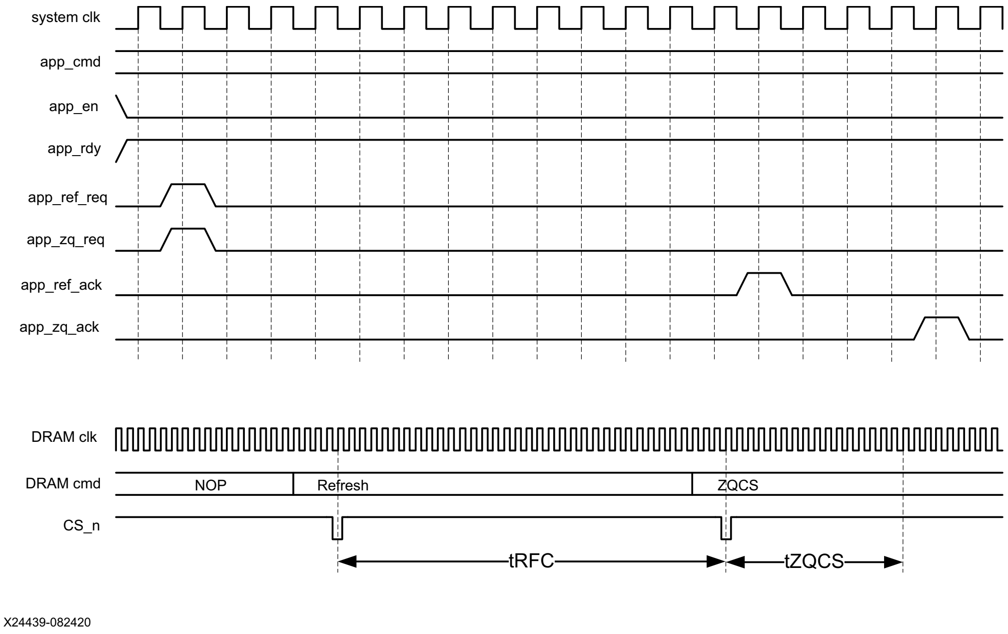


Figure 4-8: **User Mode Ports on DRAM Command Bus Timing Diagram**

Figure 4-9 shows a case where the **`app_en`** is asserted and read transactions are presented continuously to the UI when the **`app_ref_req`** and **`app_zq_req`** are asserted. The controller interrupts the DRAM traffic following DRAM protocol and timing requirements, issues the Refresh and ZQCS, and then continues issuing the read transactions. Note that the **`app_rdy`** signal deasserts during this sequence. It is likely to deassert during a sequence like this since the controller command queue can easily fill up during tRFC or tZQCS. After the maintenance commands are issued and normal traffic resumes on the bus, the **`app_rdy`** signal asserts and new transactions are accepted again into the controller.

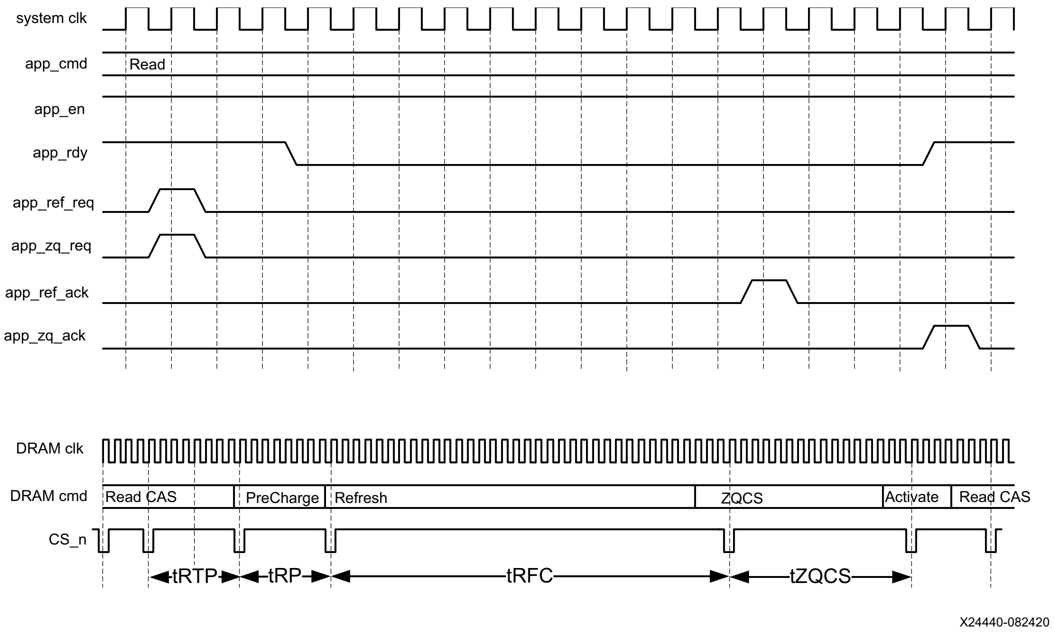


Figure 4-9: **Read Transaction on User Interface Timing Diagram**

Figure 4-9 shows the operation for a single-rank. In a multi-rank system, a single refresh request generates a DRAM Refresh command to each rank, in series, staggered by tRFC/2. The Refresh commands are staggered since they are relatively high power consumption operations. A ZQCS command request generates a ZQCS command to all ranks in parallel.

AXI4 Slave Interface

The AXI4 slave interface block maps AXI4 transactions to the UI to provide an industry-standard bus protocol interface to the Memory Controller. The AXI4 slave interface is optional in designs provided through the DDR3/DDR4 SDRAM tool. The RTL is consistent between both tools. For details on the AXI4 signaling protocol, see the Arm AMBA specifications [Ref 12].

The overall design is composed of separate blocks to handle each AXI channel, which allows for independent read and write transactions. Read and write commands to the UI rely on a simple round-robin arbiter to handle simultaneous requests.

The address read/address write modules are responsible for chopping the AXI4 incr/wrap requests into smaller memory size burst lengths of either four or eight, and also conveying the smaller burst lengths to the read/write data modules so they can interact with the user interface. Fixed burst type is not supported.

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

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

AXI4 Slave Interface Parameters

Table 4-37 lists the AXI4 slave interface parameters.

Table 4-37: **AXI4 Slave Interface Parameters**

Parameter Name	Allowable Values	Description
C_S_AXI_ADDR_WIDTH	DDR3: 25–35 DDR4: 27–37	This is the width of address read and address write signals. It depends on memory density and the configuration selected. It is calculated as: For DDR3: log2(RANKS) + ROW_WIDTH + COL_WIDTH + BANK_WIDTH + log2(PAYLOAD_WIDTH) – 3 For DDR4: log2(RANKS) + ROW_WIDTH + COL_WIDTH + BANK_WIDTH + BANK_GROUP_WIDTH + log2(PAYLOAD_WIDTH) – 3 PAYLOAD_WIDTH: This is the data width of the external memory interface which is limited to 8, 16, 32, or 64 for AXI designs.
C_S_AXI_DATA_WIDTH	32, 64, 128, 256, 512	This is the width of data signals. Width of APP_DATA_WIDTH is recommended for better performance. Using a smaller width invokes an Upsizer, which would spend clocks in packing the data.
C_S_AXI_ID_WIDTH	1–16	This is the width of ID signals for every channel.

Table 4-37: **AXI4 Slave Interface Parameters *(Cont'd)***

Parameter Name	Allowable Values	Description
C_S_AXI_SUPPORTS_NARROW_BURST	0, 1	This parameter is only applicable when the C_S_AXI_DATA_WIDTH is equal to APP_DATA_WIDTH. When C_S_AXI_DATA_WIDTH is equal to APP_DATA_WIDTH and this parameter is enabled, the AXI slave instantiates an upsizer. When Master sends AXI Narrow transfers (a transfer that is narrower than its data bus), the upsizer packs consecutive transfers to present a single request at the User Interface. Hence if this AXI slave can receive Narrow transfers, the parameter C_S_AXI_SUPPORTS_NARROW_BURST must be enabled. If not, it results in unexpected behavior when the Slave receives Narrow transfers. When C_S_AXI_DATA_WIDTH is equal to APP_DATA_WIDTH and it is known that the AXI slave never received Narrow transfers, you can disable this parameter to avoid the instantiation of upsizer, thus saving implementation area. In this case, ensure that during actual simulation the AXI Slave never receives Narrow transfers. When C_S_AXI_DATA_WIDTH is less than APP_DATA_WIDTH, upsizer is always instantiated and this parameter has no effect.
C_RD_WR_ARB_ALGORITHM	TDM, ROUND_ROBIN, RD_PRI_REG, RD_PRI_REG_STARVE_LIMIT, WRITE_PRIORITY_REG, WRITE_PRIORITY	This parameter indicates the Arbitration algorithm scheme. See Arbitration in AXI Shim, page 143 for more information.
C_ECC	ON, OFF	This parameter specifies if ECC is enabled for the design or not. ECC is always enabled for 72-bit designs and disabled for all other data widths

AXI Addressing

The AXI address from the AXI master is a TRUE byte address. The AXI shim converts the address from the AXI master to the memory based on AXI SIZE and memory data width. The LSBs of the AXI byte address are masked to 0, depending on the data width of the memory array. If the memory array is 64 bits (8 bytes) wide, AXI address[2:0] are ignored and treated as 0. If the memory array is 16 bits (2 bytes) wide, AXI address[0] is ignored and treated as 0. DDR3/DDR4 DRAM is accessed in blocks of DRAM bursts and this memory controller always uses a fixed burst length of 8. The UI Data Width is always eight times the PAYLOAD_WIDTH.

Table 4-38: **AXI Byte Address Mapping**

UI Data Width	Memory Interface Data Width	AXI Byte Address
64	8	AxADDR = app_addr[ADDR_WIDTH-1:0]
128	16	AxADDR = app_addr[ADDR_WIDTH-1:0], 1'b0
256	32	AxADDR = app_addr[ADDR_WIDTH-1:0], 2'b00
512	64	AxADDR = app_addr[ADDR_WIDTH-1:0], 3'b000

AXI4 Slave Interface Signals

Table 4-39 lists the AXI4 slave interface specific signal. **`ui_clk`** and **`ui_clk_sync_rst`** to the interface is provided from the Memory Controller. AXI interface is synchronous to **`ui_clk`**.

Table 4-39: **AXI4 Slave Interface Signals**

Name	Width	I/O	Active State	Description
ui_clk	1	O		Output clock from the core to the interface.
ui_clk_sync_rst	1	O	High	Output reset from the core to the interface.
aresetn	1	I	Low	Input reset to the AXI Shim and it should be in synchronous with FPGA logic clock.
s_axi_awid	C_S_AXI_ID_WIDTH	I		Write address ID
s_axi_awaddr	C_S_AXI_ADDR_WIDTH	I		Write address
s_axi_awlen	8	I		Burst length. The burst length gives the exact number of transfers in a burst.
s_axi_awsize	3	I		Burst size. This signal indicates the size of each transfer in the burst.
s_axi_awburst	2	I		Burst type. Only INCR/WRAP supported.
s_axi_awlock	1	I		Lock type. (This is not used in the current implementation.) ***Note:*** When an unsupported value is selected, awburst defaults to an INCR burst type.
s_axi_awcache	4	I		Cache type. (This is not used in the current implementation.)
s_axi_awprot	3	I		Protection type. (Not used in the current implementation.)
s_axi_awqos	4	I		Quality of service. (Not used in the current implementation.)
s_axi_awvalid	1	I	High	Write address valid. This signal indicates that valid write address and control information are available.
s_axi_awready	1	O	High	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
s_axi_wdata	C_S_AXI_DATA_WIDTH	I		Write data

Table 4-39: **AXI4 Slave Interface Signals *(Cont'd)***

Name	Width	I/O	Active State	Description
s_axi_wstrb	C_S_AXI_DATA_WIDTH/8	I		Write strobes
s_axi_wlast	1	I	High	Write last. This signal indicates the last transfer in a write burst.
s_axi_wvalid	1	I	High	Write valid. This signal indicates that write data and strobe are available.
s_axi_wready	1	O	High	Write ready
s_axi_bid	C_S_AXI_ID_WIDTH	O		Response ID. The identification tag of the write response.
s_axi_bresp	2	O		Write response. This signal indicates the status of the write response.
s_axi_bvalid	1	O	High	Write response valid
s_axi_bready	1	I	High	Response ready
s_axi_arid	C_S_AXI_ID_WIDTH	I		Read address ID
s_axi_araddr	C_S_AXI_ADDR_WIDTH	I		Read address
s_axi_arlen	8	I		Read burst length
s_axi_arsize	3	I		Read burst size
s_axi_arburst	2	I		Read burst type. Only INCR/WRAP supported.
s_axi_arlock	1	I		Lock type. (This is not used in the current implementation.) ***Note:*** When an unsupported value is selected, arburst defaults to an INCR burst type.
s_axi_arcache	4	I		Cache type. (This is not used in the current implementation.)
s_axi_arprot	3	I		Protection type. (This is not used in the current implementation.)
s_axi_arqos	4	I		Quality of service. (Not used in the current implementation.)
s_axi_arvalid	1	I	High	Read address valid
s_axi_arready	1	O	High	Read address ready
s_axi_rid	C_S_AXI_ID_WIDTH	O		Read ID tag
s_axi_rdata	C_S_AXI_DATA_WIDTH	O		Read data
s_axi_rresp	2	O		Read response
s_axi_rlast	1	O		Read last
s_axi_rvalid	1	O		Read valid
s_axi_rready	1	I		Read ready
dbg_clk	1	O		Debug Clock. Do not connect any signals to dbg_clk and keep the port open during instantiation.

AXI4 Slave Interface Transaction Examples

Figure 4-10 shows the write full transfer timing diagram.

Aligned (ADDR A)	AXI data width = 32-bit	AWID = 0	AWADDR = 'h0	AWSIZE = 2	AWLEN = 3	AWBURST = INCR
Unaligned (ADDR B)	AXI data width = 32-bit	AWID = 1	AWADDR = 'h3	AWSIZE = 2	AWLEN = 3	AWBURST = INCR

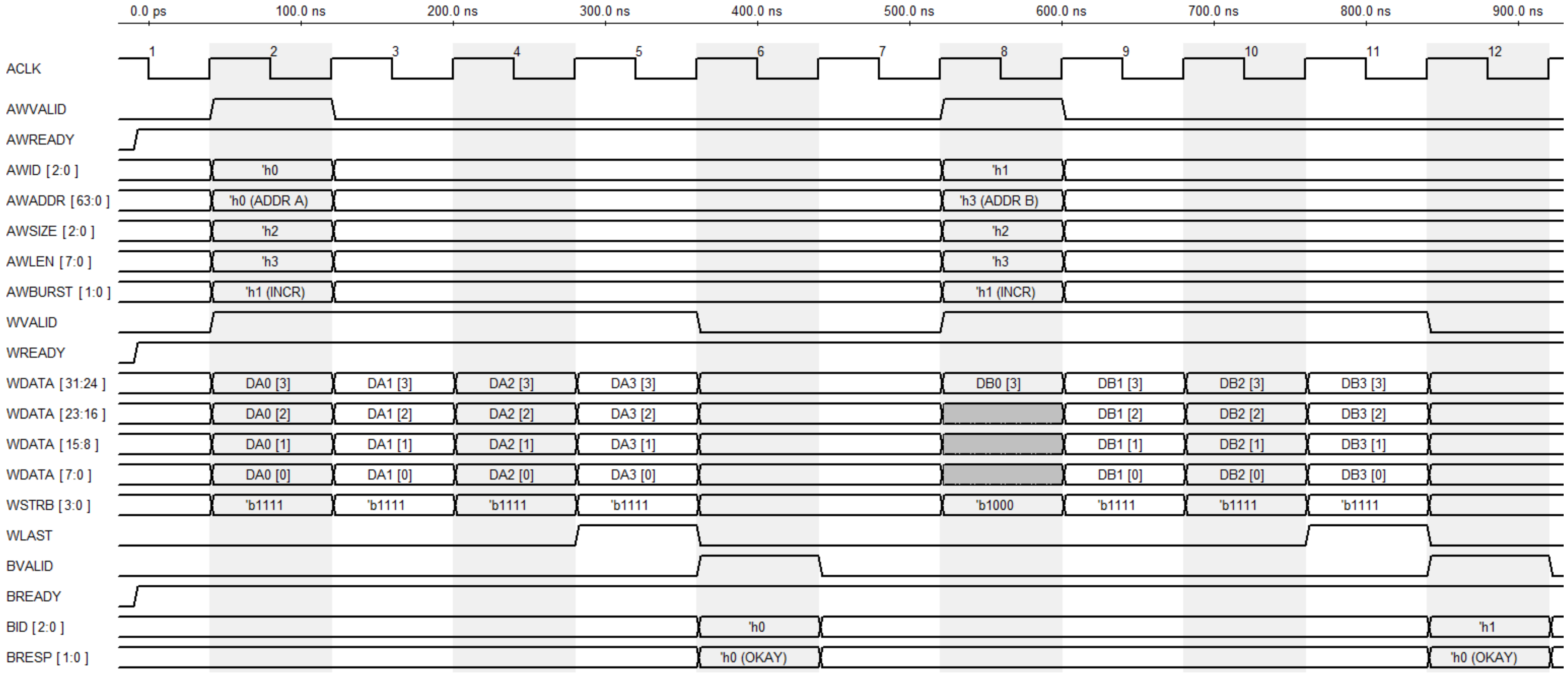


Figure 4-10: **Write Full Transfer**

Figure 4-11 shows the read full transfer timing diagram.

Aligned (ADDR A)	AXI data width = 32-bit	ARID = 0	ARADDR = 'h0	ARSIZE = 2	ARLEN = 3	ARBURST = INCR
Unaligned (ADDR B)	AXI data width = 32-bit	ARID = 1	ARADDR = 'h3	ARSIZE = 2	ARLEN = 3	ARBURST = INCR

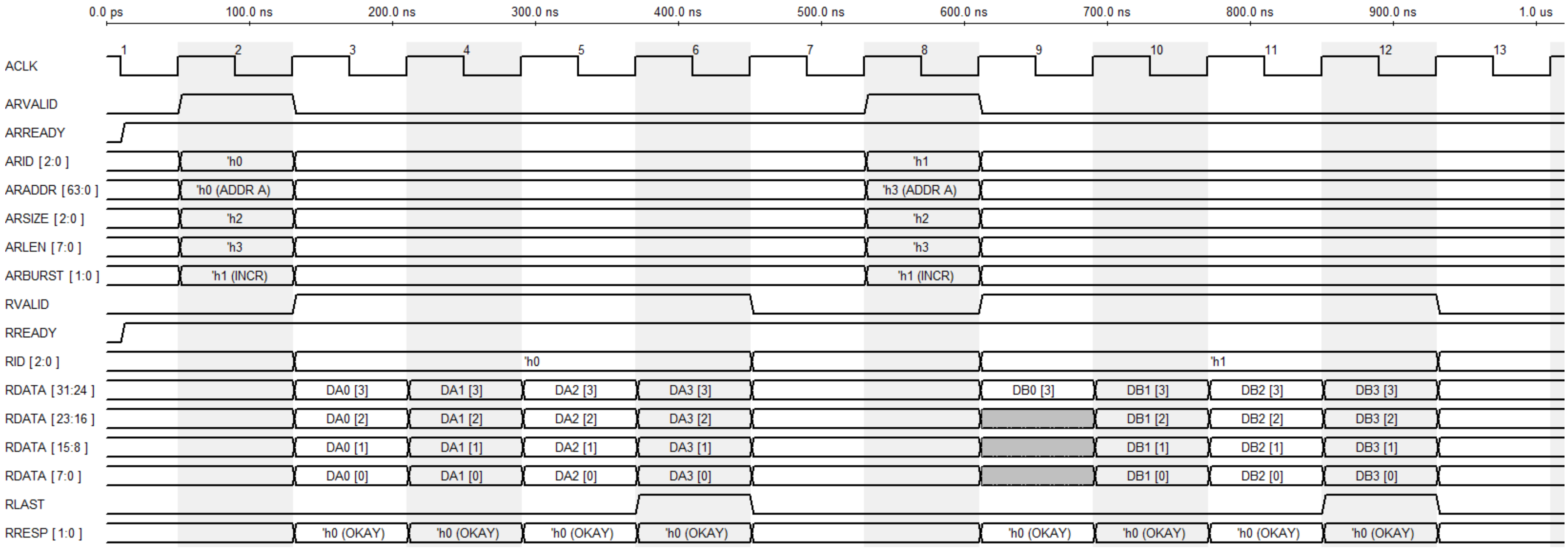


Figure 4-11: **Read Full Transfer**

Figure 4-12 shows the write narrow transfer timing diagram.

Aligned (ADDR A)	AXI data width = 32-bit	AWID = 0	AWADDR = 'h0	AWSIZE = 1	AWLEN = 3	AWBURST = INCR
Unaligned (ADDR B)	AXI data width = 32-bit	AWID = 1	AWADDR = 'h3	AWSIZE = 1	AWLEN = 3	AWBURST = INCR

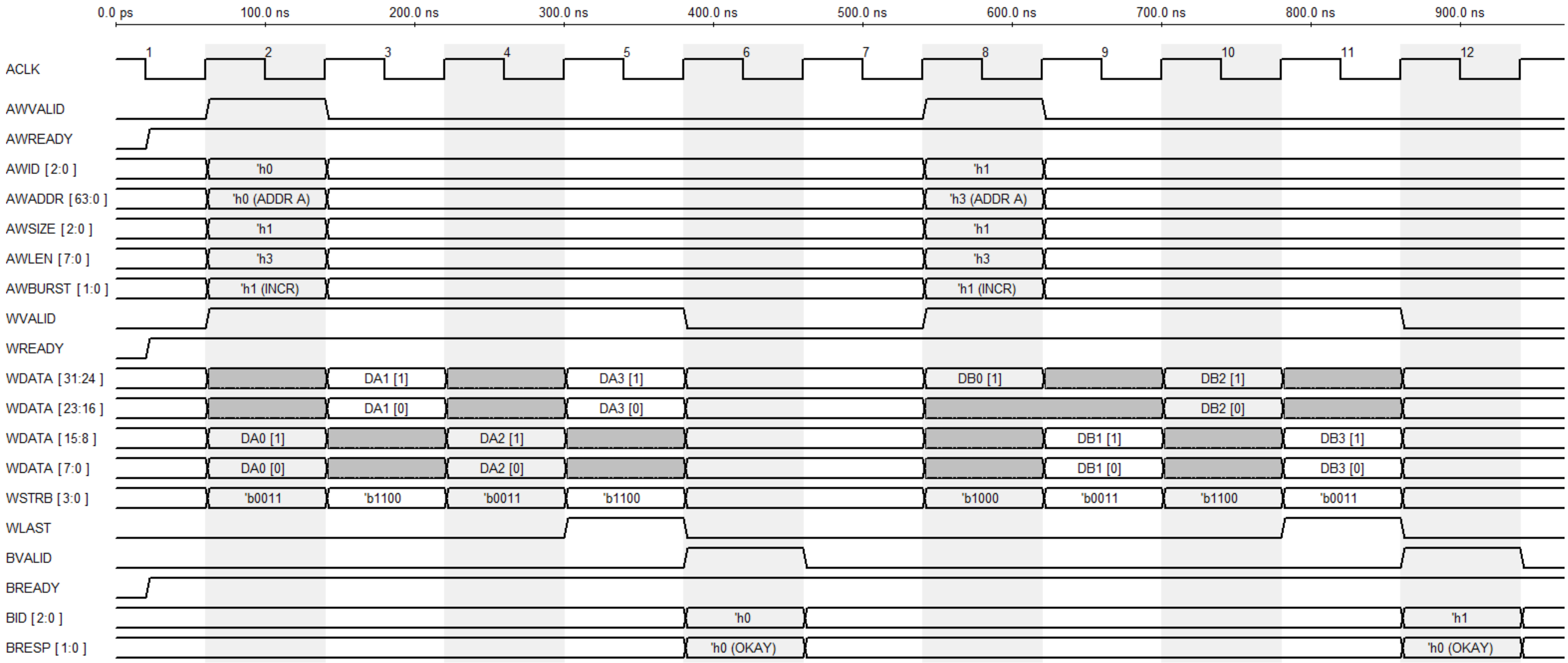


Figure 4-12: **Write Narrow Transfer**

Figure 4-13 shows the read narrow transfer timing diagram.

Aligned (ADDR A)	AXI data width = 32-bit	ARID = 0	ARADDR = 'h0	ARSIZE = 1	ARLEN = 3	ARBURST = INCR
Unaligned (ADDR B)	AXI data width = 32-bit	ARID = 1	ARADDR = 'h3	ARSIZE = 1	ARLEN = 3	ARBURST = INCR

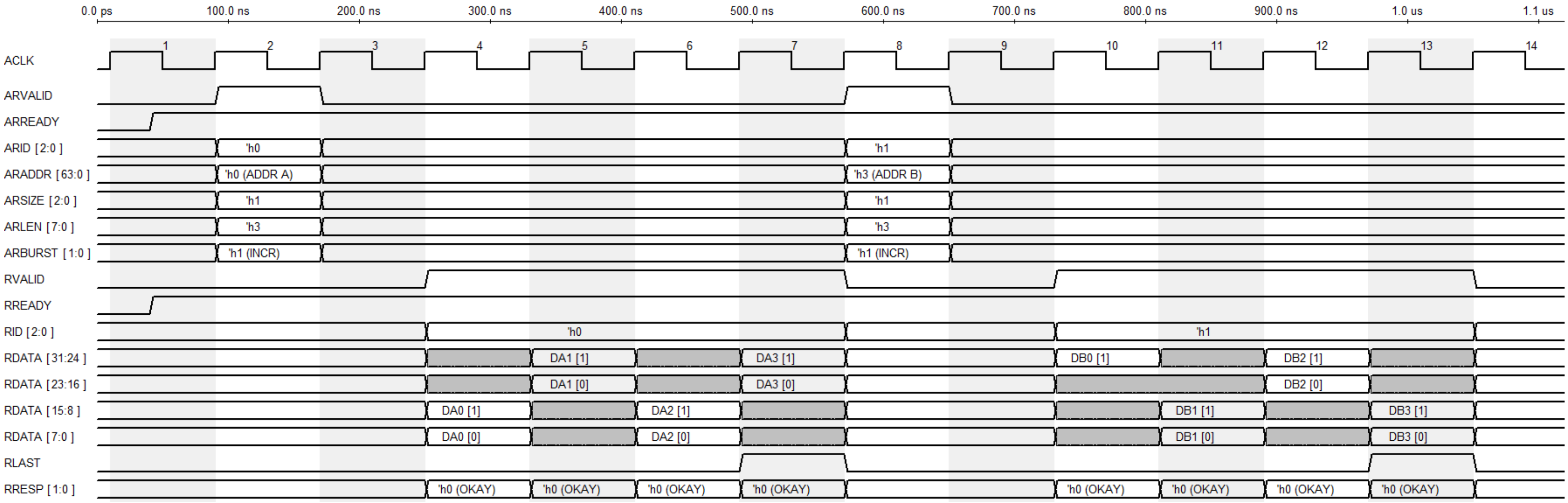


Figure 4-13: **Read Narrow Transfer**

Arbitration in AXI Shim

The AXI4 protocol calls for independent read and write address channels. The Memory Controller has one address channel. The following arbitration options are available for arbitrating between the read and write address channels.

Time Division Multiplexing (TDM)

Equal priority is given to read and write address channels in this mode. The grant to the read and write address channels alternate every clock cycle. The read or write requests from the AXI master has no bearing on the grants. For example, the read requests are served in alternative clock cycles, even when there are no write requests. The slots are fixed and they are served in their respective slots only.

Round-Robin

Equal priority is given to read and write address channels in this mode. The grant to the read and write channels depends on the last served request granted from the AXI master. For example, if the last performed operation is write, then it gives precedence for read operation to be served over write operation. Similarly, if the last performed operation is read, then it gives precedence for write operation to be served over read operation.

Read Priority (RD_PRI_REG)

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

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

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

Read Priority with Starve Limit (RD_PRI_REG_STARVE_LIMIT)

The read address channel is always given priority in this mode. The requests from the write address channel are processed when there are no pending requests from the read address channel or the starve limit for read is reached.

Write Priority (WRITE_PRIORITY, WRITE_PRIORITY_REG)

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

AXI4-Lite Slave Control/Status Register Interface Block

The AXI4-Lite Slave Control register block provides a processor accessible interface to the ECC memory option. The interface is available when ECC is enabled and the primary slave interface is AXI4. The block provides interrupts, interrupt enable, ECC status, ECC enable/disable, ECC correctable errors counter, first failing correctable/uncorrectable data, ECC, and address. Fault injection registers for software testing is provided when the ECC_TEST_FI_XOR (C_ECC_TEST) parameter is ON. The AXI4-Lite interface is fixed at 32 data bits and signaling follows the standard AMBA AXI4-Lite specifications [Ref 12].

The AXI4-Lite Control/Status register interface block is implemented in parallel to the AXI4 memory-mapped interface. The block monitors the output of the native interface to capture correctable (single bit) and uncorrectable (multiple bit) errors. When a correctable and/or uncorrectable error occurs, the interface also captures the byte address of the failure along with the failing data bits and ECC bits. Fault injection is provided by an XOR block placed in the write datapath after the ECC encoding has occurred.

Only the first memory beat in a transaction can have errors inserted. For example, in a memory configuration with a data width of 72 and a mode register set to burst length 8, only the first 72 bits are corruptible through the fault injection interface. Interrupt generation based on either a correctable or uncorrectable error can be independently configured with the register interface. SLVERR response is seen on the read response bus (rresp) in case of uncorrectable errors (if ECC is enabled).

ECC Enable/Disable

The ECC_ON_OFF register enables/disables the ECC decode functionality. However, encoding is always enabled. The default value at start-up can be parameterized with C_ECC_ONOFF_RESET_VALUE. Assigning a value of 1 for the ECC_ON_OFF bit of this register results in the `correct_en` signal input into the mem_intfc to be asserted. Writing a value of 0 to the ECC_ON_OFF bit of this register results in the `correct_en` signal to be deasserted. When `correct_en` is asserted, decoding is enabled, and the opposite is true when this signal is deasserted. ECC_STATUS/ECC_CE_CNT are not updated when ECC_ON_OFF = 0. The FI_D0, FI_D1, FI_D2, and FI_D3 registers are not writable when ECC_ON_OFF = 0.

Single Error and Double Error Reporting

Two vectored signals from the Memory Controller indicate an ECC error: `ecc_single` and `ecc_multiple`. The `ecc_single` signal indicates if there has been a correctable error and the `ecc_multiple` signal indicates if there has been an uncorrectable error. The widths of `ecc_multiple` and `ecc_single` are based on the C_NCK_PER_CLK parameter.

There can be between 0 and C_NCK_PER_CLK × 2 errors per cycle with each data beat signaled by one of the vector bits. Multiple bits of the vector can be signaled per cycle indicating that multiple correctable errors or multiple uncorrectable errors have been detected. The **`ecc_err_addr`** signal (discussed in Fault Collection) is valid during the assertion of either **`ecc_single`** or **`ecc_multiple`**.

The ECC_STATUS register sets the CE_STATUS bit and/or UE_STATUS bit for correctable error detection and uncorrectable error detection, respectively.

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

Interrupt Generation

When interrupts are enabled with the CE_EN_IRQ and/or UE_EN_IRQ bits of the ECC_EN_IRQ register, and a correctable error or uncorrectable error occurs, the interrupt signal is asserted.

Fault Collection

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

When a correctable error occurs, there is also a counter that counts the number of correctable errors that have occurred. The counter can be read from the CE_CNT register and is fixed as an 8-bit counter. It does not rollover when the maximum value is incremented.

Fault Injection

The ECC Fault Injection registers, FI_D and FI_ECC, facilitates testing of the software drivers. When set, the ECC Fault Injection register XORs with the DDR3/DDR4 SDRAM datapath to simulate errors in the memory. It is ideal for injection to occur here because this is after the encoding has been completed. There is only support to insert errors on the first data beat, therefore there are two to four FI_D registers to accommodate this. During operation, after the error has been inserted into the datapath, the register clears itself.

AXI4-Lite Slave Control/Status Register Interface Parameters

Table 4-40 lists the AXI4-Lite slave interface parameters.

Table 4-40: **AXI4-Lite Slave Control/Status Register Parameters**

Parameter Name	Default Value	Allowable Values	Description
C_S_AXI_CTRL_ADDR_WIDTH	32	32	This is the width of the AXI4-Lite address buses.
C_S_AXI_CTRL_DATA_WIDTH	32	32	This is the width of the AXI4-Lite data buses.
C_ECC_ONOFF_RESET_VALUE	1	0, 1	Controls ECC ON/OFF value at startup/reset.
C_ECC_TEST	OFF	ON, OFF	When ON, you can inject faults on the first burst of data/ECC.

AXI4-Lite Slave Control/Status Register Interface Signals

Table 4-41 lists the AXI4 slave interface specific signals. Clock/reset to the interface is provided from the Memory Controller.

Table 4-41: **List of New I/O Signals**

Name	Width	I/O	Active State	Description
s_axi_ctrl_awaddr	C_S_AXI_CTRL_ADDR_WIDTH	I		Write address
s_axi_ctrl_awvalid	1	I	High	Write address valid. This signal indicates that valid write address and control information are available.
s_axi_ctrl_awready	1	O	High	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals.
s_axi_ctrl_wdata	C_S_AXI_CTRL_DATA_WIDTH	I		Write data
s_axi_ctrl_wvalid	1	I	High	Write valid. This signal indicates that write data and strobe are available.
s_axi_ctrl_wready	1	O	High	Write ready
s_axi_ctrl_bvalid	1	O	High	Write response valid
s_axi_ctrl_bresp2	2	O		Write response
s_axi_ctrl_bready	1	I	High	Response ready
s_axi_ctrl_araddr	C_S_AXI_CTRL_ADDR_WIDTH	I		Read address
s_axi_ctrl_arvalid	1	I	High	Read address valid
s_axi_ctrl_arready	1	O	High	Read address
s_axi_ctrl_rdata	C_S_AXI_CTRL_DATA_WIDTH	O		Read data
s_axi_ctrl_rresp	2	O		Read response
s_axi_ctrl_rvalid	1	O		Read valid
s_axi_ctrl_rready	1	I		Read ready
interrupt	1	O	High	IP Global Interrupt signal

AXI4-Lite Slave Control/Status Register Map

ECC register map is shown in Table 4-42. The register map is Little Endian. Write accesses to read-only or reserved values are ignored. Read accesses to write-only or reserved values return the value `0xDEADDEAD`.

Table 4-42: **ECC Control Register Map**

Address Offset	Register Name	Access Type	Default Value	Description
0x00	ECC_STATUS	R/W	0x0	ECC Status Register
0x04	ECC_EN_IRQ	R/W	0x0	ECC Enable Interrupt Register
0x08	ECC_ON_OFF	R/W	0x0 or 0x1	ECC On/Off Register. If C_ECC_ONOFF_RESET_VALUE = 1, the default value is 0x1.
0x0C	CE_CNT	R/W	0x0	Correctable Error Count Register
(0x10–0x9C) Reserved				
0x100	CE_FFD[31:00]	R	0x0	Correctable Error First Failing Data Register
0x104	CE_FFD[63:32]	R	0x0	Correctable Error First Failing Data Register
0x108	CE_FFD[95:64](1)	R	0x0	Correctable Error First Failing Data Register
0x10C	CE_FFD [127:96](1)	R	0x0	Correctable Error First Failing Data Register
(0x110–0x17C) Reserved				
0x180	CE_FFE	R	0x0	Correctable Error First Failing ECC Register
(0x184–0x1BC) Reserved				
0x1C0	CE_FFA[31:0]	R	0x0	Correctable Error First Failing Address
0x1C4	CE_FFA[63:32]	R	0x0	Correctable Error First Failing Address
(0x1C8–0x1FC) Reserved				
0x200	UE_FFD [31:00]	R	0x0	Uncorrectable Error First Failing Data Register
0x204	UE_FFD [63:32]	R	0x0	Uncorrectable Error First Failing Data Register
0x208	UE_FFD [95:64](1)	R	0x0	Uncorrectable Error First Failing Data Register
0x20C	UE_FFD [127:96](1)	R	0x0	Uncorrectable Error First Failing Data Register
(0x210–0x27C) Reserved				
0x280	UE_FFE	R	0x0	Uncorrectable Error First Failing ECC Register
(0x284–0x2BC) Reserved				
0x2C0	UE_FFA[31:0]	R	0x0	Uncorrectable Error First Failing Address
0x2C4	UE_FFA[63:32]	R	0x0	Uncorrectable Error First Failing Address
(0x2C8–0x2FC) Reserved				
0x300	FI_D[31:0](2)	W	0x0	Fault Inject Data Register
0x304	FI_D[63:32](2)	W	0x0	Fault Inject Data Register
0x308	FI_D[95:64](1)(2)	W	0x0	Fault Inject Data Register

Table 4-42: **ECC Control Register Map *(Cont'd)***

Address Offset	Register Name	Access Type	Default Value	Description
0x30C	FI_D[127:96](1)(2)	W	0x0	Fault Inject Data Register
(0x340–0x37C) Reserved				
0x380	FI_ECC(2)	W	0x0	Fault Inject ECC Register

Notes:

1. Data bits 64–127 are only enabled if the DQ width is 144 bits.
2. FI_D* and FI_ECC* are only enabled if ECC_TEST parameter has been set to 1.

AXI4-Lite Slave Control/Status Register Map Detailed Descriptions

IMPORTANT: *The ECC_TEST parameter must be manually set to ON in the IP output products when they are generated globally for the AXI4-Lite Slave Control/Status Register Interface block to be generated.*

ECC_STATUS

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

Table 4-43: **ECC Status Register**

Bits	Name	Core Access	Reset Value	Description
1	CE_STATUS	R/W	0	If 1, a correctable error has occurred. This bit is cleared when a 1 is written to this bit position.
0	UE_STATUS	R/W	0	If 1, an uncorrectable error has occurred. This bit is cleared when a 1 is written to this bit position.

ECC_EN_IRQ

This register determines if the values of the CE_STATUS and UE_STATUS bits in the ECC Status register assert the Interrupt output signal (ECC_Interrupt). If both CE_EN_IRQ and UE_EN_IRQ are set to 1 (enabled), the value of the Interrupt signal is the logical OR between the CE_STATUS and UE_STATUS bits.

Table 4-44: **ECC Interrupt Enable Register**

Bits	Name	Core Access	Reset Value	Description
1	CE_EN_IRQ	R/W	0	If 1, the value of the CE_STATUS bit of ECC Status register is propagated to the Interrupt signal. If 0, the value of the CE_STATUS bit of ECC Status register is not propagated to the Interrupt signal.
0	UE_EN_IRQ	R/W	0	If 1, the value of the UE_STATUS bit of ECC Status register is propagated to the Interrupt signal. If 0, the value of the UE_STATUS bit of ECC Status register is not propagated to the Interrupt signal.

ECC_ON_OFF

The ECC On/Off Control register allows the application to enable or disable ECC checking. The design parameter, C_ECC_ONOFF_RESET_VALUE (default on) determines the reset value for the enable/disable setting of ECC. This facilitates start-up operations when ECC might or might not be initialized in the external memory. When disabled, ECC checking is disabled for read but ECC generation is active for write operations.

Table 4-45: **ECC On/Off Control Register**

Bits	Name	Core Access	Reset Value	Description
0	ECC_ON_OFF	R/W	Specified by design parameter, C_ECC_ONOFF_RESET_VALUE	If 0, ECC checking is disabled on read operations. (ECC generation is enabled on write operations when C_ECC = 1). If 1, ECC checking is enabled on read operations. All correctable and uncorrectable error conditions are captured and status is updated.

CE_CNT

This register counts the number of occurrences of correctable errors. It can be cleared or preset to any value using a register write. When the counter reaches its maximum value, it does not wrap around, but instead it stops incrementing and remains at the maximum value. The width of the counter is defined by the value of the C_CE_COUNTER_WIDTH parameter. The value of the CE counter width is fixed to eight bits.

Table 4-46: **Correctable Error Counter Register**

Bits	Name	Core Access	Reset Value	Description
7:0	CE_CNT	R/W	0	Holds the number of correctable errors encountered.

CE_FFA[31:0]

This register stores the lower 32 bits of the decoded DRAM address (Bits[31:0]) of the first occurrence of an access with a correctable error. The address format is defined in Table 3-1, page 32. When the CE_STATUS bit in the ECC Status register is cleared, this register is re-enabled to store the address of the next correctable error. Storing of the failing address is enabled after reset.

Table 4-47: **Correctable Error First Failing Address [31:0] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	CE_FFA[31:0]	R	0	Address (Bits[31:0]) of the first occurrence of a correctable error.

CE_FFA[63:32]

This register stores the upper 32 bits of the decoded DRAM address (Bits[55:32]) of the first occurrence of an access with a correctable error. The address format is defined in Table 3-1, page 32. In addition, the upper byte of this register stores the `ecc_single` signal. When the CE_STATUS bit in the ECC Status register is cleared, this register is re-enabled to store the address of the next correctable error. Storing of the failing address is enabled after reset.

Table 4-48: **Correctable Error First Failing Address [63:32] Register**

Bits	Name	Core Access	Reset Value	Description
31:24	CE_FFA[63:56]	R	0	ecc_single[7:0]. Indicates which bursts of the BL8 transaction associated with the logged address had a correctable error. Bit[24] corresponds to the first burst of the BL8 transfer.
23:0	CE_FFA[55:32]	R	0	Address (Bits[55:32]) of the first occurrence of a correctable error.

CE_FFD[31:0]

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

Table 4-49: **Correctable Error First Failing Data [31:0] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	CE_FFD[31:0]	R	0	Data (Bits[31:0]) of the first occurrence of a correctable error.

CE_FFD[63:32]

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

Table 4-50: **Correctable Error First Failing Data [63:32] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	CE_FFD[63:32]	R	0	Data (Bits[63:32]) of the first occurrence of a correctable error.

CE_FFD[95:64]

Note: This register is only used when DQ_WIDTH == 144.

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

Table 4-51: **Correctable Error First Failing Data [95:64] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	CE_FFD[95:64]	R	0	Data (Bits[95:64]) of the first occurrence of a correctable error.

CE_FFD[127:96]

Note: This register is only used when DQ_WIDTH == 144.

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

Table 4-52: **Correctable Error First Failing Data [127:96] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	CE_FFD[127:96]	R	0	Data (Bits[127:96]) of the first occurrence of a correctable error.

CE_FFE

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

Table 4-53 describes the register bit usage when DQ_WIDTH = 72.

Table 4-53: **Correctable Error First Failing ECC Register for 72-Bit External Memory Width**

Bits	Name	Core Access	Reset Value	Description
7:0	CE_FFE	R	0	ECC (Bits[7:0]) of the first occurrence of a correctable error.

Table 4-54 describes the register bit usage when DQ_WIDTH = 144.

Table 4-54: **Correctable Error First Failing ECC Register for 144-Bit External Memory Width**

Bits	Name	Core Access	Reset Value	Description
15:0	CE_FFE	R	0	ECC (Bits[15:0]) of the first occurrence of a correctable error.

UE_FFA[31:0]

This register stores the decoded DRAM address (Bits[31:0]) of the first occurrence of an access with an uncorrectable error. The address format is defined in Table 3-1, page 32. When the UE_STATUS bit in the ECC Status register is cleared, this register is re-enabled to store the address of the next uncorrectable error. Storing of the failing address is enabled after reset.

Table 4-55: **Uncorrectable Error First Failing Address [31:0] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	UE_FFA [31:0]	R	0	Address (Bits[31:0]) of the first occurrence of an uncorrectable error.

UE_FFA[63:32]

This register stores the decoded address (Bits[55:32]) of the first occurrence of an access with an uncorrectable error. The address format is defined in Table 3-1, page 32. In addition, the upper byte of this register stores the `ecc_multiple` signal. When the UE_STATUS bit in the ECC Status register is cleared, this register is re-enabled to store the address of the next uncorrectable error. Storing of the failing address is enabled after reset.

Table 4-56: **Uncorrectable Error First Failing Address [31:0] Register**

Bits	Name	Core Access	Reset Value	Description
31:24	UE_FFA[63:56]	R	0	ecc_multiple[7:0]. Indicates which bursts of the BL8 transaction associated with the logged address had an uncorrectable error. Bit[24] corresponds to the first burst of the BL8 transfer.
23:0	UE_FFA[55:32]	R	0	Address (Bits[55:32]) of the first occurrence of a correctable error.

UE_FFD[31:0]

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

Table 4-57: **Uncorrectable Error First Failing Data [31:0] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	UE_FFD[31:0]	R	0	Data (Bits[31:0]) of the first occurrence of an uncorrectable error.

UE_FFD[63:32]

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

Table 4-58: **Uncorrectable Error First Failing Data [63:32] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	UE_FFD [63:32]	R	0	Data (Bits[63:32]) of the first occurrence of an uncorrectable error.

UE_FFD[95:64]

Note: This register is only used when the DQ_WIDTH == 144.

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

Table 4-59: **Uncorrectable Error First Failing Data [95:64] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	UE_FFD[95:64]	R	0	Data (Bits[95:64]) of the first occurrence of an uncorrectable error.

UE_FFD[127:96]

Note: This register is only used when the DQ_WIDTH == 144.

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

Table 4-60: **Uncorrectable Error First Failing Data [127:96] Register**

Bits	Name	Core Access	Reset Value	Description
31:0	UE_FFD[127:96]	R	0	Data (Bits[127:96]) of the first occurrence of an uncorrectable error.

UE_FFE

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

Table 4-61 describes the register bit usage when DQ_WIDTH = 72.

Table 4-61: **Uncorrectable Error First Failing ECC Register for 72-Bit External Memory Width**

Bits	Name	Core Access	Reset Value	Description
7:0	UE_FFE	R	0	ECC (Bits[7:0]) of the first occurrence of an uncorrectable error.

Table 4-62 describes the register bit usage when DQ_WIDTH = 144.

Table 4-62: **Uncorrectable Error First Failing ECC Register for 144-Bit External Memory Width**

Bits	Name	Core Access	Reset Value	Description
15:0	UE_FFE	R	0	ECC (Bits[15:0]) of the first occurrence of an uncorrectable error.

FI_D0

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

The register is only implemented if C_ECC_TEST = ON or ECC_TEST_FI_XOR = ON and ECC = ON in a DDR3/DDR4 SDRAM design in the Vivado IP catalog.

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

Table 4-63: **Fault Injection Data (Word 0) Register**

Bits	Name	Core Access	Reset Value	Description
31:0	FI_D0	W	0	Bit positions set to 1 toggle the corresponding Bits[31:0] of the next data word written to the memory. This register is automatically cleared after the fault has been injected.

Special consideration must be given across FI_D0, FI_D1, FI_D2, and FI_D3 such that only a single error condition is introduced.

FI_D1

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

This register is only implemented if C_ECC_TEST = ON or ECC_TEST_FI_XOR = ON and ECC = ON in a DDR3/DDR4 SDRAM design in the Vivado IP catalog.

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

Table 4-64: **Fault Injection Data (Word 1) Register**

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

FI_D2

Note: This register is only used when DQ_WIDTH =144.

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

This register is only implemented if C_ECC_TEST = ON or ECC_TEST_FI_XOR = ON and ECC = ON in a DDR3/DDR4 SDRAM design in the Vivado IP catalog.

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

Table 4-65: **Fault Injection Data (Word 2) Register**

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

Special consideration must be given across FI_D0, FI_D1, FI_D2, and FI_D3 such that only a single error condition is introduced.

FI_D3

Note: This register is only used when DQ_WIDTH =144.

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

The register is only implemented if C_ECC_TEST = ON or ECC_TEST_FI_XOR = ON and ECC = ON in a DDR3/DDR4 SDRAM design in the Vivado IP catalog.

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

Table 4-66: **Fault Injection Data (Word 3) Register**

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

FI_ECC

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

The register is only implemented if C_ECC_TEST = ON or ECC_TEST_FI_XOR = ON and ECC = ON in a DDR3/DDR4 SDRAM design in the Vivado IP catalog.

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

Table 4-67 describes the register bit usage when DQ_WIDTH = 72.

Table 4-67: **Fault Injection ECC Register for 72-Bit External Memory Width**

Bits	Name	Core Access	Reset Value	Description
7:0	FI_ECC	W	0	Bit positions set to 1 toggle the corresponding bit of the next ECC written to the memory. The register is automatically cleared after the fault has been injected.

Table 4-68 describes the register bit usage when DQ_WIDTH = 144.

Table 4-68: **Fault Injection ECC Register for 144-Bit External Memory Width**

Bits	Name	Core Access	Reset Value	Description
15:0	FI_ECC	R	0	Bit positions set to 1 toggle the corresponding bit of the next ECC written to the memory. The register is automatically cleared after the fault has been injected.

PHY Only Interface

This section describes the FPGA logic interface signals and key parameters of the DDR3 and DDR4 PHY. The goal is to implement a "PHY Only" solution that connects your own custom Memory Controller directly to the DDR3/DDR4 SDRAM generated PHY, instead of interfacing to the user interface or AXI Interface of a DDR3/DDR4 SDRAM generated Memory Controller. The PHY interface takes DRAM commands, like Activate, Precharge, Refresh, etc. at its input ports and issues them directly to the DRAM bus.

The PHY does not take in "memory transactions" like the user and AXI interfaces, which translate transactions into one or more DRAM commands that meet DRAM protocol and timing requirements. The PHY interface does no DRAM protocol or timing checking. When using a PHY Only option, you are responsible for meeting all DRAM protocol requirements and timing specifications of all DRAM components in the system.

The PHY runs at the system clock frequency, or 1/4 of the DRAM clock frequency. The PHY therefore accepts four DRAM commands per system clock and issues them serially on consecutive DRAM clock cycles on the DRAM bus. In other words, the PHY interface has four command slots: slots 0, 1, 2, and 3, which it accepts each system clock. The command in slot position 0 is issued on the DRAM bus first, and the command in slot 3 is issued last. The PHY does have limitations as to which slots can accept read and write CAS commands. For more information, see CAS Command Timing Limitations, page 176. Except for CAS commands, each slot can accept arbitrary DRAM commands.

The PHY FPGA logic interface has an input port for each pin on a DDR3 or DDR4 bus. Each PHY command/address input port has a width that is eight times wider than its corresponding DRAM bus pin. For example, a DDR4 bus has one **`act_n`** pin, and the PHY has an 8-bit **`mc_ACT_n`** input port. Each pair of bits in the **`mc_ACT_n`** port corresponds to a "command slot." The two LSBs are slot0 and the two MSBs are slot3. The PHY address input port for a DDR4 design with 18 address pins is 144 bits wide, with each byte corresponding to the four command slots for one DDR4 address pin. There are two bits for each command slot in each input port of the PHY.

This is due to the underlying design of the PHY and its support for double data rate data buses. But as the DRAM command/address bus is single data rate, you must always drive the two bits that correspond to a command slot to the same value. See the following interface tables for additional descriptions and examples in the timing diagrams that show how bytes and bits correspond to DRAM pins and command slots.

The PHY interface has read and write data ports with eight bits for each DRAM DQ pin. Each port bit represents one data bit on the DDR DRAM bus for a BL8 burst. Therefore one BL8 data burst for the entire DQ bus is transferred across the PHY interface on each system clock. The PHY only supports BL8 data transfers. The data format is the same as the user interface data format. For more information, see PHY, page 34.

The PHY interface also has several control signals that you must drive and/or respond to when a read or write CAS command is issued. The control signals are used by the PHY to manage the transfer of read and write data between the PHY interface and the DRAM bus. See the following signal tables and timing diagrams.

Your custom Memory Controller must wait until the PHY output **`calDone`** is asserted before sending any DRAM commands to the PHY. The PHY initializes and trains the DRAM before asserting **`calDone`**. For more information on the PHY internal structures and training algorithms, see the PHY, page 34. After **`calDone`** is asserted, the PHY is ready to accept any DRAM commands.

The only required DRAM or PHY commands are related to VT tracking and DRAM refresh/ZQ. These requirements are detailed in VT Tracking, page 178 and Refresh and ZQ, page 181.

PHY Interface Signals

The PHY interface signals to the FPGA logic can be categorized into six groups:

- Clocking and Reset
- Command and Address
- Write Data
- Read Data
- PHY Control
- Debug

Clocking and Reset and Debug signals are described in other sections or documents. See the corresponding references. In this section, a description is given for each signal in the remaining four groups and timing diagrams show examples of the signals in use.

Clocking and Reset

For more information on the clocking and reset, see the Clocking, page 81 section.

Command and Address

Table 4-69 shows the command and address signals for a PHY only option.

Table 4-69: **Command and Address**

Signal	I/O	Description
mc_ACT_n[7:0]	I	DRAM ACT_n command signal for four DRAM clock cycles. Bits[1:0] correspond to the first DRAM clock cycle, Bits[3:2] to the second, Bits[5:4] to the third, and Bits[8:7] to the fourth. For center alignment to the DRAM clock with 1N timing, both bits of a given bit pair should be asserted to the same value. See timing diagrams for examples. All of the command/address ports in this table follow the same eight bits per DRAM pin format. Active-Low. This signal is not used in DDR3 systems.
mc_ADR[ADDR_WIDTH × 8 – 1:0]	I	DRAM address. Eight bits in the PHY interface for each address bit on the DRAM bus. Bits[7:0] corresponds to DRAM address Bit[0] on four DRAM clock cycles. Bits[15:8] corresponds to DRAM address Bit[1] on four DRAM clock cycles, etc. See the timing diagrams for examples. All of the multi-bit DRAM signals in this table follow the same format of 1-byte of the PHY interface port corresponding to four commands for one DRAM pin. Mixed active-Low and High depending on which type of DRAM command is being issued, but follows the DRAM pin active-High/Low behavior. The function of each byte of the mc_ADR port depends on whether the memory type is DDR4 or DDR3 and the particular DRAM command that is being issued. These functions match the DRAM address pin functions. For example, with DDR4 memory and the mc_ACT_n port bits asserted High, mc_ADR[135:112] have the function of RAS_n, CAS_n, and WE_n pins.
mc_RAS_n[7:0]	I	DDR3 DRAM RAS_n pin. Not used in DDR4 systems.
mc_CAS_n[7:0]	I	DDR3 DRAM CAS_n pin. Not used in DDR4 systems.
mc_WE_n[7:0]	I	DDR3 DRAM WE_n pin. Not used in DDR4 systems.
mc_BA[BANK_WIDTH × 8 – 1:0]	I	DRAM bank address. Eight bits for each DRAM bank address.
mc_BG[BANK_GROUP_WIDTH × 8 – 1:0]	I	DRAM bank group address. Eight bits for each DRAM pin.
mc_C[LR_WIDTH × 8 – 1:0]	I	DDR4 DRAM Chip ID pin. Valid for 3DS RDIMMs only. LR_WIDTH is $\log_2$(StackHeight) where StackHeight (S_HEIGHT) is 2 or 4.
mc_CKE[CKE_WIDTH × 8 – 1:0]	I	DRAM CKE. Eight bits for each DRAM pin.
mc_CS_n[CS_WIDTH × 8 – 1:0]	I	DRAM CS_n. Eight bits for each DRAM pin. Active-Low.
mc_ODT[ODT_WIDTH × 8– 1:0]	I	DRAM ODT. Eight bits for each DRAM pin. Active-High.
mc_PAR[7:0]	I	DRAM address parity. Eight bits for one DRAM parity pin. ***Note:*** This signal is valid for RDIMMs/LRDIMMs only.

Figure 4-14 shows the functional relationship between the PHY command/address input signals and a DDR4 command/address bus. The diagram shows an Activate command on system clock cycle N in the slot1 position. The `mc_ACT_n[3:2]` and `mc_CS_n[3:2]` are both asserted Low in cycle N, and all the other bits in cycle N are asserted High, generating an Activate in the slot1 position roughly two system clocks later and NOP/DESELECT commands on the other command slots.

On cycle N + 3, `mc_CS_n` and the `mc_ADR` bits corresponding to CAS/A15 are set to 0xFC. This asserts `mc_ADR[121:120]` and `mc_CS_n[1:0]` Low, and all other bits in cycle N + 3 High, generating a read command on slot0 and NOP/DESELECT commands on the other command slots two system clocks later. With the Activate and read command separated by three system clock cycles and taking into account the command slot position of both commands within their system clock cycle, expect the separation on the DDR4 bus to be 11 DRAM clocks, as shown in the DDR bus portion of Figure 4-14.

Note: Figure 4-14 shows the relative position of commands on the DDR bus based on the PHY input signals. Although the diagram shows some latency in going through the PHY to be somewhat realistic, this diagram does not represent the absolute command latency through the PHY to the DDR bus, or the system clock to DRAM clock phase alignment. The intention of this diagram is to show the concept of command slots at the PHY interface.

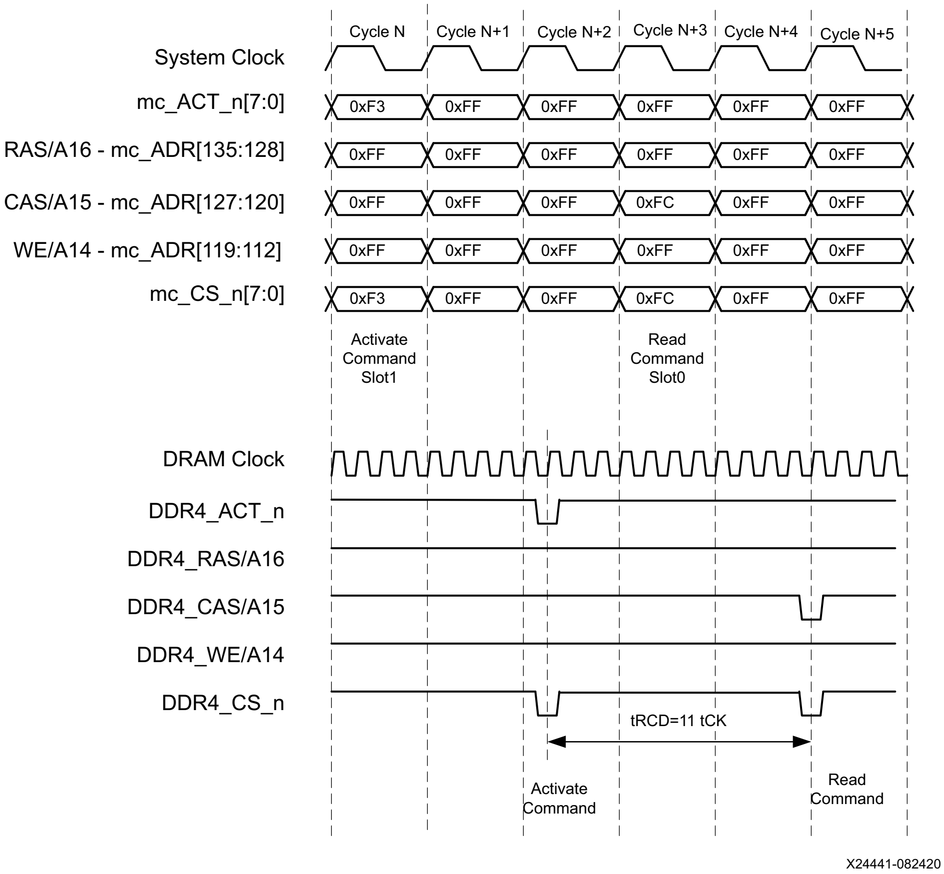


Figure 4-14: **PHY Command/Address Input Signal with DDR4 Command/Address Bus**

Figure 4-15 shows an example of using all four command slots in a single system clock. This example shows three commands to rank0, and one to rank1, in cycle N. BG and BA address pins are included in the diagram to spread the commands over different banks to not violate DRAM protocol. Table 4-70 lists the command in each command slot.

Table 4-70: **Command Slots**

Command Slot	0	1	2	3
DRAM Command	Read	Activate	Precharge	Refresh
Bank Group	0	1	2	0
Bank	0	3	1	0
Rank	0	0	0	1

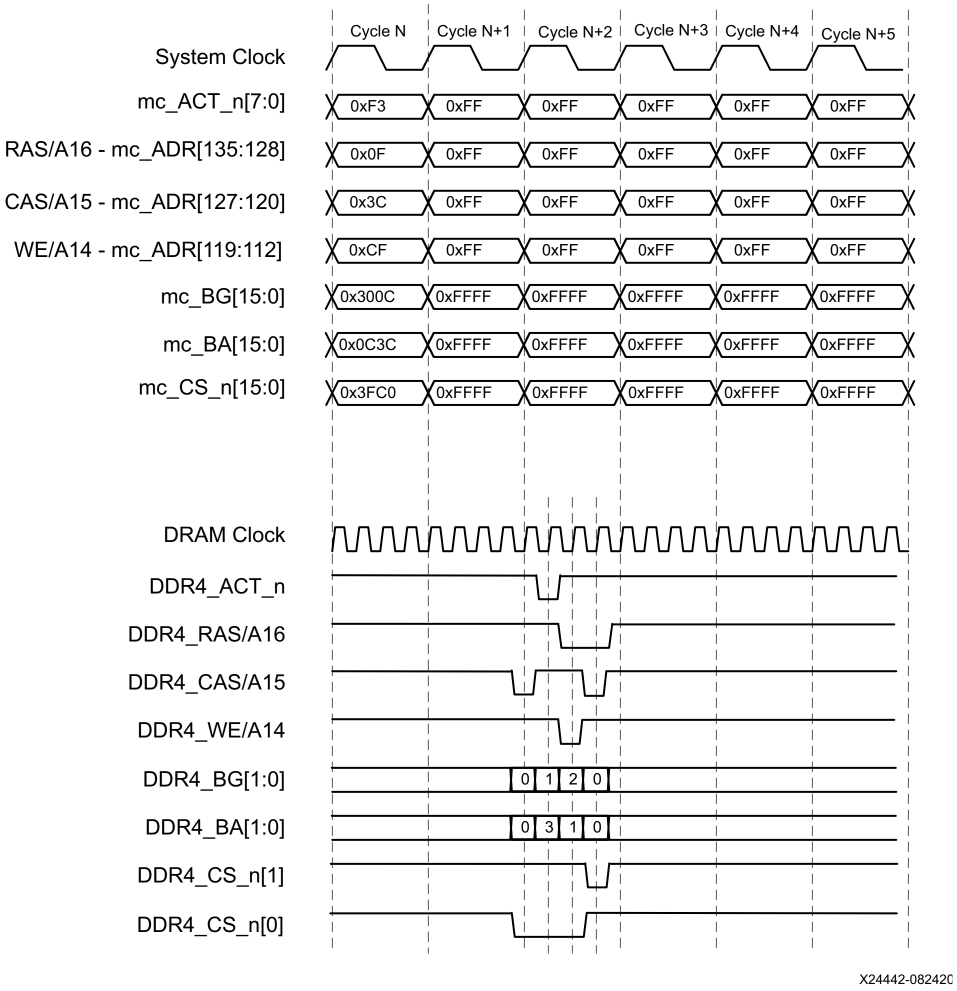


Figure 4-15: **PHY Command/Address with All Four Command Slots**

To understand how DRAM commands to different command slots are packed together, the following detailed example shows how to convert DRAM commands at the PHY interface to commands on the DRAM command/address bus. To convert PHY interface commands to DRAM commands, write out the PHY signal for one system clock in binary and reverse the bit order of each byte. You can also drop every other bit after the reversal because the bit pairs are required to have the same value. In the subsequent example, the `mc_BA[15:0]` signal has a cycle N value of `0x0C3C`:

Hex	0x0C3C
Binary	16'b0000_1100_0011_1100
Reverse bits in each byte	16'b0011_0000_0011_1100

Take the upper eight bits for DRAM `BA[1]` and the lower eight bits for DRAM `BA[0]` and the expected pattern on the DRAM bus is:

BA[1]	00	11	00	00
	0	1	0	0
	Low	High	Low	Low
BA[0]	00	11	11	00
	0	1	1	0
	Low	High	High	Low

This matches the DRAM `BA[1:0]` signal values of 0, 3, 1, and 0 shown in the Figure 4-15.

Write Data

Table 4-71 shows the write data signals for a PHY only option.

Table 4-71: **Write Data**

Signal	I/O	Description
wrData[DQ_WIDTH × 8 – 1:0]	I	DRAM write data. Eight bits for each DQ lane on the DRAM bus. This port transfers data for an entire BL8 write on each system clock cycle. Write data must be provided to the PHY one cycle after the wrDataEn output signal asserts, or two cycles after if the ECC parameter is set to ON. This protocol must be followed. There is no data buffering in the PHY.
wrDataMask[DM_WIDTH × 8 – 1:0]	I	DRAM write DM/DBI port. One bit for each byte of the wrData port, corresponding to one bit for each byte of each burst of a BL8 transfer. wrDataMask is transferred on the same system clock cycle as wrData. Active-High. For DDR3 interface, wrDataMask port appears in the Data Mask enabled option in Vivado IDE. For DDR4 interface, wrDataMask port appears in the "Data Mask and DBI" Vivado IDE option values of DM_NO_DBI and DM_DBI_RD.

Table 4-71: **Write Data *(Cont'd)***

Signal	I/O	Description
wrDataEn	O	Write data required. PHY asserts this port for one cycle for each write CAS command. Your design must provide wrData and wrDataMask at the PHY input ports on the cycle after wrDataEn asserts, or two cycles after if the ECC parameter is set to ON.
wrDataAddr[DATA_BUF_ADDR_WIDTH – 1:0]	O	Optional control signal. PHY stores and returns a data buffer address for each in-flight write CAS command. The wrDataAddr signal returns the stored addresses. It is only valid when the PHY asserts wrDataEn. You can use this signal to manage the process of sending write data into the PHY for a write CAS command, but this is completely optional.
tCWL[5:0]	O	Optional control signal. This output indicates the CAS write latency used in the PHY.
dBufAdr[DATA_BUF_ADDR_WIDTH – 1:0]	I	Reserved. Should be tied Low.

Read Data

Table 4-72 shows the read data signals for a PHY only option.

Table 4-72: **Read Data**

Signal	I/O	Description
rdData[DQ_WIDTH × 8 – 1:0]	O	DRAM read data. Eight bits for each DQ lane on the DRAM bus. This port transfers data for an entire BL8 read on each system clock cycle. rdData is only valid when the rdDataEn, per_rd_done, or rmw_rd_done is asserted. Your design must consume the read data when rdDataEn one of these "data valid" signals asserts. There is no data buffering in the PHY.
rdDataEn	O	Read data valid. This signal asserts High to indicate that the rdData and rdDataAddr signals are valid. rdDataEn asserts High for one system clock cycle for each BL8 read, unless the read was tagged as a special type of read. See the optional per_rd_done and rmw_rd_done signals for details on special reads. rdData must be consumed when rdDataEn asserts or data is lost. Active-High.
rdDataAddr[DATA_BUF_ADDR_WIDTH – 1:0]	O	Optional control signal. PHY stores and returns a data buffer address for each in-flight read CAS command. The rdDataAddr signal returns the stored addresses. It is only valid when the PHY asserts rdDataEn, per_rd_done, or rmw_rd_done. Your design can use this signal to manage the process of capturing and storing read data provided by the PHY, but this is completely optional.

Table 4-72: **Read Data *(Cont'd)***

Signal	I/O	Description
per_rd_done	O	Optional read data valid signal. This signal indicates that a special type of read has completed and its associated rdData and rdDataAddr signals are valid. When PHY input winInjTxn is asserted High at the same time as mcRdCAS, the read is tagged as a special type of read, and per_rd_done asserts instead of rdDataEn when data is returned.
rmw_rd_done	O	Optional read data valid signal. This signal indicates that a special type of read has completed and its associated rdData and rdDataAddr signals are valid. When PHY input winRmw is asserted High at the same time as mcRdCAS, the read is tagged as a special type of read, and rmw_rd_done asserts instead of rdDataEn when data is returned.
rdDataEnd	O	Unused. Tied High.

PHY Control

Table 4-73 shows the PHY control signals for a PHY only option.

Table 4-73: **PHY Control**

Signal	I/O	Description
calDone	O	Indication that the DRAM is powered up, initialized, and calibration is complete. This indicates that the PHY interface is available to send commands to the DRAM. Active-High.
mcRdCAS	I	Read CAS command issued. This signal must be asserted for one system clock if and only if a read CAS command is asserted on one of the command slots at the PHY command/address input ports. Hold at 0x0 until calDone asserts. Active-High.
mcWrCAS	I	Write CAS command issued. This signal must be asserted for one system clock if and only if a write CAS command is asserted on one of the command slots at the PHY command/address input ports. Hold at 0x0 until calDone asserts. Active-High.
winRank[1:0]	I	Target rank for CAS commands. This value indicates which rank a CAS command is issued to. It must be valid when either mcRdCAS or mcWrCAS is asserted. The PHY passes the value from this input to the XIPHY to select the calibration results for the target rank of a CAS command in multi-rank systems. In a single-rank system, this input port can be tied to 0x0.
mcCasSlot[1:0]	I	CAS command slot select. The PHY only supports CAS commands on even command slots. mcCasSlot indicates which of these two possible command slots a read CAS or write CAS was issued on. mcCasSlot is used by the PHY to generate XIPHY control signals, like DQ output enables, that need DRAM clock cycle resolution relative to the command slot used for a CAS command. Valid values after calDone asserts are 0x0 and 0x2. Hold at 0x0 until calDone asserts. This signal must be valid if mcRdCAS or mcWrCAS is asserted. For more information, see the CAS Command Timing Limitations, page 176.

Table 4-73: **PHY Control *(Cont'd)***

Signal	I/O	Description
mcCasSlot2	I	CAS slot 2 select. mcCasSlot2 serves a similar purpose as the mcCasSlot[1:0] signal, but mcCasSlot2 is used in timing critical logic in the PHY. Ideally mcCasSlot2 should be driven from separate flops from mcCasSlot[1:0] to allow synthesis/implementation to better optimize timing. mcCasSlot2 and mcCasSlot[1:0] must always be consistent if mcRdCAS or mcWrCAS is asserted. To be consistent, the following must be TRUE: mcCasSlot2==mcCasSlot[1]. Hold at 0x0 until calDone asserts. Active-High.
winInjTxn	I	Optional read command type indication. When winInjTxn is asserted High on the same cycle as mcRdCAS, the read does not generate an assertion on rdDataEn when it completes. Instead, the per_rd_done signal asserts, indicating that a special type of read has completed and that its data is valid on the rdData output. In DDR3/DDR4 SDRAM controller designs, the winInjTxn/per_rd_done signals are used to track non-system read traffic by asserting winInjTxn only on read commands issued for the purpose of VT tracking.
winRmw	I	Optional read command type indication. When winRmw is asserted High on the same cycle as mcRdCAS, the read does not generate an assertion on rdDataEn when it completes. Instead, the rmw_rd_done signal asserts, indicating that a special type of read has completed and that its data is valid on the rdData output. In DDR3/DDR4 SDRAM controller designs, the winRmw/rmw_rd_done signals are used to track reads issued as part of a read-modify-write flow. The DDR3/DDR4 SDRAM controller asserts winRmw only on read commands that are issued for the read phase of a RMW sequence.
winBuf[DATA_BUF_ADDR_WIDTH – 1:0]	I	Optional control signal. When either mcRdCAS or mcWrCAS is asserted, PHY stores the value on the winBuf signal. The value is returned on rdDataAddr or wrDataAddr, depending on whether mcRdCAS or mcWrCAS was used to capture winBuf. In DDR3/DDR4 SDRAM controller designs, these signals are used to track the data buffer address used to source write data or sink read return data.
gt_data_ready	I	Update VT Tracking. This signal triggers the PHY to read RIU registers in the XIPHY that measure how well the DQS Gate signal is aligned to the center of the read DQS preamble, and then adjust the alignment if needed. This signal must be asserted periodically to keep the DQS Gate aligned as voltage and temperature drift. For more information, see VT Tracking, page 178. Hold at 0x0 until calDone asserts. Active-High.

Figure 4-16 shows a write command example. On cycle N, write command "A" is asserted on the PHY command/address inputs in the slot0 position. The **`mcWrCAS`** input is also asserted on cycle N, and a valid rank value is asserted on the **`winRank`** signal. In Figure 4-16, there is only one **`CS_n`** pin, so the only valid **`winRank`** value is 0x0. The **`mcCasSlot[1:0]`** and **`mcCasSlot2`** signals are valid on cycle N, and specify slot0.

Write command "B" is then asserted on cycle N + 1 in the slot2 position, with **`mcWrCAS`**, **`winRank`**, **`mcCasSlot[1:0]`**, and **`mcCasSlot2`** asserted to valid values as well. On cycle

M, PHY asserts **`wrDataEn`** to indicate that **`wrData`** and **`wrDataMask`** values corresponding to command A need to be driven on cycle M + 1.

Figure 4-16 shows the data and mask widths assuming an 8-bit DDR4 DQ bus width. The delay between cycle N and cycle M is controlled by the PHY, based on the CWL and AL settings of the DRAM. **`wrDataEn`** also asserts on cycle M + 1 to indicate that **`wrData`** and **`wrDataMask`** values for command B are required on cycle M + 2. Although this example shows that **`wrDataEn`** is asserted on two consecutive system clock cycles, you should not assume this will always be the case, even if mcWrCAS is asserted on consecutive clock cycles as is shown here. There is no data buffering in the PHY and data is pulled into the PHY just in time. Depending on the CWL/AL settings and the command slot used, consecutive **`mcWrCAS`** assertions might not result in consecutive **`wrDataEn`** assertions.

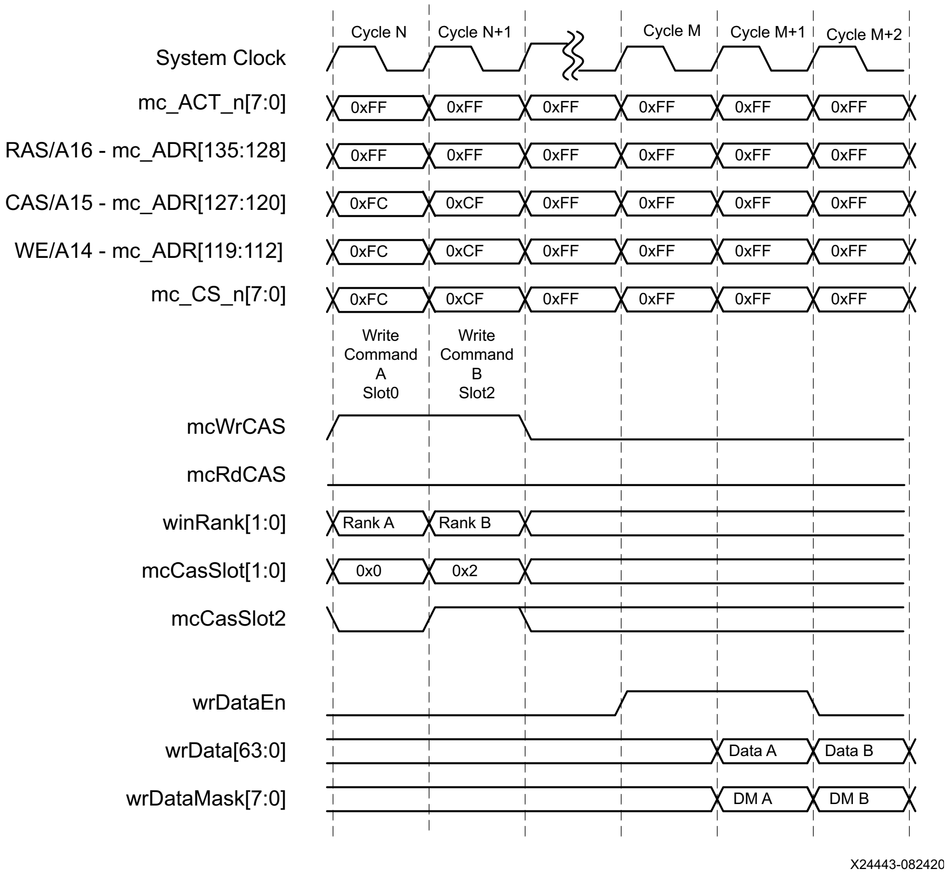


Figure 4-16: **Write Command Example**

Figure 4-17 shows a read command example. Read commands are issued on cycles N and N + 1 in slot positions 0 and 2, respectively. The **`mcRdCAS`**, **`winRank`**, **`mcCasSlot`**, and mcCasSlot2 are asserted on these cycles as well. On cycles M + 1 and M + 2, PHY asserts **`rdDataEn`** and **`rdData`**.

Note: The separation between N and M + 1 is much larger than in the write example (Figure 4-16). In the read case, the separation is determined by the full round trip latency of command output, DRAM CL/AL, and data input through PHY.

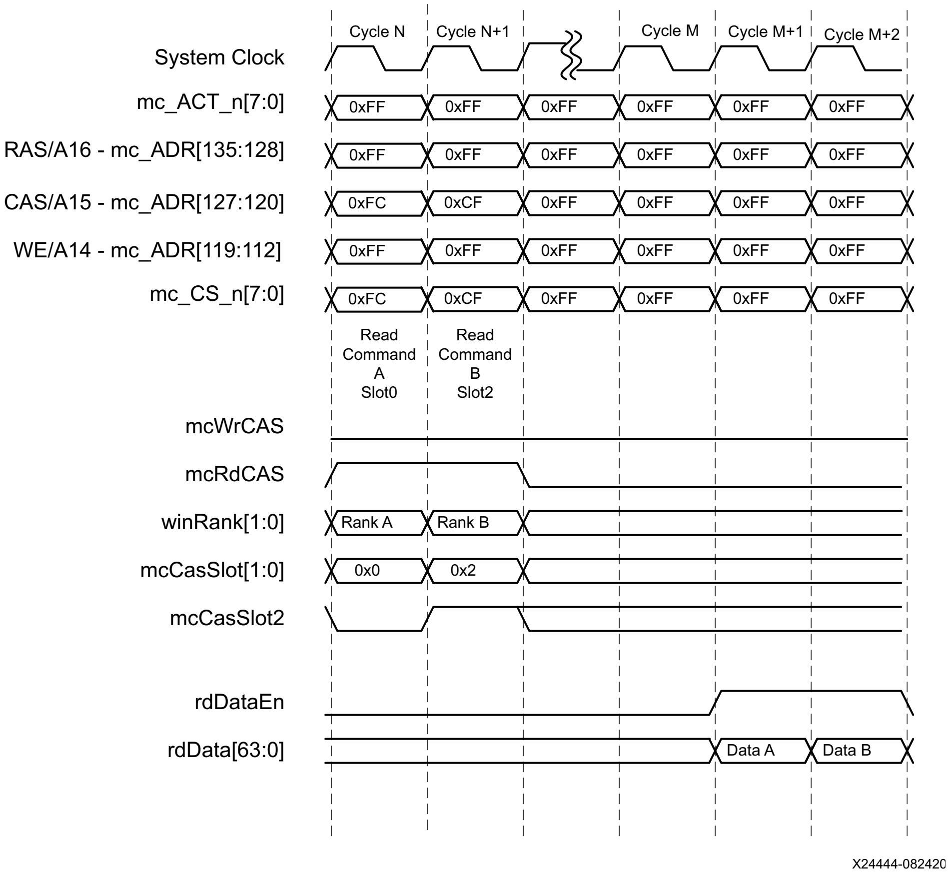


Figure 4-17: **Read Command Example**

Debug

The debug signals are explained in Debug Tools, page 581.

PHY Only Parameters

All PHY parameters are configured by the DDR3/DDR4 SDRAM software. Table 4-74 describes the PHY parameters. These parameter values must not be modified in the DDR3/DDR4 SDRAM generated designs. The parameters are set during core generation. The core must be regenerated to change any parameter settings.

Table 4-74: **PHY Only Parameters**

Parameter Name	Default Value	Allowable Values	Description
ADDR_WIDTH	18	DDR4 18.. 17 DDR3 16.. 13	Number of DRAM Address pins
BANK_WIDTH	2	DDR4 2 DDR3 3	Number of DRAM Bank Address pins
BANK_GROUP_WIDTH	2	DDR4 2.. 1 DDR3 N/A	Number of DRAM Bank Group pins

Table 4-74: **PHY Only Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
CK_WIDTH	1	2.. 1	Number of DRAM Clock pins
CKE_WIDTH	1	2.. 1	Number of DRAM CKE pins
CS_WIDTH	1	2.. 1	Number of DRAM CS pins
ODT_WIDTH	1	4.. 1	Number of DRAM ODT pins
DRAM_TYPE	"DDR4"	"DDR4," "DDR3"	DRAM Technology
DQ_WIDTH	16	Minimum = 8 Must be multiple of 8	Number of DRAM DQ pins in the channel
DQS_WIDTH	2	Minimum = 1 x8 DRAM – 1 per DQ byte x4 DRAM – 1 per DQ nibble	Number of DRAM DQS pins in the channel
DM_WIDTH	2	Minimum = 0 x8 DRAM – 1 per DQ byte x4 DRAM – 0	Number of DRAM DM pins in the channel
DATA_BUF_ADDR_WIDTH	5	5	Number of data buffer address bits stored for a read or write transaction
ODTWR	0x8421	0xFFFF .. 0x0000	Reserved for future use
ODTWRDEL	8	Set to CWL	Reserved for future use
ODTWRDUR	6	7.. 6	Reserved for future use
ODTRD	0x0000	0xFFFF.. 0x0000	Reserved for future use
ODTRDDEL	11	Set to CL	Reserved for future use
ODTRDDUR	6	7.. 6	Reserved for future use
ODTWR0DEL ODTWR0DUR ODTRD0DEL ODTRD0DUR ODTNOP	N/A	N/A	Reserved for future use
MR0	0x630	Legal SDRAM configuration	DRAM MR0 setting
MR1	0x101	Legal SDRAM configuration	DRAM MR1 setting
MR2	0x10	Legal SDRAM configuration	DRAM MR2 setting
MR3	0x0	Legal SDRAM configuration	DRAM MR3 setting
MR4	0x0	Legal SDRAM configuration	DRAM MR4 setting. DDR4 only.
MR5	0x400	Legal SDRAM configuration	DRAM MR5 setting. DDR4 only.

Table 4-74: **PHY Only Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
MR6	0x800	Legal SDRAM configuration	DRAM MR6 setting. DDR4 only.
SLOT0_CONFIG	0x1	0x1 0x3 0x5 0xF	For more information, see SLOT0_CONFIG.
SLOT1_CONFIG	0x0	0x0 0x2 0xC 0xA	For more information, see SLOT0_CONFIG.
SLOT0_FUNC_CS	0x1	0x1 0x3 0x5 0xF	Memory bus CS_n pins used to send all DRAM commands including MRS to memory. Each bit of the parameter represents 1-bit of the CS_n bus, for example, the LSB indicates CS_n[0], and the MSB indicates CS_n[3]. For DIMMs this parameter specifies the CS_n pins connected to DIMM slot 0. ***Note:*** slot 0 used here should not be confused with the "command slot0" term used in the description of the PHY command/address interface. For more information, see SLOT0_FUNC_CS.
SLOT1_FUNC_CS	0x0	0x0 0x2 0xC 0xA	See the SLOT0_FUNC_CS description. The only difference is that SLOT1_FUNC_CS specifies CS_n pins connected to DIMM slot 1.
REG_CTRL	OFF	ON OFF	Enable RDIMM RCD initialization and calibration
CA_MIRROR	OFF	ON OFF	Enable Address mirroring. This parameter is set to ON for the DIMMs that support address mirroring.
DDR4_REG_RC03	0x30	Legal RDIMM RCD configuration	RDIMM RCD control word 03
DDR4_REG_RC04	0x40	Legal RDIMM RCD configuration	RDIMM RCD control word 04
DDR4_REG_RC05	0x50	Legal RDIMM RCD configuration	RDIMM RCD control word 05
tCK	938	Minimum 833	DRAM clock period in ps
tXPR	72	Minimum 1. DRAM tXPR specification in system clocks	See JEDEC DDR SDRAM specification [Ref 1].
tMOD	6	Minimum 1. DRAM tMOD specification in system clocks	See JEDEC DDR SDRAM specification [Ref 1].

Table 4-74: **PHY Only Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
tMRD	2	Minimum 1. DRAM tMRD specification in system clocks	See JEDEC DDR SDRAM specification [Ref 1].
tZQINIT	256	Minimum 1. DRAM tZQINIT specification in system clocks	See JEDEC DDR SDRAM specification [Ref 1].
TCQ	100	100	Flop clock to Q in ps. For simulation purposes only.
EARLY_WR_DATA	OFF	OFF	Reserved for future use
EXTRA_CMD_DELAY	0	2.. 0	Added command latency in system clocks. Added command latency is required for some configurations. See details in CL/CWL section.
ECC	"OFF"	OFF	Enables early wrDataEn timing for DDR3/DDR4 SDRAM generated controllers when set to ON. PHY only designs must set this to OFF.
DM_DBI	"DM_NODBI"	"NONE" "DM_NODBI" "DM_DBIRD" "NODM_DBIWR" "NODM_DBIRD" "NODM_DBIWRRD" "NODM_NODBI"	DDR4 DM/DBI configuration. For details, see Table 4-76.
USE_CS_PORT	1	0 = no CS_n pins 1 = CS_n pins used	Controls whether or not CS_n pins are connect to DRAM. If there are no CS_n pins the PHY initialization and training logic issues NOPs between DRAM commands. If there are no CS_n pins, The DRAM chip select pin (CS#) must be tied Low externally at the DRAM.
DRAM_WIDTH	8	16, 8, 4	DRAM component DQ width
RANKS	1	4, 2, 1	Number of ranks in the memory subsystem
nCK_PER_CLK	4	4	Number of DRAM clocks per system clock
C_FAMILY	"kintexu"	"kintexu" "virtexu"	Device information used by MicroBlaze controller in the PHY.
BYTES	4	Minimum 3	Number of XIPHY "bytes" used for data, command, and address
DBYTES	2	Minimum 1	Number of bytes in the DRAM DQ bus

Table 4-74: **PHY Only Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
IOBTYPE	{39'b001_001_001_001_001_101_101_001_001_001_001_001_001, 39'b001_001_001_001_001_001_001_001_001_001_001_001_001, 39'b000_011_011_011_011_111_111_011_011_011_011_001_011, 39'b001_011_011_011_011_111_111_011_011_011_011_001_011}	3'b000 = Unused pin 3'b 001 = Single-ended output 3'b 010 = Single-ended input 3'b011 = Single-ended I/O 3'b100 = Unused pin 3'b 101 = Differential Output 3'b 110 = Differential Input 3'b 111 = Differential INOUT	IOB setting
PLL_WIDTH	1	DDR3/DDR4 SDRAM generated values	Number of PLLs
CLKOUTPHY_MODE	"VCO_2X"	VCO_2X	Determines the clock output frequency based on the VCO frequency for the BITSLICE_CONTROL block
PLLCLK_SRC	0	0 = pll_clk0 1 = pll_clk1	XIPHY PLL clock source
DIV_MODE	0	0 = DIV4 1 = DIV2	XIPHY controller mode setting
DATA_WIDTH	8	8	XIPHY parallel input data width
CTRL_CLK	0x3	0 = Internal, local div_clk used 1 = External RIU clock used	Internal or external XIPHY clock for the RIU
INIT	{(15 × BYTES){1'b1}}	1'b0 1'b1	3-state bitslice OSERDES initial value
RX_DATA_TYPE	{15'b000000_00_00000_00, 15'b000000_00_00000_00, 15'b011110_10_11110_01, 15'b011110_10_11110_01}	2'b00 = None 2'b01 = DATA(DQ_EN) 2'b10 = CLOCK(DQS_EN) 2'b11 = DATA_AND_CLOCK	XIPHY bitslice setting
TX_OUTPUT_PHASE_90	{13'b1111111111111, 13'b1111111111111, 13'b0000011000010, 13'b1000011000010}	1'b0 = No offset 1'b1 = 90° offset applied	XIPHY setting to apply 90° offset on a given bitslice

Table 4-74: **PHY Only Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
RXTX_BITSLICE_EN	{13'b1111101111111, 13'b1111111111111, 13'b0111101111111, 13'b1111101111111}	1'b0 = No bitslice 1'b1 = Bitslice enabled	XIPHY setting to enable a bitslice
NATIVE_ODLAY_BYPASS	{(13 × BYTES){1'b0}}	1'b0 = FALSE 1'b1 = TRUE (Bypass)	Bypass the ODELAY on output bitslices
EN_OTHER_PCLK	{BYTES{2'b01}}	1'b 0 = FALSE (not used) 1'b 1 = TRUE (used)	XIPHY setting to route capture clock from other bitslice
EN_OTHER_NCLK	{BYTES{2'b01}}	1'b 0 = FALSE (not used) 1'b 1 = TRUE (used)	XIPHY setting to route capture clock from other bitslice
RX_CLK_PHASE_P	{{(BYTES – DBYTES){2'b00}}, {DBYTES{2'b11}}}	2'b00 for Address/Control, 2'b11 for Data	XIPHY setting to shift the read clock DQS_P by 90° relative to the DQ
RX_CLK_PHASE_N	{{(BYTES – DBYTES){2'b00}}, {DBYTES{2'b11}}}	2'b00 for Address/Control, 2'b11 for Data	XIPHY setting to shift the read clock DQS_N by 90° relative to the DQ
TX_GATING	{{(BYTES – DBYTES){2'b00}}, {DBYTES{2'b11}}}	2'b00 for Address/Control, 2'b11 for Data	Write DQS gate setting for the XIPHY
RX_GATING	{{(BYTES – DBYTES){2'b00}}, {DBYTES{2'b11}}}	2'b00 for Address/Control, 2'b11 for Data	Read DQS gate setting for the XIPHY
EN_DYN_ODLY_MODE	{{(BYTES – DBYTES){2'b00}}, {DBYTES{2'b11}}}	2'b00 for Address/Control, 2'b11 for Data	Dynamic loading of the ODELAY by XIPHY
BANK_TYPE	"HP_IO"	"HP_IO" "HR_IO"	Indicates whether selected bank is HP or HR
SIM_MODE	"BFM"	"FULL", "BFM"	Flag to set if the XIPHY is used ("UNISIM") or the behavioral model for simulation speed up.
SELF_CALIBRATE	{(2 × BYTES){1'b0}}	{(2 × BYTES){1'b0}} for simulation, {(2 × BYTES){1'b1}} for hardware	BISC self calibration
BYPASS_CAL	"FALSE"	"TRUE" for simulation, "FALSE" for hardware	Flag to turn calibration ON/OFF
CAL_WRLVL	"FULL"	"FULL"	Flag for calibration, write-leveling setting
CAL_DQS_GATE	"FULL"	"FULL"	Flag for calibration, DQS gate setting
CAL_RDLVL	"FULL"	"FULL"	Flag for calibration, read training setting

Table 4-74: **PHY Only Parameters *(Cont'd)***

Parameter Name	Default Value	Allowable Values	Description
CAL_WR_DQS_DQ	"FULL"	"FULL"	Flag for calibration, write DQS-to-DQ setting
CAL_COMPLEX	"FULL"	"SKIP", "FULL"	Flag for calibration, complex pattern setting
CAL_RD_VREF	"SKIP"	"SKIP", "FULL"	Flag for calibration, read V_{REF} setting
CAL_WR_VREF	"SKIP"	"SKIP", "FULL"	Flag for calibration, write V_{REF} setting
CAL_JITTER	"FULL"	"FULL", "NONE"	Reserved for verification. Speed up calibration simulation. Must be set to "FULL" for all hardware test cases.
t200us	53305 decimal	0x3FFFF.. 1	Wait period after BISC complete to DRAM reset_n deassertion in system clocks
t500us	133263 decimal	0x3FFFF.. 1	Wait period after DRAM reset_n deassertion to CKE assertion in system clocks

EXTRA_CMD_DELAY Parameter

Depending on the number of ranks, ECC mode, and DRAM latency configuration, PHY must be programmed to add latency on the DRAM command address bus. This provides enough pipeline stages in the PHY programmable logic to close timing and to process **`mcWrCAS`**. Added command latency is generally needed at very low CWL in single-rank configurations, or in multi-rank configurations. Enabling ECC might also require adding command latency, but this depends on whether your controller design (outside the PHY) depends on receiving the **`wrDataEn`** signal a system clock cycle early to allow for generating ECC check bits.

The EXTRA_CMD_DELAY parameter is used to add one or two system clock cycles of delay on the DRAM command/address path. The parameter does not delay the **`mcWrCAS`** or **`mcRdCAS`** signals. This gives the PHY more time from the assertion of **`mcWrCAS`** or **`mcRdCAS`** to generate XIPHY control signals. To the PHY, an EXTRA_CMD_DELAY setting of one or two is the same as having a higher CWL or AL setting.

Table 4-75 shows the required EXTRA_CMD_DELAY setting for various configurations of CWL, CL, and AL.

Table 4-75: **EXTRA_CMD_DELAY Configuration Settings**

DRAM Configuration			Required EXTRA_CMD_DELAY	
DRAM CAS Write Latency CWL	DRAM CAS Latency CL	DRAM Additive Latency MR1[4:3]	Single-Rank without ECC	Single-Rank with ECC or Multi-Rank
5	5	0	1	2
5	5	1	0	1
5	5	2	1	2

Table 4-75: **EXTRA_CMD_DELAY Configuration Settings *(Cont'd)***

DRAM Configuration			Required EXTRA_CMD_DELAY	
DRAM CAS Write Latency CWL	**DRAM CAS Latency CL**	**DRAM Additive Latency MR1[4:3]**	**Single-Rank without ECC**	**Single-Rank with ECC or Multi-Rank**
5	5	3	1	2
5	6	0	1	2
5	6	1	0	1
5	6	2	0	1
5	6	3	0	1
6	6	0	1	2
6	6	1	0	1
6	6	2	0	1
6	6	3	0	1
6	7	0	1	2
6	7	1	0	1
6	7	2	0	1
6	7	3	0	1
6	8	0	1	2
6	8	1	0	0
6	8	2	0	1
6	8	3	0	1
7	7	0	1	2
7	7	1	0	0
7	7	2	0	1
7	7	3	0	1
7	8	0	1	2
7	8	1	0	0
7	8	2	0	0
7	8	3	0	0
7	9	0	1	2
7	9	1	0	0
7	9	2	0	0
7	9	3	0	0
7	10	0	1	2
7	10	1	0	0
7	10	2	0	0
7	10	3	0	0

Table 4-75: **EXTRA_CMD_DELAY Configuration Settings *(Cont'd)***

DRAM Configuration			Required EXTRA_CMD_DELAY	
DRAM CAS Write Latency CWL	**DRAM CAS Latency CL**	**DRAM Additive Latency MR1[4:3]**	**Single-Rank without ECC**	**Single-Rank with ECC or Multi-Rank**
8	8	0	1	2
8	8	1	0	0
8	8	2	0	0
8	8	3	0	0
8	9	0	1	2
8	9	1	0	0
8	9	2	0	0
8	9	3	0	0
8	10	0	1	2
8	10	1	0	0
8	10	2	0	0
8	10	3	0	0
8	11	0	1	2
8	11	1	0	0
8	11	2	0	0
8	11	3	0	0
9 to 12	X	0	0	1
9 to 12	X	1, 2, or 3	0	0
≥13	X	0	0	0
≥13	X	1, 2, or 3	0	0

DM_DBI Parameter

The PHY supports the DDR4 DBI function on the read path and write path. Table 4-76 shows how read and write DBI can be enabled separately or in combination.

When write DBI is enabled, Data Mask is disabled. The DM_DBI parameter only configures the PHY and the MRS parameters must also be set to configure the DRAM for DM/DBI.

Table 4-76: **DM_DBI PHY Settings**

DM_DBI Parameter Value	PHY Read DBI	PHY Write DBI	PHY Write Data Mask
None	Disabled	Disabled	Disabled
DM_NODBI	Disabled	Disabled	Enabled
DM_DBIRD	Enabled	Disabled	Enabled
NODM_DBIWR	Disabled	Enabled	Disabled
NODM_DBIRD	Enabled	Disabled	Disabled

Table 4-76: **DM_DBI PHY Settings *(Cont'd)***

DM_DBI Parameter Value	PHY Read DBI	PHY Write DBI	PHY Write Data Mask
NODM_DBIWRRD	Enabled	Enabled	Disabled
NODM_NODBI	Disabled	Disabled	Disabled

The allowed values for the DM_DBI option in the GUI are as follows for x8 and x16 parts ("X" indicates supported and "–" indicates not supported):

Table 4-77: **DM_DBI Options**

Option Value	Native		AXI	
	ECC Disable	ECC Enable	ECC Disable	ECC Enable
DM_NO_DBI[1]	X	–	X	–
DM_DBI_RD	X	–	X	–
NO_DM_DBI_RD	X	X	–	X
NO_DM_DBI_WR	X	X	–	X
NO_DM_DBI_WR_RD	X	X	–	X
NO_DM_NO_DBI[2]	–	X	–	X

Notes:

1. Default option for ECC disabled interfaces.
2. Default option for ECC enabled interfaces.

IMPORTANT: *DBI should be enabled with repeated single Burst Length = 8 (BL8) read access with all "0" on the DQ bus, followed by idle (NOP/DESELECT) inserted between each BL8 read burst as shown in Figure 1-2. Enabling the DBI feature effectively mitigates excessive power supply noise.*
If DBI is not an option, then encoding the data to remove all "0" bursts in application before it reaches the memory controller is an equally effective method for mitigating power supply noise. For x4-based RDIMM/LRDIMM interfaces which lack the DM/DBI pin, the power supply noise is mitigated by the ODT settings used for these topologies. For x4-based component interfaces wider than 16 bits, the data encoding method is recommended.

For x4 parts, the supported DM_DBI option value is "NONE."

DBI can be enabled to reduce power consumption in the interface by reducing the total number of DQ signals driven Low and thereby reduce noise in the V_{CCO} supply. For further information where this might be useful for improved signal integrity, see Answer Record AR 70006.

CAS Command Timing Limitations

The PHY only supports CAS commands on even command slots, that is, 0 and 2. This limitation is due to the complexity of the PHY logic driven by the PHY control inputs, like the `mcWrCAS` and `mcRdCAS` signals, not the actual DRAM command signals like `mc_ACT_n[7:0]`, which just pass through the PHY after `calDone` asserts. The PHY logic is complex because it generates XIPHY control signals based on the DRAM CWL and CL values with DRAM clock resolution, not just system clock resolution.

Supporting two different command slots for CAS commands adds a significant amount of logic on the XIPHY control paths. There are very few pipeline stages available to break up the logic due to protocol requirements of the XIPHY. CAS command support on all four slots would further increase the complexity and degrade timing.

Minimum Write CAS Command Spacing

The minimum Write CAS to Write CAS command spacing to different ranks is eight DRAM clocks. This is a PHY limitation. If you violate this timing, the PHY might not have enough time to switch its internal delay settings and drive Write `DQ/DQS` on the DDR bus with correct timing. The internal delay settings are determined during calibration, and it varies with system layout.

Following the memory system layout guidelines ensures that a spacing of eight DRAM clocks is sufficient for correct operation. Write to Write timing to the same rank is limited only by the DRAM specification and the command slot limitations for CAS commands discussed earlier.

System Considerations for CAS Command Spacing

System layout and timing uncertainties should be considered in how your custom controller sets minimum CAS command spacing. The controller must space the CAS commands so that there are no DRAM timing violations and no `DQ/DQS` bus drive fights. When a DDR3/DDR4 SDRAM generated memory controller is instantiated, the layout guidelines are considered and command spacing is adjusted accordingly for a worst case layout.

Consider Read to Write command spacing, the JEDEC® DRAM specification [Ref 1] shows the component requirement as: RL + BL/2 + 2 – WL. This formula only spaces the Read `DQS` post-amble and Write DQS preamble by one DRAM clock on an ideal bus with no timing skews. Any `DQS` flight time, write leveling uncertainty, jitter, etc. reduces this margin. When these timing errors add up to more than one DRAM clock, there is a drive fight at the FPGA DQS pins which likely corrupts the Read transaction. A DDR3/DDR4 SDRAM generated controller uses the following formula to delay Write CAS after a Read CAS to allow for a worst case timing budget for a system following the layout guidelines: RL + BL/2 + 4 – WL.

Read CAS to Read CAS commands to different ranks must also be spaced by your custom controller to avoid drive fights, particularly when reading first from a "far" rank and then from a "near" rank. A DDR3/DDR4 SDRAM generated controller spaces the Read CAS commands to different ranks by at least six DRAM clock cycles.

Write CAS to Read CAS to the same rank is defined by the JEDEC DRAM specification [Ref 1]. Your controller must follow this DRAM requirement, and it ensures that there is no possibility of drive fights for Write to Read to the same rank. Write CAS to Read CAS spacing to different ranks, however, must also be limited by your controller. This spacing is not defined by the JEDEC DRAM specification [Ref 1] directly.

Write to Read to different ranks can be spaced much closer together than Write to Read to the same rank, but factors to consider include write leveling uncertainty, jitter, and tDQSCK. A DDR3/DDR4 SDRAM generated controller spaces Write CAS to Read CAS to different ranks by at least six DRAM clocks.

Additive Latency

The PHY supports DRAM additive latency. The only effect on the PHY interface due to enabling Additive Latency in the MRS parameters is in the timing of the **`wrDataEn`** signal after **`mcWrCAS`** assertion. The PHY takes the AL setting into account when scheduling **`wrDataEn`**. You can also find the **`rdDataEn`** asserts much later after **`mcRdCAS`** because the DRAM returns data much later. The AL setting also has an impact on whether or not the EXTRA_CMD_DELAY parameter needs to be set to a non-zero value.

VT Tracking

The PHY requires read commands to be issued at a minimum rate to keep the read DQS gate signal aligned to the read DQS preamble after **`calDone`** is asserted. In addition, the **`gt_data_ready`** signal needs to be pulsed at regular intervals to instruct the PHY to update its read DQS training values in the RIU. Finally, the PHY requires periodic gaps in read traffic to allow the XIPHY to update its gate alignment circuits with the values the PHY programs into the RIU. Specifically, the PHY requires the following after **`calDone`** asserts:

1. At least one read command every 1 µs. For a multi-rank system any rank is acceptable within the same channel. For a Ping Pong PHY, there are multiple channels. In that case, it is necessary to read command on each channel.
2. The **`gt_data_ready`** signal is asserted for one system clock cycle after **`rdDataEn`** or **`per_rd_done`** signal asserts at least once within each 1 µs interval.

For a Ping Pong PHY, there are multiple channels. In that case, it is necessary to assert the **`gt_data_ready`** signal for multiple channels at the same time like the following figure.

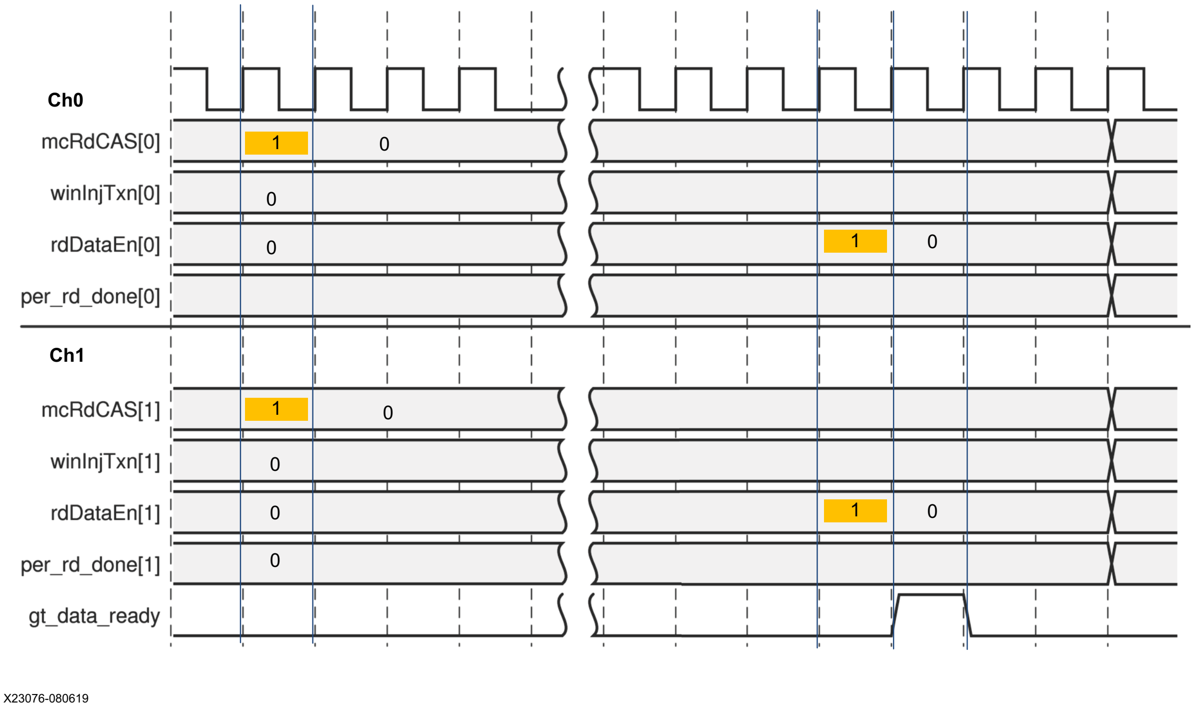


Figure 4-18: **Ping Pong PHY for Multiple Channels**

When the read is tagged as a special type of read, it is possible to assert the **`gt_data_ready`** signal after the **`per_rd_done`** signal at each channels like the following figure.

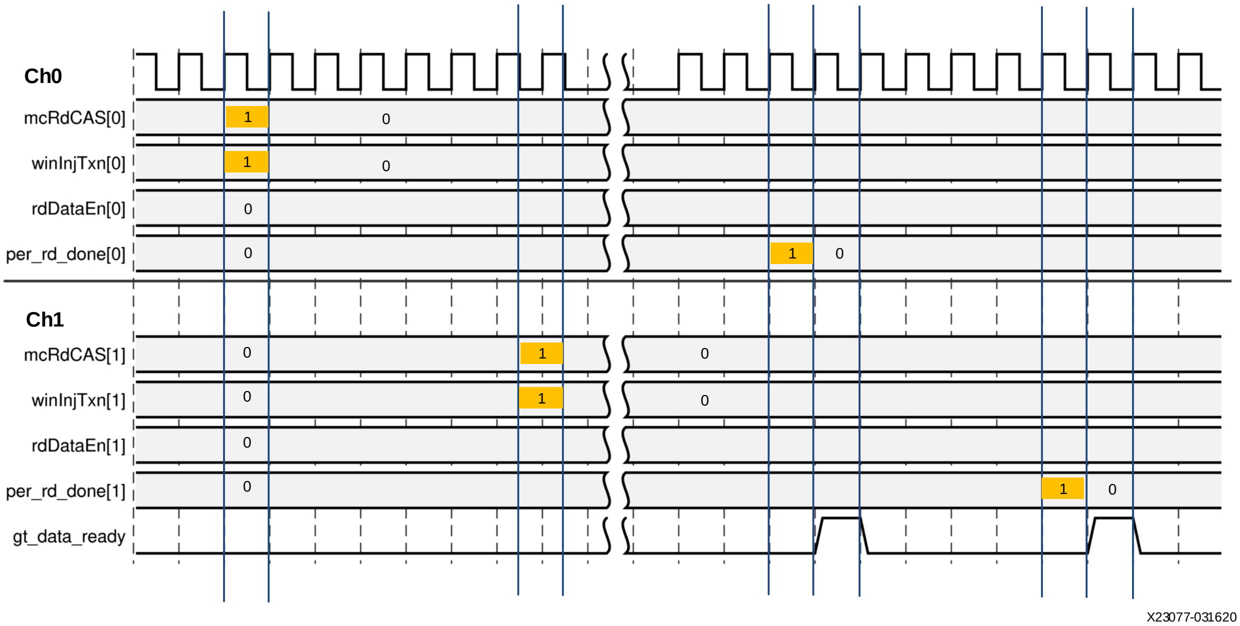


Figure 4-19: **Ping Pong PHY for Special Type of Read**

3. There is a three contiguous system clock cycle period with no read CAS commands asserted at the PHY interface every 1 μs.

The PHY cannot interrupt traffic to meet these requirements. It is therefore your custom Memory Controller's responsibility to issue DRAM commands and assert the **`gt_data_ready`** input signal in a way that meets the above requirements.

Figure 4-20 shows two examples where the custom controller must interrupt normal traffic to meet the VT tracking requirements. The first example is a High read bandwidth workload with **`mcRdCAS`** asserted continuously for almost 1 μs. The controller must stop issuing read commands for three contiguous system clocks once each 1 μs period, and assert **`gt_data_ready`** once per period.

The second example is a High write bandwidth workload with **`mcWrCAS`** asserted continuously for almost 1 μs. The controller must stop issuing writes, issue at least one read command, and then assert **`gt_data_ready`** once per 1 μs period.

IMPORTANT: *The controller must not violate DRAM protocol or timing requirements during this process.*

Note: The VT tracking diagrams are not drawn to scale.

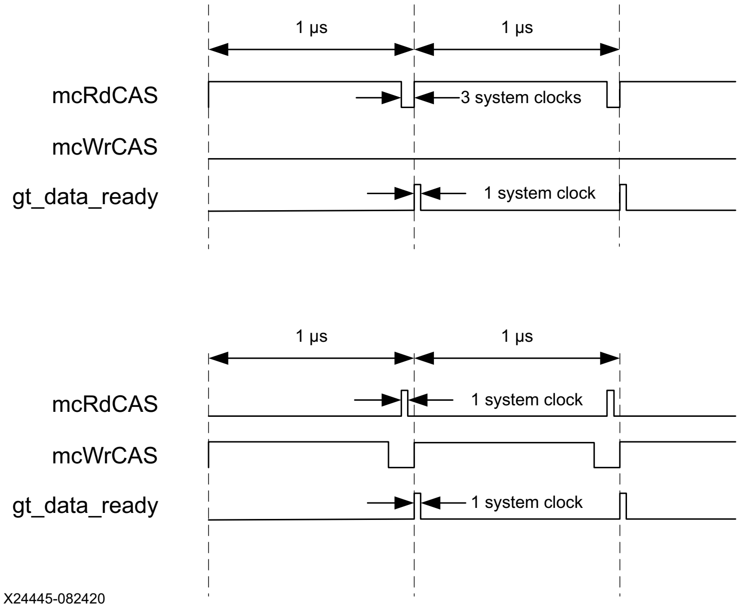


Figure 4-20: **VT Tracking Diagrams**

A workload that has a mix of read and write traffic in every 1 µs interval might naturally meet the first and third VT tracking requirements listed above. In this case, the only extra step required is to assert the `gt_data_ready` signal every 1 µs and regular traffic would not be interrupted at all. The custom controller, however, is responsible for ensuring all three requirements are met for all workloads. DDR3/DDR4 SDRAM generated controllers monitor the `mcRdCAS` and `mcWrCAS` signals and decide each 1 µs period what actions, if any, need to be taken to meet the VT tracking requirements. Your custom controller can implement any scheme that meets the requirements described here.

Refresh and ZQ

After `calDone` is asserted by the PHY, periodic DRAM refresh and ZQ calibration are the responsibility of your custom Memory Controller. Your controller must issue refresh and ZQ commands, meet DRAM refresh and ZQ interval requirements, while meeting all other DRAM protocol and timing requirements. For example, if a refresh is due and you have open pages in the DRAM, you must precharge the pages, wait tRP, and then issue a refresh command, etc. The PHY does not perform the precharge or any other part of this process for you.

Ping Pong PHY

Overview

This section describes the Ping Pong PHY in the UltraScale architecture. It includes the Ping Pong PHY overview, configuration supported, and interface.

RECOMMENDED: *The Ping Pong PHY is based on the PHY only design. Read the PHY Only Interface section before starting this section.*

In the Ping Pong PHY, two memory channels are supported. The two channels share most of the control/address signals except **`CS_n`**, **`CKE`**, and **`ODT`** are duplicated for Channel1. Each channel has its own Data (**`DQ`**/**`DQS`**/**`DM`**) signals. The advantage of using Ping Pong PHY is that the control/address signals are pin saving.

Figure 4-21 shows a Ping Pong PHY design with a total channel width of DQ_WIDTH. The total channel width, DQ_WIDTH, is split into two evenly split channels. Each channel has a width of DQ_WIDTH/2. The solid arrows indicate shared control/address signals. The dashed arrows indicate **`CS_n[1:0]`**, **`CKE[1:0]`**, and **`ODT[1:0]`** connected to two separated channels. The dotted arrows indicate **`DQ`**/**`DQS`**/**`DM`** signals connected to two separated channels.

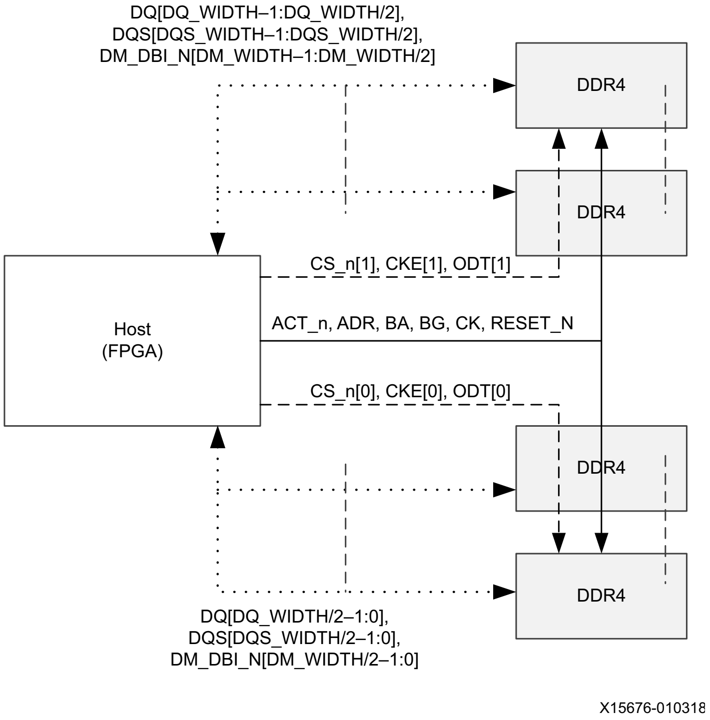


Figure 4-21: **Ping Pong PHY Topology in DDR4**

Supported Configuration

The following rules outline the configuration supported by the Ping Pong PHY:

1. The number of channels supported is two.
2. Supports for up to two ranks.
3. Supports for components only which includes twin-die components.
4. Memory components supported are x4, x8, and x16.
5. Each channel only has device(s) with the same device width (x4, x8, or x16).
6. Each channel has the same width and the same configuration. Each channel width must be a multiple of eight.
7. The maximum total channel width (DQ_WIDTH) of both channels is 64-bit.
8. Pin allocation is based on the Memory IP rules. Address/control signals map to any bank which is similar to the Memory IP pin rules. Skip bytes and mix of two channel data byte groups are allowed.
9. Ping Pong PHY design should follow the PCB layout requirement as a regular PHY only design in total channel width (DQ_WIDTH).
10. One MMCM is instantiated in the middle bank.
11. CAS command can be issued to Command Slot0/Slot2 only

    Note: The same restriction applies to PHY only designs.
12. Command issued should meet JEDEC timing specification per channel.
13. You have the option to share a **`CKE`** pin. In the case when the shared **`CKE`** is enabled, **`CKE[0]`** is used for both channels. When **`CKE`** sharing is enabled, connect Ch0 **`CKE`** to Ch1 as well. If you need to use power down mode, the same command needs to be issued to both channels.
14. Chip select disable is not available for Ping Pong PHY because chip select is used to distinguish if a given command is sent to Channel0 or Channel1.
15. I/O pin planner byte selection view is the same as the regular Memory IP. You must map Channel-0 to DQ[DQ_WIDTH/2 – 1:0] and Channel-1 to DQ[DQ_WIDTH – 1:DQ_WIDTH/2].

Table 4-78: **Ping Pong PHY Configuration Summary**

Width/Device	x4	x8	x16
16	X	X	–
32	X	X	X
48	X	X	–
64	X	X	X

Ping Pong PHY Interface

The Ping Pong PHY interface is very similar to the PHY only interface except command/address signals are shared by both Channel0 and Channel1 in the Ping Pong PHY. Because command/address signals are shared between Channel0 and Channel1, they are qualified separately by `CS_n`, `CKE`, and `ODT` per channel.

Table 4-79 to Table 4-82 show the Ping Pong PHY signal interfaces.

Table 4-79: **Ping Pong PHY Command/Address Interface**

Signal	I/O	Description
mc_ACT_n[7:0]	I	DRAM ACT_n command signal for four DRAM clock cycles. Bits[1:0] correspond to the first DRAM clock cycle, Bits[3:2] to the second, Bits[5:4] to the third, and Bits[8:7] to the fourth. For center alignment to the DRAM clock with 1N timing, both bits of a given bit pair should be asserted to the same value. See the timing diagrams for examples (PHY Only Interface). All of the command/address ports in this table follow the same eight bits per DRAM pin format. Active-Low. This signal is not used in DDR3 systems.
mc_ADR [ADDR_WIDTH × 8 – 1:0]	I	DRAM address. There are eight bits in the PHY interface for each address bit on the DRAM bus. Bits[7:0] corresponds to DRAM address bit zero on four DRAM clock cycles. Bits[15:8] corresponds to DRAM address bit one on four DRAM clock cycles, and so on. See the timing diagrams for examples (PHY Only Interface). All of the multi-bit DRAM signals in this table follow the same format of one byte of the PHY interface port corresponding to four commands for one DRAM pin. Mixed active-Low and High depending on which type of DRAM command is being issued, but follows the DRAM pin active-High/Low behavior. The function of each byte of the mc_ADR port depends on whether the memory type is DDR4 or DDR3, and the particular DRAM command that is being issued. These functions match the DRAM address pin functions. For example, with DDR4 memory and the mc_ACT_n port bits asserted High, mc_ADR[135:112] have the function of RAS_n, CAS_n, and WE_n pins.
mc_RAS_n[7:0]	I	DDR3 DRAM RAS_n pin. Not used in DDR4 systems.
mc_CAS_n[7:0]	I	DDR3 DRAM CAS_n pin. Not used in DDR4 systems.
mc_WE_n[7:0]	I	DDR3 DRAM WE_n pin. Not used in DDR4 systems.
mc_BA [BANK_WIDTH × 8 – 1:0]	I	DRAM bank address. Eight bits for each DRAM bank address.
mc_BG [BANK_GROUP_WIDTH × 8 – 1:0]	I	DRAM bank group address. Eight bits for each DRAM pin.
mc_CKE [2 × CKE_WIDTH × 8 – 1:0]	I	DRAM CKE. Eight bits for each DRAM pin. mc_CKE has a width of CKE_WIDTH × 8 if "Is CKE to be shared across 2 channels" option is enabled in Vivado IDE. In Ping Pong PHY, bits [CKE_WIDTH × 8/2 – 1:0] is used for Channel0, bits [CKE_WIDTH × 8 – 1:CKE_WIDTH × 8/2] is used for Channel1. In case of dual-rank design, mc_CKE is defines as {Ch1-CKE1, Ch1-CKE0, Ch0-CKE1, Ch0-CKE0}.

Table 4-79: **Ping Pong PHY Command/Address Interface *(Cont'd)***

Signal	I/O	Description
mc_CS_n [2 × CS_WIDTH × 8 – 1:0]	I	DRAM CS_n. Eight bits for each DRAM pin. Active-Low. In Ping Pong PHY, bits [CS_WIDTH × 8/2 – 1:0] is used for Channel0, bits [CS_WIDTH × 8 – 1:CS_WIDTH × 8/2] is used for Channel1. In case of dual-rank design, mc_CS_n is defines as {Ch1-CS1, Ch1-CS0, Ch0-CS1, Ch0-CS0}.
mc_ODT [2 × ODT_WIDTH × 8 – 1:0]	I	DRAM ODT. Eight bits for each DRAM pin. Active-High. In Ping Pong PHY, bits [ODT_WIDTH × 8/2 – 1:0] is used for Channel0, bits [ODT_WIDTH × 8 – 1:ODT_WIDTH × 8/2] is used for Channel1. In case of dual-rank design, mc_ODT_n is defines as {Ch1-ODT1_n, Ch1-ODT0_n, Ch0-ODT1_n, Ch0-ODT0_n}.
mc_C[LR_WIDTH ×8 – 1:0]	I	DRAM (3DS) Logical rank select address. Eight bits for each DRAM pin.

Table 4-80: **Ping Pong PHY Write Data Interface**

Signal	I/O	Description
wrData [DQ_WIDTH × 8 – 1:0]	I	DRAM write data. There are eight bits for each DQ lane on the DRAM bus. This port transfers data for an entire BL8 write on each system clock cycle. Write data must be provided to the PHY one cycle after the wrDataEn output signal asserts. This protocol must be followed. There is no data buffering in the PHY. For Ping Pong PHY, wrData[DQ_WIDTH × 8/2 – 1:0] corresponds to channel0, wrData[DQ_WIDTH × 8 – 1:DQ_WIDTH × 8/2] corresponds to channel1.
wrDataMask [DM_WIDTH × 8 – 1:0]	I	DRAM write DM/DBI port. There is one bit for each byte of the wrData port, corresponding to one bit for each byte of each burst of a BL8 transfer. wrDataMask is transferred on the same system clock cycle as wrData. Active-High. For DDR3 interface, wrDataMask port appears for Data Mask enabled option in Vivado IDE. For DDR4 interface, wrDataMask port appears in the "Data Mask and DBI" Vivado IDE option values of DM_NO_DBI and DM_DBI_RD. For Ping Pong PHY, wrDataMask[DM_WIDTH × 8/2 – 1:0] corresponds to channel0, wrDataMask[DM_WIDTH × 8 –1: DM_WIDTH × 8/2] corresponds to channel1.
wrDataEn[1:0]	O	Write data required. The PHY asserts this port for one cycle for each write CAS command. Your design must provide wrData and wrDataMask at the PHY input ports on the cycle after wrDataEn asserts. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.

Table 4-80: **Ping Pong PHY Write Data Interface *(Cont'd)***

Signal	I/O	Description
wrDataAddr [2 × DATA_BUF_ADDR_WIDTH – 1:0]	O	Optional control signal. The PHY stores and return a data buffer address for each in-flight write CAS command. The wrDataAddr signal returns the stored addresses. It is only valid when the PHY asserts wrDataEn. You can use this signal to manage the process of sending write data into the PHY for a write CAS command, but this is completely optional. For Ping Pong PHY, wrDataAddr[2 8 DATA_BUF_ADDR_WIDTH – 1: DATA_BUF_ADDR_WIDTH] corresponds to channel1, wrDataAddr[DATA_BUF_ADDR_WIDTH – 1:0] corresponds to channel0.
tCWL[5:0]	O	Optional control signal. This output indicates the CAS write latency used in the PHY.
dBufAdr [2 × DATA_BUF_ADDR_WIDTH – 1:0]	I	Reserved. Should be tied Low.

Table 4-81: **Ping Pong PHY Read Data Interface**

Signal	I/O	Description
rdData [DQ_WIDTH × 8 – 1:0]	O	DRAM read data. There are eight bits for each DQ lane on the DRAM bus. This port transfers data for an entire BL8 read on each system clock cycle. rdData is only valid when rdDataEn is asserted. Your design must consume the read data when rdDataEn asserts. There is no data buffering in the PHY. For Ping Pong PHY, rdData[DQ_WIDTH × 8/2 – 1:0] corresponds to channel0, rdData[DQ_WIDTH × 8 – 1:DQ_WIDTH × 8/2] corresponds to channel1.
rdDataEn[1:0]	O	Read data valid. This signal asserts for one system clock cycle for each completed read operation, indicating that the rdData, rdDataAddr, per_rd_done, and rmw_rd_done signals are valid. These signals are only valid when rdDataEn asserts. rdData must be consumed when rdDataEn asserts or data will be lost. Active-High. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
rdDataAddr [2 × DATA_BUF_ADDR_WIDTH – 1:0]	O	Optional control signal. The PHY stores and returns a data buffer address for each in-flight read CAS command. The rdDataAddr signal returns the stored addresses. It is only valid when the PHY asserts rdDataEn. Your design can use this signal to manage the process of capturing and storing read data provided by the PHY, but this is completely optional. For Ping Pong PHY, rdDataAddr[2 × DATA_BUF_ADDR_WIDTH – 1: DATA_BUF_ADDR_WIDTH] corresponds to channel1, rdDataAddr[DATA_BUF_ADDR_WIDTH – 1:0] corresponds to channel0.

Table 4-81: **Ping Pong PHY Read Data Interface *(Cont'd)***

Signal	I/O	Description
per_rd_done[1:0]	O	Optional read status signal. The PHY stores and returns the winInjTxn signal on per_rd_done. The Memory IP generated controller uses this bit to indicate the return of a "periodic read", and to distinguish this from normal system traffic. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
rmw_rd_done[1:0]	O	Optional read status signal. The PHY stores and returns the winRmw signal on rmw_rd_done. The Memory IP generated controller uses this bit to indicate the return of a read for a read-modify-write flow, and to distinguish this from normal system traffic. For Ping Pong PHY, bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
rdDataEnd[1:0]	O	Unused. Tied High.

Table 4-82: **Ping Pong PHY Control Interface**

Signal	I/O	Description
calDone	O	Indication that the DRAM is powered up, initialized, and calibration is complete. This indicates that the PHY interface is available to send commands to the DRAM. Active-High.
mcRdCAS[1:0]	I	Read CAS command issued. This signal must be asserted for one system clock if and only if a read CAS command is asserted on one of the command slots at the PHY command/address input ports. Hold at 0x0 until calDone asserts. Active-High. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
mcWrCAS[1:0]	I	Write CAS command issued. This signal must be asserted for one system clock if and only if a write CAS command is asserted on one of the command slots at the PHY command/address input ports. Hold at 0x0 until calDone asserts. Active-High. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
winRank[3:0]	I	Target rank for CAS commands. This value indicates which rank a CAS command is issued to. It must be valid when either mcRdCAS or mcWrCAS is asserted. The PHY passes the value from this input to the XIPHY to select the calibration results for the target rank of a CAS command in multi-rank systems. In a single-rank system this input port can be tied to 0x0. For Ping Pong PHY, Bit[1:0] corresponds to channel0, Bit[3:2] corresponds to channel1.

Table 4-82: **Ping Pong PHY Control Interface *(Cont'd)***

Signal	I/O	Description
mcCasSlot[3:0]	I	CAS command slot select. The PHY only supports CAS commands on even command slots. mcCasSlot indicates which of these two possible command slots a read CAS or write CAS was issued on. mcCasSlot is used by the PHY to generate XIPHY control signals, like DQ output enables, that need DRAM clock cycle resolution relative to the command slot used for a CAS command. Valid values after calDone asserts are 0x0 and 0x2. Hold at 0x0 until calDone asserts. This signal must be valid if mcRdCAS or mcWrCAS is asserted. For more information, see CAS Command Timing Limitations. For Ping Pong PHY, Bit[1:0] corresponds to channel0, Bit[3:2] corresponds to channel1.
mcCasSlot2[1:0]	I	CAS slot 2 select. mcCasSlot2 serves a similar purpose as the mcCasSlot[1:0] signal, but mcCasSlot2 is used in timing critical logic in the PHY. Ideally mcCasSlot2 should be driven from separate flops from mcCasSlot[1:0] to allow synthesis/implementation to better optimize timing. mcCasSlot2 and mcCasSlot[1:0] must always be consistent if mcRdCAS or mcWrCAS is asserted. To be consistent, the following must be true: mcCasSlot2==mcCasSlot[1]. Hold at 0x0 until calDone asserts. Active-High. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
winInjTxn[1:0]	I	Optional read command type indication. When mcRdCAS is asserted, the PHY stores the value of winInjTxn and returns the value when read data is output. The return value is driven on the per_rd_done PHY output port. In Memory IP controller designs, the winInjTxn/per_rd_done signals are used to track non-system read traffic by asserting winInjTxn only on read commands issued for the purpose of VT tracking. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.
winRmw[1:0]	I	Optional read command type indication. When mcRdCAS is asserted, the PHY stores the value of winRmw and returns the value when read data is output. The return value is driven on the rmw_rd_done output port. In Memory IP controller designs, the winRmw/rmw_rd_done signals are used to track reads issued as part of a read-modify-write flow. The Memory IP controller asserts winRmw only on read commands that are issued for the read phase of a RMW sequence. For Ping Pong PHY, Bit[0] corresponds to channel0, Bit[1] corresponds to channel1.

Table 4-82: **Ping Pong PHY Control Interface *(Cont'd)***

Signal	I/O	Description
winBuf [2 × DATA_BUF_ADDR_WIDTH – 1:0]	I	Optional control signal. When either mcRdCAS or mcWrCAS is asserted, the PHY stores the value on the winBuf signal. The value is returned on rdDataAddr or wrDataAddr, depending on whether mcRdCAS or mcWrCAS was used to capture winBuf. In Memory IP controller designs, these signals are used to track the data buffer address used to source write data or sink read return data. For Ping Pong PHY, winBuf[2 × DATA_BUF_ADDR_WIDTH – 1: DATA_BUF_ADDR_WIDTH] corresponds to channel1, winBuf[DATA_BUF_ADDR_WIDTH – 1:0] corresponds to channel0.
gt_data_ready	I	Update VT Tracking. This signal triggers the PHY to read RIU registers in the XIPHY that measure how well the DQS Gate signal is aligned to the center of the read DQS preamble, and then adjust the alignment if needed. This signal must be asserted periodically to keep the DQS Gate aligned as voltage and temperature drift. For more information, see VT Tracking. Hold at 0x0 until calDone asserts. Active-High.

Performance

The efficiency of a memory system is affected by many factors including limitations due to the memory, such as cycle time (tRC) within a single bank, or Activate to Activate spacing to the same DDR4 bank group (tRRD_L). When given multiple transactions to work on, the Memory Controller schedules commands to the DRAM in a way that attempts to minimize the impact of these DRAM timing requirements. But there are also limitations due to the Memory Controller architecture itself. This section explains the key controller limitations and options for obtaining the best performance out of the controller.

Address Map

The **`app_addr`** to the DRAM address map is described in the User Interface. Six mapping options are included:

- ROW_COLUMN_BANK
- ROW_BANK_COLUMN
- BANK_ROW_COLUMN
- ROW_COLUMN_LRANK_BANK
- ROW_LRANK_COLUMN_BANK
- ROW_COLUMN_BANK_INTLV

For a purely random address stream at the user interface, all of the options would result in a similar efficiency. For a sequential `app_addr` address stream, or any workload that tends to have a small stride through the `app_addr` memory space, the ROW_COLUMN_BANK mapping generally provides a better overall efficiency. This is due to the Memory Controller architecture and the interleaving of transactions across the Group FSMs. The Group FSMs are described in the Memory Controller, page 25. This controller architecture impact on efficiency should be considered even for situations where DRAM timing is not limiting efficiency. Table 4-83 shows two mapping options for the 4 Gb (x8) DRAM components.

Table 4-83: **DDR3/DDR4 4 Gb (x8) DRAM Address Mapping without 3DS Options**

DRAM Address	DDR3 4 Gb (x8)		DDR4 4 Gb (x8)	
	ROW_BANK_COLUMN	ROW_COLUMN_BANK	ROW_BANK_COLUMN	ROW_COLUMN_BANK
Row 15	28	28	–	–
Row 14	27	27	28	28
Row 13	26	26	27	27
Row 12	25	25	26	26
Row 11	24	24	25	25
Row 10	23	23	24	24
Row 9	22	22	23	23
Row 8	21	21	22	22
Row 7	20	20	21	21
Row 6	19	19	20	20
Row 5	18	18	19	19
Row 4	17	17	18	18
Row 3	16	16	17	17
Row 2	15	15	16	16
Row 1	14	14	15	15
Row 0	13	13	14	14
Column 9	9	12	9	13
Column 8	8	11	8	12
Column 7	7	10	7	11
Column 6	6	9	6	10
Column 5	5	8	5	9
Column 4	4	7	4	8
Column 3	3	6	3	7
Column 2	2	2	2	2
Column 1	1	1	1	1
Column 0	0	0	0	0

Table 4-83: **DDR3/DDR4 4 Gb (x8) DRAM Address Mapping without 3DS Options *(Cont'd)***

DRAM Address	DDR3 4 Gb (x8)		DDR4 4 Gb (x8)	
	ROW_BANK_COLUMN	ROW_COLUMN_BANK	ROW_BANK_COLUMN	ROW_COLUMN_BANK
Bank 2	12	4	–	–
Bank 1	11	3	11	6
Bank 0	10	5	10	5
Bank Group 1	–	–	13	4
Bank Group 0	–	–	12	3

Note: Highlighted bits are used to map addresses to Group FSMs in the controller.

From the DDR3 map, you might expect reasonable efficiency with the ROW_BANK_COLUMN option with a simple address increment pattern. The increment pattern would generate page hits to a single bank, which DDR3 could handle as a stream of back-to-back CAS commands resulting in high efficiency. But looking at the italic bank bits in Table 4-84 show that the address increment pattern also maps the long stream of page hits to the same controller Group FSM.

For example, Table 4-84 shows how the first 12 **`app_addr`** addresses decode to the DRAM addresses and map to the Group FSMs for both mapping options. The ROW_BANK_COLUMN option only maps to the Group FSM 0 over this address range.

Table 4-84: **DDR3 4 Gb (x8) app_addr Mapping Options**

app_addr	DDR3 4 Gb (x8) ROW_BANK_COLUMN				DDR3 4 Gb (x8) ROW_COLUMN_BANK			
	Row	Column	Bank	Group_FSM	Row	Column	Bank	Group_FSM
0x58	0x0	0x58	0x0	0	0x0	0x8	0x6	3
0x50	0x0	0x50	0x0	0	0x0	0x8	0x4	2
0x48	0x0	0x48	0x0	0	0x0	0x8	0x2	1
0x40	0x0	0x40	0x0	0	0x0	0x8	0x0	0
0x38	0x0	0x38	0x0	0	0x0	0x0	0x7	3
0x30	0x0	0x30	0x0	0	0x0	0x0	0x5	2
0x28	0x0	0x28	0x0	0	0x0	0x0	0x3	1
0x20	0x0	0x20	0x0	0	0x0	0x0	0x1	0
0x18	0x0	0x18	0x0	0	0x0	0x0	0x6	3
0x10	0x0	0x10	0x0	0	0x0	0x0	0x4	2
0x8	0x0	0x8	0x0	0	0x0	0x0	0x2	1
0x0	0x0	0x0	0x0	0	0x0	0x0	0x0	0

The same address to Group FSM mapping issue applies to x16 DRAMs. The map for DDR4 4 Gb (x16) is shown in Table 4-85. The ROW_COLUMN_BANK option gives the best efficiency with sequential address patterns. The bits used to map to the Group FSMs are highlighted.

Table 4-85: **DDR4 4 Gb (x16) Address Mapping**

DRAM Address	ROW_BANK_COLUMN	ROW_COLUMN_BANK
Row 15	–	–
Row 14	27	27
Row 13	26	26
Row 12	25	25
Row 11	24	24
Row 10	23	23
Row 9	22	22
Row 8	21	21
Row 7	20	20
Row 6	19	19
Row 5	18	18
Row 4	17	17
Row 3	16	16
Row 2	15	15
Row 1	14	14
Row 0	13	13
Column 9	9	12
Column 8	8	11
Column 7	7	10
Column 6	6	9
Column 5	5	8
Column 4	4	7
Column 3	3	6
Column 2	2	2
Column 1	1	1
Column 0	0	0
Bank 2	–	–
Bank 1	12	5
Bank 0	11	4
Bank Group 1	–	–
Bank Group 0	10	3

For example, Table 4-86 shows how the first 12 `app_addr` decodes to the DRAM address and maps to the Group FSMs for the ROW_COLUMN_BANK mapping option.

Table 4-86: **DDR4 4 Gb (x16) app_addr Mapping Options**

app_addr	ROW_COLUMN_BANK			
	Row	**Column**	**Bank**	**Group_FSM**
0xF8	0x0	0x18	0x3	3
0XE8	0x0	0x18	0x2	2
0XF0	0x0	0x18	0x3	1
0XE0	0x0	0x18	0x2	0
0X78	0x0	0x8	0x3	3
0X68	0x0	0x8	0x2	2
0X70	0x0	0x8	0x3	1
0x60	0x0	0x8	0x2	0
0x38	0x0	0x0	0x3	3
0x28	0x0	0x0	0x2	2
0x30	0x0	0x0	0x3	1
0x20	0x0	0x0	0x2	0

As mentioned in the Memory Controller, page 25, a Group FSM can issue one CAS command every three system clock cycles, or every 12 DRAM clock cycles, even for page hits. Therefore with only a single Group FSM issuing page hit commands to the DRAM for long periods, the maximum efficiency is 33%.

Table 4-84 shows that the ROW_COLUMN_BANK option maps these same 12 addresses evenly across all eight DRAM banks and all four controller Group FSMs. This generates eight “page empty” transactions which open up all eight DRAM banks, followed by page hits to the open banks.

With all four Group FSMs issuing page hits, the efficiency can hit 100%, for as long as the address increment pattern continues, or until a refresh interrupts the pattern, or there is bus dead time for a DQ bus turnaround, etc. Figure 4-22 shows the Group FSM issue over a larger address range for the ROW_BANK_COLUMN option. Note that the first 2k addresses map to two DRAM banks, but only one Group FSM.

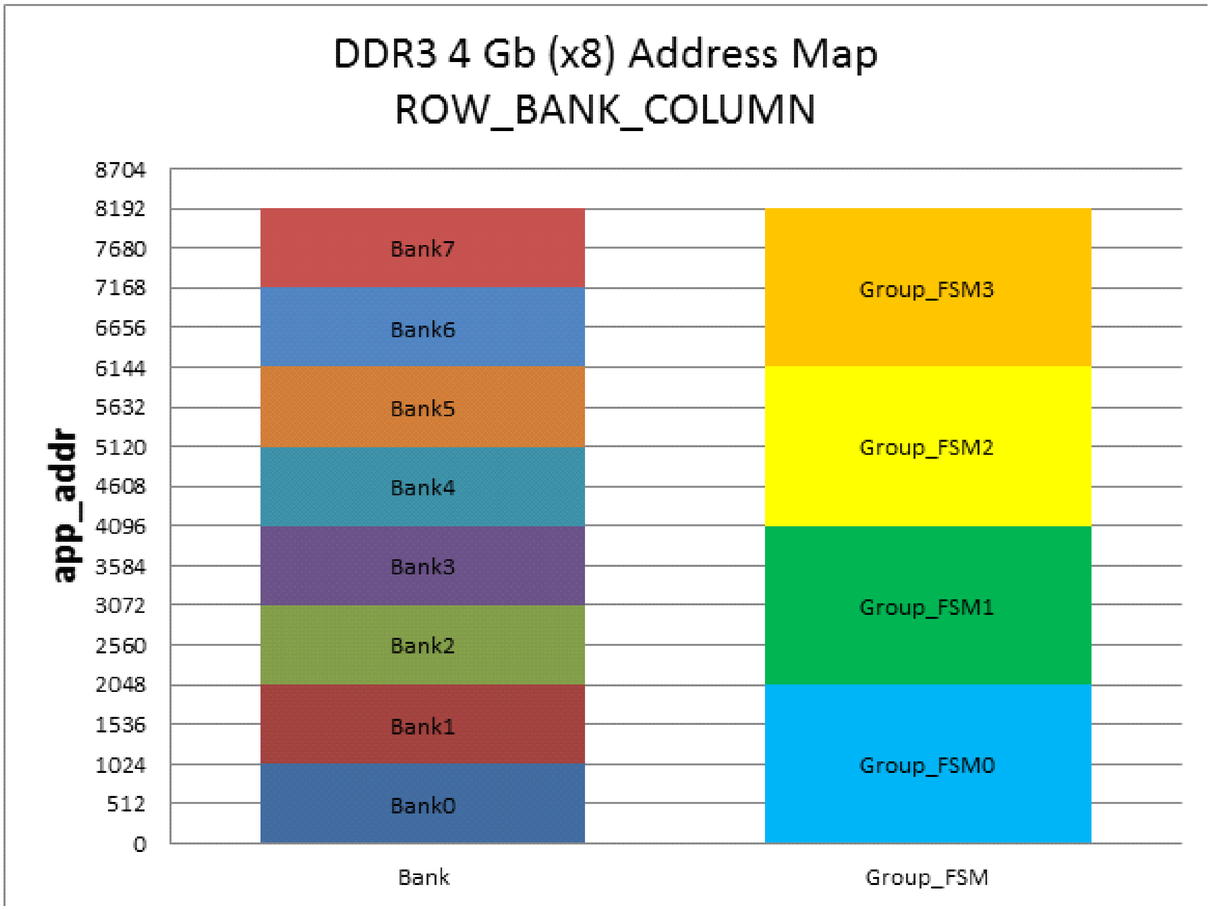


Figure 4-22: **DDR3 4 Gb (x8) Address Map ROW_BANK_COLUMN Graph**

The address map graph for the ROW_COLUMN_BANK option is shown in Figure 4-23. Note that the address range in this graph is only 64 bytes, not 8k bytes. This graph is showing the same information as in the Address Decode in Table 4-84. With an address pattern that tends to stride through memory in minimum sized steps, efficiency tends to be High with the ROW_COLUMN_BANK option.

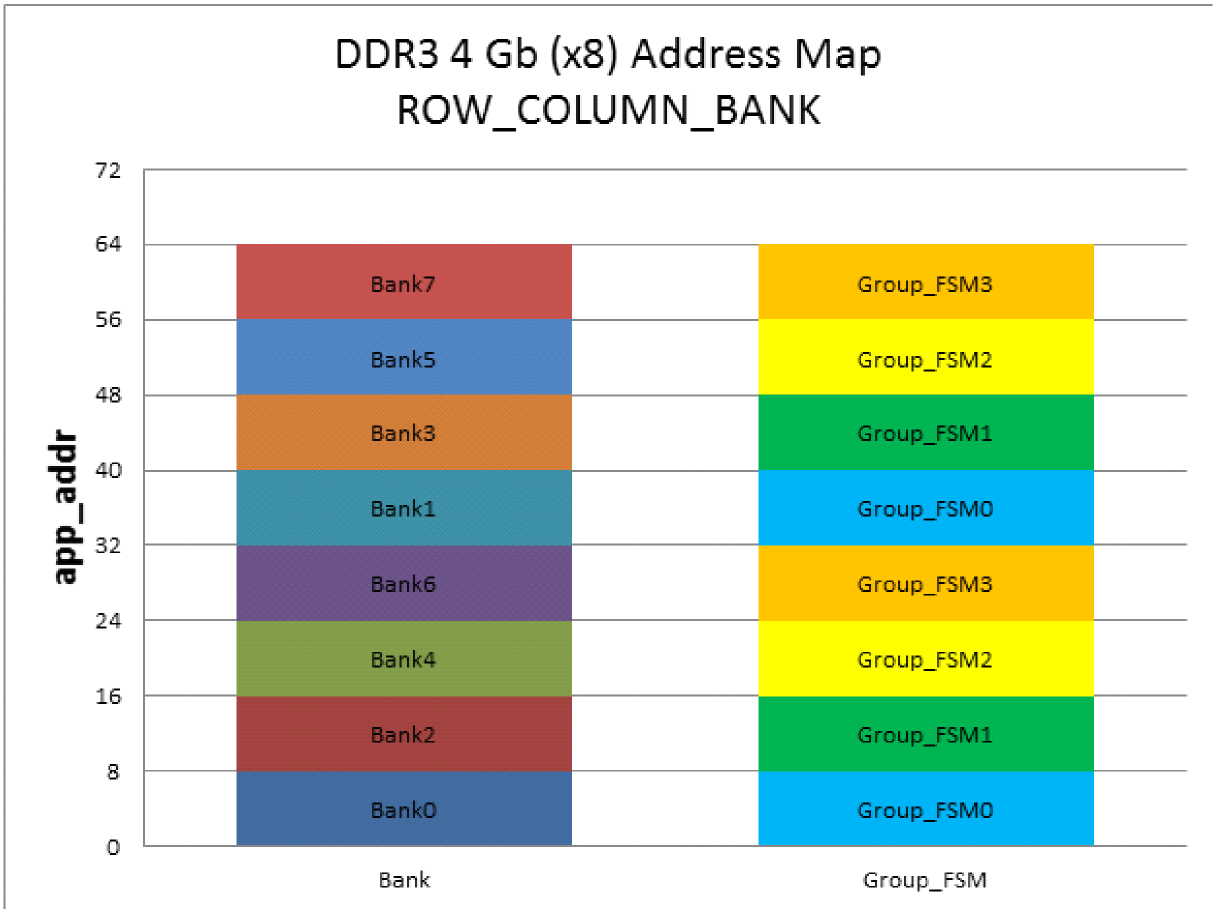


Figure 4-23: **DDR3 4 Gb (x8) Address Map ROW_COLUMN_BANK Graph**

Note that the ROW_COLUMN_BANK option does not result in High bus efficiency for all strides through memory. Consider the case of a stride of 16 bytes. This maps to only two Group FSMs resulting in a maximum efficiency of 67%. A stride of 32 bytes maps to only one Group FSM and the maximum efficiency is the same as the ROW_BANK_COLUMN option, just 33%. For an address pattern with variable strides, but strides that tend to be < 1k in the **`app_addr`** address space, the ROW_COLUMN_BANK option is much more likely to result in good efficiency.

The same Group FSM issue exists for DDR4. With an address increment pattern and the DDR4 ROW_BANK_COLUMN option, the first 4k transactions map to a single Group FSM, as well as mapping to banks within a single DRAM bank group. The DRAM would limit the address increment pattern efficiency due to the tCCD_L timing restriction. The controller limitation in this case is even more restrictive, due to the single Group FSM. Again the efficiency would be limited to 33%.

With the ROW_COLUMN_BANK option, the address increment pattern interleaves across all the DRAM banks and bank groups and all of the Group FSMs over a small address range.

Figure 4-24 shows how the DDR4 4 Gb (x8) ROW_COLUMN_BANK address map for the first 128 bytes of `app_addr`. This graph shows how the addresses map evenly across all DRAM banks and bank groups, and all four controller Group FSMs.

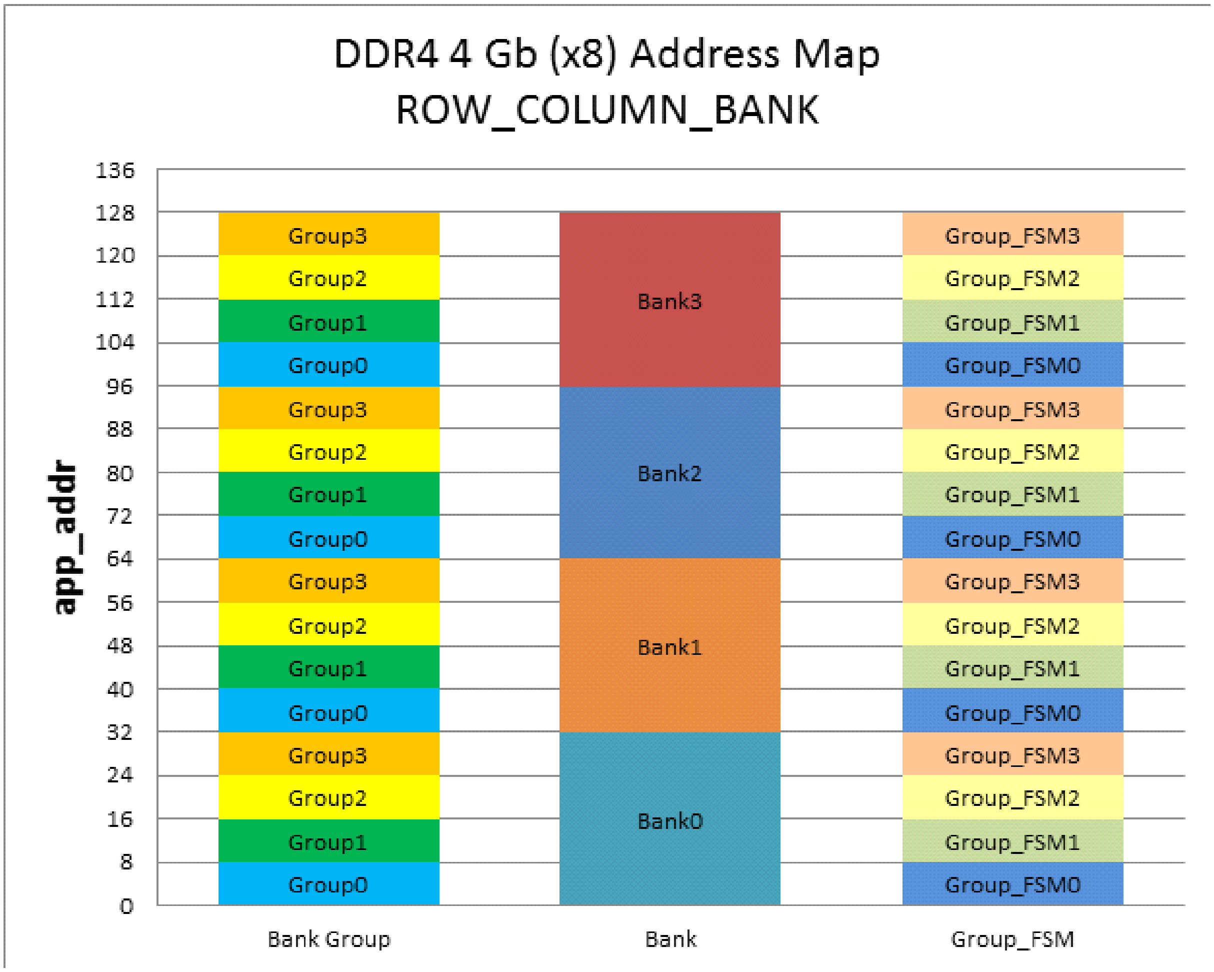


Figure 4-24: **DDR4 4 Gb (x8) Address Map ROW_COLUMN_BANK Graph**

Figure 4-25 shows the first 64 bytes of **`app_addr`** mapping evenly across banks, bank groups, and Group FSMs.

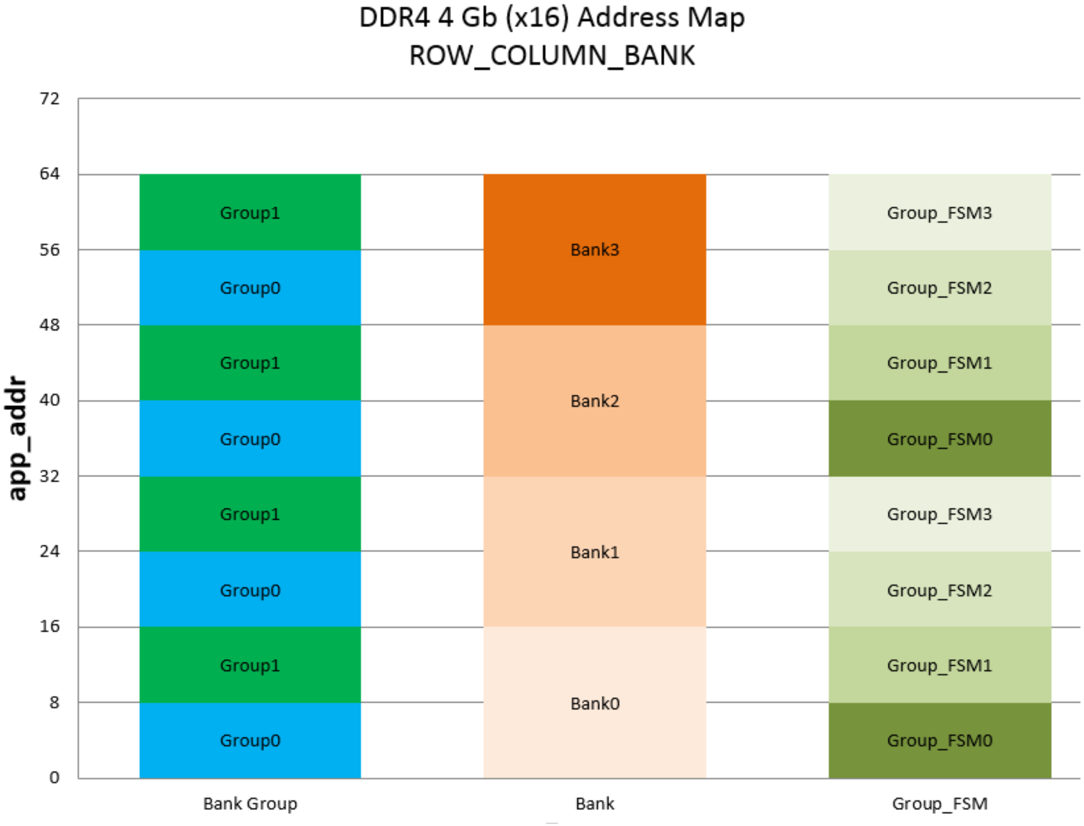


Figure 4-25: **DDR4 4 Gb (x16) Address Map ROW_COLUMN_BANK Graph**

When considering whether an address pattern at the user interface results in good DRAM efficiency, the mapping of the pattern to the controller Group FSMs is just as important as the mapping to the DRAM address. The **`app_addr`** bits that map **`app_addr`** addresses to the Group FSMs are shown in Table 4-87 for 4 Gb and 8 Gb components.

Table 4-87: **DDR3/DDR4 Map Options for 4 Gb and 8 Gb**

Memory Type	**DDR3**				**DDR4**		
Map Option	**ROW_BANK_COLUMN**			**ROW_COLUMN_BANK**	**ROW_BANK_COLUMN**		**ROW_COLUMN_BANK**
DRAM Component Width	x4	x8	x16	x4, x8, x16	x4, x8	x16	x4, x8, x16
Component Density	–	–	–	–	–	–	–
4 Gb	13,12	12,11	12,11	4,3	13,12	12,10	4,3
8 Gb	14,13	13,12	12,11	4,3	13,12	12,10	4,3

Consider an example where you try to obtain good efficiency using only four DDR3 banks at a time. Assume you are using a 4 Gb (x8) with the ROW_COLUMN_BANK option and you decide to open a page in banks 0, 1, 2, and 3, and issue transactions to four column addresses in each bank. Using the address map from Address Map, determine the **`app_addr`** pattern that decodes to this DRAM sequence. Applying the Group FSM map from Table 4-87, determine how this **`app_addr`** pattern maps to the FSMs. The result is shown in Table 4-88.

Table 4-88: **Four Banks Sequence on DDR3 4 Gb (x8)**

app_addr	**Bank 0, 1, 2, 3 Sequence DDR3 4 Gb (x8) ROW_COLUMN_BANK**			
	Row	**Column**	**Bank**	**Group_FSM**
0xE8	0x0	0x18	0x3	1
0xC8	0x0	0x18	0x2	1
0xE0	0x0	0x18	0x1	0
0xC0	0x0	0x18	0x0	0
0xA8	0x0	0x10	0x3	1
0x88	0x0	0x10	0x2	1
0xA0	0x0	0x10	0x1	0
0x80	0x0	0x10	0x0	0
0x68	0x0	0x8	0x3	1
0x48	0x0	0x8	0x2	1
0x60	0x0	0x8	0x1	0
0x40	0x0	0x8	0x0	0
0x28	0x0	0x0	0x3	1
0x08	0x0	0x0	0x2	1
0x20	0x0	0x0	0x1	0
0x00	0x0	0x0	0x0	0

The four bank pattern in Table 4-88 works well from a DRAM point of view, but the controller only uses two of its four Group FSMs and the maximum efficiency is 67%. In practice it is even lower due to other timing restrictions like tRCD. A better bank pattern would be to open all the even banks and send four transactions to each as shown in Table 4-89.

Table 4-89: **Four Even Banks Sequence on DDR3 4 Gb (x8)**

app_addr	**Bank 0, 2, 4, 6 Sequence DDR3 4 Gb (x8) ROW_COLUMN_BANK**			
	Row	**Column**	**Bank**	**Group_FSM**
0xD8	0x0	0x18	0x6	3
0xD0	0x0	0x18	0x4	2
0xC8	0x0	0x18	0x2	1

Table 4-89: **Four Even Banks Sequence on DDR3 4 Gb (x8)** ***(Cont'd)***

app_addr	Bank 0, 2, 4, 6 Sequence DDR3 4 Gb (x8) ROW_COLUMN_BANK			
	Row	**Column**	**Bank**	**Group_FSM**
0xC0	0x0	0x18	0x0	0
0x98	0x0	0x10	0x6	3
0x90	0x0	0x10	0x4	2
0x88	0x0	0x10	0x2	1
0x80	0x0	0x10	0x0	0
0x58	0x0	0x8	0x6	3
0x50	0x0	0x8	0x4	2
0x48	0x0	0x8	0x2	1
0x40	0x0	0x8	0x0	0
0x18	0x0	0x0	0x6	3
0x10	0x0	0x0	0x4	2
0x08	0x0	0x0	0x2	1
0x00	0x0	0x0	0x0	0

The "even bank" pattern uses all of the Group FSMs and therefore has better efficiency than the previous pattern.

Controller Head of Line Blocking and Look Ahead

As described in the Memory Controller, page 25, each Group FSM has an associated transaction FIFO that is intended to improve efficiency by reducing "head of line blocking." Head of line blocking occurs when one or more Group FSMs are fully occupied and cannot accept any new transactions for the moment, but the transaction presented to the user interface command port maps to one of the unavailable Group FSMs. This not only causes a delay in issuing new transactions to those busy FSMs, but to all the other FSMs as well, even if they are idle.

For good efficiency, you want to keep as many Group FSMs busy in parallel as you can. You could try changing the transaction presented to the user interface to one that maps to a different FSM, but you do not have visibility at the user interface as to which FSMs have space to take new transactions. The transaction FIFOs prevent this type of head of line blocking until a UI command maps to an FSM with a full FIFO.

A Group FSM FIFO structure can hold up to six transactions, depending on the page status of the target rank and bank. The FIFO structure is made up of two stages that also implement a "Look Ahead" function. New transactions are placed in the first FIFO stage and are operated on when they reach the head of the FIFO. Then depending on the transaction page status, the Group FSM either arbitrates to open the transaction page, or if the page is already open, the FSM pushes the page hit into the second FIFO stage. This scheme allows multiple page hits to be queued up while the FSM looks ahead into the logical FIFO structure for pages that need to be opened. Looking ahead into the queue allows an FSM to interleave DRAM commands for multiple transactions on the DDR bus. This helps to hide DRAM tRCD and tRP timing associated with opening and closing pages.

The following conceptual timing diagram shows the transaction flow from the UI to the DDR command bus, through the Group FSMs, for a series of transactions. The diagram is conceptual in that the latency from the UI to the DDR bus is not considered and not all DRAM timing requirements are met. Although not completely timing accurate, the diagram does follow DRAM protocol well enough to help explain the controller features under discussion.

Four transactions are presented at the UI, the first three mapping to the Group FSM0 and the fourth to FSM1. On system clock cycle 1, FSM0 accepts transaction 1 to Row 0, Column 0, and Bank 0 into its stage 1 FIFO and issues an Activate command.

On clock 2, transaction 1 is moved into the FSM0 stage 2 FIFO and transaction 2 is accepted into FSM0 stage 1 FIFO. On clock cycles 2 through 4, FSM0 is arbitrating to issue a CAS command for transaction 1, and an Activate command for transaction 2. FSM0 is looking ahead to schedule commands for transaction 2 even though transaction 1 is not complete. Note that the time when these DRAM commands win arbitration is determined by DRAM timing such as tRCD and controller pipeline delays, which explains why the commands are spaced on the DDR command bus as shown.

On cycle 3, transaction 3 is accepted into FSM0 stage 1 FIFO, but it is not processed until clock cycle 5 when it comes to the head of the stage 1 FIFO. Cycle 5 is where FSM0 begins looking ahead at transaction 3 while also arbitrating to issue the CAS command for transaction 2. Finally on cycle 4, transaction 4 is accepted into FSM1 stage 1 FIFO. If FSM0 did not have at least a three deep FIFO, transaction 4 would have been blocked until cycle 6.

Table 4-90: **Conceptional Timing Diagram for UI to DDR**

Transaction Flow													
System Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
UI Transaction Number	1	2	3	4	–	–	–	–	–	–	–	–	–

Table 4-90: **Conceptional Timing Diagram for UI to DDR *(Cont'd)***

Transaction Flow													
UI Transaction	R0, C0, B0	R0, C0, B1	R1, C0, B0	R0, C0, B2	–	–	–	–	–	–	–	–	–
FSM0 FIFO Stage 2	–	R0, C0, B0	R0, C0, B0	R0, C0, B0	R0, C0, B1	R0, C0, B1	R0, B0, B1	–	–	R1, C0, B0	R1, C0, B0	R1, C0, B0	–
FSM0 FIFO Stage 1	R0, C0, B0	R0, C0, B1	R0, C0, B1 R1, C0, B0	R0, C0, B1 R1, C0, B0	R1, C0, B0	R1, C0, B0	R1, C0, B0	R1, C0, B0	R1, C0, B0	–	–	–	–
FSM1 FIFO Stage 2	–	–	–	–	–	R0, C0, B2	R0, C0, B2	R0, C0, B2	–	–	–	–	–
FSM1 FIFO Stage 1	–	–	–	R0, C0, B2	R0, C0, B2	–	–	–	–	–	–	–	–
DDR Command Bus	Act R0, B0	–	–	Act R0, B1	ACT R0, B2 CAS C0, B0	Pre B0	–	CAS C0, B1	Act R1, B0 CAS C0, B2	–	–	–	CAS C0, B0

This diagram does not show a high efficiency transaction pattern. There are no page hits and only two Group FSMs are involved. But the example does show how a single Group FSM interleaves DRAM commands for multiple transactions on the DDR bus and minimizes blocking of the UI, thereby improving efficiency.

Autoprecharge

The Memory Controller defaults to a page open policy. It leaves banks open, even when there are no transactions pending. It only closes banks when a refresh is due, a page miss transaction is being processed, or when explicitly instructed to issue a transaction with a RDA or WRA CAS command. The `app_autoprecharge` port on the UI allows you to explicitly instruct the controller to issue a RDA or WRA command in the CAS command phase of processing a transaction, on a per transaction basis. You can use this signal to improve efficiency when you have knowledge of what transactions will be sent to the UI in the future.

The following diagram is a modified version of the "look ahead" example from the previous section. The page miss transaction that was previously presented to the UI in cycle 3 is now moved out to cycle 9. The controller can no longer "look ahead" and issues the Precharge to Bank 0 in cycle 6 because it does not know about the page miss until cycle 9. But if you know that transaction 1 in cycle 1 is the only transaction to Row 0 in Bank0, assert the `app_autoprecharge` port in cycle 1. Then, the CAS command for transaction 1 in cycle 5 is a RDA or WRA, and the transaction to Row 1, Bank 0 in cycle 9 is no longer a page miss. The transaction in cycle 9 is only needed as an Activate command instead of a Precharge followed by an Activate tRP later.

Table 4-91: **Conceptional Timing Diagram with Autoprecharge**

Transaction Flow													
System Clock Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
UI Transaction Number	1	2	–	3	–	–	–	–	4	–	–	–	–
UI Transaction	R0, C0, B0 AutoPrecharge	R0, C0, B1	–	R0, C0, B2	–	–	–	–	R1, C0, B0	–	–	–	–
FSM0 FIFO Stage 2	–	R0, C0, B0	R0, C0, B0	R0, C0, B0	R0, C0, B1	R0, C0, B1	R0, B0, B1	–	–	R1, C0, B0	R1, C0, B0	R1, C0, B0	–
FSM0 FIFO Stage 1	R0, C0, B0	R0, C0, B1	R0, C0, B1	R0, C0, B1	–	–	–	–	R1, C0, B0	–	–	–	–
FSM1 FIFO Stage 2	–	–	–	–	–	R0, C0, B2	R0, C0, B2	R0, C0, B2	–	–	–	–	–
FSM1 FIFO Stage 1	–	–	–	R0, C0, B2	R0, C0, B2	–	–	–	–	–	–	–	–
DDR Command Bus	Act R0, B0	–	–	Act R0, B1	Act R0, B2 CAS-A C0, B0	–	–	CAS C0, B1	Act R1, B0 CAS C0, B2	–	–	–	CAS C0, B0

A general rule for improving efficiency is to assert `app_autoprecharge` on the last transaction to a page. An extreme example is an address pattern that never generates page hits. In this situation, it is best to assert `app_autoprecharge` on every transactions issued to the UI.

The controller has an option to automatically inject an autoprecharge on a transaction. When the **Force Read and Write commands to use AutoPrecharge** option is selected, the Memory Controller issues a transaction to memory with an AutoPrecharge if Column address bit A3 is set High. This feature disables the `app_autoprecharge` input signal on the User Interface. The **Force** option when used with the ROW_COLUMN_BANK_INTLV address mapping improves efficiency for transaction patterns with bursts of 16 sequential addresses before switching to a different random address. Patterns like this are often seen in typical AXI configurations.

User Refresh and ZQCS

The Memory Controller can be configured to automatically generate DRAM refresh and ZQCS maintenance commands to meet DRAM timing requirements. In this mode, the controller blocks the UI transactions on a regular basis to issue the maintenance commands, reducing efficiency.

If you have knowledge of the UI traffic pattern, you might be able to schedule DRAM maintenance commands with less impact on system efficiency. You can use the **`app_ref`** and **`app_zq`** ports at the UI to schedule these commands when the controller is configured for User Refresh and ZQCS. In this mode, the controller does not schedule the DRAM maintenance commands and only issues them based on the **`app_ref`** and **`app_zq`** ports. You are responsible for meeting all DRAM timing requirements for refresh and ZQCS.

Consider a case where the system needs to move a large amount of data into or out of the DRAM with the highest possible efficiency over a 50 µs period. If the controller schedules the maintenance commands, this 50 µs data burst would be interrupted multiple times for refresh, reducing efficiency roughly 4%. In User Refresh mode, however, you can decide to postpone refreshes during the 50 µs burst and make them up later. The DRAM specification allows up to eight refreshes to be postponed, giving you flexibility to schedule refreshes over a 9 × tREFI period, more than enough to cover the 50 µs in this example.

While User Refresh and ZQCS enable you to optimize efficiency, their incorrect use can lead to DRAM timing violations and data loss in the DRAM. Use this mode only if you thoroughly understand DRAM refresh and ZQCS requirements as well as the operation of the **`app_ref`** and **`app_zq`** UI ports. The UI port operation is described in the User Interface.

Periodic Reads

The FPGA DDR PHY requires at least one DRAM RD or RDA command to be issued every 1 µs. This requirement is described in the User Interface. If this requirement is not met by the transaction pattern at the UI, the controller detects the lack of reads and injects a read transaction into Group FSM0. This injected read is issued to the DRAM following the normal mechanisms of the controller issuing transactions. The key difference is that no read data is returned to the UI. This is wasted DRAM bandwidth.

User interface patterns with long strings of write transactions are affected the most by the PHY periodic read requirement. Consider a pattern with a 50/50 read/write transaction ratio, but organized such that the pattern alternates between 2 µs bursts of 100% page hit reads and 2 µs bursts of 100% page hit writes. There is at least one injected read in the 2 µs write burst, resulting in a loss of efficiency due to the read command and the turnaround time to switch the DRAM and DDR bus from writes to reads back to writes. This 2 µs alternating burst pattern is slightly more efficient than alternating between reads and writes every 1 µs. A 1 µs or shorter alternating pattern would eliminate the need for the controller to inject reads, but there would still be more read-write turnarounds.

Bus turnarounds are expensive in terms of efficiency and should be avoided if possible. Long bursts of page hit writes, > 2 µs in duration, are still the most efficient way to write to the DRAM, but the impact of one write-read-write turnaround each 1 µs must be taken into account when calculating the maximum write efficiency.

DIMM Configurations

DDR3/DDR4 SDRAM memory interface supports UDIMM, RDIMM, LRDIMM, and SODIMM in multiple slot configurations.

IMPORTANT: *Note that the chip select order generated by Vivado is dependent to your board design. Also, the DDR3/DDR4 IP core does not read SPD. If the DIMM configuration changes, the IP must be regenerated.*

In the following configurations, the empty slot is not used and it is optional to be implemented on the board.

DDR3/DDR4 UDIMM/SODIMM

Table 4-92 and Figure 4-26 show the four configurations supported for DDR3/DDR4 UDIMM and SODIMM.

For a dual-rank DIMM, Dual Slot configuration, follow the chip select order shown in Figure 4-26, where CS0 and CS1 are connected to Slot0 and CS2 and CS3 are connected to Slot1.

Table 4-92: **DDR3/DDR4 UDIMM Configuration**

Slot0	Slot1
Single-rank	Empty
Dual-rank	Empty
Dual-rank	Dual-rank
Single-rank	Single-rank

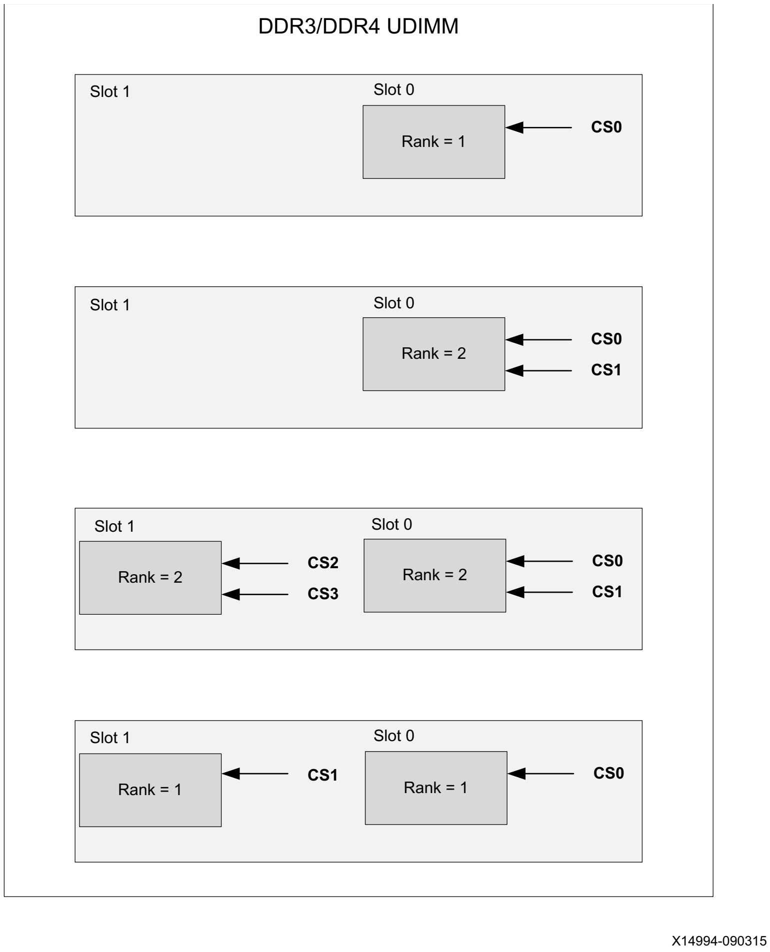


Figure 4-26: **DDR3/DDR4 UDIMM Configuration**

DDR3 RDIMM

Table 4-93 and Figure 4-27 show the five configurations supported for DDR3 RDIMM. DDR3 RDIMM requires two chip selects for a single-rank RDIMM to program the register chip.

For a single-rank DIMM, Dual slot configuration, you must follow the chip select order shown in Figure 4-27, where CS0 and CS2 are connected to Slot0 and CS1 and CS3 are connected to Slot1.

For a dual-rank DIMM, Dual Slot configuration, follow the chip select order shown in Figure 4-27, where CS0 and CS1 are connected to Slot0 and CS2 and CS3 are connected to Slot1.

Table 4-93: **DDR3 RDIMM Configuration**

Slot0	Slot1
Single-rank	Empty
Single-rank	Single-rank

Table 4-93: **DDR3 RDIMM Configuration *(Cont'd)***

Slot0	Slot1
Dual-rank	Empty
Dual-rank	Dual-rank
Quad-rank	Empty

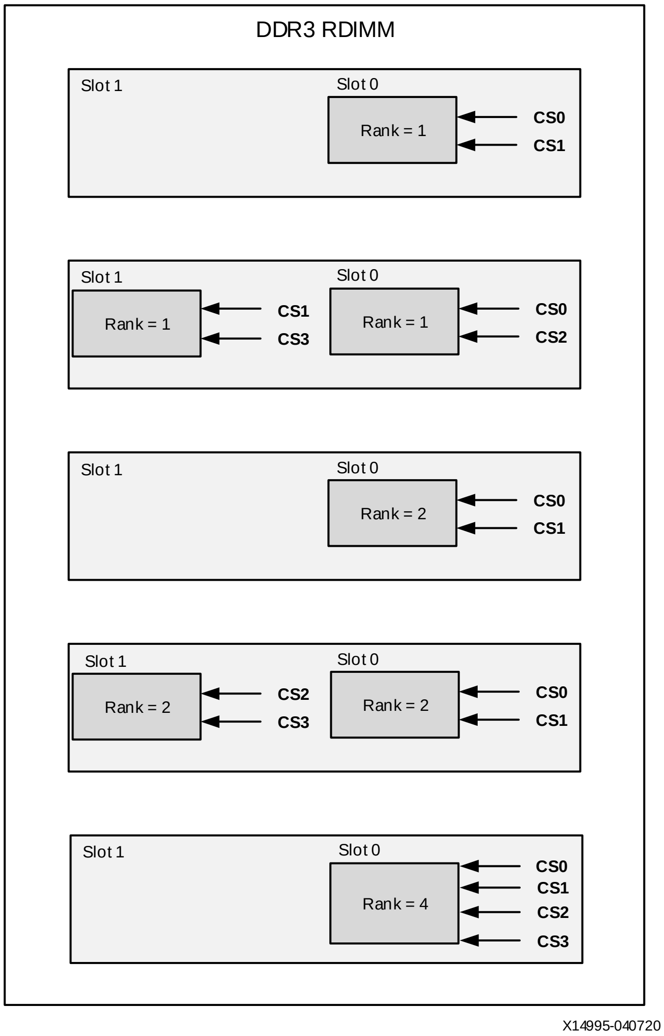


Figure 4-27: **DDR3 RDIMM Configuration**

DDR4 RDIMM

Table 4-94 and Figure 4-28 show the four configurations supported for DDR4 RDIMM. For dual-rank DIMM, Dual Slot configuration, follow the chip select order shown in Figure 4-28, where CS0 and CS1 are connected to Slot0 and CS2 and CS3 are connected to Slot1.

Table 4-94: **DDR4 RDIMM Configuration**

Slot0	Slot1
Single-rank	Empty
Single-rank	Single-rank
Dual-rank	Empty
Dual-rank	Dual-rank

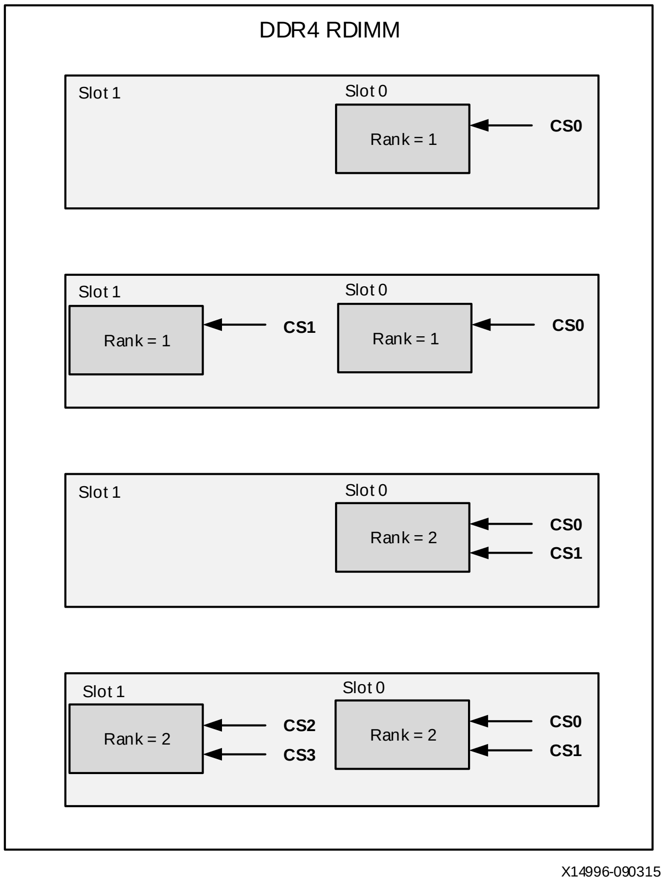


Figure 4-28: **DDR4 RDIMM Configuration**

SLOT0_CONFIG

In a given DIMM configuration, the logic chip select is mapped to physical slot using an 8-bit number per SLOT. Each bit corresponds to a logic chip select connectivity in a SLOT.

Example 1: Dual-Rank DIMM, Dual Slot system (total of four ranks):

```
SLOT0_CONFIG = 8'b0000_0011 // describes CS0 and CS1 are connected to SLOT0.
SLOT1_CONFIG = 8'b0000_1100 // describes CS2 and CS3 are connected to SLOT1.
SLOT0_FUNC_CS = 8'b0000_0011 // describes CS0 and CS1 in SLOT0 are functional chip
select.
SLOT1_FUNC_CS = 8'b0000_1100 // describes CS2 and CS3 in SLOT1 are functional chip
select.
SLOT0_ODD_CS  = 8'b0000_0010 // describes CS1 bit corresponding to ODD functional
chip select located in slot0.
SLOT1_ODD_CS  = 8'b0000_1000 // describes CS3 bit corresponding to ODD functional
chip select located in slot1.
```

Example 2: Single-Rank DIMM, Dual Slot system (total of two ranks):

```
SLOT0_CONFIG = 8'b0000_0001 // describes CS0 is connected to SLOT0.
SLOT1_CONFIG = 8'b0000_0010 // describes CS1 is connected to SLOT1.
SLOT0_FUNC_CS = 8'b0000_0001 // describes CS0 in SLOT0 is functional chip select.
SLOT1_FUNC_CS = 8'b0000_0010 // describes CS1 in SLOT1 is functional chip select.
SLOT0_ODD_CS  = 8'b0000_0000 // describes there is no ODD functional chip select
located in slot0.
SLOT1_ODD_CS  = 8'b0000_0000 // describes there is no ODD functional chip select
located in slot1.
```

SLOT0_FUNC_CS

A DDR3 single-rank RDIMM and two chip selects are needed to access the register chip. However, only the lower rank chip select is used as functional chip select. SLOT0_FUNC_CS describes the functional chip select per SLOT. For any DIMM other than a DDR3 single-rank RDIMM, SLOT0_CONFIG is the same as SLOT0_FUNC_CS and SLOT1_CONFIG is the same as SLOT1_FUNC_CS.

Example 1: DDR3 RDIMM, Single-Rank DIMM, Single Slot system:

```
SLOT0_CONFIG = 8'b0000_0011 // describes CS0 and CS1 are connected to SLOT0.
SLOT1_CONFIG = 8'b0000_0000 // describes no DIMM is connected to SLOT1.
SLOT0_FUNC_CS = 8'b0000_0001 // describes CS0 is functional chip select. CS1 is not
functional chip select and is only used for register chip access.
SLOT1_FUNC_CS = 8'b0000_0000 // describes there is no functional chip select in
SLOT1.
SLOT0_ODD_CS  = 8'b0000_0010 // describes CS1 bit corresponding to ODD functional
chip select located in slot0.
SLOT1_ODD_CS  = 8'b0000_0000 // describes there is no ODD functional chip select
located in slot1.
```

Example 2: DDR3 RDIMM, Single-Rank DIMM, Dual Slot system:

```
SLOT0_CONFIG = 8'b0000_0101 // describes CS0 and CS2 are connected to SLOT0.
SLOT1_CONFIG = 8'b0000_1010 // describes CS1 and CS3 are connected to SLOT1.
SLOT0_FUNC_CS = 8'b0000_0001 // describes CS0 is functional chip select. CS1 is not
functional chip select and is only used for Register Chip access.
SLOT1_FUNC_CS = 8'b0000_0100 // describes CS2 is functional chip select. CS3 is not
functional chip select and is only used for register chip access.
SLOT0_ODD_CS  = 8'b0000_0010 // describes CS1 bit corresponding to ODD functional
chip select located in slot0.
```

```
SLOT1_ODD_CS  = 8'b0000_1000 // describes CS3 bit corresponding to ODD functional
chip select located in slot1.
```

DDR4 LRDIMM

Table 4-95 and Figure 4-29 show the three configurations supported for DDR4 LRDIMM.

For Dual Slot, dual-rank configuration, follow the chip select order shown in Figure 4-29, where CS0 and CS1 are connected to Slot0 and CS2 and CS3 are connected to Slot1.

Table 4-95: **DDR4 LRDIMM Configuration**

Slot0	Slot1
Dual-rank	Empty
Quad-rank	Empty
Dual-rank	Dual-rank

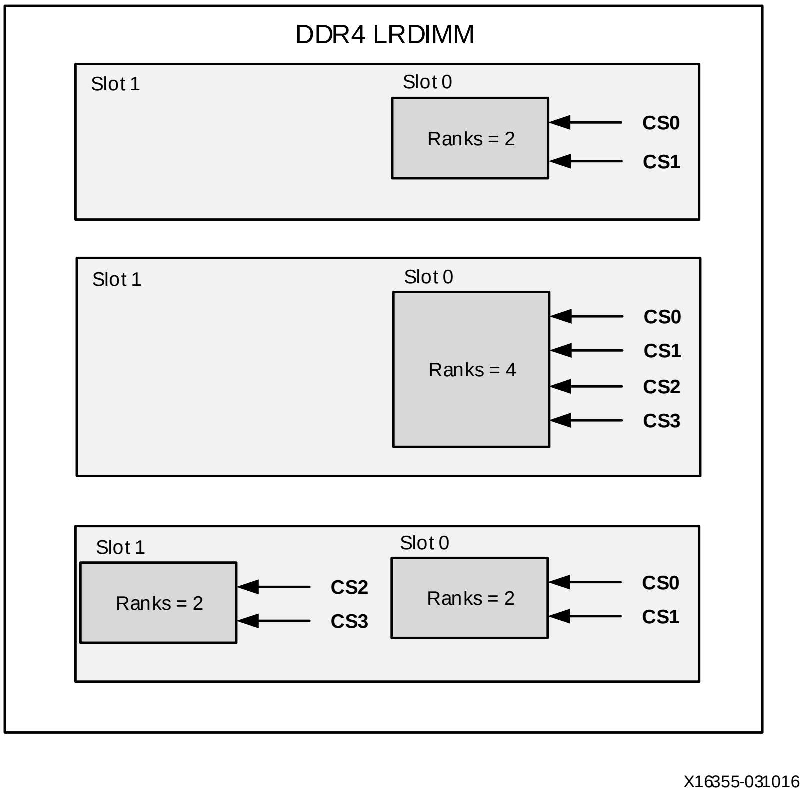


Figure 4-29: **DDR4 LRDIMM Configuration**

Setting Timing Options

The DDR3/DDR4 interfaces are on the edge of meeting timing for certain configurations. Due to controller complexity, designs are failing in timing with levels of logic from eight to 11 in controller modules (`u_ddr_mc` instance). To meet timing for such cases, Tcl command options are supported. These Tcl commands are supported for Controller/PHY mode of the **Controller and Physical Layer**. Based on the Tcl command set in the console, a few RTL parameters are going to change which are listed in Table 4-96. These parameters are valid for all DDR3/DDR4 designs.

Table 4-96: **Parameter Values Based on Tcl Command Option**

Parameters	Default	Better timing, +4tCK Latency (TIMING_OP1 Tcl Option)	Best timing, +4 to +8tCK Latency Depending on Transaction Pattern (TIMING_OP2 Tcl Option)
CAS_FIFO_BYPASS	ON	ON	OFF
PER_RD_PERF	1’b1	1’b1	1’b0
TXN_FIFO_BYPASS	ON	OFF	OFF
TXN_FIFO_PIPE	OFF	ON	ON

The default values of four parameters are given in Table 4-96. These parameters can be changed through the Tcl command using user parameter `TIMING_OP1` or `TIMING_OP2` for Controller/PHY mode of the **Controller and Physical Layer**. These Tcl options are not valid for any PHY_ONLY (**Physical Layer Only** and **Physical Layer Ping Pong**) designs.

Steps to Change RTL Parameters

1. Generate DDR3 or DDR4 IP with **Controller and Physical Layer** selected.
2. In the **Generate Output Products** option do not select **Generate** instead select **Skip**. See Figure 4-30.

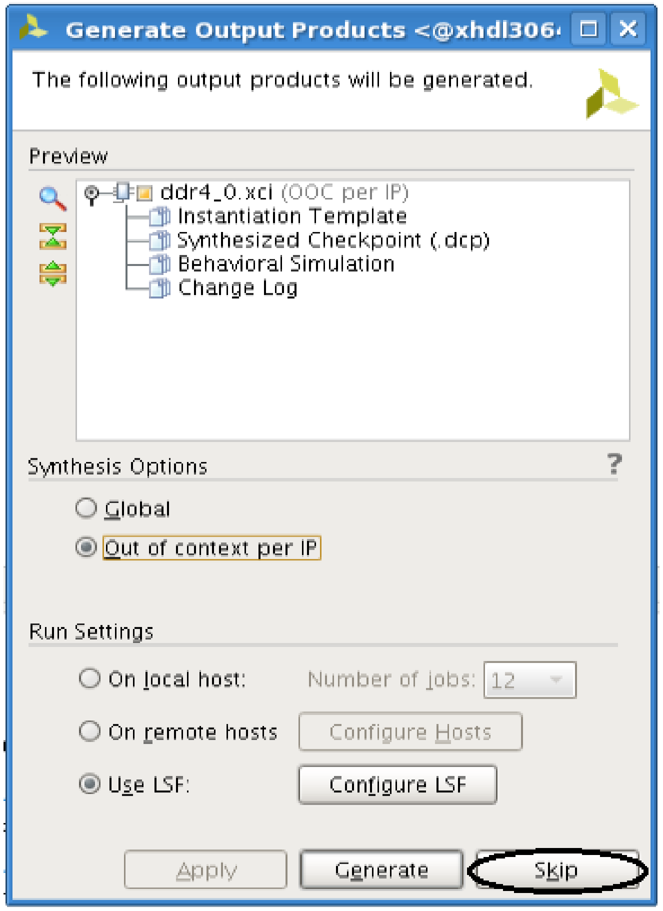


Figure 4-30: **Generate Output Products Window – Skip**

3. Set the following command on the Tcl Console:

```
set_property -dict [list config.TIMING_OP1 {true}] [get_ips <ip_name>]
For example: set_property -dict [list config.TIMING_OP1 {true}] [get_ips ddr4_0]

set_property -dict [list config.TIMING_OP2 {true}] [get_ips <ip_name>]
For example: set_property -dict [list config.TIMING_OP2 {true}] [get_ips ddr4_0]
```

4. Generate output files by selecting **Generate Output Products** after right-clicking IP. See Figure 4-31.

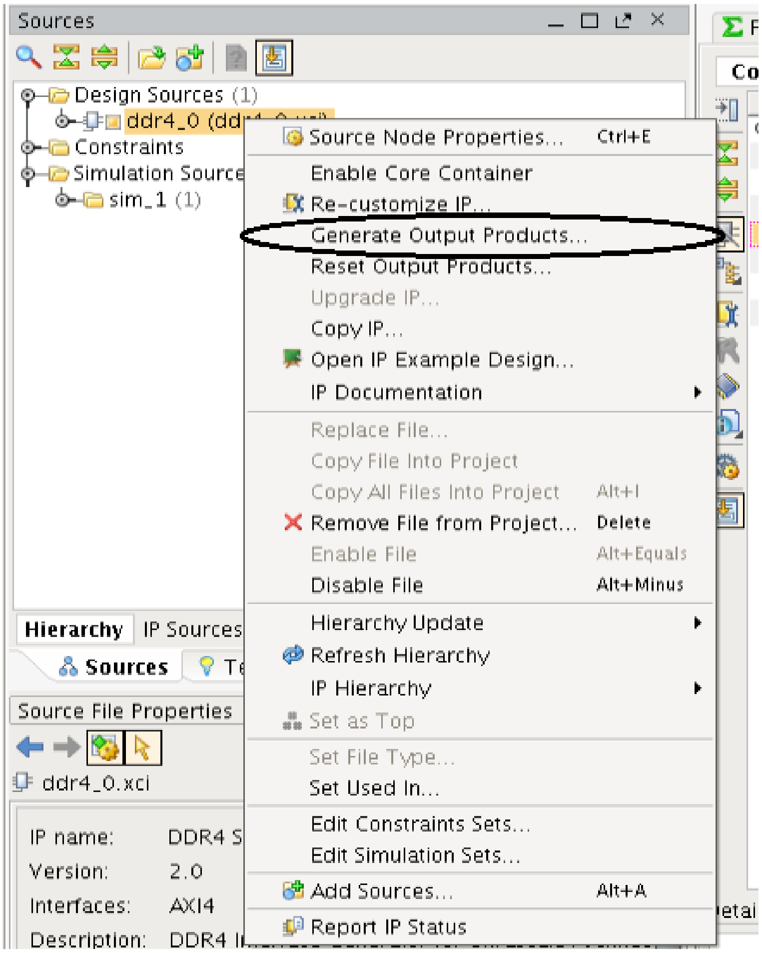


Figure 4-31: **Sources Window – Generate Output Products**

The generated output files have the RTL parameter values set as per Table 4-96.

Timing Improvements for 3DS Designs

The DDR4 3DS interfaces are not meeting timing for certain configurations. The failing timing paths are in the controller modules (`u_ddr_mc` instance). To meet timing for such cases, the Tcl command option is supported. Tcl command is supported for the Controller/PHY mode of **Controller and Physical Layer** and valid for 3DS parts only (S_HEIGHT parameter value of 2 or 4). Based on the Tcl command that is set in the console, a few RTL parameters are going to change which are listed in Table 4-97.

Table 4-97: **Parameter Values Based on Tcl Command Option for 3DS**

Parameters	Default	Better Timing (TIMING_3DS Tcl Option)
ALIAS_PAGE	OFF	ON
ALIAS_P_CNT	OFF	ON

DRAM pages are kept open as long as possible to reduce number of precharges. The controller contains a page table per bank and rank for each bank group. With 3DS, a third dimension is added to these page tables for logical ranks. This increases gate counts and makes timing closures harder. But the DRAM access performance is improved. ALIAS_PAGE = ON removes this dimension.

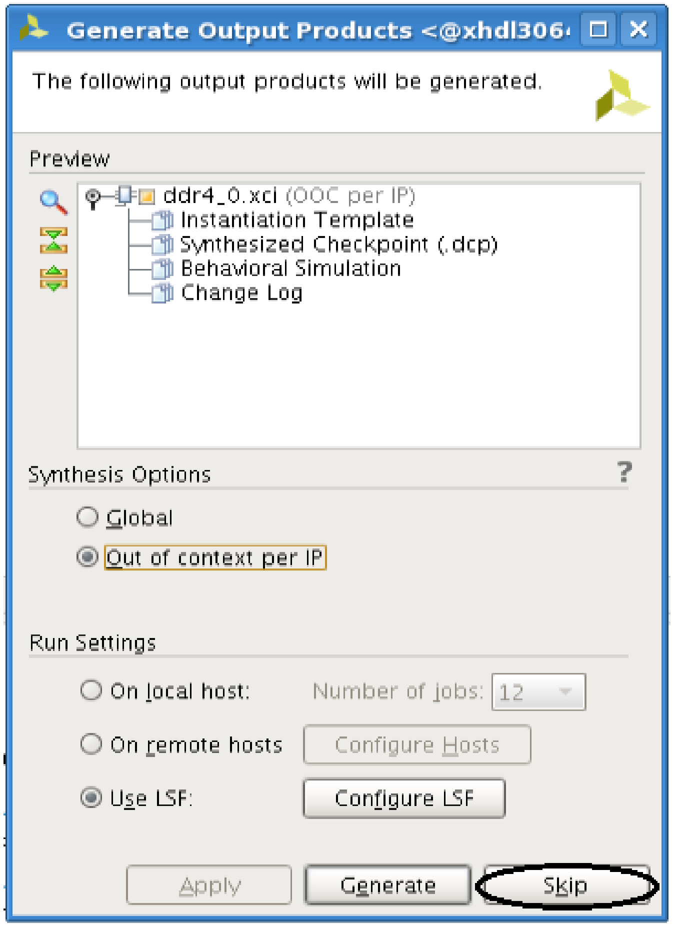
Similarly for 3DS, another dimension is added for logical rank to some per rank/bank counters which keeps track of tRAS, tRTP, and tWTP. ALIAS_P_CNT = ON removes the logical rank dimension.

Removing the third dimension does not affect correct operation of DRAM. However, it removes some of the performance advantages.

The default values of two parameters are given in Table 4-97. These parameters can be changed through the Tcl command using user parameter `TIMING_3DS` for Controller/PHY mode of **Controller and Physical Layer**. These Tcl options are not valid for any PHY_ONLY (**Physical Layer Only** and **Physical Layer Ping Pong**) designs.

Steps to Change RTL Parameters

1. Generate the DDR3 or DDR4 IP with **Controller and Physical Layer** selected.
2. In the **Generate Output Products** option do not select **Generate** instead select **Skip**. See Figure 4-32.



The following output products will be generated.

- ddr4_0.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Behavioral Simulation
  - Change Log

Synthesis Options

○ Global
● Out of context per IP

Run Settings

○ On local host: Number of jobs: 12
○ On remote hosts
● Use LSF:

Figure 4-32: **Generate Output Products Window – Skip**

3. Set the following command on the Tcl Console:

```
set_property -dict [list config.TIMING_3DS {true}] [get_ips <ip_name>]
For example:  set_property -dict [list config.TIMING_3DS {true}] [get_ips ddr4_0]
```

4. Generate output files by selecting **Generate Output Products** after right-clicking IP. See Figure 4-33.

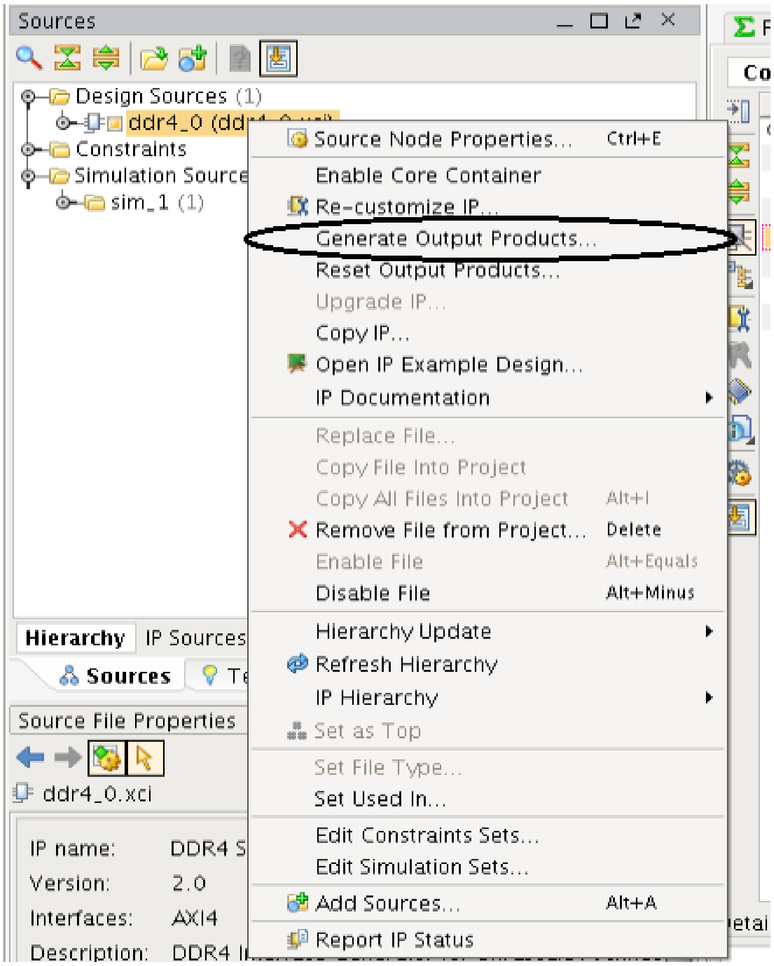


Figure 4-33: **Sources Window – Generate Output Products**

M and D Support for Reference Input Clock Speed

Memory IPs provide two possibilities to select the Reference Input Clock Speed. Value allowed for Reference Input Clock Speed (ps) is always ≥ Memory Device Interface Speed (ps).

- Memory IP lists the possible Reference Input Clock Speed values based on the targeted memory frequency (based on selected Memory Device Interface Speed).
- Otherwise, select **M and D Options** and target for desired Reference Input Clock Speed which is calculated based on selected CLKFBOUT_MULT (M), DIVCLK_DIVIDE (D), and CLKOUT0_DIVIDE (D0) values in the Advanced Clocking Tab.

The required Reference Input Clock Speed is calculated from the M, D, and D0 values entered in the GUI using the following formulas:

- MMCM_CLKOUT (MHz) = tCK / Phy_Clock_Ratio

  Where tCK is the Memory Device Interface Speed selected in the Basic tab.

- CLKIN (MHz) = (MMCM_CLKOUT (MHz) × D × D0) / M

  CLKIN (MHz) is the calculated Reference Input Clock Speed.

- VCO (MHz) = (CLKIN (MHz)) / D

  VCO (MHz) is the calculated VCO frequency.

- PFD (MHz) = CLKIN (MHz) / D

  PFD (MHz) is the calculated PFD frequency.

Calculated Reference Input Clock Speed from M, D, and D0 values are validated as per clocking guidelines. For more information on clocking rules, see Clocking.

Apart from the memory specific clocking rules, validation of the possible MMCM input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values are completed for M, D, and D0 in the GUI.

For UltraScale devices, see *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2] and *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For UltraScale+ devices, see *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4], *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5], and *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For possible M, D, and D0 values and detailed information on clocking and the MMCM, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]

Customizing and Generating the Core

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

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the **`validate_bd_design`** command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

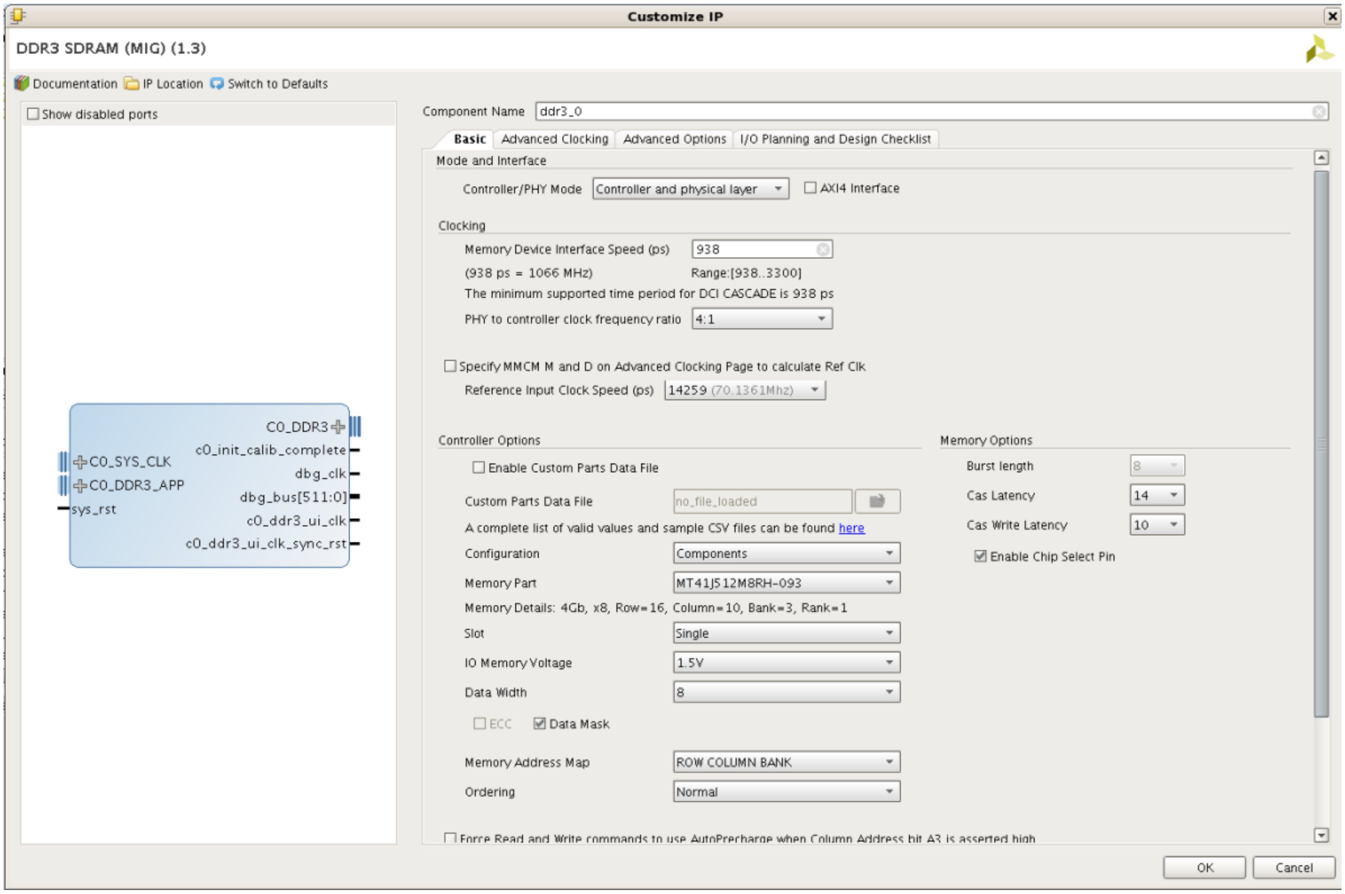
1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For more information about generating the core in Vivado, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

Basic Tab

Figure 5-1 and Figure 5-2 show the **Basic** tab when you start up the DDR3/DDR4 SDRAM.



Component Name: ddr3_0

Controller/PHY Mode: Controller and physical layer
☐ AXI4 Interface

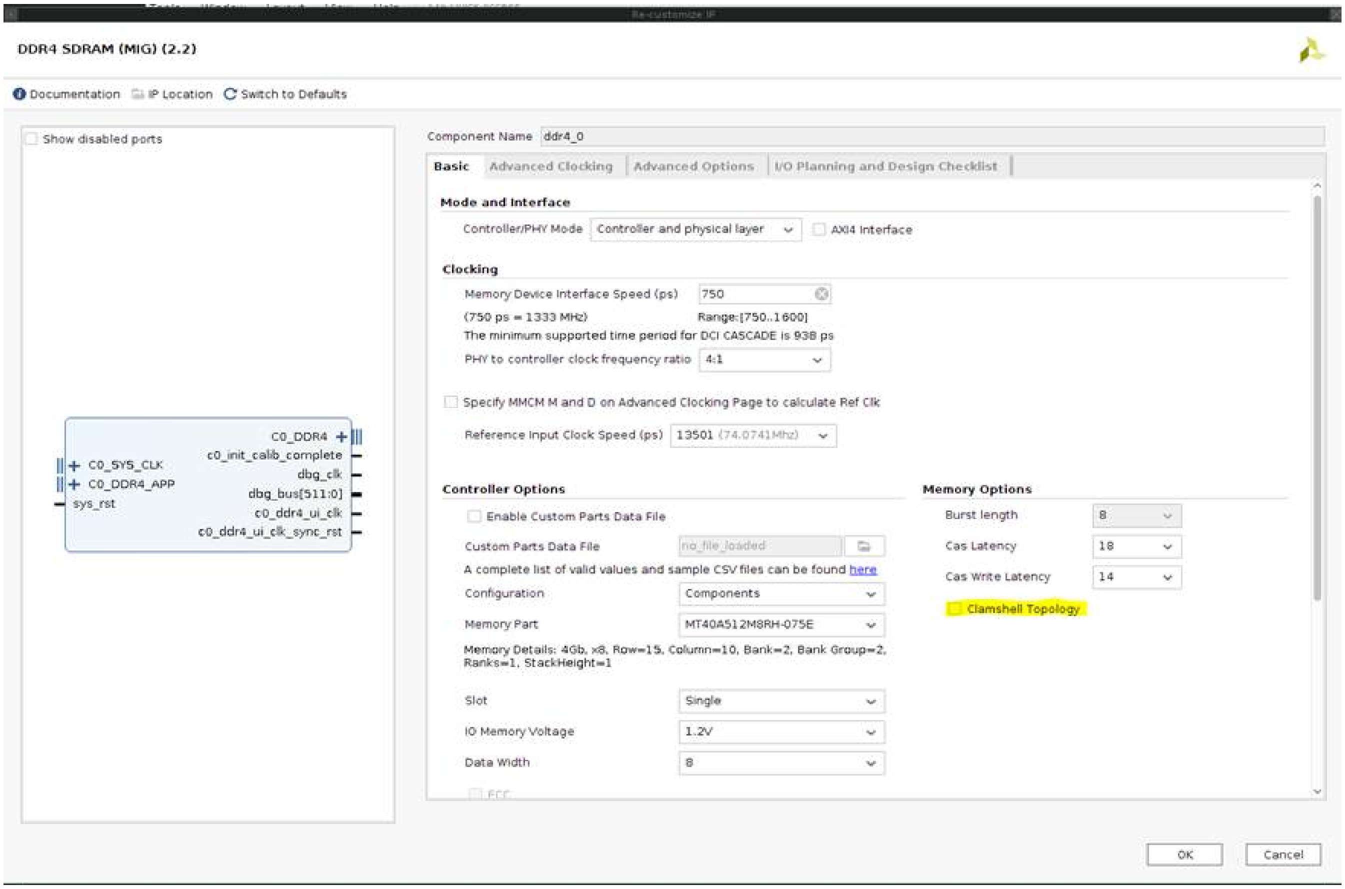
Memory Device Interface Speed (ps): 938
(938 ps = 1066 MHz) Range:[938..3300]
The minimum supported time period for DCI CASCADE is 938 ps
PHY to controller clock frequency ratio: 4:1

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk
Reference Input Clock Speed (ps): 14259 (70.1361Mhz)

☐ Enable Custom Parts Data File
Custom Parts Data File: no_file_loaded
A complete list of valid values and sample CSV files can be found here
Configuration: Components
Memory Part: MT41J512M8RH-093
Memory Details: 4Gb, x8, Row=16, Column=10, Bank=3, Rank=1
Slot: Single
IO Memory Voltage: 1.5V
Data Width: 8
☐ ECC ☑ Data Mask
Memory Address Map: ROW COLUMN BANK
Ordering: Normal
☐ Force Read and Write commands to use AutoPrecharge when Column Address bit A3 is asserted high

Burst length: 8
Cas Latency: 14
Cas Write Latency: 10
☑ Enable Chip Select Pin

Figure 5-1: **Vivado Customize IP Dialog Box for DDR3 – Basic**



☐ Show disabled ports

Component Name: ddr4_0

Mode and Interface

Controller/PHY Mode: Controller and physical layer

☐ AXI4 Interface

Clocking

Memory Device Interface Speed (ps): 750

(750 ps = 1333 MHz) Range:[750..1600]

The minimum supported time period for DCI CASCADE is 938 ps

PHY to controller clock frequency ratio: 4:1

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps): 13501 (74.0741Mhz)

Controller Options

☐ Enable Custom Parts Data File

Custom Parts Data File: no_file_loaded

A complete list of valid values and sample CSV files can be found here

Configuration: Components

Memory Part: MT40A512M8RH-075E

Memory Details: 4Gb, x8, Row=15, Column=10, Bank=2, Bank Group=2, Ranks=1, StackHeight=1

Slot: Single

IO Memory Voltage: 1.2V

Data Width: 8

☐ ECC

Memory Options

Burst length: 8

Cas Latency: 18

Cas Write Latency: 14

☐ Clamshell Topology

Figure 5-2: **Vivado Customize IP Dialog Box for DDR4 – Basic**

IMPORTANT: *All parameters shown in the controller options dialog box are limited selection options in this release.*

For the Vivado IDE, all controllers (DDR3, DDR4, LPDDR3, QDR II+, QDR-IV, and RLDRAM 3) can be created and available for instantiation.

In IP integrator, only one controller instance can be created and only two kinds of controllers are available for instantiation:

- DDR3
- DDR4

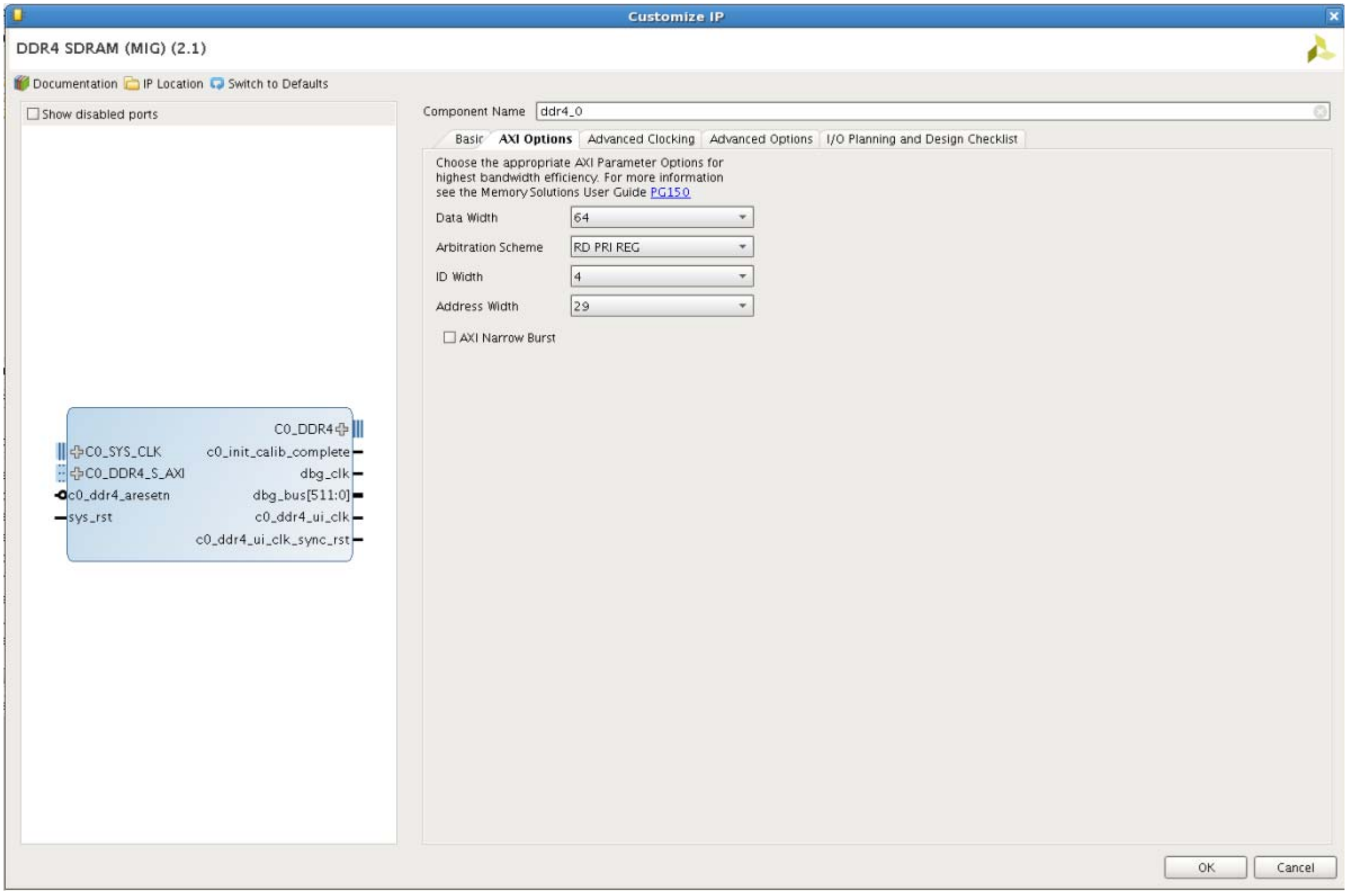
1. After a controller is added in the pull-down menu, select the **Mode and Interface** for the controller. Select the **AXI4 Interface** or have the option to select the **Generate the PHY component only**.
2. Select the settings in the **Clocking**, **Controller Options**, **Memory Options**, and **Advanced User Request Controller Options**.

   In **Clocking**, the **Memory Device Interface Speed** sets the speed of the interface. The speed entered drives the available **Reference Input Clock Speeds**. For more information on the clocking structure, see the Clocking, page 81.
3. To use memory parts which are not available by default through the DDR3/DDR4 SDRAM Vivado IDE, you can create a custom parts CSV file, as specified in the AR: 63462. This CSV file has to be provided after enabling the **Custom Parts Data File** option. After selecting this option. you are able to see the custom memory parts along with the default memory parts. Note that, simulations are not supported for the custom part. Custom part simulations require manually adding the memory model to the simulation and might require modifying the test bench instantiation.
4. All available options of **Data Mask and DBI** and their functionality is described in Table 4-76. Also, the dependency of ECC on the DM_DBI input is mentioned in Table 4-77 for both user and AXI interfaces.

IMPORTANT: *To support partial writes, AXI designs require Data Mask (DM) to always be selected and it is grayed out. This is for all AXI interfaces except 72 bits, which requires the use of ECC. Having ECC and DM in the same design causes the ECC to fail, so turning off the DM when ECC is enabled is required.*

AXI Options Tab

Figure 5-3 shows the next tab called **AXI Options** when the AXI4 interface is selected in the Basic page. This displays the settings for **AXI Options** for the specific controller.



Component Name: ddr4_0

Choose the appropriate AXI Parameter Options for highest bandwidth efficiency. For more information see the Memory Solutions User Guide PG150

Data Width: 64

Arbitration Scheme: RD PRI REG

ID Width: 4

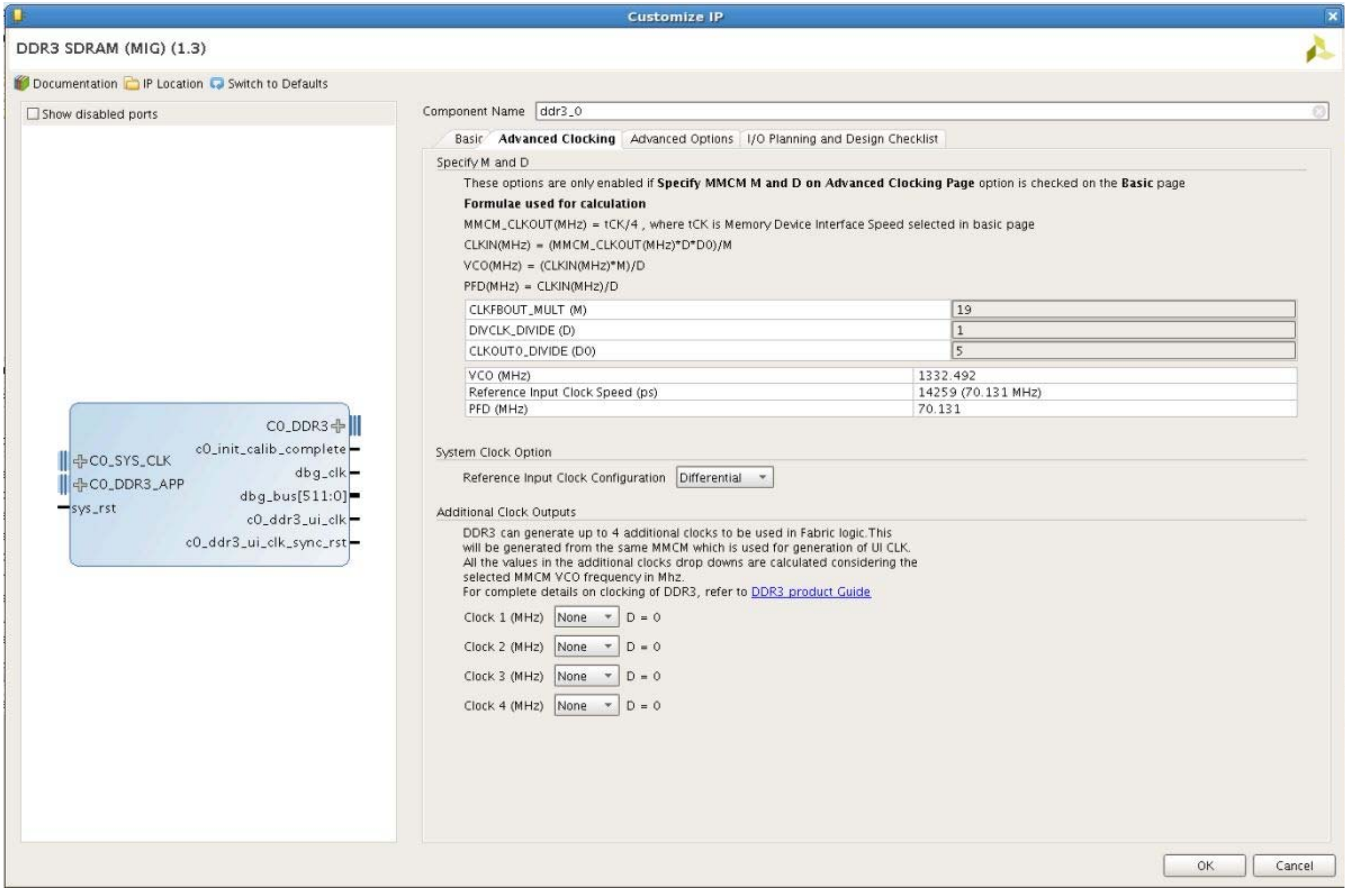
Address Width: 29

☐ AXI Narrow Burst

Figure 5-3: **Vivado Customize IP Dialog Box for DDR4 – AXI Options**

Advanced Clocking Tab

Figure 5-4 shows the next tab called **Advanced Clocking**. This displays the settings for **Specify M and D** value, **System Clock Options**, and **Additional Clock Outputs** for the specific controller.



Component Name: ddr3_0

Specify M and D

These options are only enabled if **Specify MMCM M and D on Advanced Clocking Page** option is checked on the **Basic** page

Formulae used for calculation

MMCM_CLKOUT(MHz) = tCK/4 , where tCK is Memory Device Interface Speed selected in basic page

CLKIN(MHz) = (MMCM_CLKOUT(MHz)*D*D0)/M

VCO(MHz) = (CLKIN(MHz)*M)/D

PFD(MHz) = CLKIN(MHz)/D

CLKFBOUT_MULT (M)	19
DIVCLK_DIVIDE (D)	1
CLKOUT0_DIVIDE (D0)	5

VCO (MHz)	1332.492
Reference Input Clock Speed (ps)	14259 (70.131 MHz)
PFD (MHz)	70.131

System Clock Option

Reference Input Clock Configuration: Differential

Additional Clock Outputs

DDR3 can generate up to 4 additional clocks to be used in Fabric logic.This will be generated from the same MMCM which is used for generation of UI CLK. All the values in the additional clocks drop downs are calculated considering the selected MMCM VCO frequency in Mhz. For complete details on clocking of DDR3, refer to DDR3 product Guide

Clock 1 (MHz): None D = 0

Clock 2 (MHz): None D = 0

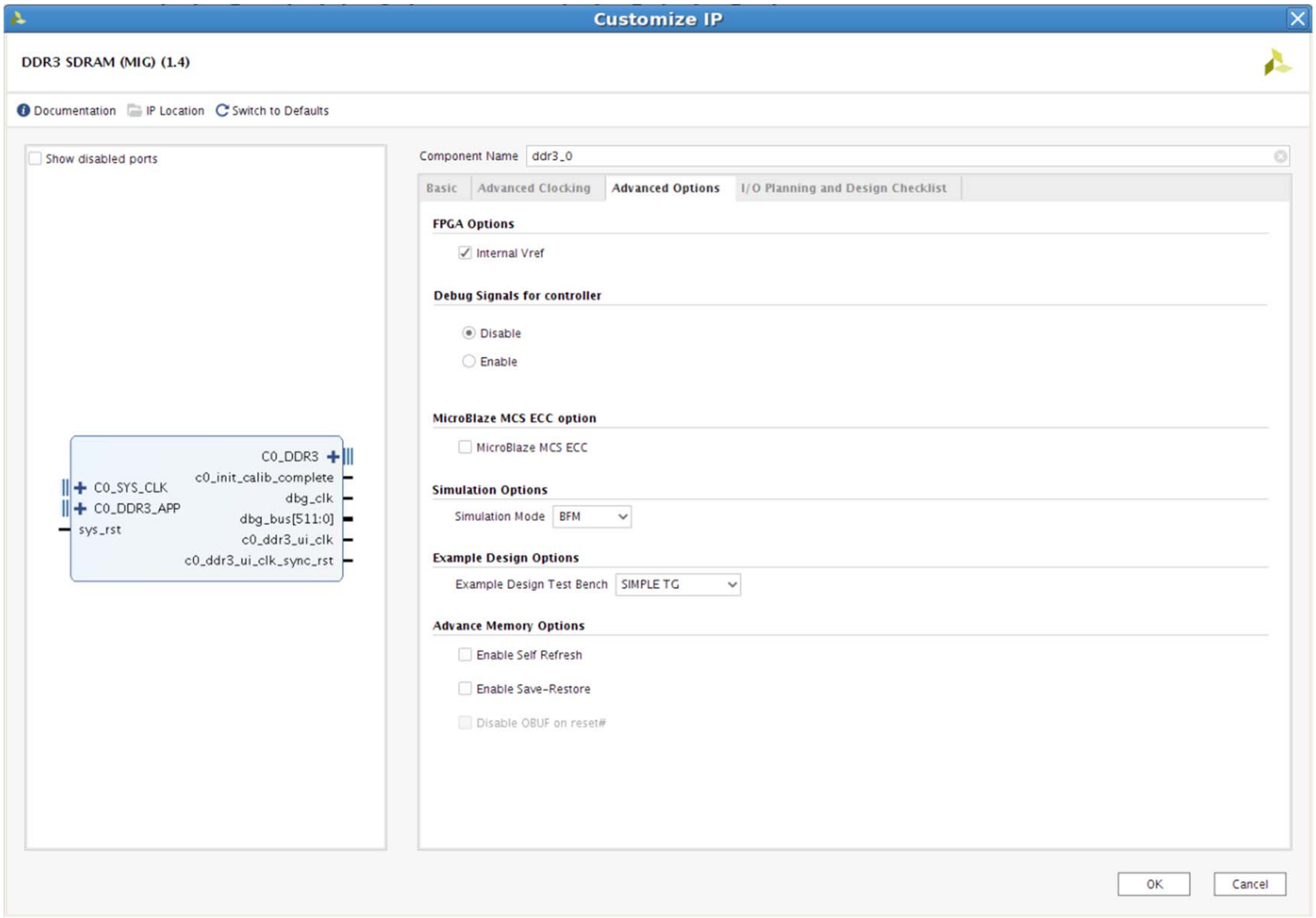
Clock 3 (MHz): None D = 0

Clock 4 (MHz): None D = 0

Figure 5-4: **Vivado Customize IP Dialog Box for DDR3 – Advanced Clocking**

Advanced Options Tab

Figure 5-5 and Figure 5-6 show the next tab called **Advanced Options**. This displays the advanced memory options for the specific controller.



Component Name: ddr3_0

FPGA Options

☑ Internal Vref

Debug Signals for controller

● Disable

○ Enable

MicroBlaze MCS ECC option

☐ MicroBlaze MCS ECC

Simulation Options

Simulation Mode: BFM

Example Design Options

Example Design Test Bench: SIMPLE TG

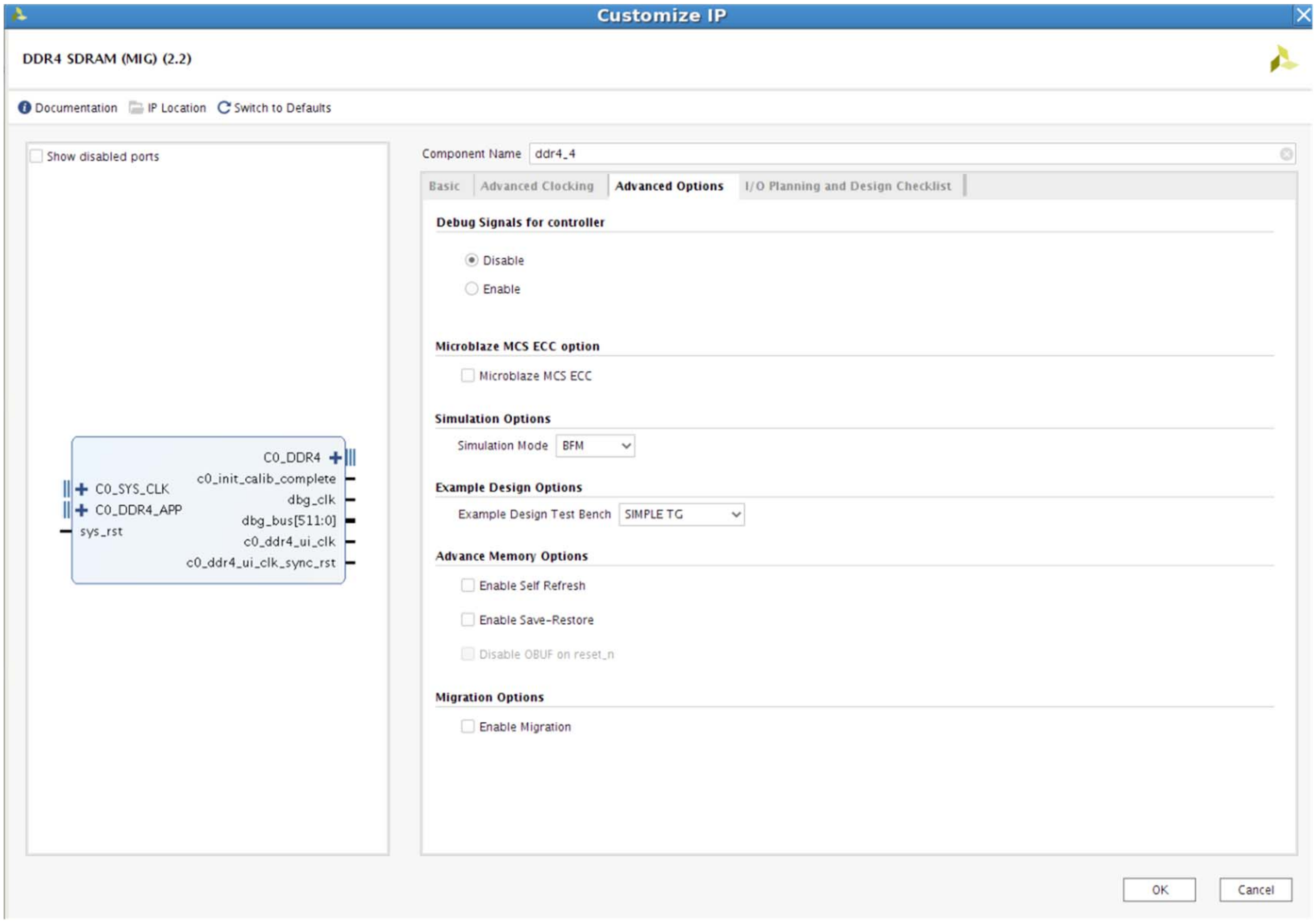
Advance Memory Options

☐ Enable Self Refresh

☐ Enable Save-Restore

☐ Disable OBUF on reset#

Figure 5-5: **Vivado Customize IP Dialog Box for DDR3 – Advanced Options**



Component Name: ddr4_4

Debug Signals for controller

● Disable

○ Enable

Microblaze MCS ECC option

☐ Microblaze MCS ECC

Simulation Options

Simulation Mode: BFM

Example Design Options

Example Design Test Bench: SIMPLE TG

Advance Memory Options

☐ Enable Self Refresh

☐ Enable Save-Restore

☐ Disable OBUF on reset_n

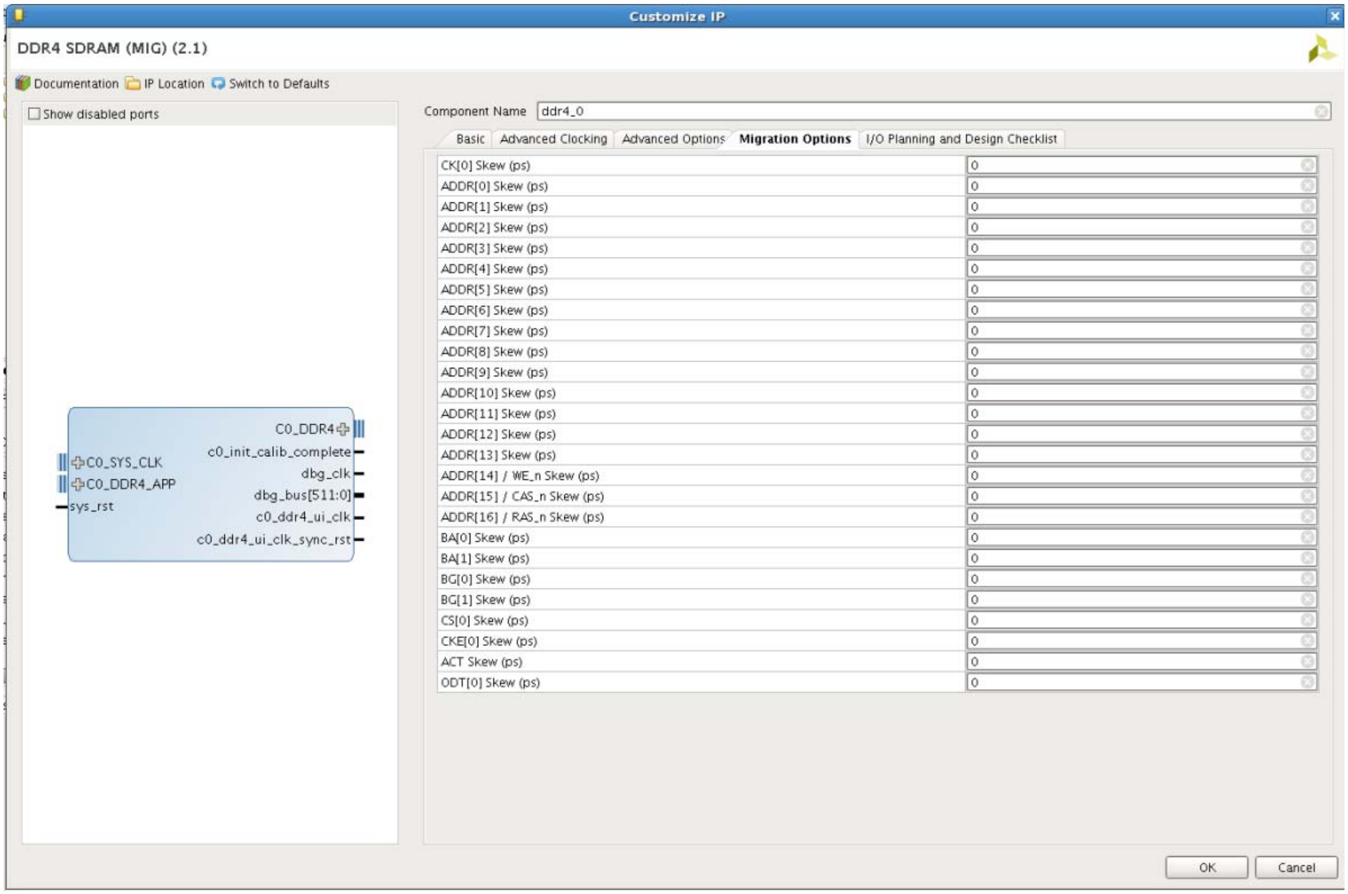
Migration Options

☐ Enable Migration

Figure 5-6: **Vivado Customize IP Dialog Box for DDR4 – Advanced Options**

Migration Options Tab

Figure 5-7 shows the next tab called **Migration Options** only for DDR4 displays when **Enable Migration** option is selected in **Advanced Options** tab.



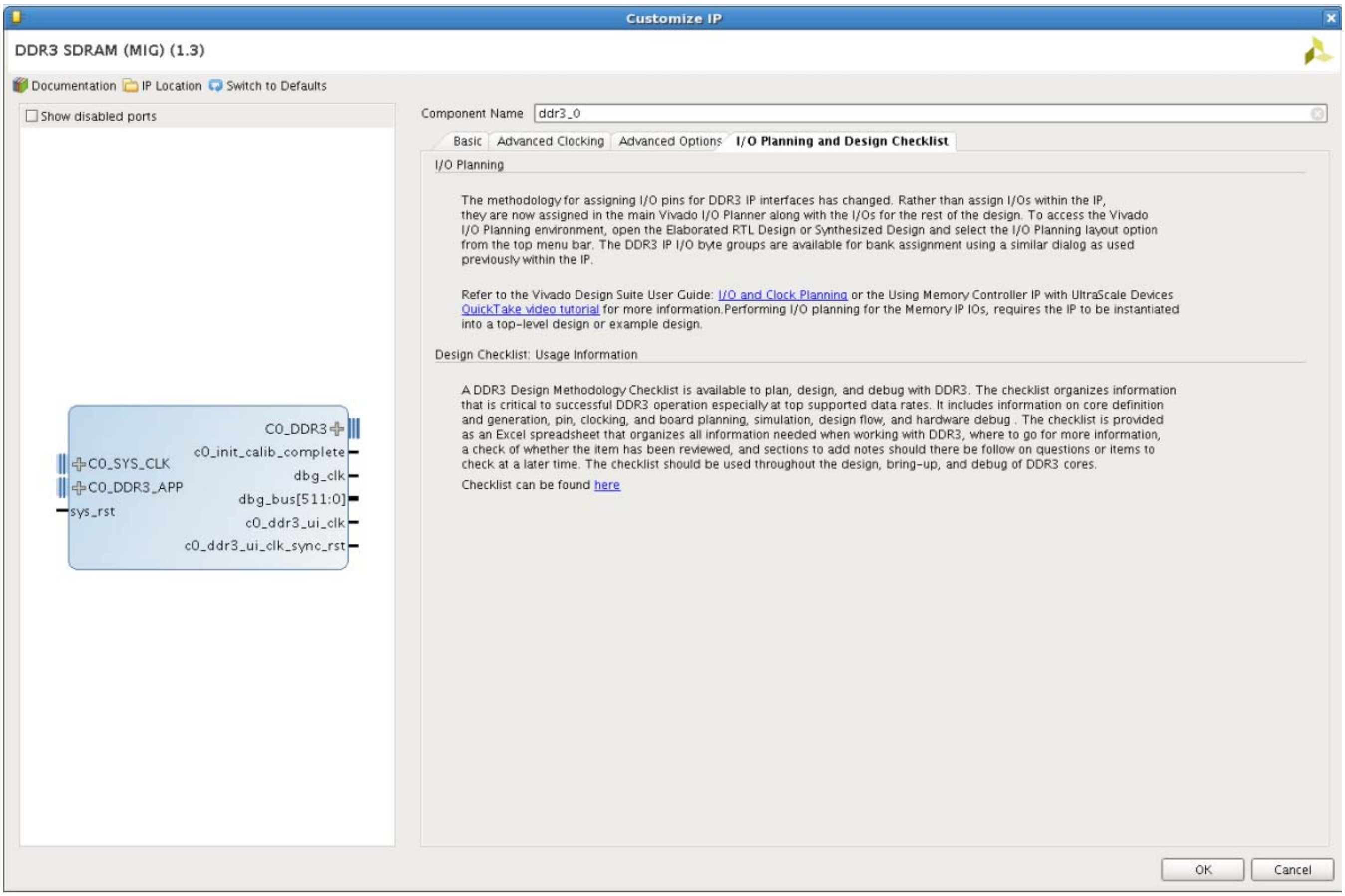
Component Name: ddr4_0

CK[0] Skew (ps)	0
ADDR[0] Skew (ps)	0
ADDR[1] Skew (ps)	0
ADDR[2] Skew (ps)	0
ADDR[3] Skew (ps)	0
ADDR[4] Skew (ps)	0
ADDR[5] Skew (ps)	0
ADDR[6] Skew (ps)	0
ADDR[7] Skew (ps)	0
ADDR[8] Skew (ps)	0
ADDR[9] Skew (ps)	0
ADDR[10] Skew (ps)	0
ADDR[11] Skew (ps)	0
ADDR[12] Skew (ps)	0
ADDR[13] Skew (ps)	0
ADDR[14] / WE_n Skew (ps)	0
ADDR[15] / CAS_n Skew (ps)	0
ADDR[16] / RAS_n Skew (ps)	0
BA[0] Skew (ps)	0
BA[1] Skew (ps)	0
BG[0] Skew (ps)	0
BG[1] Skew (ps)	0
CS[0] Skew (ps)	0
CKE[0] Skew (ps)	0
ACT Skew (ps)	0
ODT[0] Skew (ps)	0

Figure 5-7: **Vivado Customize IP Dialog Box for DDR4 – Migration Options**

DDR3/DDR4 SDRAM I/O Planning and Design Checklist Tab

Figure 5-8 and Figure 5-9 show the **DDR3/DDR4 SDRAM I/O Planning and Design Checklist** usage information.



Component Name ddr3_0

I/O Planning

The methodology for assigning I/O pins for DDR3 IP interfaces has changed. Rather than assign I/Os within the IP, they are now assigned in the main Vivado I/O Planner along with the I/Os for the rest of the design. To access the Vivado I/O Planning environment, open the Elaborated RTL Design or Synthesized Design and select the I/O Planning layout option from the top menu bar. The DDR3 IP I/O byte groups are available for bank assignment using a similar dialog as used previously within the IP.

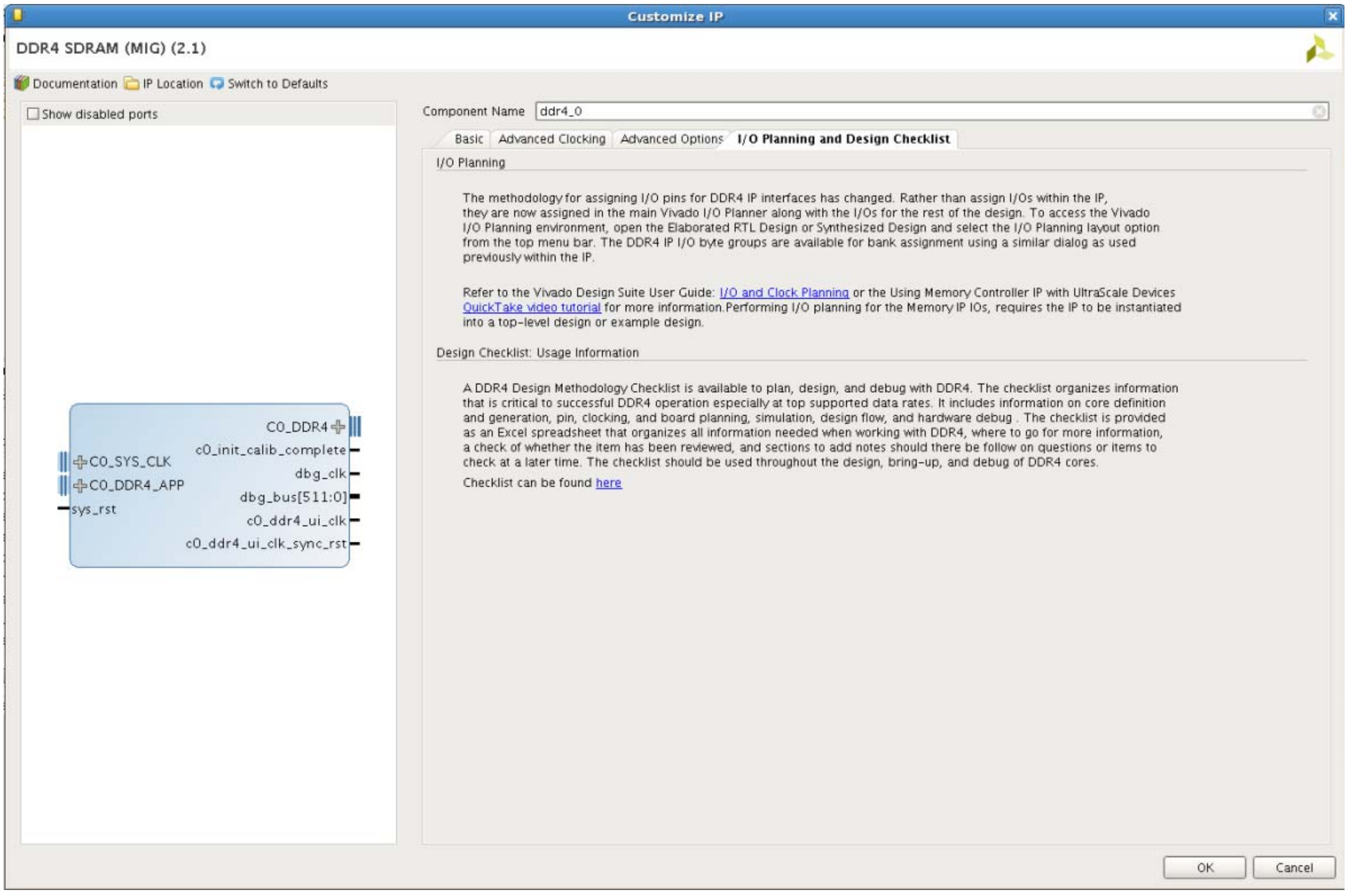
Refer to the Vivado Design Suite User Guide: I/O and Clock Planning or the Using Memory Controller IP with UltraScale Devices QuickTake video tutorial for more information.Performing I/O planning for the Memory IP IOs, requires the IP to be instantiated into a top-level design or example design.

Design Checklist: Usage Information

A DDR3 Design Methodology Checklist is available to plan, design, and debug with DDR3. The checklist organizes information that is critical to successful DDR3 operation especially at top supported data rates. It includes information on core definition and generation, pin, clocking, and board planning, simulation, design flow, and hardware debug . The checklist is provided as an Excel spreadsheet that organizes all information needed when working with DDR3, where to go for more information, a check of whether the item has been reviewed, and sections to add notes should there be follow on questions or items to check at a later time. The checklist should be used throughout the design, bring-up, and debug of DDR3 cores.

Checklist can be found here

Figure 5-8: **Vivado Customize IP Dialog Box – DDR3 SDRAM I/O Planning and Design Checklist**



Component Name ddr4_0

I/O Planning

The methodology for assigning I/O pins for DDR4 IP interfaces has changed. Rather than assign I/Os within the IP, they are now assigned in the main Vivado I/O Planner along with the I/Os for the rest of the design. To access the Vivado I/O Planning environment, open the Elaborated RTL Design or Synthesized Design and select the I/O Planning layout option from the top menu bar. The DDR4 IP I/O byte groups are available for bank assignment using a similar dialog as used previously within the IP.

Refer to the Vivado Design Suite User Guide: I/O and Clock Planning or the Using Memory Controller IP with UltraScale Devices QuickTake video tutorial for more information.Performing I/O planning for the Memory IP IOs, requires the IP to be instantiated into a top-level design or example design.

Design Checklist: Usage Information

A DDR4 Design Methodology Checklist is available to plan, design, and debug with DDR4. The checklist organizes information that is critical to successful DDR4 operation especially at top supported data rates. It includes information on core definition and generation, pin, clocking, and board planning, simulation, design flow, and hardware debug . The checklist is provided as an Excel spreadsheet that organizes all information needed when working with DDR4, where to go for more information, a check of whether the item has been reviewed, and sections to add notes should there be follow on questions or items to check at a later time. The checklist should be used throughout the design, bring-up, and debug of DDR4 cores.

Checklist can be found here

Figure 5-9: **Vivado Customize IP Dialog Box – DDR4 SDRAM I/O Planning and Design Checklist**

User Parameters

Table 5-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 5-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value
System Clock Configuration	System_Clock	Differential
Internal V_{REF}	Internal_Vref	TRUE
DCI Cascade	DCI_Cascade	FALSE
Debug Signal for Controller	Debug_Signal	Disable
Clock 1 (MHz)	ADDN_UI_CLKOUT1_FREQ_HZ	None
Clock 2 (MHz)	ADDN_UI_CLKOUT2_FREQ_HZ	None
Clock 3 (MHz)	ADDN_UI_CLKOUT3_FREQ_HZ	None
Clock 4 (MHz)	ADDN_UI_CLKOUT4_FREQ_HZ	None
Enable System Ports	Enable_SysPorts	TRUE
Default Bank Selections	Default_Bank_Selections	FALSE

Table 5-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value
Reference Clock	Reference_Clock	FALSE
Enable System Ports	Enable_SysPorts	TRUE
DDR3		
AXI4 Interface	C0.DDR3_AxiSelection	FALSE
Clock Period (ps)	C0.DDR3_TimePeriod	1,071
Input Clock Period (ps)	C0.DDR3_InputClockPeriod	13,947
General Interconnect to Memory Clock Ratio	C0.DDR3_PhyClockRatio	4:1
Data Width	C0.DDR3_AxiDataWidth	64
Arbitration Scheme	C0.DDR3_AxiArbitrationScheme	RD_PRI_REG
Address Width	C0.DDR3_AxiAddressWidth	27
AXI4 Narrow Burst	C0.DDR3_AxiNarrowBurst	FALSE
Configuration	C0.DDR3_MemoryType	Components
Memory Part	C0.DDR3_MemoryPart	MT41J128M16JT-093
Data Width	C0.DDR3_DataWidth	8
Data Mask	C0.DDR3_DataMask	TRUE
Burst Length	C0.DDR3_BurstLength	8
R_{TT} (nominal)-ODT	C0.DDR3_OnDieTermination	RZQ/6
CAS Latency	C0.DDR3_CasLatency	11
CAS Write Latency	C0.DDR3_CasWriteLatency	9
Chip Select	C0.DDR3_ChipSelect	TRUE
Memory Address Map	C0.DDR3_Mem_Add_Map	ROW_COLUMN_BANK
Memory Voltage	C0.DDR3_MemoryVoltage	1.5
ECC	C0.DDR3_Ecc	FALSE
Ordering	C0.DDR3_Ordering	Normal
Burst Type	C0.DDR3_BurstType	Sequential
Output Driver Impedance Control	C0.DDR3_OutputDriverImpedanceControl	RZQ/6
AXI ID Width	C0.DDR3_AxiIDWidth	4
Capacity	C0.DDR3_Capacity	512
DDR4		
AXI4 Interface	C0.DDR4_AxiSelection	FALSE
Clock Period (ps)	C0.DDR4_TimePeriod	938
Input Clock Period (ps)	C0.DDR4_InputClockPeriod	104,045
General Interconnect to Memory Clock Ratio	C0.DDR4_PhyClockRatio	4:1
Data Width	C0.DDR4_AxiDataWidth	64
Arbitration Scheme	C0.DDR4_AxiArbitrationScheme	RD_PRI_REG
Address Width	C0.DDR4_AxiAddressWidth	27

Table 5-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value
AXI4 Narrow Burst	C0.DDR4_AxiNarrowBurst	FALSE
Configuration	C0.DDR4_MemoryType	Components
Memory Part	C0.DDR4_MemoryPart	MT40A256M16HA-083
Data Width	C0.DDR4_DataWidth	8
Data Mask	C0.DDR4_DataMask	TRUE
Burst Length	C0.DDR4_BurstLength	8
R_{TT} (nominal)-ODT	C0.DDR4_OnDieTermination	RZQ/6
CAS Latency	C0.DDR4_CasLatency	14
CAS Write Latency	C0.DDR4_CasWriteLatency	11
Chip Select	C0.DDR4_ChipSelect	TRUE
Memory Address Map	C0.DDR4_Mem_Add_Map	ROW_COLUMN_BANK
Memory Voltage	C0.DDR4_MemoryVoltage	1.2
ECC	C0.DDR4_Ecc	FALSE
Ordering	C0.DDR4_Ordering	Normal
Burst Type	C0.DDR4_BurstType	Sequential
Output Driver Impedance Control	C0.DDR4_OutputDriverImpedenceControl	RZQ/7
AXI ID Width	C0.DDR4_AxiIDWidth	4
Capacity	C0.DDR4_Capacity	512

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Setting Burst Type for PHY_ONLY Designs

For DDR3 or DDR4, the default value of Burst Type is set to Sequential. This can be changed through the Tcl command using the user parameter C0.DDR3_BurstType for DDR3 and C0.DDR4_BurstType for DDR4. Table 5-2 shows details of the C0.DDR3_BurstType and C0.DDR4_BurstType user parameters.

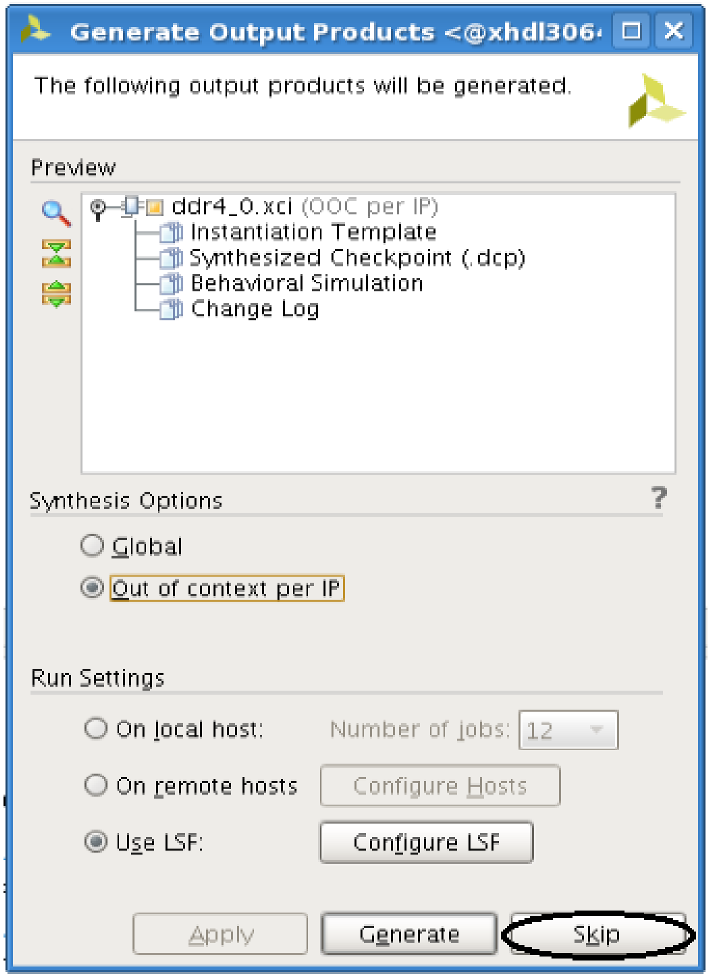
Table 5-2: **Burst Type User Parameter**

User Parameter	Value Format	Default Value	Possible Values
C0.DDR3_BurstType	String	Sequential	Sequential, Interleaved
C0.DDR4_BurstType	String	Sequential	Sequential, Interleaved

Follow these steps to change the Burst Type value.

1. Generate DDR3 or DDR4 PHY_ONLY IP.

2. In the **Generate Output Products** option, do not select **Generate** instead select **Skip** (Figure 5-10).



The following output products will be generated.

Preview

- ddr4_0.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Behavioral Simulation
  - Change Log

Synthesis Options

○ Global

● Out of context per IP

Run Settings

○ On local host: Number of jobs: 12

○ On remote hosts

● Use LSF:

Figure 5-10: **Generate Output Products Window – Skip**

3. Set the Burst Type value by running the following command on the Tcl Console:

   a. For DDR3 IP:

```
set_property -dict [list CONFIG.C0.DDR3_BurstType <value_to_be_set>] [get_ips
<ip_name>]
```

For example:

```
set_property -dict [list CONFIG.C0.DDR3_BurstType {Interleaved}] [get_ips <ddr3_0>]
```

   b. For DDR4 IP:

```
set_property -dict [list CONFIG.C0.DDR4_BurstType <value_to_be_set>] [get_ips
<ip_name>]
```

For example:

```
set_property -dict [list CONFIG.C0.DDR4_BurstType {Interleaved}] [get_ips <ddr4_0>]
```

4. Generate output files by selecting **Generate Output Products** after right-clicking IP (Figure 5-11).

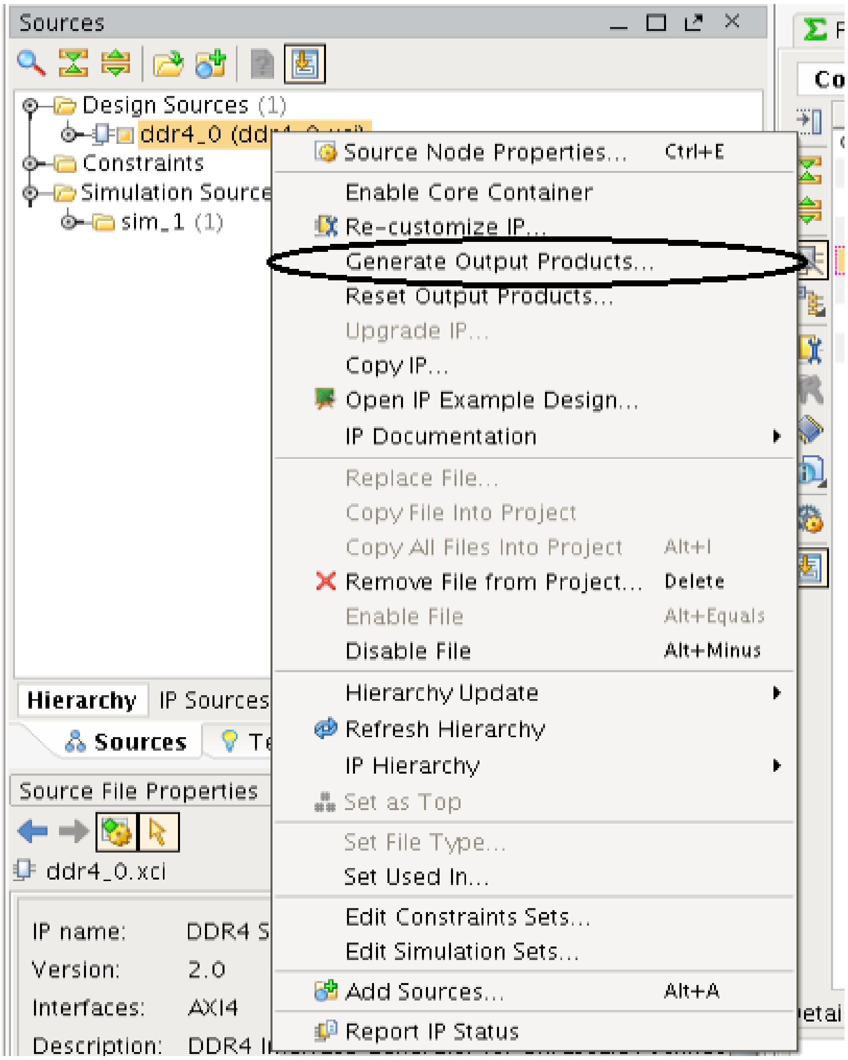


Figure 5-11: **Generate Output Products – Output Files**

The generated output files have the Burst Type value set as per the selected value.

Setting Additive Latency for PHY_ONLY Designs

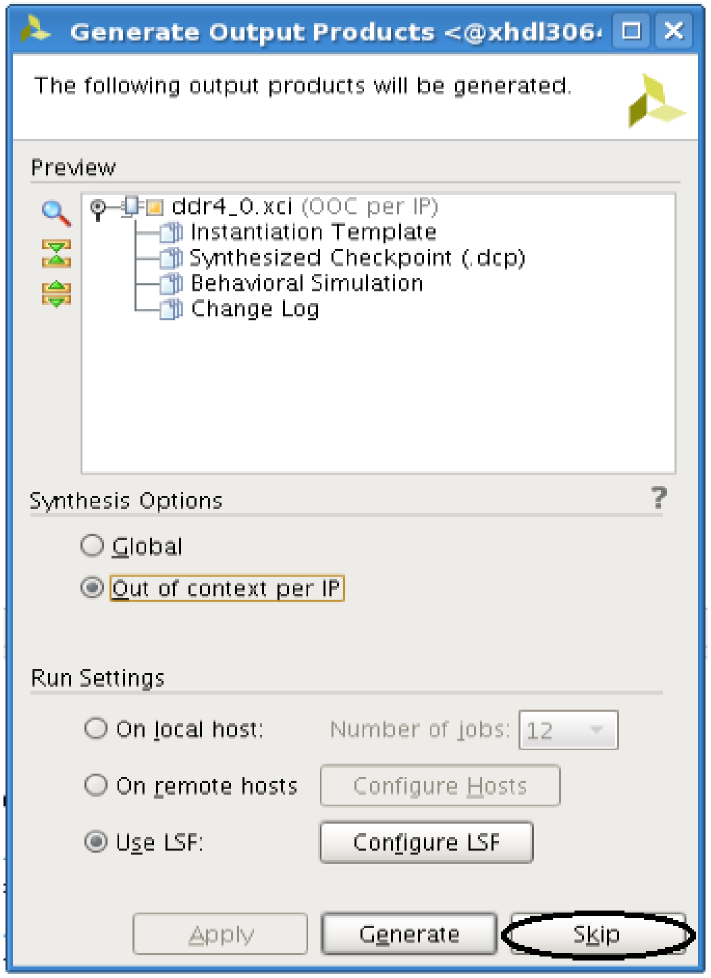
For DDR3/DDR4, the default value of Additive Latency is set to 0. This can be changed through the Tcl command using the user parameter AL_SEL for any PHY_ONLY (Physical Layer Only and Physical Layer Ping Pong designs). Table 5-3 shows details of the AL_SEL user parameter.

Table 5-3: **Additive Latency User Parameter**

User Parameter	Value Format	Default Value	Possible Values (Non-3DS Memories)	Possible Values (3DS Memories)
AL_SEL	String	0	0 – Additive Latency = 0 CL-1 – Additive Latency = CL - 1 CL-2 – Additive Latency = CL - 2	0 – Additive Latency = 0 CL-2 – Additive Latency = CL - 2 CL-3 – Additive Latency = CL - 3

Follow these steps to change the Additive Latency value.

1. Generate DDR3 or DDR4 PHY_ONLY IP.
2. In the **Generate Output Products** option, do not select **Generate** instead select **Skip** (Figure 5-12).



The following output products will be generated.

○ Global

● Out of context per IP

○ On local host: Number of jobs: 12

○ On remote hosts

● Use LSF:

Figure 5-12: **Generate Output Products Window – Skip**

3. Set the Additive Latency value by running the following command on the Tcl Console:

```
set_property -dict [list config.AL_SEL <value_to_be_set>] [get_ips <ip_name>]
```

For example:

```
set_property -dict [list config.AL_SEL CL-1] [get_ips ddr4_0]
```

4. Generate output files by selecting **Generate Output Products** after right-clicking IP (Figure 5-13).

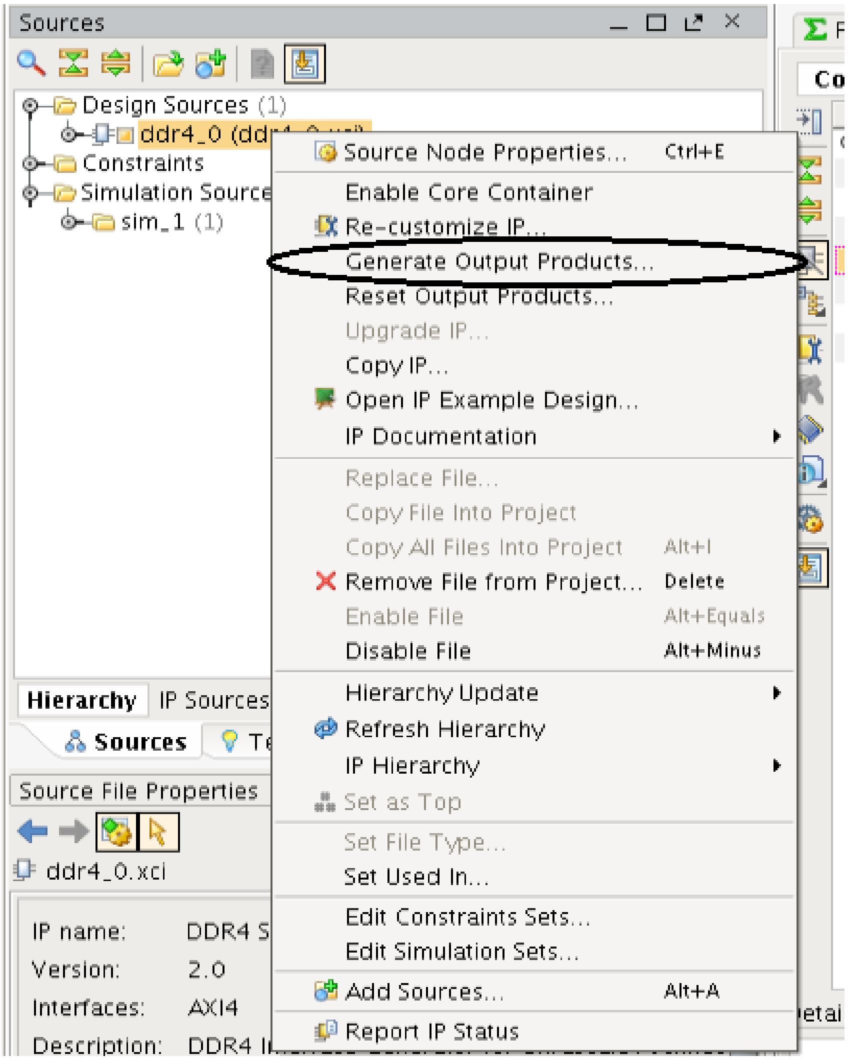


Figure 5-13: **Generate Output Products – Output Files**

The generated output files have the Additive Latency value set as per the selected value.

Setting Timing Parameters for DDR4 Non-Custom Memory Parts

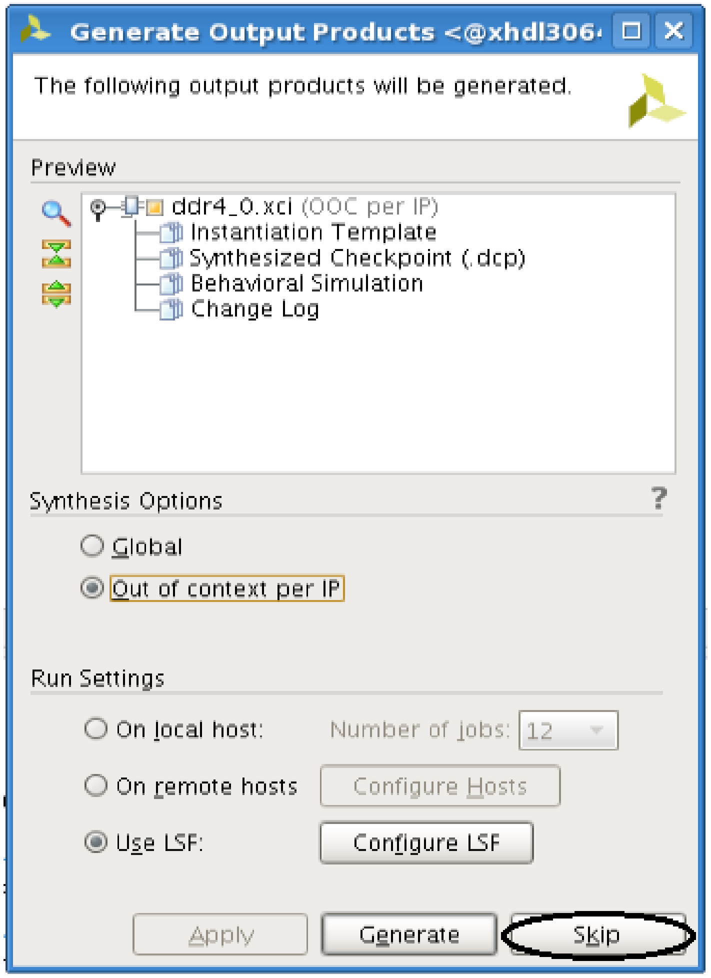
To set timing parameters for DDR4 non-custom memory parts, see Table 5-4 and the following steps.

Table 5-4: **User Parameters for DDR Non-Custom Memory Parts**

User Parameter	Value Format	Default Value	Units
C0.DDR4_TREFI	Long	0	ps
C0.DDR4_TRFC	Long	0	ps
C0.DDR4_TRFC_DLR	Long	0	ps

IMPORTANT: *The values entered are not validated, it is your responsibility to enter the right values.*

1. Generate the DDR4 IP.
2. In the **Generate Output Products** option, do not select **Generate** instead select **Skip** (Figure 5-14).



Generate Output Products <@xhdl306‹

The following output products will be generated.

Preview

- ddr4_0.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Behavioral Simulation
  - Change Log

Synthesis Options

○ Global

● Out of context per IP

Run Settings

○ On local host: Number of jobs: 12

○ On remote hosts [Configure Hosts]

● Use LSF: [Configure LSF]

Figure 5-14: **Generate Output Products – Skip**

3. To set tREFI, run the following command on the Tcl Console:

   ```
   set_property -dict [list CONFIG.C0.DDR4_TREFI <value_to_be_set>] [get_ips <ip_name>]
   ```

   For example:

   ```
   set_property -dict [list CONFIG.C0.DDR4_TREFI {7800000}] [get_ips ddr4_0]
   ```

4. To set tRFC, run the following command on the Tcl Console:

   ```
   set_property -dict [list CONFIG.C0.DDR4_TRFC <value_to_be_set>] [get_ips <ip_name>]
   ```

   For example:

   ```
   set_property -dict [list CONFIG.C0.DDR4_TRFC {260000}] [get_ips ddr4_0]
   ```

5. To set tRFC_dlr, run the following command on the Tcl Console:

   ```
   set_property -dict [list CONFIG.C0.DDR4_TRFC_DLR <value_to_be_set>] [get_ips
   <ip_name>]
   ```

   For example:

   ```
   set_property -dict [list CONFIG.C0.DDR4_TRFC_DLR {40000}] [get_ips ddr4_0]
   ```

Note: C0.DDR4_TRFC_DLR can only be set for 3DS-memory parts.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

I/O Planning

DDR3/DDR4 SDRAM I/O pin planning is completed with the full design pin planning using the Vivado I/O Pin Planner. DDR3/DDR4 SDRAM I/O pins can be selected through several Vivado I/O Pin Planner features including assignments using I/O Ports view, Package view, or Memory Bank/Byte Planner. Pin assignments can additionally be made through importing an XDC or modifying the existing XDC file.

These options are available for all DDR3/DDR4 SDRAM designs and multiple DDR3/DDR4 SDRAM IP instances can be completed in one setting. To learn more about the available Memory IP pin planning options, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 18].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

For DDR3/DDR4 SDRAM Vivado IDE, you specify the pin location constraints. For more information on I/O standard and other constraints, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 18]. The location is chosen by the Vivado IDE according to the banks and byte lanes chosen for the design.

The I/O standard is chosen by the memory type selection and options in the Vivado IDE and by the pin type. A sample for `dq[0]` is shown here.

```
set_property PACKAGE_PIN AF20 [get_ports "c0_ddr4_dq[0]"]
set_property IOSTANDARD POD12_DCI [get_ports "c0_ddr4_dq[0]"]
```

The system clock must have the period set properly:

```
create_clock -name c0_sys_clk -period 10 [get_ports c0_sys_clk_p]
```

For HR banks, update the `output_impedance` of all the ports assigned to HR banks pins using the `reset_property` command. For more information, see AR: 63852.

IMPORTANT: *Do not alter these constraints. If the pin locations need to be altered, rerun the DDR3/DDR4 SDRAM Vivado IDE to generate a new XDC file.*

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For more information on clocking, see Clocking, page 81.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The DDR3/DDR4 SDRAM tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

IMPORTANT: *The* `set_input_delay` *and* `set_output_delay` *constraints are not needed on the external memory interface pins in this design due to the calibration process that automatically runs at start-up. Warnings seen during implementation for the pins can be ignored.*

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]. For more information on simulation, see Chapter 6, Example Design and Chapter 7, Test Bench.

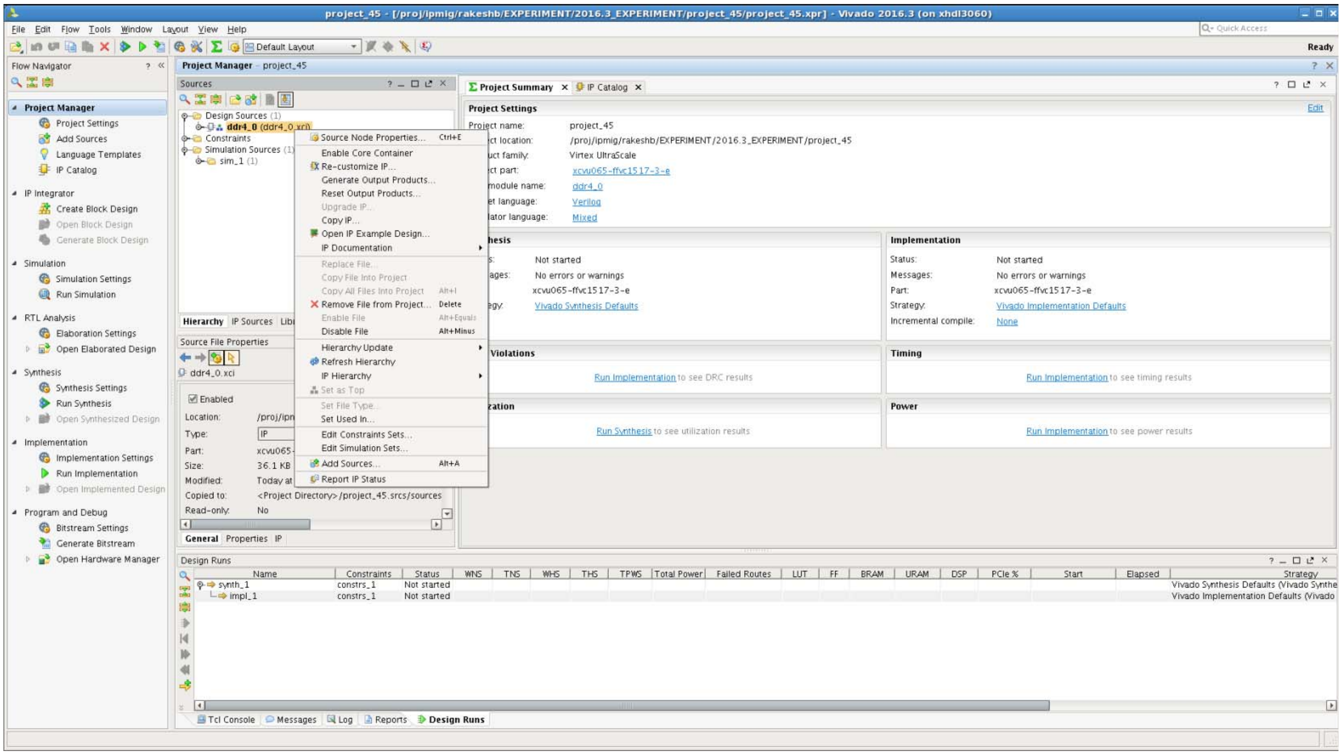
Note: The Example Design is a **Mixed Language** IP and simulations should be run with the **Simulation Language** set to **Mixed**. If the **Simulation Language** is set to **Verilog**, then it attempts to run a netlist simulation.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite. Vivado supports Open IP Example Design flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in Figure 6-1 and select **Open IP Example Design**.



☑ Enabled
Location: /proj/ipn
Type: IP
Part: xcvu065-
Size: 36.1 KB
Modified: Today at
Copied to: <Project Directory>/project_45.srcs/sources
Read-only: No

Project name: project_45
ct location: /proj/ipmig/rakeshb/EXPERIMENT/2016.3_EXPERIMENT/project_45
uct family: Virtex UltraScale
ct part: xcvu065-ffvc1517-3-e
module name: ddr4_0
et language: Verilog
lator language: Mixed

hesis
s: Not started
ages: No errors or warnings
xcvu065-ffvc1517-3-e
egy: Vivado Synthesis Defaults

Implementation
Status: Not started
Messages: No errors or warnings
Part: xcvu065-ffvc1517-3-e
Strategy: Vivado Implementation Defaults
Incremental compile: None

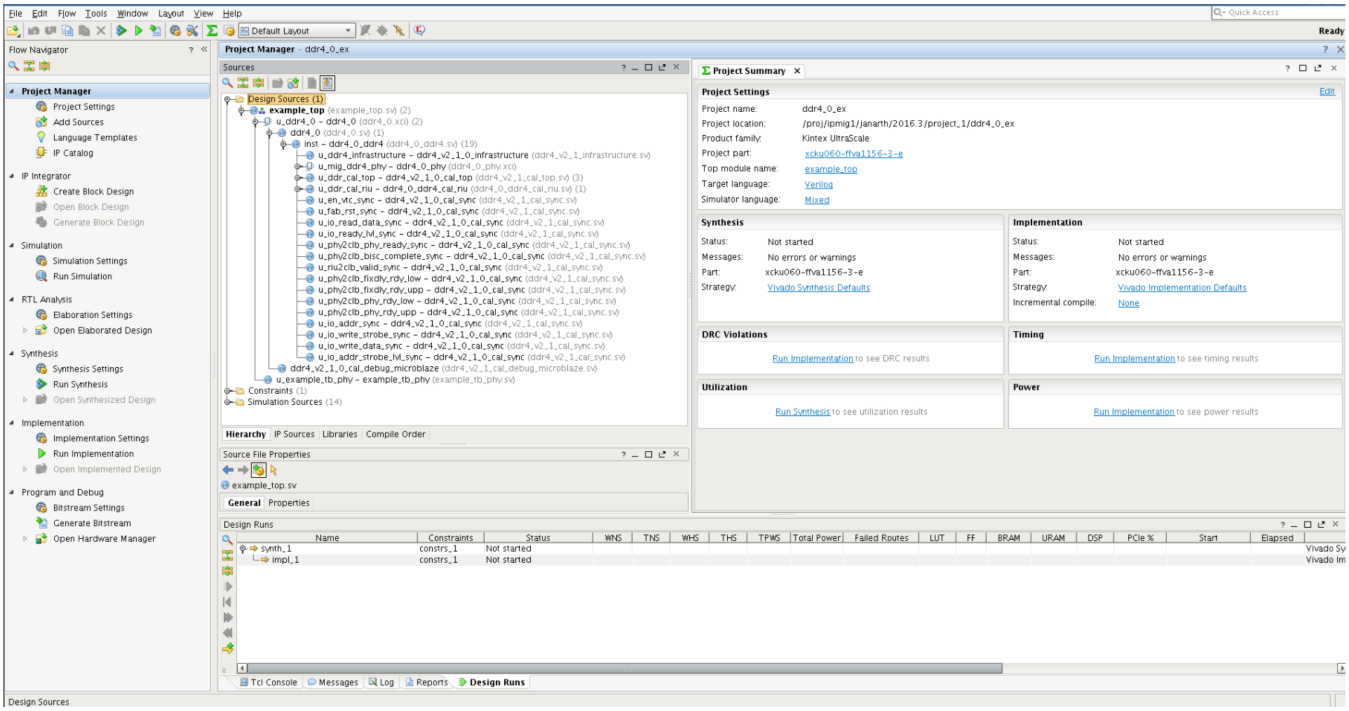
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	PCIe %	Start	Elapsed	Strategy
synth_1	constrs_1	Not started																Vivado Synthesis Defaults (Vivado Synthe
impl_1	constrs_1	Not started																Vivado Implementation Defaults (Vivado

Figure 6-1: **DDR4 Open IP Example Design**

This option creates a new Vivado project. Upon selecting the menu, a dialog box to enter the directory information for the new design project opens.

Select a directory, or use the defaults, and click **OK**. This launches a new Vivado with all of the example design files and a copy of the IP.

Figure 6-2 shows the example design with the PHY only option selected (controller module does not get generated).



```
Design Sources (1)
  example_top (example_top.sv) (2)
    u_ddr4_0 - ddr4_0 (ddr4_0.xci) (2)
      ddr4_0 (ddr4_0.sv) (1)
        inst - ddr4_0_ddr4 (ddr4_0_ddr4.sv) (19)
          u_ddr4_infrastructure - ddr4_v2_1_0_infrastructure (ddr4_v2_1_infrastructure.sv)
          u_mig_ddr4_phy - ddr4_0_phy (ddr4_0_phy.xci)
          u_ddr_cal_top - ddr4_v2_1_0_cal_top (ddr4_v2_1_cal_top.sv) (3)
          u_ddr_cal_riu - ddr4_0_ddr4_cal_riu (ddr4_0_ddr4_cal_riu.sv) (1)
          u_en_vtc_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_fab_rst_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_io_read_data_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_io_ready_lvl_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_phy2clb_phy_ready_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_phy2clb_bisc_complete_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_riu2clb_valid_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_phy2clb_fixdly_rdy_low - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_phy2clb_fixdly_rdy_upp - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_phy2clb_phy_rdy_low - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_phy2clb_phy_rdy_upp - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_io_addr_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_io_write_strobe_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_io_write_data_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
          u_io_addr_strobe_lvl_sync - ddr4_v2_1_0_cal_sync (ddr4_v2_1_cal_sync.sv)
      ddr4_v2_1_0_cal_debug_microblaze (ddr4_v2_1_cal_debug_microblaze.sv)
    u_example_tb_phy - example_tb_phy (example_tb_phy.sv)
Constraints (1)
Simulation Sources (14)
```

example_top.sv

Project Settings

Project name: ddr4_0_ex
Project location: /proj/ipmig1/janarth/2016.3/project_1/ddr4_0_ex
Product family: Kintex UltraScale
Project part: xcku060-ffva1156-3-e
Top module name: example_top
Target language: Verilog
Simulator language: Mixed

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xcku060-ffva1156-3-e
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xcku060-ffva1156-3-e
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization

Run Synthesis to see utilization results

Power

Run Implementation to see power results

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	PCIe %	Start	Elapsed	
synth_1	constrs_1	Not started																Vivado Sy
impl_1	constrs_1	Not started																Vivado Im

Figure 6-2: **Open IP Example Design with PHY Only Option Selected**

Simulating the Example Design (Designs with Standard User Interface)

The example design provides a synthesizable test bench to generate a fixed simple data pattern. DDR3/DDR4 SDRAM generates the Simple Traffic Generator (STG) module as **`example_tb`** for native interface and **`example_tb_phy`** for PHY only interface. The STG native interface generates 100 writes and 100 reads. The STG PHY only interface generates 10 writes and 10 reads.

The example design can be simulated using one of the methods in the following sections.

RECOMMENDED: *If a custom wrapper is used to simulate the example design, the following parameter should be used in the custom wrapper:*

```
parameter SIMULATION = "TRUE"
```

The parameter SIMULATION is used to disable the calibration during simulation.

Project-Based Simulation

This method can be used to simulate the example design using the Vivado Integrated Design Environment (IDE). Memory IP delivers memory models for DDR3 and IEEE encrypted memory models for DDR4.

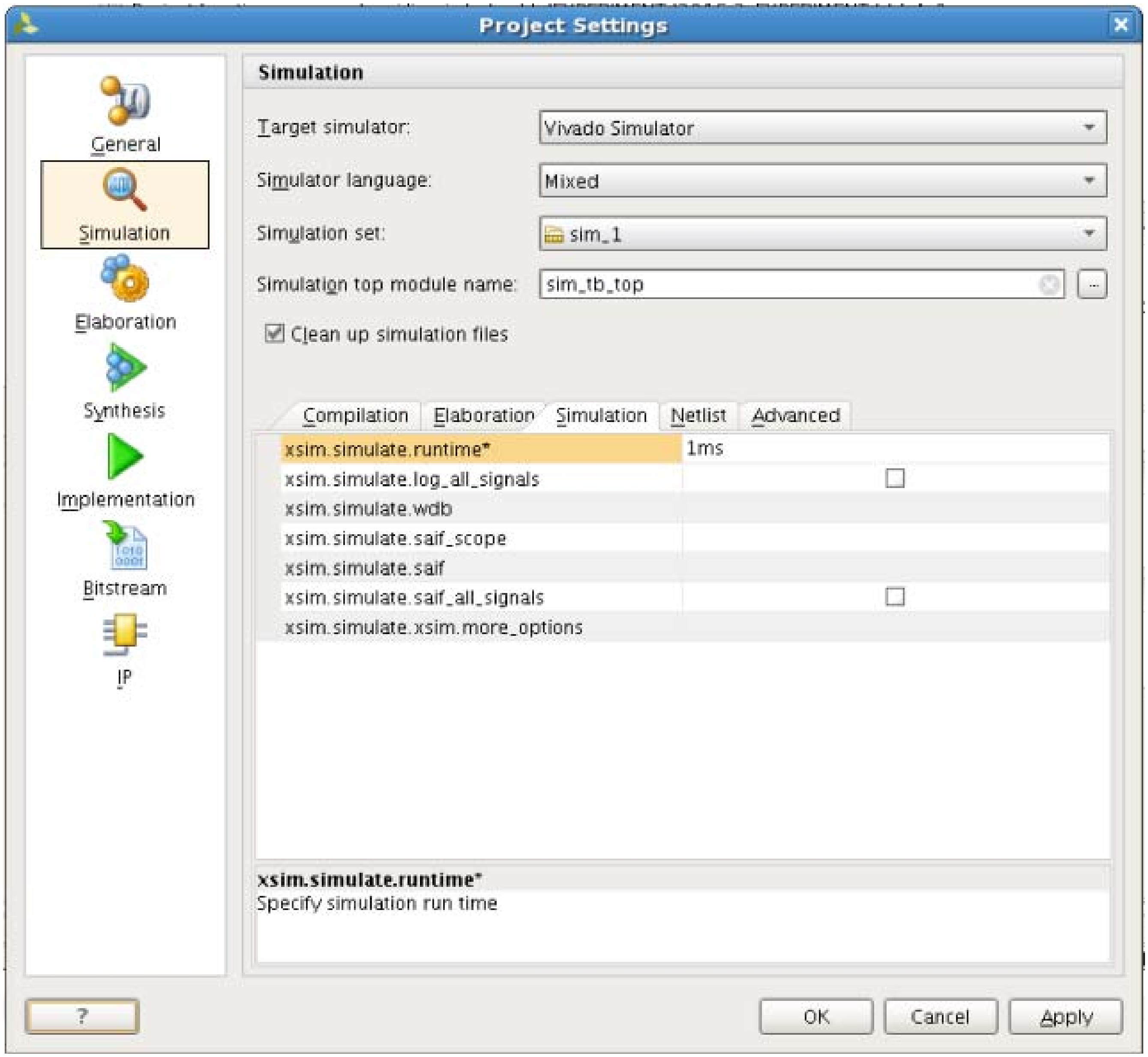
The Vivado simulator, Questa Advanced Simulator, IES, and VCS tools are used for DDR3/DDR4 IP verification at each software release. The Vivado simulation tool is used for DDR3/DDR4 IP verification from 2015.1 Vivado software release. The following subsections describe steps to run a project-based simulation using each supported simulator tool.

Project-Based Simulation Flow Using Vivado Simulator

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Vivado Simulator**.

   Under the **Simulation** tab, set the **`xsim.simulate.runtime`** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 6-3. For DDR3 simulation, set the **`xsim.simulate.xsim.more_options`** to **`-testplusarg model_data+./`**. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.

3. Set the **Simulation Language** to **Mixed**.
4. Apply the settings and select **OK**.



Simulation

Target simulator: Vivado Simulator

Simulator language: Mixed

Simulation set: sim_1

Simulation top module name: sim_tb_top

☑ Clean up simulation files

xsim.simulate.runtime*	1ms
xsim.simulate.log_all_signals	☐
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐
xsim.simulate.xsim.more_options	

xsim.simulate.runtime*

Specify simulation run time

Figure 6-3: **Simulation with Vivado Simulator**

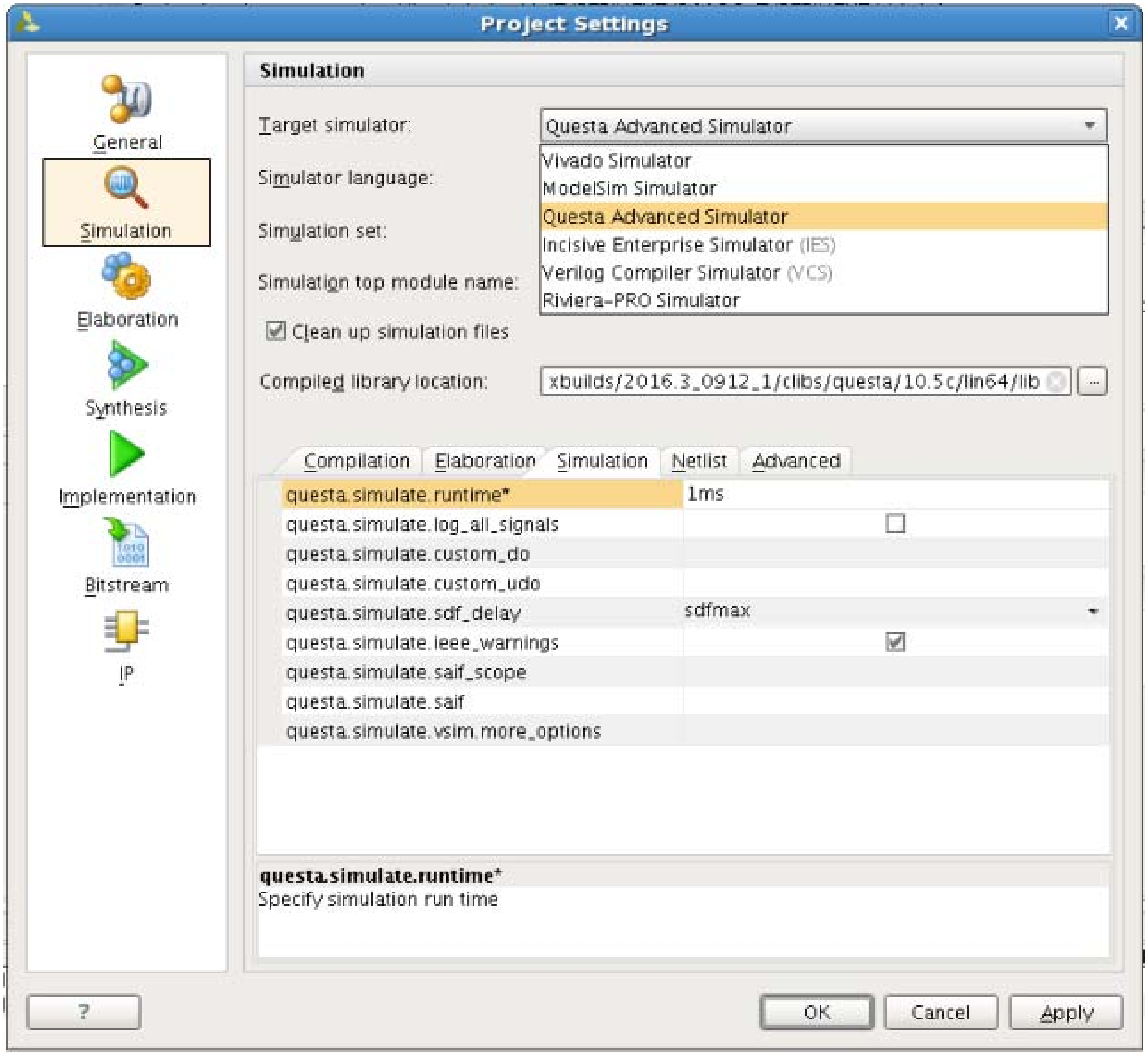
5. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 6-4.

Figure 6-4: **Run Behavioral Simulation**

6. Vivado invokes Vivado simulator and simulations are run in the Vivado simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using Questa Advanced Simulator

1. Open a DDR3/DDR4 SDRAM example Vivado project (**Open IP Example Design**...), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as **Questa Advanced Simulator**.
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `modelsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 6-5. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected. For DDR3 simulation, set the `modelsim.simulate.vsim.more_options` to `+model_data+./`.
3. Apply the settings and select **OK**.



Project Settings

Simulation

Target simulator: Questa Advanced Simulator

Vivado Simulator
ModelSim Simulator
Questa Advanced Simulator
Incisive Enterprise Simulator (IES)
Verilog Compiler Simulator (VCS)
Riviera-PRO Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: xbuilds/2016.3_0912_1/clibs/questa/10.5c/lin64/lib

Compilation | Elaboration | Simulation | Netlist | Advanced

questa.simulate.runtime*	1ms
questa.simulate.log_all_signals	☐
questa.simulate.custom_do	
questa.simulate.custom_udo	
questa.simulate.sdf_delay	sdfmax
questa.simulate.ieee_warnings	☑
questa.simulate.saif_scope	
questa.simulate.saif	
questa.simulate.vsim.more_options	

questa.simulate.runtime*
Specify simulation run time

Figure 6-5: **Simulation with Questa Advanced Simulator**

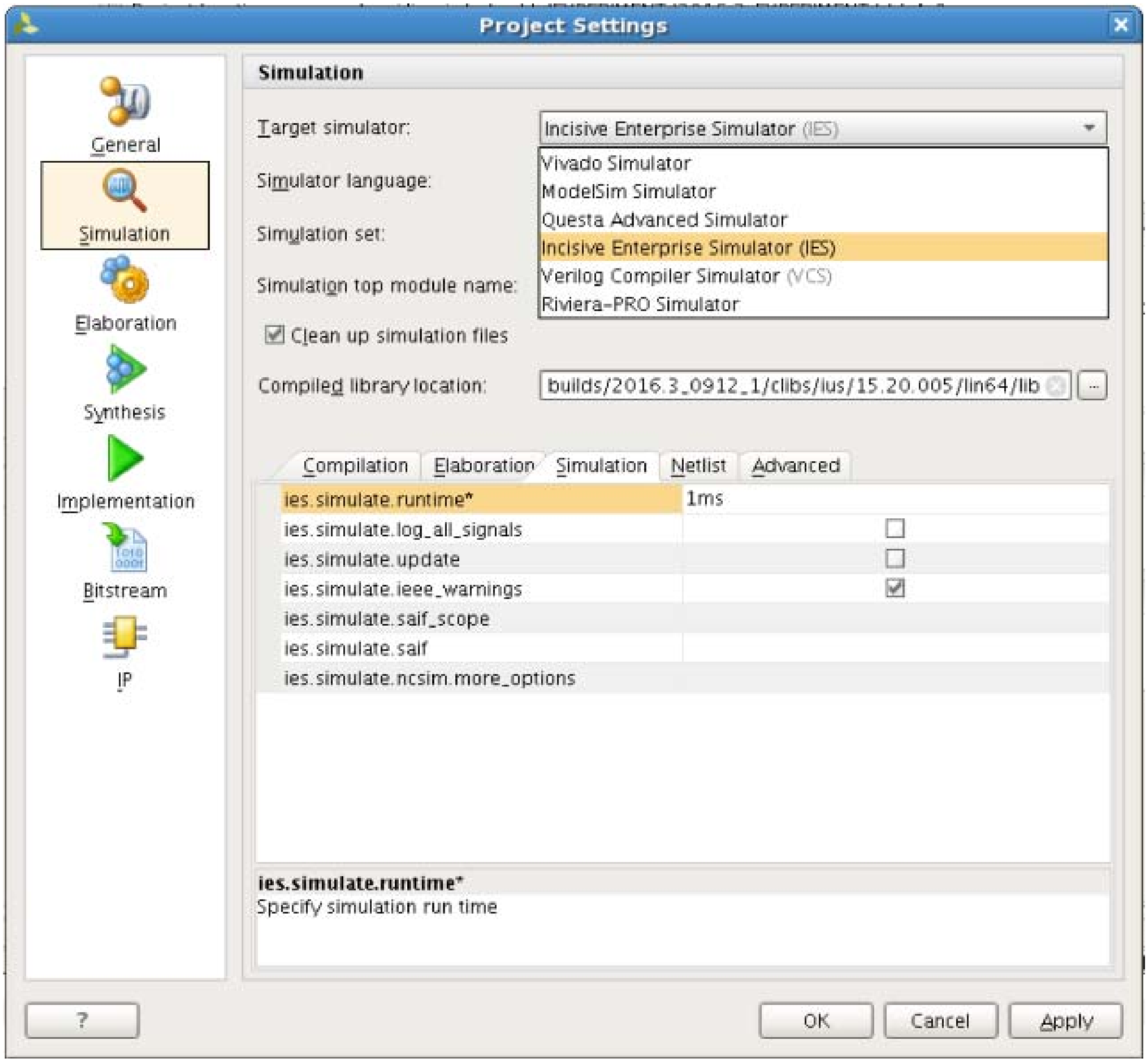
4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 6-6.

Figure 6-6: **Run Behavioral Simulation**

5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using IES

1. Open a DDR3/DDR4 SDRAM example Vivado project (**Open IP Example Design**...), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `ies.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 6-7. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected. For DDR3 simulation, set the `modelsim.simulate.vsim.more_options` to `+model_data+./`.
3. Apply the settings and select **OK**.



Simulation

Target simulator: Incisive Enterprise Simulator (IES)

- Vivado Simulator
- ModelSim Simulator
- Questa Advanced Simulator
- Incisive Enterprise Simulator (IES)
- Verilog Compiler Simulator (VCS)
- Riviera-PRO Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: builds/2016.3_0912_1/clibs/ius/15.20.005/lin64/lib

ies.simulate.runtime*	1ms
ies.simulate.log_all_signals	☐
ies.simulate.update	☐
ies.simulate.ieee_warnings	☑
ies.simulate.saif_scope	
ies.simulate.saif	
ies.simulate.ncsim.more_options	

ies.simulate.runtime*
Specify simulation run time

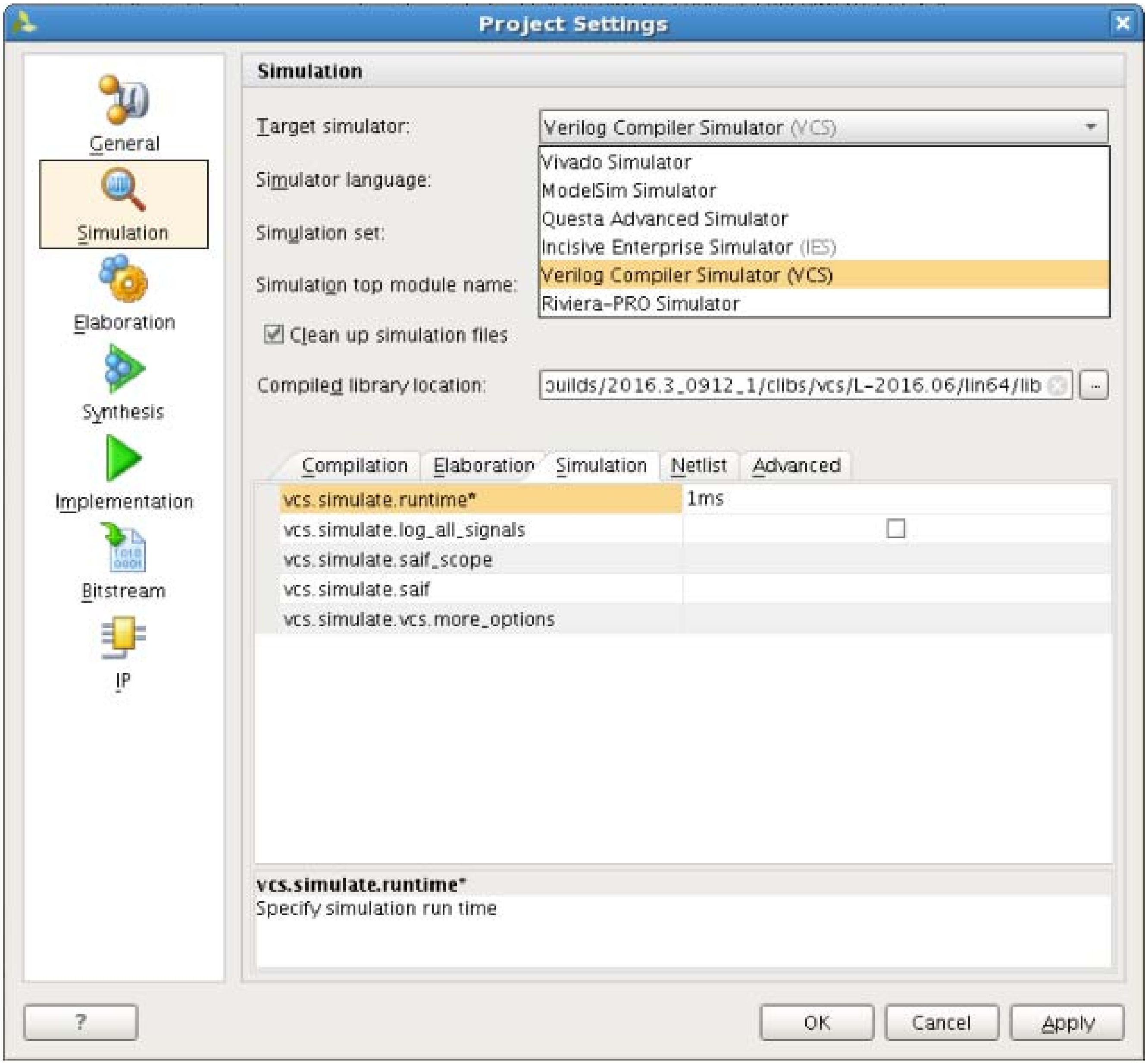
Figure 6-7: **Simulation with IES Simulator**

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 6-6.

5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using VCS

1. Open a DDR3/DDR4 SDRAM example Vivado project (**Open IP Example Design...**), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as Verilog Compiler Simulator (VCS).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `vcs.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 6-8. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected. For DDR3 simulation, set the `modelsim.simulate.vsim.more_options` to `+model_data+./`.
3. Apply the settings and select **OK**.



Target simulator: Verilog Compiler Simulator (VCS)

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: ouilds/2016.3_0912_1/clibs/vcs/L-2016.06/lin64/lib

vcs.simulate.runtime*	1ms
vcs.simulate.log_all_signals	☐
vcs.simulate.saif_scope	
vcs.simulate.saif	
vcs.simulate.vcs.more_options	

vcs.simulate.runtime*
Specify simulation run time

Figure 6-8: **Simulation with VCS Simulator**

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 6-6.
5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Simulation Speed

DDR3/DDR4 SDRAM provides a Vivado IDE option to reduce the simulation speed by selecting behavioral XIPHY model instead of UNISIM XIPHY model. Behavioral XIPHY model simulation is a default option for DDR3/DDR4 SDRAM designs. To select the simulation mode, click the **Advanced Options** tab and find the **Simulation Options** as shown in Figure 5-5.

The SIM_MODE parameter in the RTL is given a different value based on the Vivado IDE selection.

- **SIM_MODE = BFM** – If BFM mode is selected in the Vivado IDE, the RTL parameter reflects this value for the SIM_MODE parameter. This is the default option.
- **SIM_MODE = FULL** – If UNISIM mode is selected in the Vivado IDE, XIPHY UNISIMs are selected and the parameter value in the RTL is FULL.

Using Xilinx IP with Third-Party Synthesis Tools

For more information on how to use Xilinx IP with third-party synthesis tools, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation

If the GCIO pin and MMCM are not allocated in the same bank, the CLOCK_DEDICATED_ROUTE constraint must be set to BACKBONE. To use the BACKBONE route, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM input. DDR3/DDR4 SDRAM manages these constraints for designs generated with the **Reference Input Clock** option selected as **Differential** (at **Advanced > FPGA Options > Reference Input**). Also, DDR3/DDR4 SDRAM handles the IP and example design flows for all scenarios.

If the design is generated with the **Reference Input Clock** option selected as **No Buffer** (at **Advanced > FPGA Options > Reference Input**), the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation based on GCIO and MMCM allocation needs to be handled manually for the IP flow. DDR3/DDR4 SDRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations for the IP flow.

For an example design flow with **No Buffer** configurations, DDR3/DDR4 SDRAM generates the example design with differential buffer instantiation for system clock pins. DDR3/DDR4 SDRAM generates clock constraints in the `example_design.xdc`. It also generates a CLOCK_DEDICATED_ROUTE constraint as the "BACKBONE" and instantiates BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM input if the GCIO and MMCM are not in same bank to provide a complete solution. This is done for the example design flow as a reference when it is generated for the first time.

If in the example design, the I/O pins of the system clock pins are changed to some other pins with the I/O pin planner, the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation need to be managed manually. A DRC error is reported for the same.

Chapter 7

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

The intent of the performance test bench is for you to obtain an estimate on the efficiency for a given traffic pattern with the DDR3/DDR4 SDRAM controller. The test bench passes your supplied commands and address to the Memory Controller and measures the efficiency for the given pattern. The efficiency is measured by the occupancy of the `dq` bus. The primary use of the test bench is for efficiency measurements so no data integrity checks are performed. Static data is written into the memory during write transactions and the same data is always read back.

The stimulus to the Traffic Generator is provided through a `ddr3_v1_4_0_ddr3_stimulus.txt` file. The stimulus consists of command, address, and command repetition count. Each line in the stimulus file represents one stimulus (command repetition, address, and command). Multiple stimuli can be provided in a stimulus file and each stimulus is separated by the new line.

Table 7-1: **Modules for Performance Traffic Generator**

File Name	Description
ddr4_v2_2_ddr4_traffic_generator.sv	This file has the Traffic Generator code for sending out the traffic for DDR4 and also for the calculation of bus utilized.
ddr4_v2_2_ddr4_stimulus_mem_x4_x8_3ds_2h.txt ddr4_v2_2_ddr4_stimulus_mem_x4_x8_3ds_4h.txt ddr4_v2_2_0_ddr4_stimulus_mem_x4_x8.txt ddr4_v2_2_0_ddr4_stimulus_mem_x16.txt	File name depends on 3DS stack height and component width of memory part selected.
ddr3_v1_4_ddr3_traffic_generator.sv	This file has the Traffic Generator code for sending out the traffic for DDR3 and also for the calculation of bus utilized.
ddr3_v1_4_0_ddr3_stimulus.txt	These files have the stimulus with Writes, Reads, and NOPs for DDR3 for the calculation of bus utilization.

Stimulus Pattern

Stimulus pattern for non-3DS part is 48 bits and the format is described in Table 7-2. For a 3DS part, stimulus pattern is 52 bits and is described in Table 7-3. The stimulus pattern description for non-3DS and 3DS parts are shown in Table 7-4.

Table 7-2: **Stimulus Command Pattern for Non-3DS**

Command Repeat[47:40]	Address [39:4]	Command[3:0]

Table 7-3: **Stimulus Command Pattern for 3DS**

Command Repeat[51:44]	Address [43:4]	Command[3:0]

Table 7-4: **Stimulus Pattern Description**

Signal	Description
Command[3:0]	This corresponds to the WRITE/READ/NOP command that is sent to the user interface.
Address[35:0]/ Address[39:0]	This corresponds to the address to the user interface. For non-3DS part, the width is 36 bits and for 3DS the width is 40 bits.
Command Repeat[7:0]	This corresponds to the repetition count of the command. Up to 128 repetitions can be made for a command. In the burst length of eight mode, 128 transactions fill up the page in the memory.

Command Encoding (Command[3:0])

Table 7-5: **Command Description**

Command	Code	Description
WRITE	0	This corresponds to the Write operation that needs to be performed.
READ	1	This corresponds to the Read operation that needs to be performed.
NOP	7	This corresponds to the idle situation for the bus.

Address Encoding (Address[35:0]/Address[39:0])

Address is encoded in the stimulus as per Figure 7-1 to Figure 7-6. All the address fields need to be entered in the hexadecimal format. All the address fields are the width that is divisible by four to enter in the hexadecimal format. The test bench only sends the required bits of an address field to the Memory Controller.

For example, an eight bank configuration only bank Bits[2:0] is sent to the Memory Controller and the remaining bits are ignored. The extra bits for an address field are provided for you to enter the address in a hexadecimal format. You must confirm the value entered corresponds to the width of a given configuration.

Table 7-6: **Address Encoded for Non-3DS**

Rank[3:0]	Bank[3:0]	Row[15:0]	Column[11:0]

Table 7-7: **Address Encoded for 3DS**

Logical Rank[3:0]	Rank[3:0]	Bank[3:0]	Row[15:0]	Column[11:0]

- **Column Address (Column[11:0])** – Column Address in the stimulus is provided with a maximum of 12 bits, but you need to address this based on the column width parameter set in your design.
- **Row Address (Row[15:0])** – Row address in the stimulus is provided with a maximum of 16 bits, but you need to address this based on the row width parameter set in your design.
- **Bank Address (Bank[3:0])** – Bank address in the stimulus is provided with a maximum of four bits, but you need to address this based on the bank width parameter set in your design.

  Note: For DDR4, use the 2-bit LSB for Bank Address and two bits of MSB for Bank Groups.
- **Rank Address (Rank[3:0])** – Rank address in the stimulus is provided with a maximum of four bits, but you need to address this based on the rank width parameter set in your design.
- **Logical Rank[3:0]** – Logical rank in the stimulus is provided with a maximum of four bits, This is based on a stack height parameter set in your design.

The address is assembled based on the top-level MEM_ADDR_ORDER parameter and sent to the user interface.

Command Repeat (Command Repeat[7:0])

The command repetition count is the number of time the respective command is repeated at the User Interface. The address for each repetition is incremented by 8. The maximum repetition count is 128. The test bench does not check for the column boundary and it wraps around if the maximum column limit is reached during the increments. The 128 commands fill up the page. For any column address other than 0, the repetition count of 128 ends up crossing the column boundary and wrapping around to the start of the column address.

Bus Utilization

The bus utilization is calculated at the User Interface taking total number of Reads and Writes into consideration and the following equation is used:

$$bw\_cumulative = \frac{((rd\_command\_cnt + wr\_command\_cnt) \times (BURST\_LEN / 2) \times 100)}{((end\_of\_stimulus - calib\_done) / tCK);}$$

Equation 7-1

- **`BURST_LEN`** equals 8 for DDR3 and DDR4. **`BURST_LEN`** is divided by 2 in the BW formula to give the number tCK of data activity on the DDR bus for each read and write.
- **`rd_command_cnt`** and **`wr_command_cnt`** are the total number of read and write commands accepted at the User Interface between **`calib_done`** and **`end_of_stimulus`**.
- **`end_of_stimulus`** is the time when all the commands are done.
- **`calib_done`** is the time when the calibration is done.

Example Patterns

These examples are based on the MEM_ADDR_ORDER set to BANK_ROW_COLUMN.

Single Read Pattern

00_0_2_000F_00A_1 – This pattern is a single read from 10th column, 15th row, and second bank.

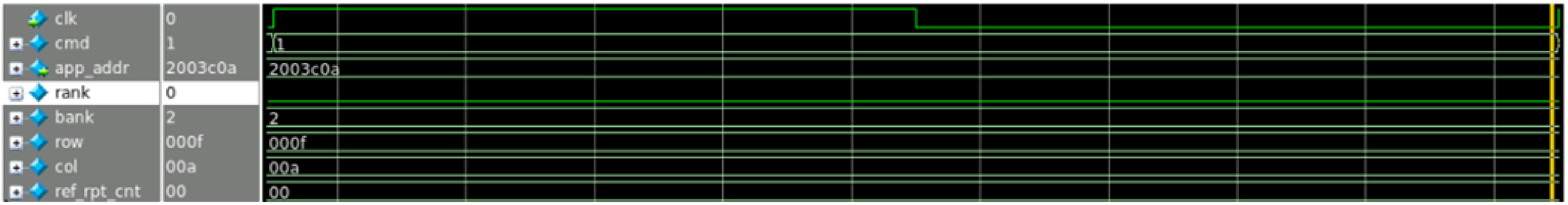


Figure 7-1: **Single Read Pattern**

Single Write Pattern

00_0_1_0040_010_0 – This pattern is a single write to the 32nd column, 128th row, and first bank.

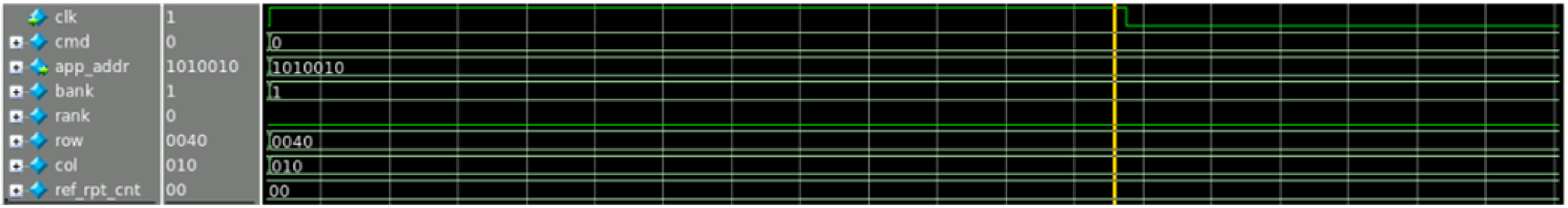


Figure 7-2: **Single Write Pattern**

Single Write and Read to Same Address

00_0_2_000F_00A_0 – This pattern is a single write to 10th column, 15th row, and second bank.

00_0_2_000F_00A_1 – This pattern is a single read from 10th column, 15th row, and second bank.

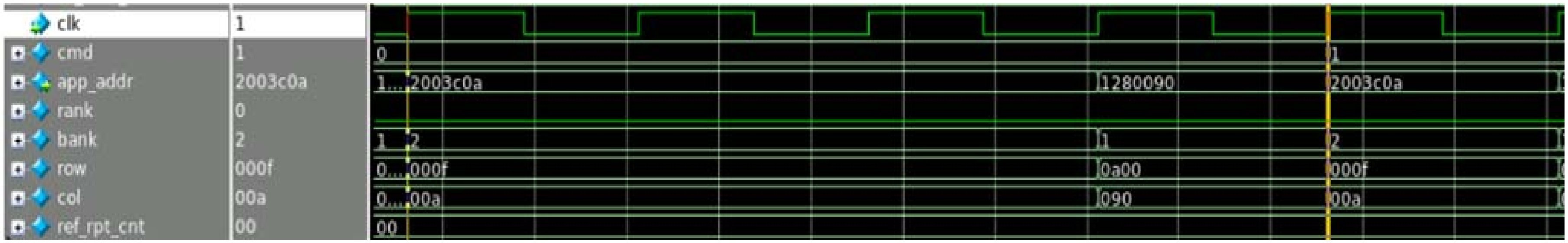


Figure 7-3: **Single Write and Read to Same Address**

Multiple Writes and Reads with Same Address

0A_0_0_0010_000_0 – This corresponds to 11 writes with address starting from 0 to 80 which can be seen in the column.

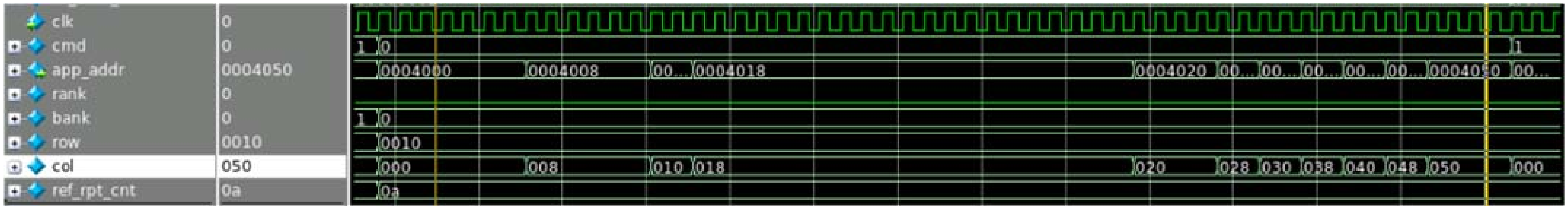


Figure 7-4: **Multiple Writes with Same Address**

0A_0_0_0010_000_1 – This corresponds to 11 reads with address starting from 0 to 80 which can be seen in the column.

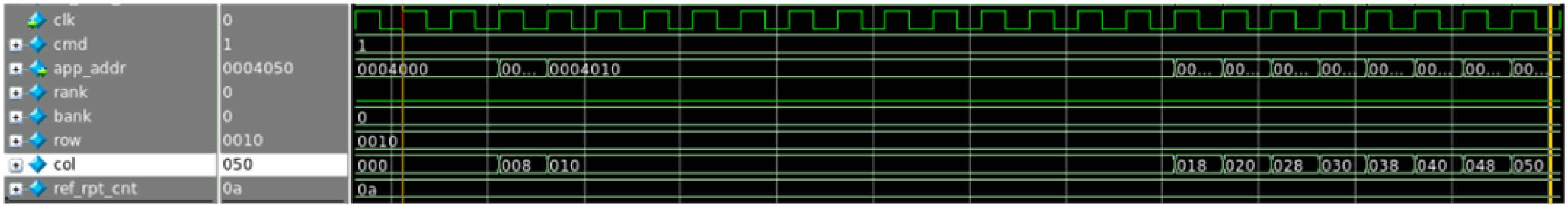


Figure 7-5: **Multiple Reads with Same Address**

Page Wrap during Writes

0A_0_2_000F_3F8_0 – This corresponds to 11 writes with column address wrapped to the starting of the page after one write.

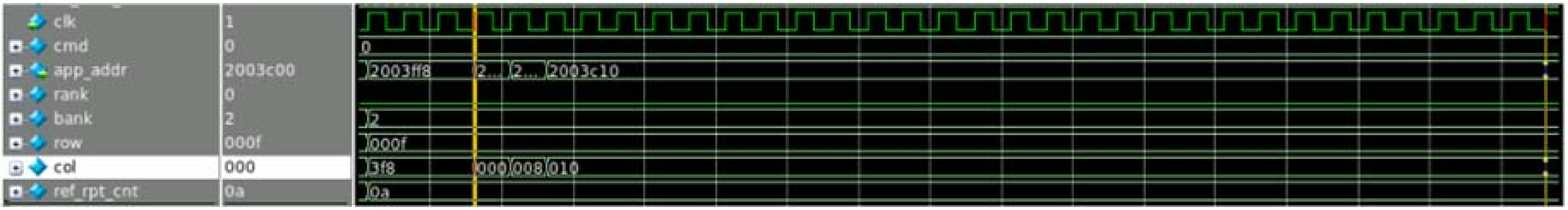


Figure 7-6: **Page Wrap during Writes**

Simulating the Performance Traffic Generator

Note: This is not supported when the AXI interface is enabled.

After opening the `example_design` project, follow the steps to run the performance traffic generator.

1. In the Vivado Integrated Design Environment (IDE), open the **Simulation Sources** section and double-click the `sim_tb_top.sv` file to open it in **Edit** mode. Or open the file from the following location, `<project_dir>/example_project/<component_name>_example/<component_name>_example.srcs/sim_1/imports/tb/sim_tb_top.sv`.
2. Add a `` `define BEHV line in the file[sim_tb_to.sv] `` and save it.
3. Go to the **Simulation Settings** in the Vivado IDE.
   a. Select **Target Simulator** from the supported simulators (supported simulators are Questa Advanced Simulator, Incisive Enterprise Simulator (IES), Verilog Compiler Simulator (VCS), and Vivado simulator). Browse to the compiled libraries location and set the path on the **Compiled Libraries Location** option as per the **Target Simulator**.
   b. Under the **Simulation** tab, set the simulation run-time to 1 ms (there are simulation RTL directives which stop the simulation after a certain period of time, which is less than 1 ms). The **Generate Scripts Only** option generates simulation scripts only.

      To run behavioral simulation, the **Generate Scripts Only** option must be de-selected. For DDR3 simulation, set the `more_options` for the following:

      ```
      +model_data+./ for Questa/IES/VCS simulators
      -testplusarg model_data+./for Vivado simulator
      ```
   c. Click **Apply** to save these settings.
4. Click **Run Simulations**.
5. Check the transcript for the results.

SECTION III: LPDDR3

Overview

IMPORTANT: *This document supports LPDDR3 SDRAM core v1.0.*

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
    - Clocking
    - Resets
    - Protocol Description
    - Customizing and Generating the Core
    - Example Design

Core Overview

The Xilinx UltraScale™ architecture includes the LPDDR3 SDRAM core. This core provides solutions for interfacing with the SDRAM memory type. The UltraScale architecture for the LPDDR3 core is organized in the following high-level blocks:

- **Controller** – The controller accepts burst transactions from the user interface and generates transactions to and from the SDRAM. The controller takes care of the SDRAM timing parameters and refresh. It coalesces write and read transactions to reduce the number of dead cycles involved in turning the bus around. The controller also reorders commands to improve the utilization of the data bus to the SDRAM.

- **Physical Layer** – The physical layer provides a high-speed interface to the SDRAM. This layer includes the hard blocks inside the FPGA and the soft blocks calibration logic necessary to ensure optimal timing of the hard blocks interfacing to the SDRAM.

  The new hard blocks in the UltraScale architecture allow interface rates of up to 1,600 Mb/s to be achieved. The application logic is responsible for all SDRAM transactions, timing, and refresh.

  These hard blocks include:

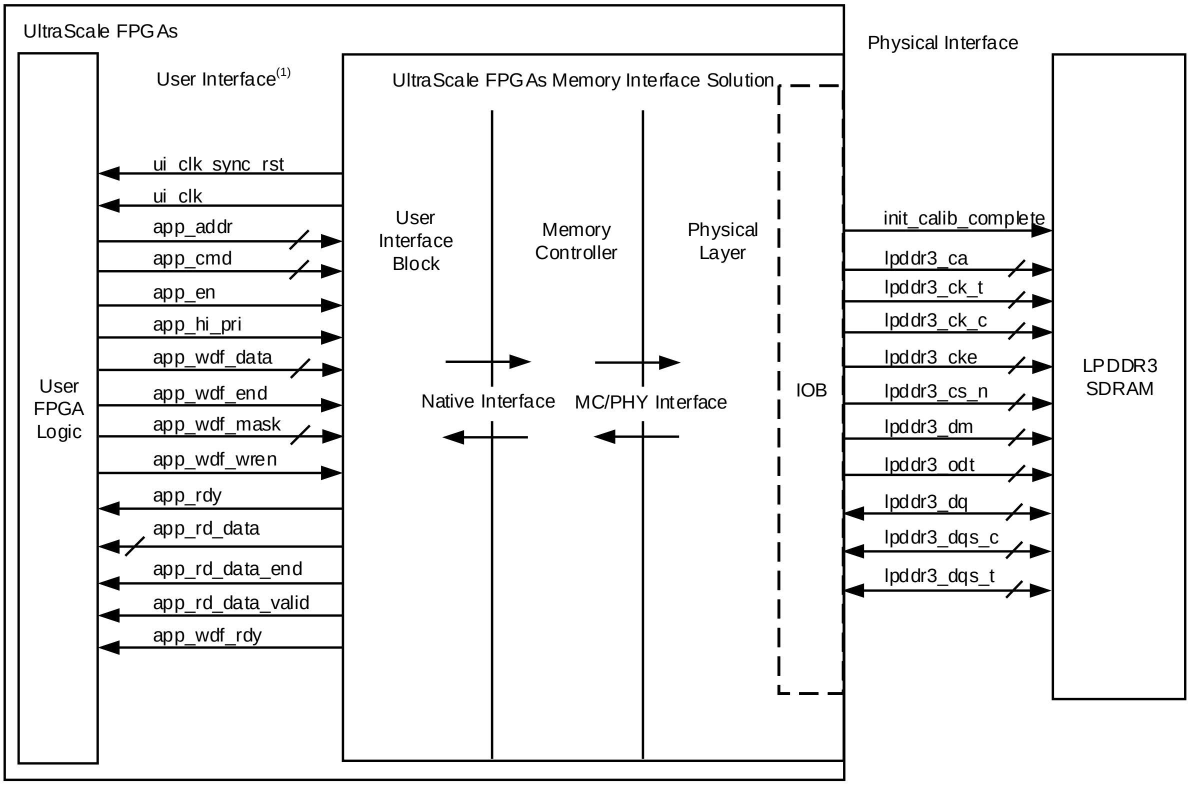
    - Data serialization and transmission
    - Data capture and deserialization
    - High-speed clock generation and synchronization
    - Coarse and fine delay elements per pin with voltage and temperature tracking

  - The soft blocks include:
    - **Memory Initialization** – The calibration modules provide a JEDEC®-compliant initialization routine for the particular memory type. The delays in the initialization process can be bypassed to speed up simulation time, if desired.
    - **Calibration** – The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance.

      Results of the calibration process are available through the Xilinx debug tools. After completion of calibration, the PHY layer presents raw interface to the SDRAM.

- **Application Interface** – The user interface layer provides a simple FIFO-like interface to the application. Data is buffered and read data is presented in request order.

  The above user interface is layered on top of the native interface to the controller. The native interface is not accessible by the user application and has no buffering and presents return data to the user interface as it is received from the SDRAM which is not necessarily in the original request order. The user interface then buffers the read and write data and reorders the data as needed.



1. System clock (sys_clk_p and sys_clk_n/sys_clk_i) and system reset (sys_rst_n) port connections are not shown in block diagram.

X18838-082117

Figure 8-1: **UltraScale Architecture-Based FPGAs LPDDR3 Memory Interface Solution**

Feature Summary

- Density support
    - Support 8 GB for component
    - Other densities for memory device support is available through custom part selection
- 8-bank support
- x32 device support
    - x16 memory device support is available through custom part selection
- 8:1 DQ:DQS ratio support for all devices
- 8-word burst support
- Support for 6 to 12 cycles of column-address strobe (CAS) latency (CL)
- On-die termination (ODT) support
- Support for 3 to 6 cycles of CAS write latency
- JEDEC® -compliant LPDDR3 initialization support
- Source code delivery in Verilog
- 4:1 memory to FPGA logic interface clock ratio
- Open, closed, and transaction based pre-charge controller policy
- Interface calibration and training information available through the Vivado Design Suite hardware manager

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

Standards

This core supports DRAMs that are compliant to the JESD209-3C, *LPDDR3 SDRAM Standard,* JEDEC® Solid State Technology Association [Ref 1].

For more information on UltraScale™ architecture documents, see References, page 789.

Performance

Maximum Frequencies

For more information on the maximum frequencies, see the following documentation:

- *Kintex UltraScale FPGAs Data Sheet, DC and AC Switching Characteristics* (DS892) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6]
- *UltraScale Maximum Memory Performance Utility* (XTP414) [Ref 21]

Resource Utilization

For full details about performance and resource utilization, visit Performance and Resource Utilization.

Port Descriptions

For a complete Memory Controller solution there are three port categories at the top-level of the memory interface core called the "user design."

- The first category is the memory interface signals that directly interfaces with the SDRAM. These are defined by the JEDEC specification.
- The second category is the application interface signals. These are described in the Protocol Description, page 296.
- The third category includes other signals necessary for proper operation of the core. These include the clocks, reset, and status signals from the core. The clocking and reset signals are described in their respective sections.

  The active-High **`init_calib_complete`** signal indicates that the initialization and calibration are complete and that the interface is now ready to accept commands for the interface.

Core Architecture

This chapter describes the UltraScale™ architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Overview

The UltraScale architecture-based FPGAs Memory Interface Solutions is shown in Figure 10-1.

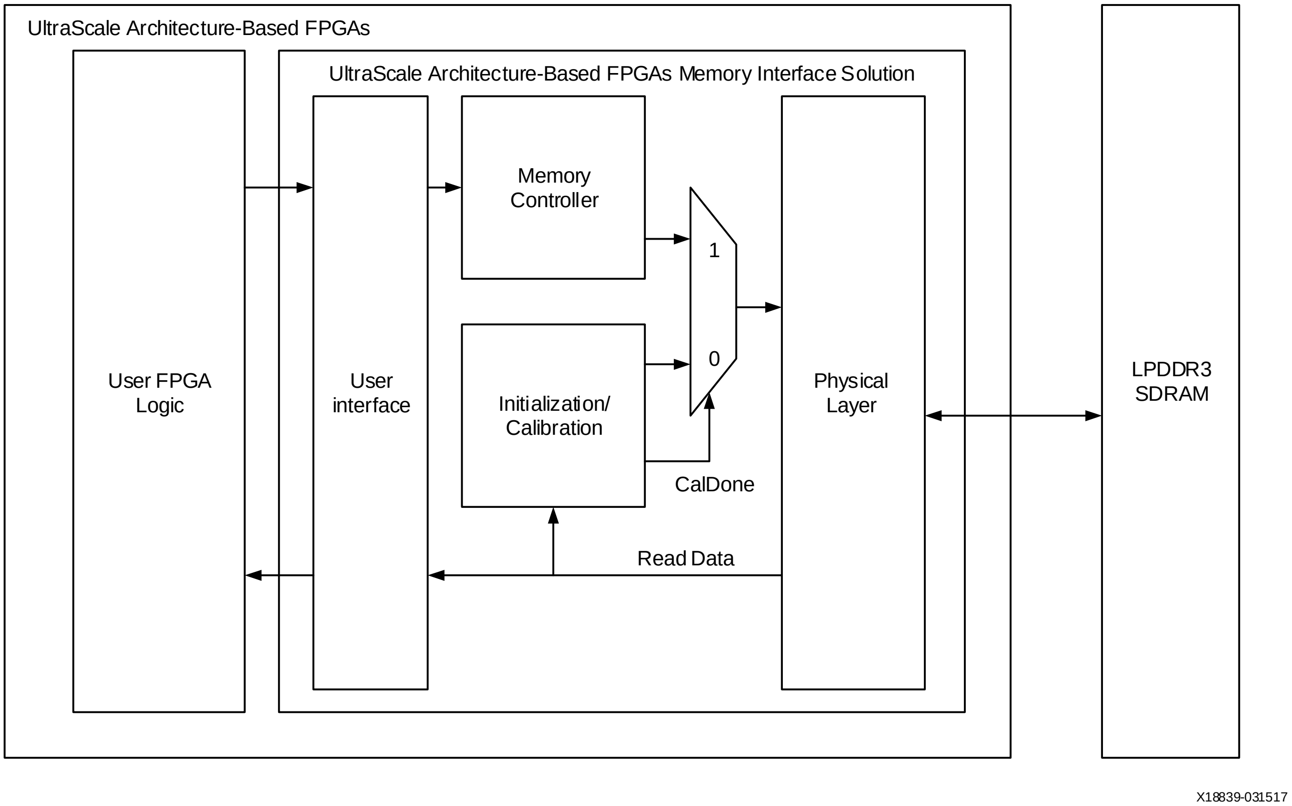


Figure 10-1: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core Architecture**

Memory Controller

In the core default configuration, the Memory Controller (MC) resides between the user interface (UI) block and the physical layer. This is depicted in Figure 10-2.

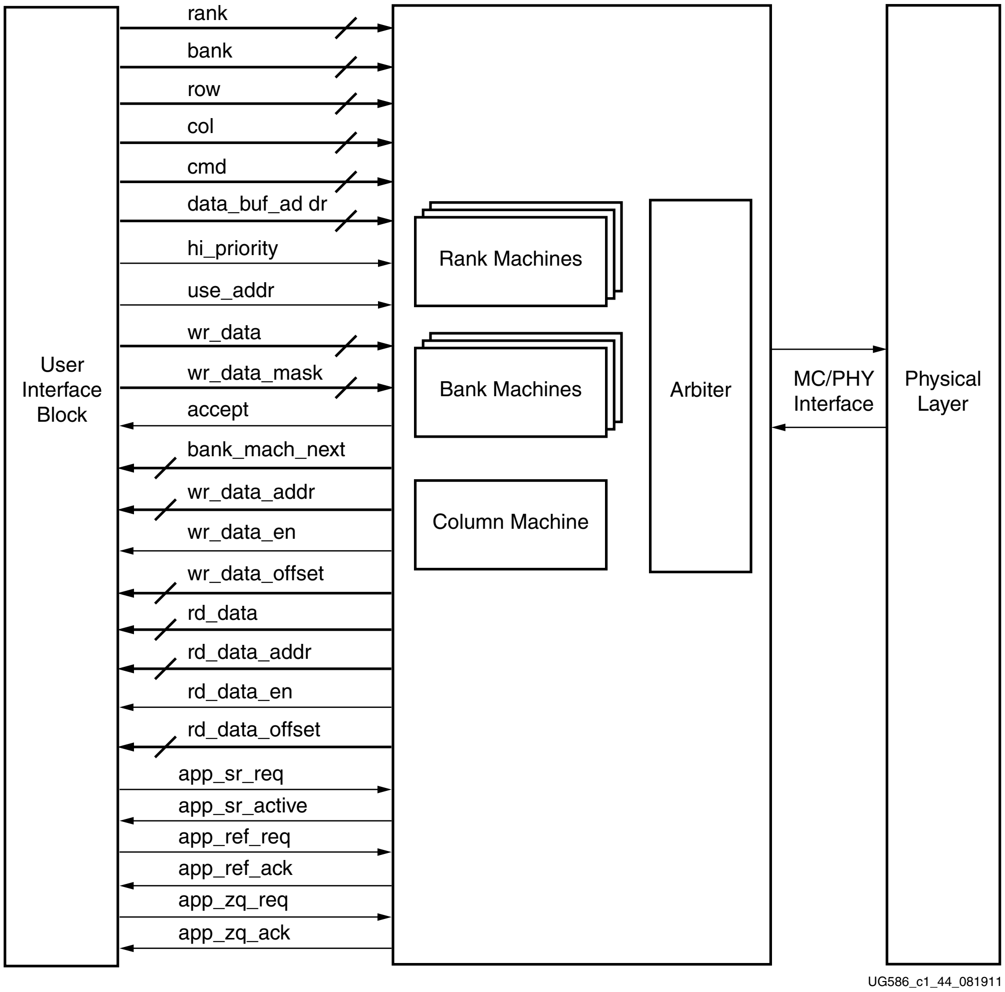


Figure 10-2: **Memory Controller Block Diagram**

The Memory Controller is the primary logic block of the memory interface. The Memory Controller receives requests from the UI and stores them in a logical queue. Requests are optionally reordered to optimize system throughput and latency.

The Memory Controller block is organized as four main pieces:

- A configurable number of "bank machines"
- A configurable number of "rank machines"
- A column machine
- An arbitration block

Bank Machines

Most of the Memory Controller logic resides in the bank machines. Bank machines correspond to DRAM banks. A given bank machine manages a single DRAM bank at any given time. However, bank machine assignment is dynamic, so it is not necessary to have a bank machine for each physical bank. The number of banks can be configured to trade off between area and performance. This is discussed in greater detail in the Precharge Policy section.

The duration of a bank machine assignment to a particular DRAM bank is coupled to user requests rather than the state of the target DRAM bank. When a request is accepted, it is assigned to a bank machine. When a request is complete, the bank machine is released and is made available for assignment to another request. Bank machines issue all the commands necessary to complete the request.

On behalf of the current request, a bank machine must generate row commands and column commands to complete the request. Row and column commands are independent but must adhere to DRAM timing requirements.

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

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

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

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

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

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

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

Rank Machines

The rank machines correspond to DRAM ranks. Rank machines monitor the activity of the bank machines and track rank or device-specific timing parameters. For example, a rank machine monitors the number of activate commands sent to a rank within a time window. After the allowed number of activates have been sent, the rank machine generates an inhibit signal that prevents the bank machines from sending any further activates to the rank until the time window has shifted enough to allow more activates. Rank machines are statically assigned to a physical DRAM rank.

Column Machine

The single column machine generates the timing information necessary to manage the DQ data bus. Although there can be multiple DRAM ranks, because there is a single DQ bus, all the columns in all DRAM ranks are managed as a single unit. The column machine monitors commands issued by the bank machines and generates inhibit signals back to the bank machines so that the DQ bus is utilized in an orderly manner.

Arbitration Block

The arbitration block receives requests to send commands to the DRAM array from the bank machines. Row commands and column commands are arbitrated independently. For each command opportunity, the arbiter block selects a row and a column command to forward to the physical layer. The arbitration block implements a round-robin protocol to ensure forward progress.

Reordering

DRAM accesses are broken into two quasi-independent parts, row commands and column commands. Each request occupies a logical queue entry, and each queue entry has an associated bank machine. These bank machines track the state of the DRAM rank or bank it is currently bound to, if any.

If necessary, the bank machine attempts to activate the proper rank, bank, or row on behalf of the current request. In the process of doing so, the bank machine looks at the current state of the DRAM to decide if various timing parameters are met. Eventually, all timing parameters are met and the bank machine arbitrates to send the activate. The arbitration is done in a simple round-robin manner. Arbitration is necessary because several bank machines might request to send row commands (activate and precharge) at the same time.

Not all requests require an activate. If a preceding request has activated the same rank, bank, or row, a subsequent request might inherit the bank machine state and avoid the precharge/activate penalties.

After the necessary rank, bank, or row is activated and the RAS to CAS delay timing is met, the bank machine tries to issue the CAS-READ or CAS-WRITE command. Unlike the row command, all requests issue a CAS command. Before arbitrating to send a CAS command, the bank machine must look at the state of the DRAM, the state of the DQ bus, priority, and ordering. Eventually, all these factors assume their favorable states and the bank machine arbitrates to send a CAS command. In a manner similar to row commands, a round-robin arbiter uses a priority scheme and selects the next column command.

The round-robin arbiter itself is a source of reordering. Assume for example that an otherwise idle Memory Controller receives a burst of new requests while processing a refresh. These requests queue up and wait for the refresh to complete. After the DRAM is ready to receive a new activate, all waiting requests assert their arbitration requests simultaneously. The arbiter selects the next activate to send based solely on its round-robin algorithm, independent of request order. Similar behavior can be observed for column commands.

The controller supports NORM ordering mode. In this mode, the controller reorders reads but not writes as needed to improve efficiency. All write requests are issued in the request order relative to all other write requests, and requests within a given rank-bank retire in order. This ensures that it is not possible to observe the result of a later write before an earlier write completes.

Precharge Policy

The controller implements an aggressive precharge policy. The controller examines the input queue of requests as each transaction completes. If no requests are in the queue for a currently open bank/row, the controller closes it to minimize latency for requests to other rows in the bank. Because the queue depth is equal to the number of bank machines, greater efficiency can be obtained by increasing the number of bank machines (nBANK_MACHS). As this number is increased, FPGA logic timing becomes more challenging. In some situations, the overall system efficiency can be greater with an increased number of bank machines and a lower memory clock frequency. Simulations should be performed with the target design command behavior to determine the optimum setting.

PHY

The PHY is considered the low-level physical interface to an external LPDDR3 SDRAM device as well as all calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

IMPORTANT: *The PHY interface is not DFI-compliant.*

Overall PHY Architecture

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

The Memory Controller and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is either divided by four. A more detailed block diagram of the PHY design is shown in Figure 10-3.

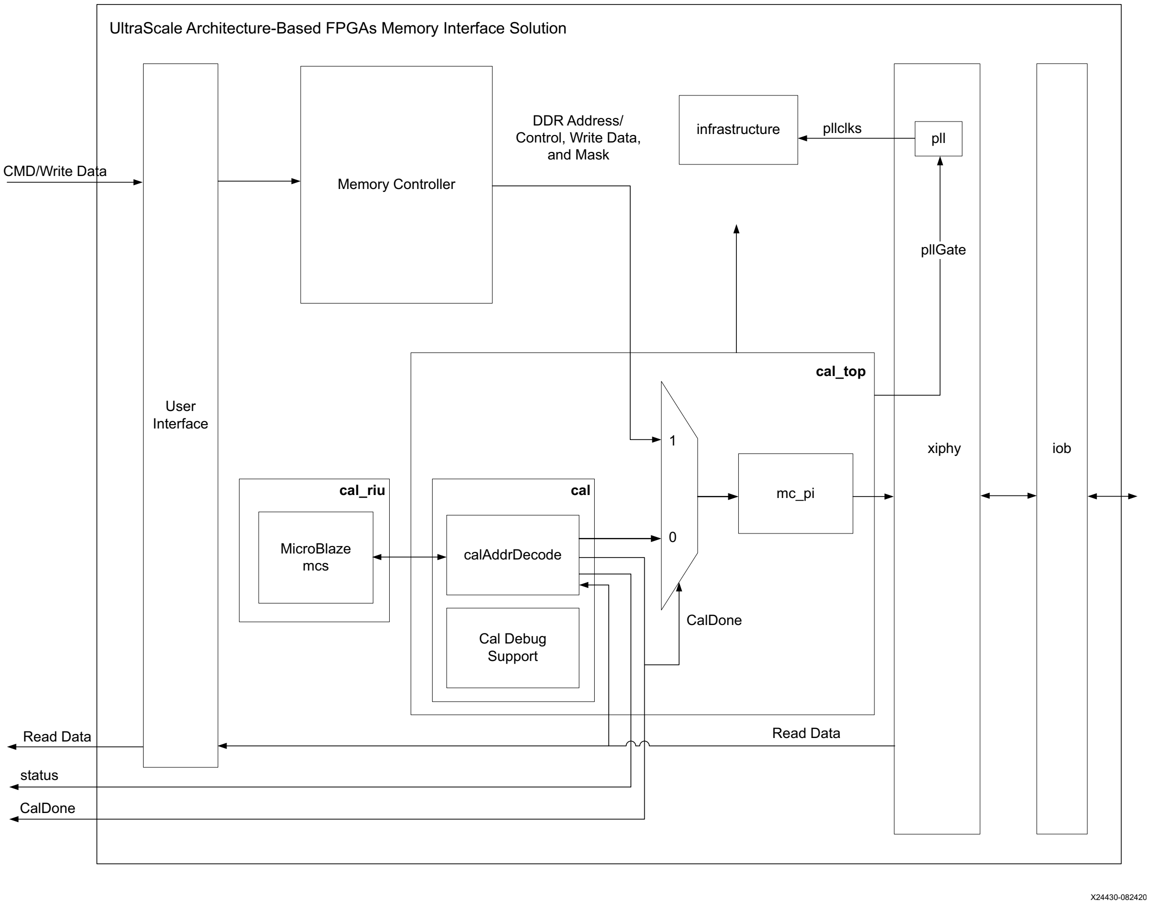


Figure 10-3: **PHY Block Diagram**

The Memory Controller is designed to separate out the command processing from the low-level PHY requirements to ensure a clean separation between the controller and physical layer. The command processing can be replaced with custom logic if desired, while the logic for interacting with the PHY stays the same and can still be used by the calibration logic.

Table 10-1: **PHY Modules**

Module Name	Description
<module>_...cal.sv	Contains <module>_...mc_pi.sv, MUXes, and MicroBlaze processing system and associated logic.
<module>_...cal_addr_decode.sv	FPGA logic interface for the MicroBlaze processor.
<module>_...config_rom.sv	Configuration storage for calibration options.
microblaze_mcs_0.sv	MicroBlaze MCS module
<module>_...iob.sv	Instantiates all byte IOB modules.
<module>_...iob_byte.sv	Generates the I/O buffers for all the signals in a given byte lane.
<module>_...xiphy.sv	Top-level XIPHY module.
<module>_...phy.sv	Top-level of the PHY, contains pll and xiphy.sv modules.

The PHY architecture encompasses all of the logic contained in **`<module>_...phy.sv`**. The PHY contains wrappers around dedicated hard blocks to build up the memory interface from smaller components. A byte lane contains all of the clocks, resets, and datapaths for a given subset of I/O. Multiple byte lanes are grouped together, along with dedicated clocking resources, to make up a single bank memory interface. Each nibble in the PHY contains a Register Interface Unit (RIU), a dedicated integrated block in the XIPHY that provides an interface to the general interconnect logic for changing settings and delays for calibration. For more information on the hard silicon physical layer architecture, see the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7].

The memory initialization, calibration, and training are implemented by an embedded MicroBlaze™ processor. The MicroBlaze Controller System (MCS) is configured with an I/O Module and a block RAM. The **`<module>_...cal_addr_decode.sv`** module provides the interface for the processor to the rest of the system and implements helper logic. The **`<module>_...config_rom.sv`** module stores settings that control the operation of initialization and calibration, providing run time options that can be adjusted without having to recompile the source code.

The address unit connects the MCS to the local register set and the PHY by performing address decode and control translation on the I/O module bus from spaces in the memory map and MUXing return data (**`<module>_...cal_addr_decode.sv`**). In addition, it provides address translation (also known as “mapping”) from a logical conceptualization of the DRAM interface to the appropriate pinout-dependent location of the delay control in the PHY address space.

Although the calibration architecture presents a simple and organized address map for manipulating the delay elements for individual data, control and command bits, there is flexibility in how those I/O pins are placed. For a given I/O placement, the path to the FPGA logic is locked to a given pin. To enable a single binary software file to work with any memory interface pinout, a translation block converts the simplified RIU addressing into the pinout-specific RIU address for the target design (see Table 10-2).

The specific address translation is written by LPDDR3 SDRAM after a pinout is selected and cannot be modified. The code shows an example of the RTL structure that supports this.

```
Casez(io_address)// MicroBlaze I/O module address
  // … static address decoding skipped
  //========================================//
  //===========DQ ODELAYS===================//
  //========================================//
  //Byte0
28`h0004100: begin //c0_lpddr3_dq[0] IO_L20P_T3L_N2_AD1P_44
    riu_addr_cal = 6`hD;
    riu_nibble = 'h6;
  end
  // … additional dynamic addressing follows
```

In this example, `DQ0` is pinned out on Bit[0] of nibble 0 (nibble 0 according to instantiation order). The RIU address for the ODELAY for Bit[0] is `0x0D`. When `DQ0` is addressed — indicated by address `0x000_4100`), this snippet of code is active. It enables nibble 0 (decoded to one-hot downstream) and forwards the address `0x0D` to the RIU address bus.

The MicroBlaze I/O module interface is not always fast enough for implementing all of the functions required in calibration. A helper circuit implemented in `<module>_...cal_addr_decode.sv` is required to obtain commands from the registers and translate at least a portion into single-cycle accuracy for submission to the PHY. In addition, it supports command repetition to enable back-to-back read transactions and read data comparison.

Table 10-2: **XIPHY RIU Addressing and Description**

RIU Address	Name	Description
0x00	NIBBLE_CTRL0	Nibble Control 0. Control for enabling DQS gate in the XIPHY, GT_STATUS for gate feedback, and clear gate which resets gate circuit.
0x01	NIBBLE_CTRL1	Nibble Control 1. TX_DATA_PHASE control for every bit in the nibble.
0x02	CALIB_CTRL	Calibration Control. XIPHY control and status for BISC.
0x03	Reserved	Reserved
0x04	Reserved	Reserved
0x05	BS_CTRL	Bit slice reset. Resets the ISERDES and IFIFOs in a given nibble.
0x06	Reserved	Reserved
0x07	PQTR	Rising edge delay for DQS.
0x08	NQTR	Falling edge delay for DQS.
0x09	Reserved	Reserved
0x0A	TRISTATE_ODELAY	Output delay for 3-state.
0x0B	ODELAY0	Output delay for bit slice 0.
0x0C	ODELAY1	Output delay for bit slice 1.
0x0D	ODELAY2	Output delay for bit slice 2.
0x0E	ODELAY3	Output delay for bit slice 3.
0x0F	ODELAY4	Output delay for bit slice 4.
0x10	ODELAY5	Output delay for bit slice 5.
0x11	ODELAY6	Output delay for bit slice 6.
0x12	IDELAY0	Input delay for bit slice 0.
0x13	IDELAY1	Input delay for bit slice 1.
0x14	IDELAY2	Input delay for bit slice 2.
0x15	IDELAY3	Input delay for bit slice 3.
0x16	IDELAY4	Input delay for bit slice 4.
0x17	IDELAY5	Input delay for bit slice 5.
0x18	IDELAY6	Input delay for bit slice 6.

Table 10-2: **XIPHY RIU Addressing and Description *(Cont'd)***

RIU Address	Name	Description
0x19	PQTR Align	BISC edge alignment computation for rising edge DQS.
0x1A	NQTR Align	BISC edge alignment computation for falling edge DQS.
0x1B to 0x2B	Reserved	Reserved
0x2C	WL_DLY_RNK0	Write Level register for Rank 0. Coarse and fine delay, WL_TRAIN.
0x2D	WL_DLY_RNK1	Write Level register for Rank 1. Coarse and fine delay.
0x2E	WL_DLY_RNK2	Write Level register for Rank 2. Coarse and fine delay.
0x2F	WL_DLY_RNK3	Write Level register for Rank 3. Coarse and fine delay.
0x30	RL_DLY_RNK0	DQS Gate register for Rank 0. Coarse and fine delay.
0x31	RL_DLY_RNK1	DQS Gate register for Rank 1. Coarse and fine delay.
0x32	RL_DLY_RNK2	DQS Gate register for Rank 2. Coarse and fine delay.
0x33	RL_DLY_RNK3	DQS Gate register for Rank 3. Coarse and fine delay.
0x34 to 0x3F	Reserved	Reserved

Memory Initialization and Calibration Sequence

After deassertion of the system reset, the PHY performs some required internal calibration steps first.

1. The built-in self-check of the PHY (BISC) is run. BISC is used in the PHY to compute internal skews for use in voltage and temperature tracking after calibration is completed.
2. After BISC is completed, calibration logic performs the required power-on initialization sequence for the memory.
3. This is followed by several stages of timing calibration for the write and read datapaths.
4. After calibration is completed, PHY calculates internal offsets to be used in voltage and temperature tracking.
5. PHY indicates calibration is finished and the controller begins issuing commands to the memory.

Figure 10-4 shows the overall flow of memory initialization and the different stages of calibration. The dark gray color is not available for this release.

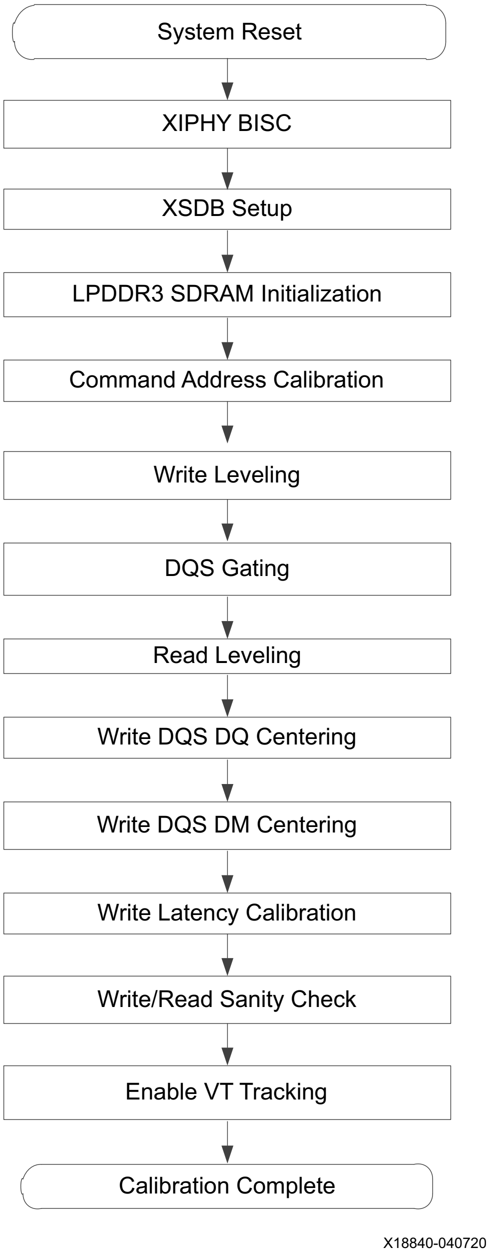


Figure 10-4: **PHY Overall Initialization and Calibration Sequence**

When simulating a design out of LPDDR3 SDRAM, the calibration it set to be bypassed to enable you to generate traffic to and from the DRAM as quickly as possible. When running in hardware or simulating with calibration, enabled signals are provided to indicate what step of calibration is running or, if an error occurs, where an error occurred.

The first step in determining calibration status is to check the CalDone port. After the CalDone port is checked, the status bits should be checked to indicate the steps that were ran and completed. Calibration halts on the very first error encountered, so the status bits indicate which step of calibration was last run. The status and error signals can be checked through either connecting the Vivado analyzer signals to these ports or through the XSDB tool (also through Vivado).

The calibration status is provided through the XSDB port, which stores useful information regarding calibration for display in the Vivado IDE. The calibration status and error signals are also provided as ports to allow for debug or triggering. Table 10-3 lists the pre-calibration status signal description.

Table 10-3: **Pre-Calibration XSDB Status Signal Description**

XSDB Status Register	XSDB Bits[8:0]	Description	Pre-Calibration Step
DDR_PRE_CAL_STATUS	0	Done	MicroBlaze has started up
	1	Done	Reserved
	2	Done	Reserved
	3	Done	Reserved
	4	Done	XSDB Setup Complete
	5	–	Reserved
	6	–	Reserved
	7	–	Reserved
	8	–	Reserved

Table 10-4 lists the status signals in the port as well as how they relate to the core XSDB data. In the status port, the mentioned bits are valid and the rest are reserved.

Table 10-4: **XSDB Status Signal Descriptions**

XSDB Status Register	XSDB Bits[8:0]	Status Port Bits[127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_RANKx_0	0	0	Start	Command Address Calibration	1
	1	1	Done	–	–
	2	2	Start	Write Leveling	2
	3	3	Done	–	–
	4	4	Start	DQS Gating	3
	5	5	Done	–	–
	6	6	Start	Read Leveling	4
	7	7	Done	–	–
	8	8	Start	Write DQS DQ Centering	5

Table 10-4: **XSDB Status Signal Descriptions *(Cont'd)***

XSDB Status Register	XSDB Bits[8:0]	Status Port Bits[127:0]	Description	Calibration Stage Name	Calibration Stage Number
DDR_CAL_STATUS_RANKx_1	0	9	Done	–	–
	1	10	Start	Write DQS DM Centering	6
	2	11	Done	–	–
	3	12	Start	Write Latency Calibration	7
	4	13	Done	–	–
	5	14	Start	Sanity Check	8
	6	15	Done	–	–

Table 10-5 lists the post-calibration XSDB status signal descriptions.

Table 10-5: **Post-Calibration XSDB Status Signal Description**

XSDB Status Register	XSDB Bits[8:0]	Description	Post-Calibration Step
DDR_POST_CAL_STATUS	0	Running	DQS Gate Tracking
	1	Idle	
	2	Fail	
	3	–	Reserved
	4	–	Reserved
	5	–	Reserved
	6	–	Reserved
	7	–	Reserved
	8	–	Reserved

Command Address Calibration

Command address bus in LPDDR3 is a Double Data Rate (DDR) type and hence, centering the CK in each CA bit is necessary. This calibration stage allows the controller to adjust each CA bit with respect to the CK forwarded to the LPDDR3 SDRAM device. The controller uses the CA training mode of the LPDDR3 device to achieve the centering.

During the CA training mode, the data sent on the address bus is returned on the DQ bus. Each CA bit is delayed until a 0 to 1 transition is detected to find the left margin. Once the left edge detection is complete, CK is moved to find the right edge. The ODELAY elements are used on both CA and CK during this alignment.

Write Leveling

LPDDR3 write leveling allows the controller to adjust each write DQS phase independently with respect to the CK forwarded to the LPDDR3 SDRAM device. This compensates for the skew between DQS and CK and meets the tDQSS specification.

During write leveling, DQS is driven by the FPGA memory interface and DQ is driven by the LPDDR3 SDRAM device to provide feedback. DQS is delayed until the 0 to 1 edge transition on DQ is detected. The DQS delay is achieved using both ODELAY and coarse tap delays.

After the edge transition is detected, the write leveling algorithm centers on the noise region around the transition to maximize margin. This second step is completed with only the use of ODELAY taps. Any reference to “FINE” is the ODELAY search.

DQS Gate

During this stage of calibration, the read DQS preamble is detected and the gate to enable data capture within the FPGA is calibrated to be one clock cycle before the first valid data on DQ. The coarse and fine DQS gate taps (RL_DLY_COARSE and RL_DLY_FINE) are adjusted during this stage. Read commands are issued with gaps in between to continually search for the DQS preamble position. During this stage of calibration, only the read DQS signals are monitored and not the read DQ signals. DQS Preamble Detection is performed sequentially on a per byte basis.

During this stage of calibration, the coarse taps are first adjusted while searching for the low preamble position and the first rising DQS edge, in other words, a DQS pattern of 00X1.

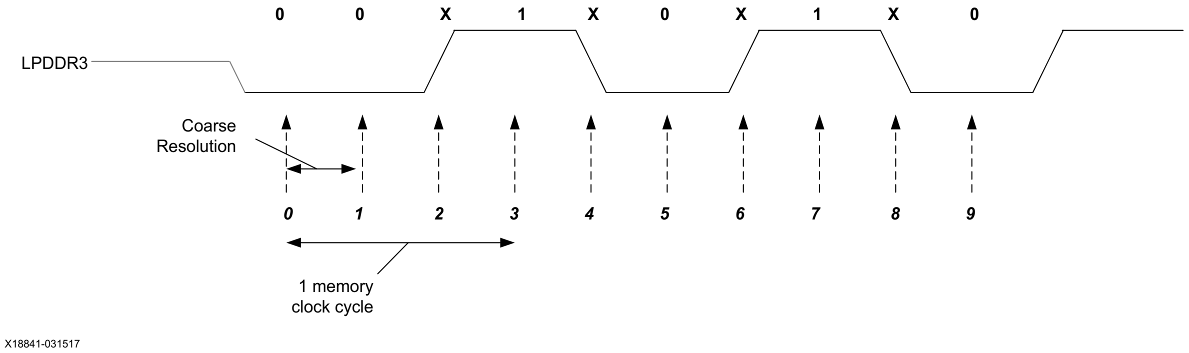


Figure 10-5: **LPDDR3 Preamble**

If the preamble is not found, the read latency is increased by one. The coarse taps are reset and then adjusted again while searching for the low preamble and first rising DQS edge. After the preamble position is properly detected, the fine taps are then adjusted to fine tune and edge align the position of the sample clock with the DQS.

Read Leveling

Read Leveling is performed over multiple stages to maximize the data eye and center the internal read sampling clock in the read DQ window for robust sampling. To perform this, Read Leveling performs the following sequential steps:

1. Maximizes the DQ eye by removing skew and OCV effects using per bit read DQ deskew.
2. Sweeps DQS across all DQ bits and finds the center of the data eye using the Multi-Purpose register data pattern. Centering of the data eye is completed for both the DQS and DQS#.
3. Post calibration, continuously maintains the relative delay of DQS versus DQ across the VT range.

Read Per-Bit Deskew

Per-bit deskew is performed on a per-bit basis whereas Read Leveling DQS centering is performed on a per-nibble basis.

During per-bit deskew, Read Leveling Calibration, a pattern of `00000000_11111111` is written and read back while DQS adjustments (PQTR and NQTR individual fine taps on DQS) and DQ adjustments (IDELAY) are made.

At the end of this stage, the DQ bits are internally deskewed to the left edge of the incoming DQS.

Write DQS DQ Centering

This stage of calibration is required to center align the write DQS in the write DQ window per bit. At the start of Write DQS Centering and Per-Bit Deskew, DQS is aligned to CK but no adjustments on the write window have been made. Write window adjustments are made in the following two sequential stages:

- Write Per-Bit Deskew
- Write DQS Centering

Write DQS-to-DQ Per-Bit Deskew

During write per-bit deskew, a toggling `10101010` pattern is continuously written and read back while making 90^o clock phase adjustments on the write DQ along with individual fine ODELAY adjustments on DQS and DQ. At the end of per-bit write DQ deskew, the write DQ bits are aligned as they are transmitted to the memory.

Write DQS-to-DQ Centering

During Write DQS Centering, the same toggling `10101010` pattern is continuously written and read back. ODELAY adjustments on DQS and DQ are also made but all of the DQ ODELAY adjustments for a given byte are made in step to maintain the previously deskewed alignment.

Write DQS DM Calibration

In all previous stages of calibration, data mask signals are driven low before and after the required amount of time to ensure they have no impact on calibration. Now, both the read and the writes have been calibrated and data mask can reliably be adjusted. If DM signals are not used within the interface, this stage of calibration is skipped.

During DM Calibration, a data pattern of `55555555_55555555` is first written to address `0x000` followed by a write to the same address but with a data pattern of `BBBBBBBB_BBBBBBBB` with DM asserted during the rising edge of DQS. A read is then issued where the expected read back pattern is all "B" except for the data where DM was asserted. In these masked locations, a 5 is expected. The same series of steps completed during Write Per-Bit Deskew and Write DQS Centering is then completed but for the DM bits.

Write Latency Calibration

Write Latency Calibration is required to align DQS to the correct CK edge. During write leveling, DQS is aligned to the nearest rising edge of CK. However, this might not be the edge that captures the write command.

Depending on the interface type, the DQS could either be one CK cycle earlier than, two CK cycles earlier than, or aligned to the CK edge that captures the write command.

This is a pattern based calibration where coarse adjustments are made on a per byte basis until the expected on time write pattern is read back. The process is as follows:

1. Issue extended writes followed by a single read.
2. Check the pattern readback against the expected patterns.
3. If necessary add coarse adjustments.
4. Repeat until the on time write pattern is read back, signifying DQS is aligned to the correct CK cycle, or an incorrect pattern is received resulting in a Write Latency failure.

The following data is written at address `0x000`:

- Data pattern before (with extra DQS pulses): `0000000000000000`
- Data pattern written to address `0x000`: `FF00AA5555AA9966`

- Data pattern after (with extra DQS pulses): `FFFFFFFFFFFFFFFF`

Reads are then performed where the following patterns can be calibrated:

- On time write pattern read back: `FF00AA5555AA9966` (no adjustments needed)
- One DQS early write pattern read back: `AA5555AA9966FFFF`
- Two DQS early write pattern read back: `55AA9966FFFFFFFF`
- Three DQS early write pattern read back: `9966FFFFFFFFFFFF`

Write Latency Calibration can fail for the following cases and signify a board violation between DQS and CK trace matching:

- Four DQS early pattern `FFFFFFFFFFFFFFFF`
- One DQS late write pattern read back: `0000FF00AA5555AA`
- Two DQS late write pattern read back: `00000000FF00AA55`
- Three DQS late write pattern read back: `000000000000FF00`

Write/Read Sanity Check

At the end of all calibration stages, a check of the data is made to ensure the previous stage of calibration did not inadvertently leave the write or read path in a bad spot. A single write burst followed by a single read command to the same location is sent to the DRAM, and the data is checked against the expected data across all bytes before continuing. During this step, the expected data pattern as seen on a nibble is `937EC924`.

Enable VT Tracking

After all stages of calibration, a signal is sent to the XIPHY to recalibrate internal delays to start voltage and temperature tracking. The XIPHY asserts a signal when complete, `phy2clb_phy_rdy_upp` for upper nibbles and `phy2clb_phy_rdy_low` for lower nibbles.

Reset Sequence

The **`sys_rst`** signal resets the entire memory design which includes general interconnect (fabric) logic which is driven by the MMCM clock (clkout0) and RIU logic. MicroBlaze™ and calibration logic are driven by the MMCM clock (clkout6). The **`sys_rst`** input signal is synchronized internally to create the **`ui_clk_sync_rst`** signal. The **`ui_clk_sync_rst`** reset signal is synchronously asserted and synchronously deasserted.

Figure 10-6 shows the **`ui_clk_sync_rst`** (fabric reset) is synchronously asserted with a few clock delays after **`sys_rst`** is asserted. When **`ui_clk_sync_rst`** is asserted, there are a few clocks before the clocks are shut off.

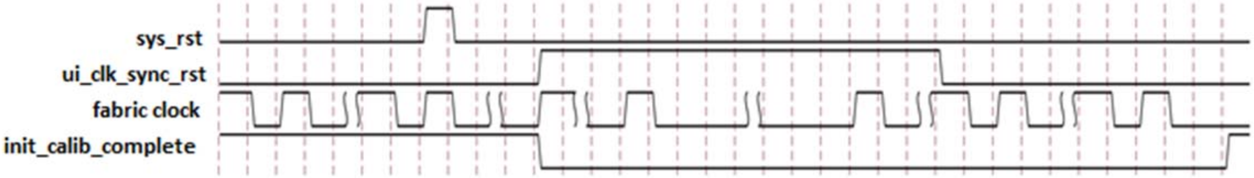


Figure 10-6: **Reset Sequence Waveform**

The following are the reset sequencing steps:

1. Reset to design is initiated after **`ui_clk_sync_rst`** goes High.
2. **`init_calib_complete`** signal goes Low when **`ui_clk_sync_rst`** is High.
3. Reset to design is deactivated after **`ui_clk_sync_rst`** is Low.
4. After **`ui_clk_sync_rst`** is deactivated, the **`init_calib_complete`** is asserted after calibration is completed.

Chapter 11

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The memory interface requires one mixed-mode clock manager (MMCM), one TXPLL per I/O bank used by the memory interface, and two BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used.

Note: LPDDR3 SDRAM generates the appropriate clocking structure and no modifications to the RTL are supported.

The LPDDR3 SDRAM tool generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to MMCM (located in center bank of memory interface)
- MMCM to BUFG (located at center bank of memory interface) driving FPGA logic and all TXPLLs
- MMCM to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate FPGA logic
- Clocking pair of the interface must be in the same SLR of memory interface for the stacked silicon interconnect (SSI) technology devices

Requirements

GCIO

- Must use a differential I/O standard
- Must be in the same I/O column as the memory interface
- Must be in the same SLR of memory interface for the SSI technology devices
- The I/O standard and termination scheme are system dependent. For more information, consult the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 7].

MMCM

- MMCM is used to generate the FPGA logic system clock (1/4 of the memory clock)
- Must be located in the center bank of memory interface
- Must use internal feedback
- Input clock frequency divided by input divider must be ≥ 70 MHz (CLKINx / D ≥ 70 MHz)
- Must use integer multiply and output divide values

Input Clock Requirement

- The clock generator driving the GCIO should have jitter < 3 ps RMS.
- The input clock should always be clean and stable. The IP functionality is not guaranteed if this input system clock has a glitch, discontinuous, etc.
- No spread spectrum clock is allowed.

BUFGs and Clock Roots

- One BUFG is used to generate the system clock to FPGA logic and another BUFG is used to divide the system clock by two.
- BUFGs and clock roots must be located in center most bank of the memory interface.
    - For two bank systems, the bank with the higher number of bytes selected is chosen as the center bank. If the same number of bytes is selected in two banks, then the top bank is chosen as the center bank.
    - Both the BUFGs must be in the same bank.

TXPLL

- CLKOUTPHY from TXPLL drives XIPHY within its bank
- TXPLL must be set to use a CLKFBOUT phase shift of 90°
- TXPLL must be held in reset until the MMCM lock output goes High
- Must use internal feedback

Figure 11-1 shows an example of the clocking structure for a three bank memory interface. The GCIO drives the MMCM located at the center bank of the memory interface. MMCM drives both the BUFGs located in the same bank. The BUFG (which is used to generate system clock to FPGA logic) output drives the TXPLLs used in each bank of the interface.

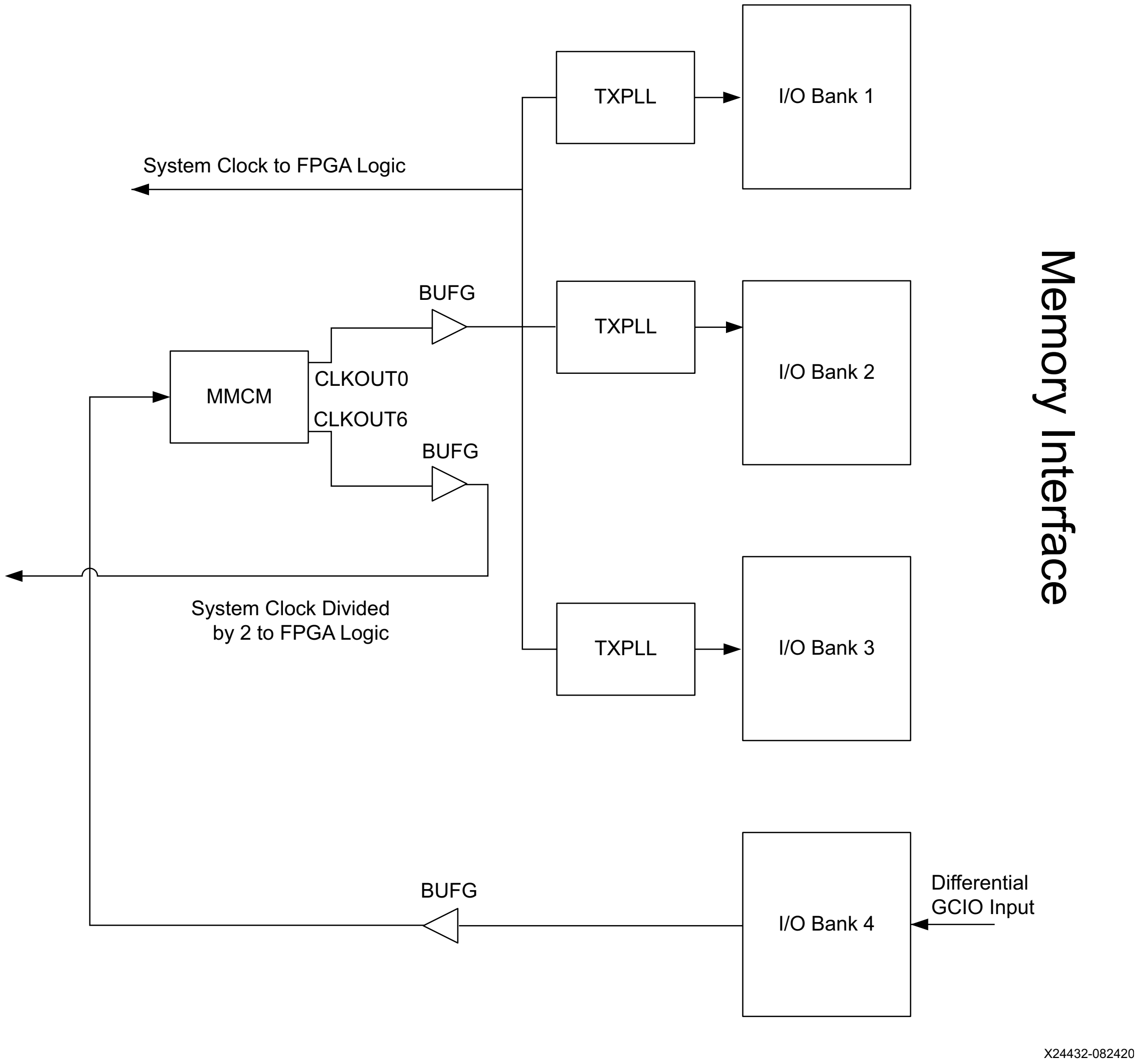


Figure 11-1: **Clocking Structure for Three Bank Memory Interface**

The MMCM is placed in the center bank of the memory interface.

- For two bank systems, MMCM is placed in a bank with the most number of bytes selected. If they both have the same number of bytes selected in two banks, then MMCM is placed in the top bank.
- For four bank systems, MMCM is placed in a second bank from the top.

For designs generated with System Clock configuration of **No Buffer**, MMCM must not be driven by another MMCM/PLL. Cascading clocking structures MMCM → BUFG → MMCM and PLL → BUFG → MMCM are not allowed.

If the MMCM is driven by the GCIO pin of the other bank, then the CLOCK_DEDICATED_ROUTE constraint with value "BACKBONE" must be set on the net that is driving MMCM or on the MMCM input. Setting up the CLOCK_DEDICATED_ROUTE constraint on the net is preferred. But when the same net is driving two MMCMs, the CLOCK_DEDICATED_ROUTE constraint must be managed by considering which MMCM needs the BACKBONE route.

In such cases, the CLOCK_DEDICATED_ROUTE constraint can be set on the MMCM input. To use the "BACKBONE" route, any clock buffer that exists in the same CMT tile as the GCIO must exist between the GCIO and MMCM input. The clock buffers that exists in the I/O CMT are BUFG, BUFGCE, BUFGCTRL, and BUFGCE_DIV. So LPDDR3 SDRAM instantiates BUFG between the GCIO and MMCM when the GCIO pins and MMCM are not in the same bank (see Figure 11-1).

If the GCIO pin and MMCM are allocated in different banks, LPDDR3 SDRAM generates CLOCK_DEDICATED_ROUTE constraints with value as "BACKBONE." If the GCIO pin and MMCM are allocated in the same bank, there is no need to set any constraints on the MMCM input.

Similarly when designs are generated with System Clock Configuration as a **No Buffer** option, you must take care of the "BACKBONE" constraint and the BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM if GCIO pin and MMCM are allocated in different banks. LPDDR3 SDRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations. For more information on clocking, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

XDC syntax for CLOCK_DEDICATED_ROUTE constraint is given here:

```
For LPDDR3:
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hier -filter {NAME =~ */
u_ddr_infrastructure/gen_mmcme*.u_mmcme_adv_inst/CLKIN1}]
```

For more information on the CLOCK_DEDICATED_ROUTE constraints, see the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 9].

Note: If two different GCIO pins are used for two LPDDR3 SDRAM IP cores in the same bank, center bank of the memory interface is different for each IP. LPDDR3 SDRAM generates MMCM LOC and CLOCK_DEDICATED_ROUTE constraints accordingly.

Sharing of Input Clock Source (sys_clk_p)

If the same GCIO pin must be used for two IP cores, generate the two IP cores with the same frequency value selected for option **Reference Input Clock Period (ps)** and **System Clock Configuration** option as **No Buffer**. Perform the following changes in the wrapper file in which both IPs are instantiated:

1. LPDDR3 SDRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) of both the IP cores.
2. System clock pins must be allocated within the same I/O column of the memory interface pins allocated. Add the pin LOC constraints for system clock pins and clock constraints in your top-level XDC.
3. You must add a "BACKBONE" constraint on the net that is driving the MMCM or on the MMCM input if GCIO pin and MMCM are not allocated in the same bank. Apart from this, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM to use the "BACKBONE" route.

Note:

- The UltraScale architecture includes an independent XIPHY power supply and TXPLL for each XIPHY. This results in clean, low jitter clocks for the memory system.
- Skew spanning across multiple BUFGs is not a concern because single point of contact exists between BUFG → TXPLL and the same BUFG → System Clock Logic.
- System input clock cannot span I/O columns because the longer the clock lines span, the more jitter is picked up.

TXPLL Usage

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used. One PLL per bank is used if a bank is used by a single memory interface. You can use a second PLL for other usage. To use a second PLL, you can perform the following steps:

1. Generate the design for the **System Clock Configuration** option as **No Buffer**.
2. LPDDR3 SDRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) and also to the input of PLL (PLL instance that you have in your design).
3. You can use the PLL output clocks.

Additional Clocks

You can produce up to four additional clocks which are created from the same MMCM that generates `ui_clk`. Additional clocks can be selected from the **Clock Options** section in the **Advanced** tab. The GUI lists the possible clock frequencies from MMCM and the frequencies for additional clocks vary based on selected memory frequency (**Memory Device Interface Speed (ps)** value in the **Basic** tab), selected FPGA, and FPGA speed grade.

Reduce System Noise during Calibration

The system design should be as quiet as possible during the calibration process. In particular, the Soft Error Mitigation (SEM) IP, if used, should be disabled during calibration. For calibration that occurs immediately after the configuration or reconfiguration of the FPGA, use the ICAP arbitration interface to hold off the SEM IP in the boot stage. For more information on the ICAP Arbitration Interface, see "ICAP Arbitration Interface" section in Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10].

For situations where the memory interface is reset and recalibrated without a reconfiguration of the FPGA, the SEM IP must be set into IDLE state to disable the memory scan and to send the SEM IP back into the scanning (Observation or Detect only) states afterwards. This can be done in two methods, through the "Command Interface" or "UART interface." See Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10] for more information.

Resets

An asynchronous reset (`sys_rst`) input is provided. This is an active-High reset and the `sys_rst` must assert for a minimum pulse width of 5 ns. The `sys_rst` can be an internal or external pin.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must have a common reset input.*

For more information on reset, see the Reset Sequence in Chapter 10, Core Architecture.

Note: The best possible calibration results are achieved when the FPGA activity is minimized from the release of this reset input until the memory interface is fully calibrated as indicated by the `init_calib_complete` port (see the User Interface section of this document).

PCB Guidelines for LPDDR3

Strict adherence to all documented LPDDR3 PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

Pin and Bank Rules

LPDDR3 Pin Rules

Here is some description on the terminology used in this section:

- Address/control means **`ck_t/ck_c`**, **`cs_n`**, **`ca[9:0]`**, **`cke`**, and **`odt`**.
- Pins in a byte lane are numbered N0 to N12.
- Byte lanes in a bank are designed by T0, T1, T2, or T3. Nibbles within a byte lane are distinguished by a "U" or "L" designator added to the byte lane designator (T0, T1, T2, or T3). Thus they are T0L, T0U, T1L, T1U, T2L, T2U, T3L, and T3U.

Note: There are two PLLs per bank and a controller uses one PLL in every bank that is being used by the interface.

1. **`dqs`**, **`dq`**, and **`dm`** location.
    a. **`dqs`** must be located on a dedicated **`dqs`** pair in the upper nibble designated with "U." **`dq`** associated with a **`dqs`** must be in same byte lane on any of the other pins except pins 1 and 12.
    b. The **`dm`** associated with a **`dqs`** must be located on pin N0 in the byte lane.
2. Byte lanes are configured as either data or address/control. No data signals (**`dqs`**, **`dq`**, **`dm`**) can be in a byte lane that is configured for address/control. Only pins 1 and 12 can be used for **`cke`** and **`odt`** pins in a data byte lane.
3. Address/control can be on any of the 13 pins in the address/control byte lanes. Address/control must be contained within the same bank.
    a. Address/control: **`ck_t/ck_c`**, **`ca[9:0]`**, and **`cs_n`** must be placed in the same byte.
    b. **`ck_t/ck_c`** must be placed in 0/1 or 6/7 pins of the byte.
    c. **`cke`** and **`odt`** can be allocated in data or address/control byte lanes.
    d. All address/control signals must be placed in the same bank.
4. There is one **`vrp`** pin per bank and DCI cascade is required for the interface when placed in a HP bank. DCI cascade option is supported.

   a. If DCI cascade option is disabled, **`vrp`** pin per bank is needed for DCI termination for memory pins allocated banks. So one **`vrp`** pin per bank is reserved in memory pins allocated banks during pin allocation.
   b. If the bank contains any memory port(s), **`vrp`** must be reserved and must not be allocated to any Memory Port or any other general I/O.
   c. If the bank contains system clock signals (**`sys_clk_p`** and **`sys_clk_n`**) and status output pins (**`init_calib_complete`**, **`data_compare_error`** and **`sys_rst_n`**) only, **`vrp`** can be used as normal I/O.
   d. If DCI cascade option is enabled, **`vrp`** pin can be used for any memory port or any other general I/O. DCI cascade rules are the same as the I/O DCI rules and there are no specific DCI cascade rules for memory specific.
   e. DCI cascade is valid for HP banks only.

RECOMMENDED: *Xilinx strongly recommends that the DCIUpdateMode option is kept with the default value of ASREQUIRED so that the DCI circuitry is allowed to operate normally.*

5. All I/O banks used by the memory interface must be in the same column and must be in the same SLR.
6. Maximum height of interface is two contiguous banks.
7. Bank skipping is not allowed.
8. The input clock must be connected to GCIO. The highest performance is achieved when the input clock for the MMCM in the interface comes from the clock capable pair in the I/O column used for the memory interface.
9. System clock pins (**`sys_clk_p`** and **`sys_clk_n`**) restricted to the same column of memory I/Os allocated banks. Also, they must be in the same SLR of the memory interface for the SSI technology devices.
10. System clock pins can also be allocated in the memory banks.
11. System clock pins must be allocated within the same SLR of the memory pins allocated SLR.
12. System control/status signals (**`init_calib_complete`**, **`data_compare_error`**, and **`sys_rst_n`**) can be allocated in any bank in the device which also includes memory banks. These signals can also be allocated across SLR.
13. There are dedicated V_{REF} pins (not included in the rules above). Either internal or external V_{REF} is permitted. If an external V_{REF} is not used, the V_{REF} pins must be pulled to ground by a resistor value specified in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7]. These pins must be connected appropriately for the standard in use. When using external V_{REF} for a LPDDR3 interface, provide the FPGA V_{REF} pins a 0.75V reference.

☆ **IMPORTANT:** *The system reset pin (`sys_rst_n`) must not be allocated to pin N0 or N6 if the byte is used in a memory interface. Consult the UltraScale Architecture Select IO Resources User Guide (UG571) [Ref 7] for more information.*

Pinout Swapping

- Pins can swap freely within each byte group (data and address/control), except for the DQS pair which must be on the dedicated DQS pair in the nibble (for more information, see the **`dqs`**, **`dq`**, and **`dm`** location in LPDDR3 Pin Rules).
- Byte groups (data and address/control) can swap easily with each other.
- Pins in the address/control byte groups can swap freely within and between their byte groups.
- No other pin swapping is permitted.

Pinout Examples

☆ **IMPORTANT:** *Due to the calibration stage, there is no need for `set_input_delay/` `set_output_delay` on the LPDDR3 SDRAM. Ignore the unconstrained inputs and outputs for LPDDR3 SDRAM and the signals which are calibrated.*

Table 11-1 shows an example of a 32-bit LPDDR3 interface contained in two banks. This example is for a component interface using x32 LPDDR3 components.

Table 11-1: **32-Bit LPDDR3 Interface Contained in Two Banks**

Bank	Signal Name	Byte Group	I/O Type
Bank 1			
1	–	T3U_12	–
1	–	T3U_11	N
1	–	T3U_10	P
1	–	T3U_9	N
1	–	T3U_8	P
1	–	T3U_7	N
1	–	T3U_6	P
1	–	T3L_5	N
1	–	T3L_4	P
1	–	T3L_3	N
1	–	T3L_2	P
1	–	T3L_1	N

Table 11-1: **32-Bit LPDDR3 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	–	T3L_0	P
1	–	T2U_12	–
1	–	T2U_11	N
1	–	T2U_10	P
1	–	T2U_9	N
1	–	T2U_8	P
1	–	T2U_7	N
1	–	T2U_6	P
1	–	T2L_5	N
1	–	T2L_4	P
1	–	T2L_3	N
1	–	T2L_2	P
1	–	T2L_1	N
1	–	T2L_0	P
1	–	T1U_12	–
1	dq31	T1U_11	N
1	dq30	T1U_10	P
1	dq29	T1U_9	N
1	dq28	T1U_8	P
1	dqs3_c	T1U_7	N
1	dqs3_t	T1U_6	P
1	dq27	T1L_5	N
1	dq26	T1L_4	P
1	dq25	T1L_3	N
1	dq24	T1L_2	P
1	–	T1L_1	N
1	dm3	T1L_0	P
1	vrp	T0U_12	–
1	dq23	T0U_11	N
1	dq22	T0U_10	P

Table 11-1: **32-Bit LPDDR3 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	dq21	T0U_9	N
1	dq20	T0U_8	P
1	dqs2_c	T0U_7	N
1	dqs2_t	T0U_6	P
1	dq19	T0L_5	N
1	dq18	T0L_4	P
1	dq17	T0L_3	N
1	dq16	T0L_2	P
1	–	T0L_1	N
1	dm2	T0L_0	P
Bank 2			
2	ca0	T3U_12	–
2	ca1	T3U_11	N
2	ca2	T3U_10	P
2	ca3	T3U_9	N
2	ca4	T3U_8	P
2	ca5	T3U_7	N
2	ca6	T3U_6	P
2	ca7	T3L_5	N
2	ca8	T3L_4	P
2	ca9	T3L_3	N
2	cs_n	T3L_2	P
2	ck_c	T3L_1	N
2	ck_t	T3L_0	P
2	–	T2U_12	–
2	–	T2U_11	N
2	–	T2U_10	P
2	–	T2U_9	N
2	–	T2U_8	P
2	–	T2U_7	N
2	–	T2U_6	P
2	–	T2L_5	N

Table 11-1: **32-Bit LPDDR3 Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
2	–	T2L_4	P
2	–	T2L_3	N
2	–	T2L_2	P
2	sys_clk_n	T2L_1	N
2	sys_clk_p	T2L_0	P
2	–	T1U_12	–
2	dq15	T1U_11	N
2	dq14	T1U_10	P
2	dq13	T1U_9	N
2	dq12	T1U_8	P
2	dqs1_c	T1U_7	N
2	dqs1_t	T1U_6	P
2	dq11	T1L_5	N
2	dq10	T1L_4	P
2	dq9	T1L_3	N
2	dq8	T1L_2	P
2	odt	T1L_1	N
2	dm1	T1L_0	P
2	vrp	T0U_12	–
2	dq7	T0U_11	N
2	dq6	T0U_10	P
2	dq5	T0U_9	N
2	dq4	T0U_8	P
2	dqs0_c	T0U_7	N
2	dqs0_t	T0U_6	P
2	dq3	T0L_5	N
2	dq2	T0L_4	P
2	dq1	T0L_3	N
2	dq0	T0L_2	P
2	cke	T0L_1	N
2	dm0	T0L_0	P

Protocol Description

This core has a user interface.

User Interface

The user interface signals are described in Table 11-2 and connects to an FPGA user design to allow access to an external memory device. The user interface is layered on top of the native interface which is described earlier in the controller description.

Table 11-2: **User Interface**

Signal	I/O	Description
app_addr[APP_ADDR_WIDTH – 1:0]	I	This input indicates the address for the current request.
app_cmd[2:0]	I	This input selects the command for the current request.
app_en	I	This is the active-High strobe for the app_addr[], app_cmd[2:0], and app_hi_pri inputs.
app_rdy	O	This output indicates that the user interface is ready to accept commands. If the signal is deasserted when app_en is enabled, the current app_cmd, app_autoprecharge, and app_addr must be retried until app_rdy is asserted.
app_hi_pri	I	This input is reserved and should be tied to 0.
app_rd_data [APP_DATA_WIDTH – 1:0]	O	This provides the output data from read commands.
app_rd_data_end	O	This active-High output indicates that the current clock cycle is the last cycle of output data on app_rd_data[].
app_rd_data_valid	O	This active-High output indicates that app_rd_data[] is valid.
app_wdf_data [APP_DATA_WIDTH – 1:0]	I	This provides the data for write commands.
app_wdf_end	I	This active-High input indicates that the current clock cycle is the last cycle of input data on app_wdf_data[].
app_wdf_mask [APP_MASK_WIDTH – 1:0]	I	This provides the mask for app_wdf_data[].
app_wdf_rdy	O	This output indicates that the write data FIFO is ready to receive data. Write data is accepted when app_wdf_rdy = 1'b1 and app_wdf_wren = 1'b1.
app_wdf_wren	I	This is the active-High strobe for app_wdf_data[].
ui_clk	O	This user interface clock must be one quarter of the DRAM clock.
init_calib_complete	O	PHY asserts init_calib_complete when calibration is finished.
ui_clk_sync_rst	O	This is the active-High user interface reset.
addn_ui_clkout1	O	Additional clock outputs provided based on user requirement.

Table 11-2: **User Interface *(Cont'd)***

Signal	I/O	Description
addn_ui_clkout2	O	Additional clock outputs provided based on user requirement.
addn_ui_clkout3	O	Additional clock outputs provided based on user requirement.
addn_ui_clkout4	O	Additional clock outputs provided based on user requirement.
dbg_clk	O	Debug Clock. Do not connect any signals to dbg_clk and keep the port open during instantiation.
sl_iport0	I [36:0]	Input Port 0 (* KEEP = "true" *)
sl_oport0	O [16:0]	Output Port 0 (* KEEP = "true" *)

app_addr[APP_ADDR_WIDTH – 1:0]

This input indicates the address for the request currently being submitted to the user interface. The user interface aggregates all the address fields of the external SDRAM and presents a flat address space.

The MEM_ADDR_ORDER parameter determines how `app_addr` is mapped to the SDRAM address bus and chip select pins. This mapping can have a significant impact on memory bandwidth utilization. "ROW_BANK_COLUMN" is the recommended MEM_ADDR_ORDER setting.

The address mapping in ROW_BANK_COLUMN ordering has been depicted in Table 11-3 and Figure 11-2.

Table 11-3: **LPDDR3 ROW_BANK_COLUMN**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0:
Row	app_addr[COL_WIDTH + BANK_WIDTH +: ROW_WIDTH]
Column	app_addr[0 +: COL_WIDTH]
Bank	app_addr[COL_WIDTH +: BANK_WIDTH]

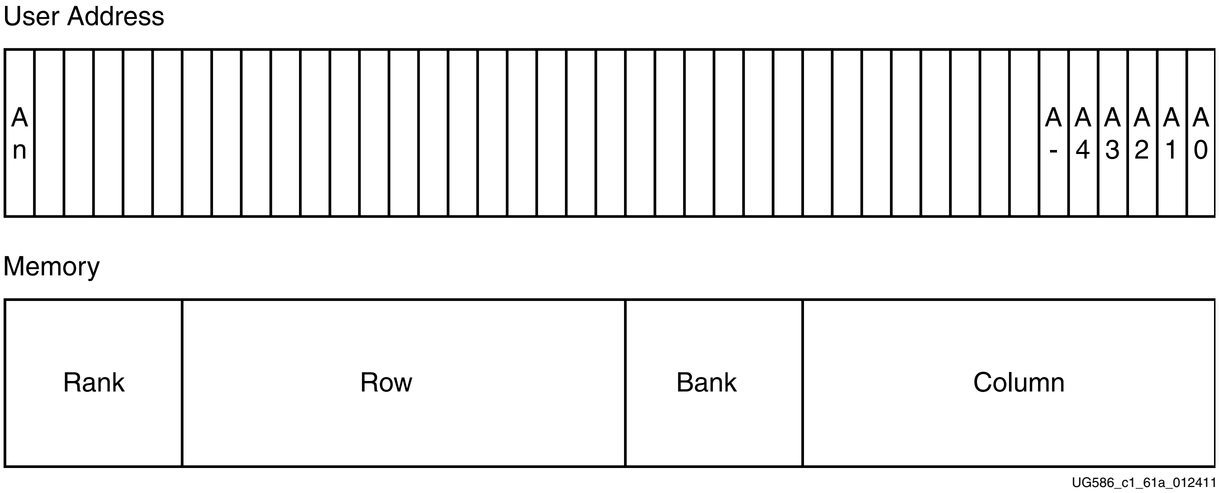


Figure 11-2: **Address Ordering for ROW_BANK_COLUMN**

The address mapping in BANK_ROW_COLUMN ordering has been depicted in Table 11-4 and Figure 11-3.

Table 11-4: **LPDDR3 BANK_ROW_COLUMN**

SDRAM	app_addr Mapping
Rank	(RANK == 1) ? 1'b0:
Row	app_addr[COL_WIDTH +: ROW_WIDTH]
Column	app_addr[0 +: COL_WIDTH]
Bank	app_addr[COL_WIDTH + ROW_WIDTH +: BANK_WIDTH]

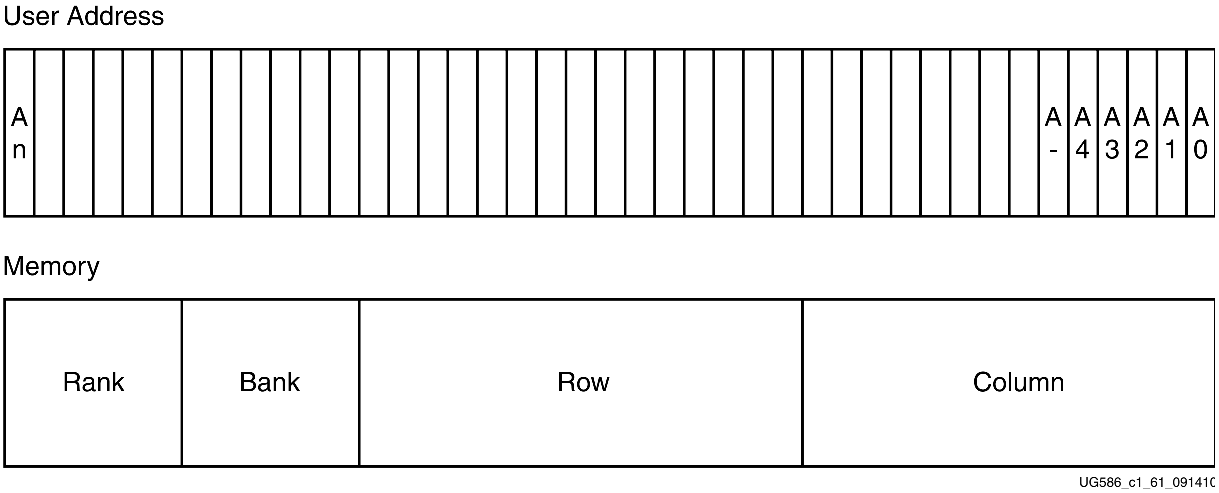


Figure 11-3: **Address Ordering for BANK_ROW_COLUMN**

app_cmd[2:0]

This input specifies the command for the request currently being submitted to the user interface. The available commands are shown in Table 11-5.

Table 11-5: **Commands for app_cmd[2:0]**

Operation	app_cmd[2:0] Code
Write	000
Read	001

app_en

This input strobes in a request. Apply the desired values to **`app_addr`**[], **`app_cmd`**[2:0], and **`app_hi_pri`**, and then assert **`app_en`** to submit the request to the user interface. This initiates a handshake that the user interface acknowledges by asserting **`app_rdy`**.

app_hi_pri

This input indicates that the current request is a high priority.

app_wdf_data[APP_DATA_WIDTH – 1:0]

This bus provides the data currently being written to the external memory.

app_wdf_end

This input indicates that the data on the **`app_wdf_data[]`** bus in the current cycle is the last data for the current request.

app_wdf_mask[APP_MASK_WIDTH – 1:0]

This bus indicates which bits of **`app_wdf_data`**[] are written to the external memory and which bits remain in their current state. The bytes are masked by setting a value of 1 to the corresponding bits in **`app_wdf_mask`**. For example, if the application data width is 256, the mask width takes a value of 32. The least significant byte [7:0] of **`app_wdf_data`** is masked using Bit[0] of **`app_wdf_mask`** and the most significant byte [255:248] of **`app_wdf_data`** is masked using Bit[31] of **`app_wdf_mask`**. Hence if you have to mask the last DWORD, that is, bytes 0, 1, 2, and 3 of **`app_wdf_data`**, the **`app_wdf_mask`** should be set to **`32'h0000_000F`**.

app_wdf_wren

This input indicates that the data on the **`app_wdf_data`**[] bus is valid.

app_rdy

This output indicates whether the request currently being submitted to the user interface is accepted. If the user interface does not assert this signal after **`app_en`** is asserted, the current request must be retried. The **`app_rdy`** output is not asserted if:

- PHY/Memory initialization is not yet completed.
- All the bank machines are occupied (can be viewed as the command buffer being full).
    - A read is requested and the read buffer is full.
    - A write is requested and no write buffer pointers are available.
- A periodic read is being inserted.

app_rd_data[APP_DATA_WIDTH – 1:0]

This output contains the data read from the external memory.

app_rd_data_end

This output indicates that the data on the **`app_rd_data`**[] bus in the current cycle is the last data for the current request.

app_rd_data_valid

This output indicates that the data on the **`app_rd_data`**[] bus is valid.

app_wdf_rdy

This output indicates that the write data FIFO is ready to receive data. Write data is accepted when both **`app_wdf_rdy`** and **`app_wdf_wren`** are asserted.

ui_clk_sync_rst

This is the reset from the user interface which is in synchronous with **`ui_clk`**.

ui_clk

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

init_calib_complete

PHY asserts **`init_calib_complete`** when calibration is finished. The application has no need to wait for **`init_calib_complete`** before sending commands to the Memory Controller.

Command Path

When the user logic **`app_en`** signal is asserted and the **`app_rdy`** signal is asserted from the user interface, a command is accepted and written to the FIFO by the user interface. The command is ignored by the user interface whenever **`app_rdy`** is deasserted. The user logic needs to hold **`app_en`** High along with the valid command, autoprecharge, and address values until **`app_rdy`** is asserted as shown for the "write with autoprecharge" transaction in Figure 11-4.

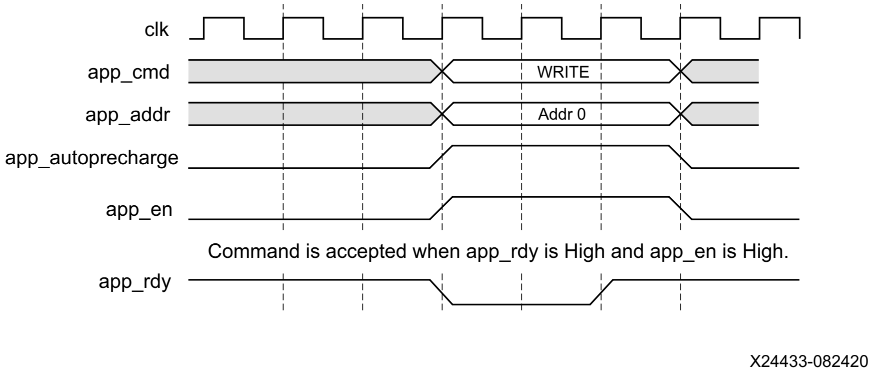


Figure 11-4: **User Interface Command Timing Diagram with app_rdy Asserted**

A non back-to-back write command can be issued as shown in Figure 11-5. This figure depicts three scenarios for the **`app_wdf_data`**, **`app_wdf_wren`**, and **`app_wdf_end`** signals as follows:

1. Write data is presented along with the corresponding write command.
2. Write data is presented before the corresponding write command.
3. Write data is presented after the corresponding write command, but should not exceed the limitation of two clock cycles.

For write data that is output after the write command has been registered, as shown in Note 3 (Figure 11-5), the maximum delay is two clock cycles.

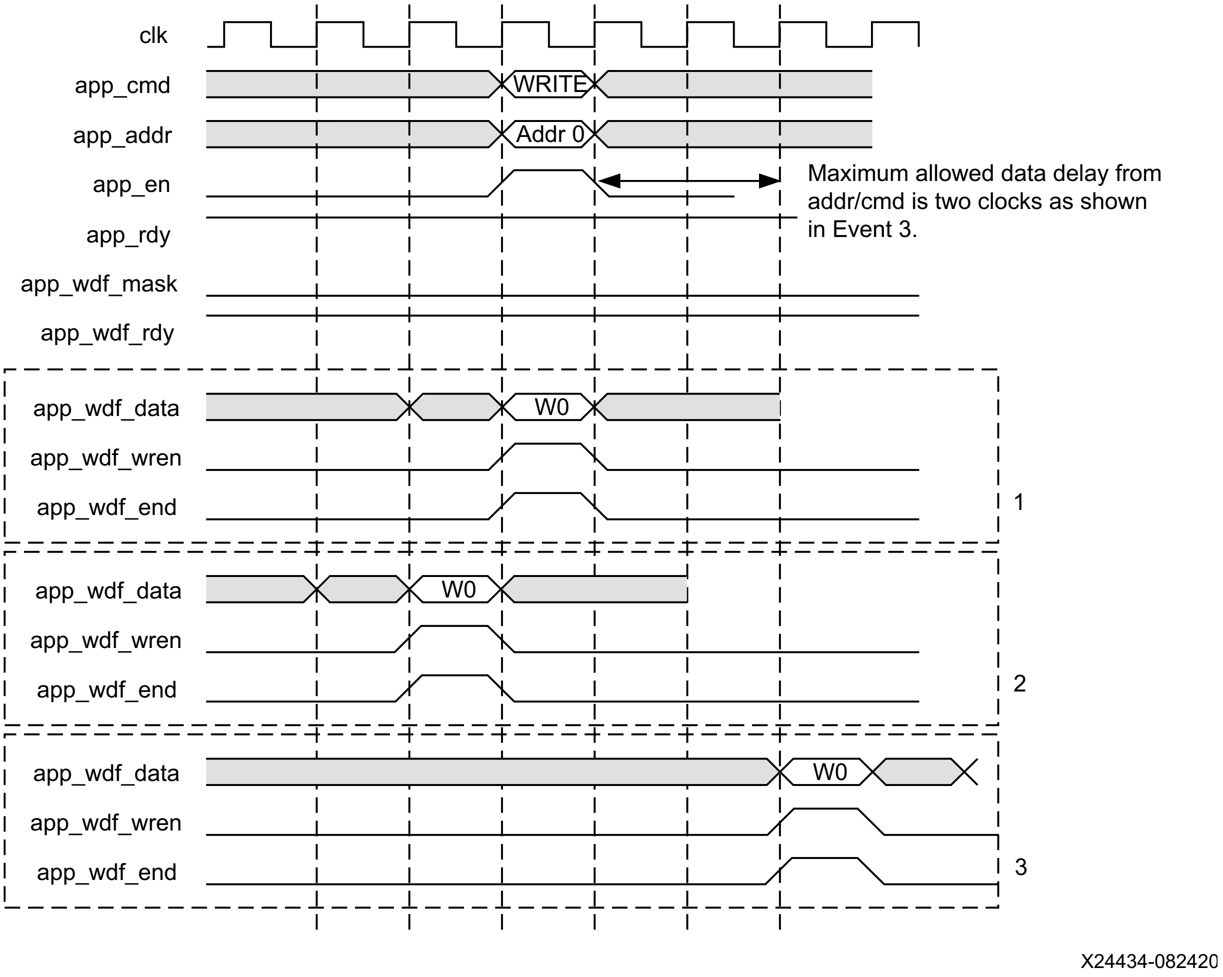


Figure 11-5: **4:1 Mode User Interface Write Timing Diagram (Memory Burst Type = BL8)**

Write Path

The write data is registered in the write FIFO when **`app_wdf_wren`** is asserted and **`app_wdf_rdy`** is High (Figure 11-6). If **`app_wdf_rdy`** is deasserted, the user logic needs to hold **`app_wdf_wren`** and **`app_wdf_end`** High along with the valid **`app_wdf_data`** value until **`app_wdf_rdy`** is asserted. The **`app_wdf_mask`** signal can be used to mask out the bytes to write to external memory.

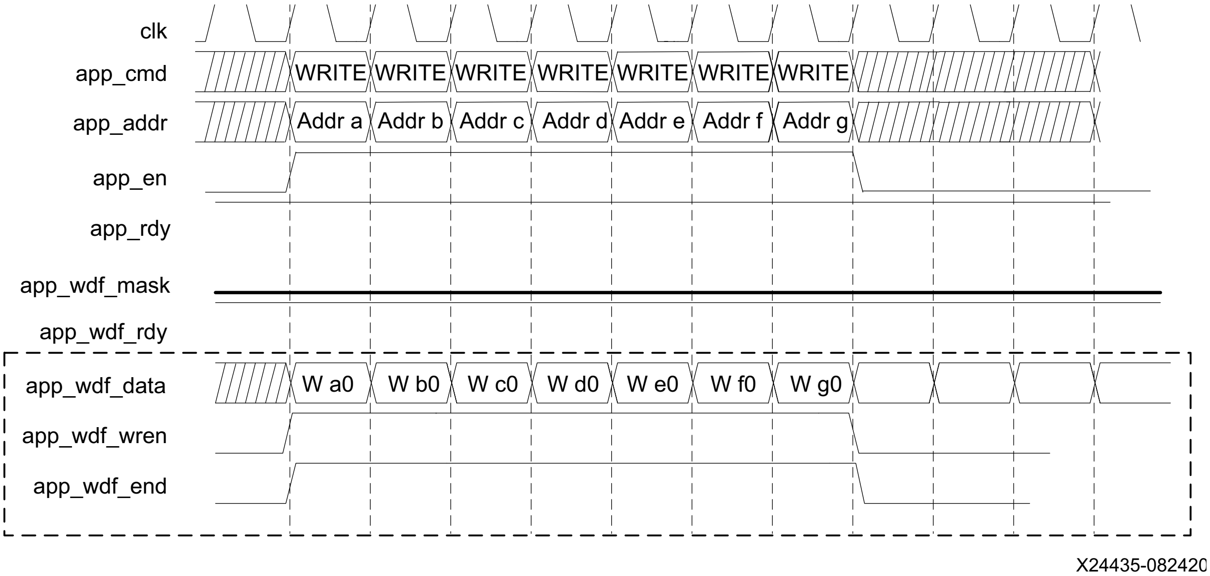


Figure 11-6: **4:1 Mode User Interface Back-to-Back Write Commands Timing Diagram (Memory Burst Type = BL8)**

The timing requirement for **`app_wdf_data`**, **`app_wdf_wren`**, and **`app_wdf_end`** relative to their associated write command is the same for back-to-back writes as it is for single writes, as shown in Figure 11-5.

The map of the application interface data to the DRAM output data can be explained with an example. For a 4:1 Memory Controller to DRAM clock ratio with an 8-bit memory, at the application interface, if the 64-bit data driven is **`0000_0806_0000_0805`** (Hex), the data values at different clock edges are as shown in Table 11-6. This is for a BL8 (Burst Length 8) transaction.

Table 11-6: **Data Values at Different Clock Edges**

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

Table 11-7 shows a generalized representation of how DRAM DQ bus data is concatenated to form application interface data signals. **`app_wdf_data`** is shown in Table 11-7, but the table applies equally to **`app_rd_data`**. Each byte of the DQ bus has eight bursts, Rise0 (burst 0) through Fall3 (burst 7) as shown previously in Table 11-6, for a total of 64 data bits. When concatenated with Rise0 in the LSB position and Fall3 in the MSB position, a 64-bit chunk of the **`app_wdf_data`** signal is formed.

For example, the eight bursts of **`lpddr3_dq[7:0]`** corresponds to DQ bus byte 0, and when concatenated as described here, they map to **`app_wdf_data[63:0]`**. To be clear on the concatenation order, **`lpddr3_dq[0]`** from Rise0 (burst 0) maps to **`app_wdf_data[0]`**, and **`lpddr3_dq[7]`** from Fall3 (burst 7) maps to **`app_wdf_data[63]`**. The table shows a second example, mapping DQ byte 1 to **`app_wdf_data[127:64]`**, as well as the formula for DQ byte N.

Table 11-7: **DRAM DQ Bus Data Map**

DQ Bus Byte	App Interface Signal	DDR Bus Signal at Each BL8 Burst Position				
		Fall3	**...**	**Rise1**	**Fall0**	**Rise0**
N	app_wdf_data [(N + 1) × 64 – 1: N × 64]	lpddr3_dq[(N + 1) × 8 – 1:N × 8]	...	lpddr3_dq[(N + 1) × 8 – 1:N × 8]	lpddr3_dq[(N + 1) × 8 – 1:N × 8]	lpddr3_dq[(N + 1) × 8 – 1:N × 8]
1	app_wdf_data[127:64]	lpddr3_dq[15:8]	...	lpddr3_dq[15:8]	lpddr3_dq[15:8]	lpddr3_dq[15:8]
0	app_wdf_data[63:0]	lpddr3_dq[7:0]	...	lpddr3_dq[7:0]	lpddr3_dq[7:0]	lpddr3_dq[7:0]

In a similar manner to the DQ bus mapping, the DM bus maps to **`app_wdf_mask`** by concatenating the DM bits in the same burst order. Example for the first two bytes of the DRAM bus are shown in Table 11-8, and the formula for mapping DM for byte N is also given.

Table 11-8: **DRAM DM Bus Data Map**

DM Bus Byte	App Interface Signal	DDR Bus Signal at Each BL8 Burst Position				
		Fall3	**...**	**Rise1**	**Fall0**	**Rise0**
N	app_wdf_mask [(N + 1) × 8 – 1:N × 8]	lpddr3_dm[N]	...	lpddr3_dm[N]	lpddr3_dm[N]	lpddr3_dm[N]
1	app_wdf_mask[15:0]	lpddr3_dm[1]	...	lpddr3_dm[1]	lpddr3_dm[1]	lpddr3_dm[1]
0	app_wdf_mask[7:0]	lpddr3_dm[0]	...	lpddr3_dm[0]	lpddr3_dm[0]	lpddr3_dm[0]

Read Path

The read data is returned by the user interface in the requested order and is valid when **`app_rd_data_valid`** is asserted (Figure 11-7 and Figure 11-8). The **`app_rd_data_end`** signal indicates the end of each read command burst and is not needed in user logic.

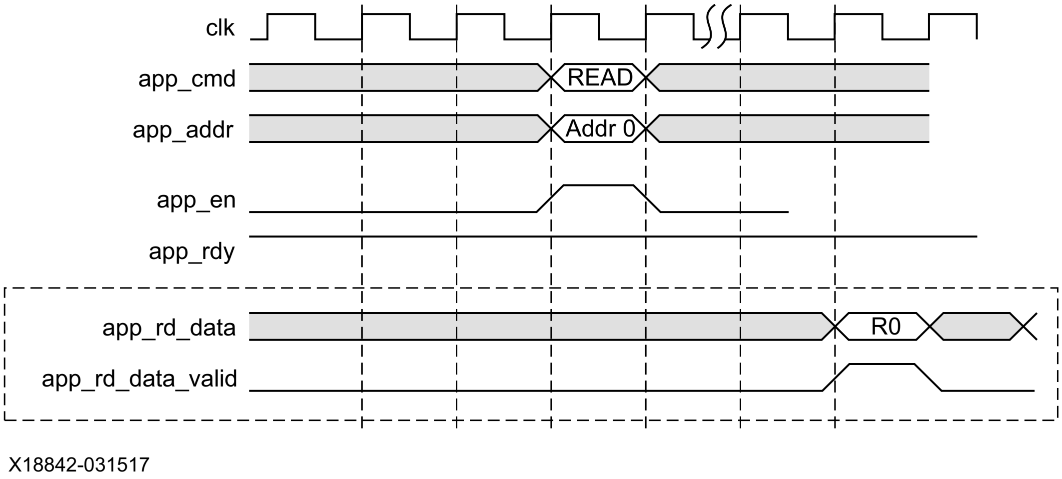


Figure 11-7: **4:1 Mode User Interface Read Timing Diagram (Memory Burst Type = BL8) #1**

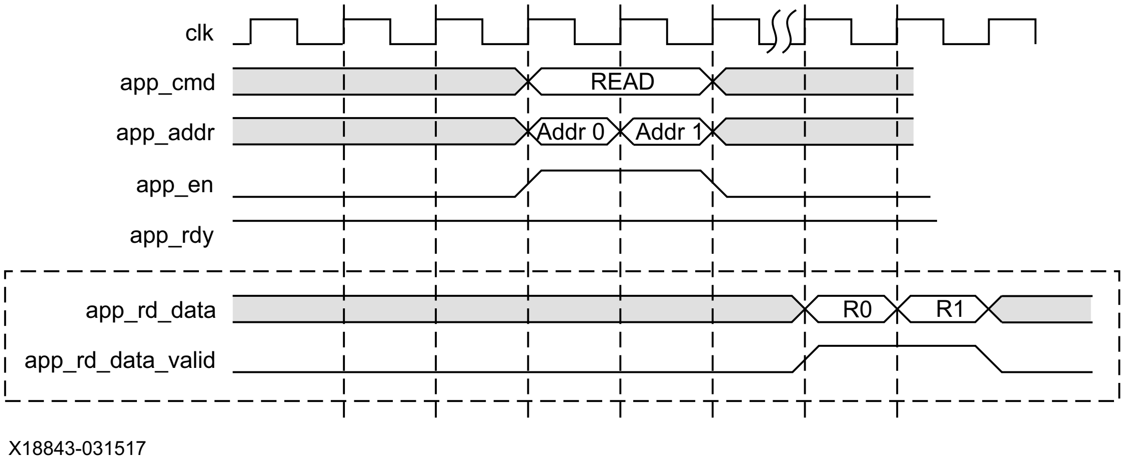


Figure 11-8: **4:1 Mode User Interface Read Timing Diagram (Memory Burst Type = BL8) #2**

In Figure 11-8, the read data returned is always in the same order as the requests made on the address/control bus.

Periodic Reads

The FPGA DDR PHY requires two back-to-back DRAM RD or RDA command to be issued every 1 µs. This requirement is described in the User Interface. When the controller is writing and the 1 µs periodic reads are due, the reads are injected by the controller to the address of the next read/write in the queue. When the controller is idle and no reads or writes are requested, the periodic reads use the last address accessed. If this address has been closed, an activate is required. This injected read is issued to the DRAM following the normal mechanisms of the controller issuing transactions. The key difference is that no read data is returned to the UI. This is wasted DRAM bandwidth.

User interface patterns with long strings of write transactions are affected the most by the PHY periodic read requirement. Consider a pattern with a 50/50 read/write transaction ratio, but organized such that the pattern alternates between 2 µs bursts of 100% page hit reads and 2 µs bursts of 100% page hit writes. The periodic reads are injected in the 2 µs write burst, resulting in a loss of efficiency due to the read command and the turnaround time to switch the DRAM and DDR bus from writes to reads back to writes. This 2 µs alternating burst pattern is slightly more efficient than alternating between reads and writes every 1 µs. A 1 µs or shorter alternating pattern would eliminate the need for the controller to inject reads, but there would still be more read-write turnarounds.

Bus turnarounds are expensive in terms of efficiency and should be avoided if possible. Long bursts of page hit writes, > 2 µs in duration, are still the most efficient way to write to the DRAM, but the impact of one write-read-write turnaround each 1 µs must be taken into account when calculating the maximum write efficiency.

M and D Support for Reference Input Clock Speed

Memory IPs provide two possibilities to select the Reference Input Clock Speed. Value allowed for Reference Input Clock Speed (ps) is always ≥ Memory Device Interface Speed (ps).

- Memory IP lists the possible Reference Input Clock Speed values based on the targeted memory frequency (based on selected Memory Device Interface Speed).
- Otherwise, select **M and D Options** and target for desired Reference Input Clock Speed which is calculated based on selected CLKFBOUT_MULT (M), DIVCLK_DIVIDE (D), and CLKOUT0_DIVIDE (D0) values in the Advanced Clocking Tab.

The required Reference Input Clock Speed is calculated from the M, D, and D0 values entered in the GUI using the following formulas:

- MMCM_CLKOUT (MHz) = tCK / Phy_Clock_Ratio

  Where tCK is the Memory Device Interface Speed selected in the Basic tab.

- CLKIN (MHz) = (MMCM_CLKOUT (MHz) × D × D0) / M

  CLKIN (MHz) is the calculated Reference Input Clock Speed.

- VCO (MHz) = (CLKIN (MHz)) / D

  VCO (MHz) is the calculated VCO frequency.

- PFD (MHz) = CLKIN (MHz) / D

  PFD (MHz) is the calculated PFD frequency.

Calculated Reference Input Clock Speed from M, D, and D0 values are validated as per clocking guidelines. For more information on clocking rules, see Clocking.

Apart from the memory specific clocking rules, validation of the possible MMCM input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values are completed for M, D, and D0 in the GUI.

For UltraScale devices, see *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2] and *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For UltraScale+ devices, see *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4], *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5], and *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For possible M, D, and D0 values and detailed information on clocking and the MMCM, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]

Customizing and Generating the Core

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

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the **`validate_bd_design`** command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

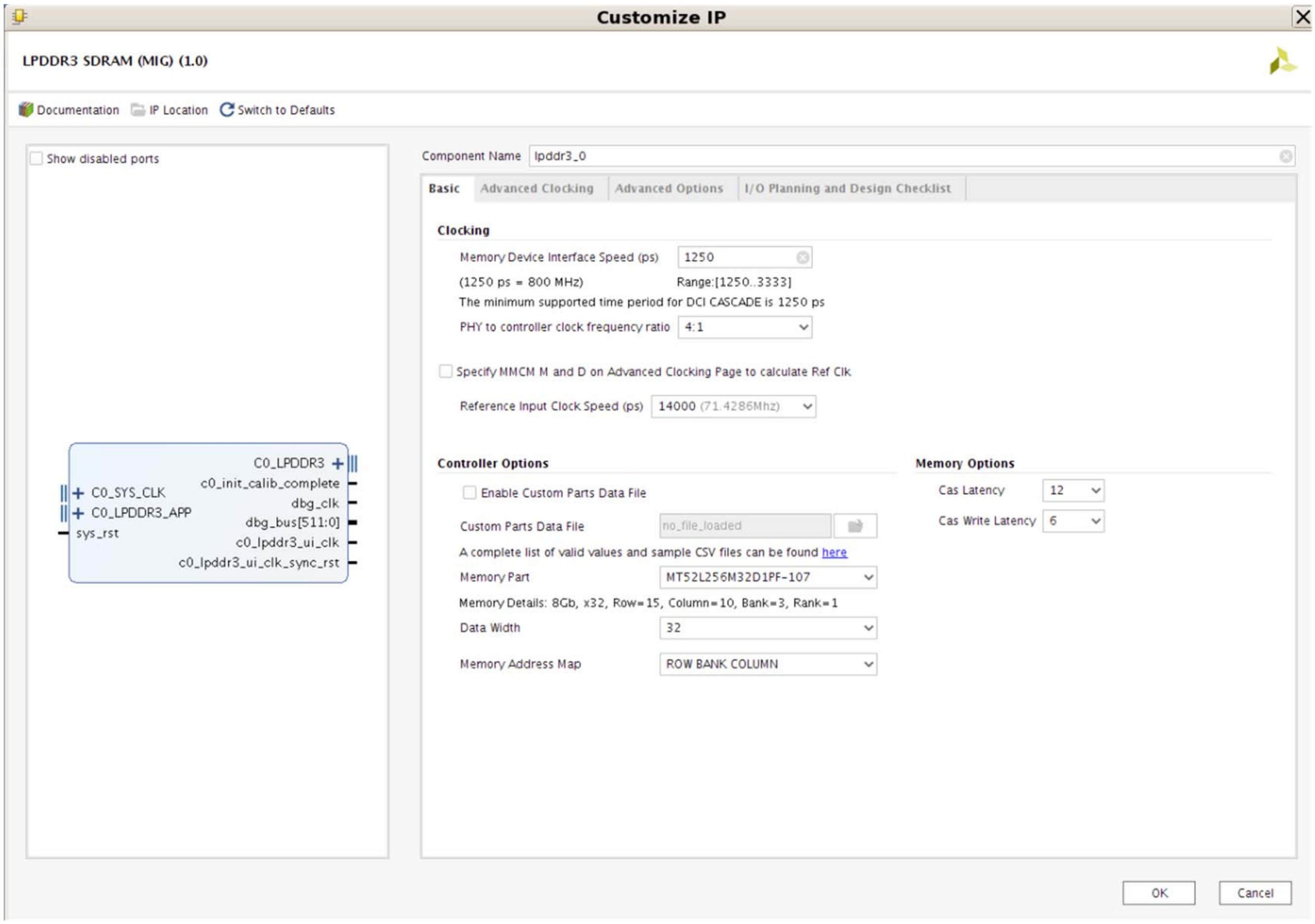
1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For more information about generating the core in Vivado, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

Basic Tab

Figure 12-1 shows the **Basic** tab when you start up the LPDDR3 SDRAM.



Component Name lpddr3_0

Clocking

Memory Device Interface Speed (ps) 1250

(1250 ps = 800 MHz) Range:[1250..3333]

The minimum supported time period for DCI CASCADE is 1250 ps

PHY to controller clock frequency ratio 4:1

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps) 14000 (71.4286Mhz)

Controller Options

☐ Enable Custom Parts Data File

Custom Parts Data File no_file_loaded

A complete list of valid values and sample CSV files can be found here

Memory Part MT52L256M32D1PF-107

Memory Details: 8Gb, x32, Row=15, Column=10, Bank=3, Rank=1

Data Width 32

Memory Address Map ROW BANK COLUMN

Memory Options

Cas Latency 12

Cas Write Latency 6

Figure 12-1: **Vivado Customize IP Dialog Box for LPDDR3 – Basic**

IMPORTANT: *All parameters shown in the controller options dialog box are limited selection options in this release.*

For the Vivado IDE, all controllers (DDR3, DDR4, LPDDR3, QDR II+, QDR-IV, and RLDRAM 3) can be created and available for instantiation.

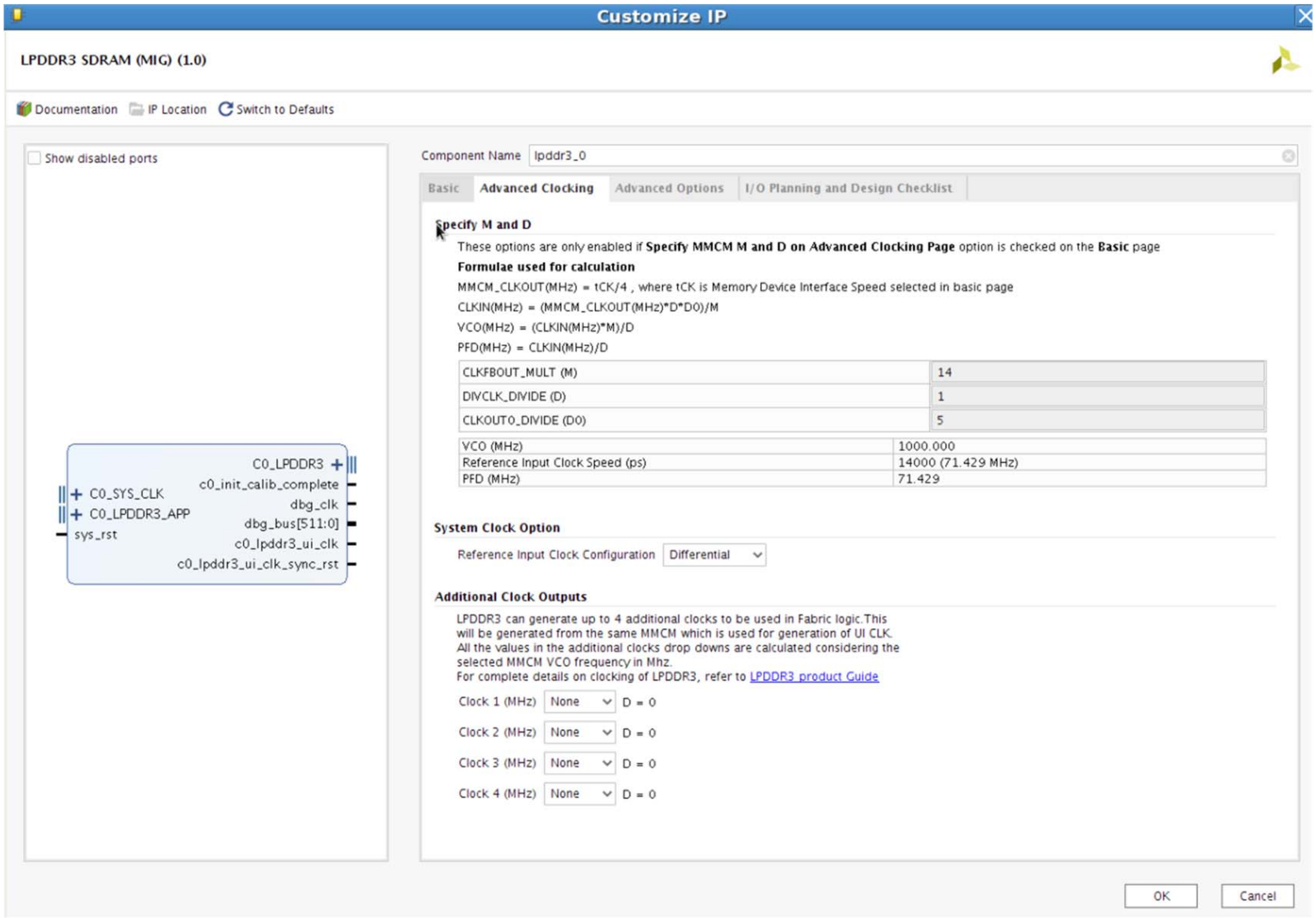
1. Select the settings in the **Clocking**, **Controller Options**, **Memory Options**, and **Advanced User Request Controller Options**.

   In **Clocking**, the **Memory Device Interface Speed** sets the speed of the interface. The speed entered drives the available **Reference Input Clock Speeds**. For more information on the clocking structure, see the Clocking, page 284.

2. To use memory parts which are not available by default through the LPDDR3 SDRAM Vivado IDE, you can create a custom parts CSV file, as specified in the AR: 63462. This CSV file has to be provided after enabling the **Custom Parts Data File** option. After selecting this option. you are able to see the custom memory parts along with the default memory parts. Note that, simulations are not supported for the custom part. Custom part simulations require manually adding the memory model to the simulation and might require modifying the test bench instantiation.

Advanced Clocking Tab

Figure 12-2 shows the next tab called **Advanced Clocking**. This displays the settings for **Specify M and D** value, **System Clock Options**, and **Additional Clock Outputs** for the specific controller.



Component Name: lpddr3_0

Specify M and D

These options are only enabled if **Specify MMCM M and D on Advanced Clocking Page** option is checked on the **Basic** page

Formulae used for calculation

MMCM_CLKOUT(MHz) = tCK/4 , where tCK is Memory Device Interface Speed selected in basic page

CLKIN(MHz) = (MMCM_CLKOUT(MHz)*D*D0)/M

VCO(MHz) = (CLKIN(MHz)*M)/D

PFD(MHz) = CLKIN(MHz)/D

CLKFBOUT_MULT (M)	14
DIVCLK_DIVIDE (D)	1
CLKOUT0_DIVIDE (D0)	5

VCO (MHz)	1000.000
Reference Input Clock Speed (ps)	14000 (71.429 MHz)
PFD (MHz)	71.429

System Clock Option

Reference Input Clock Configuration: Differential

Additional Clock Outputs

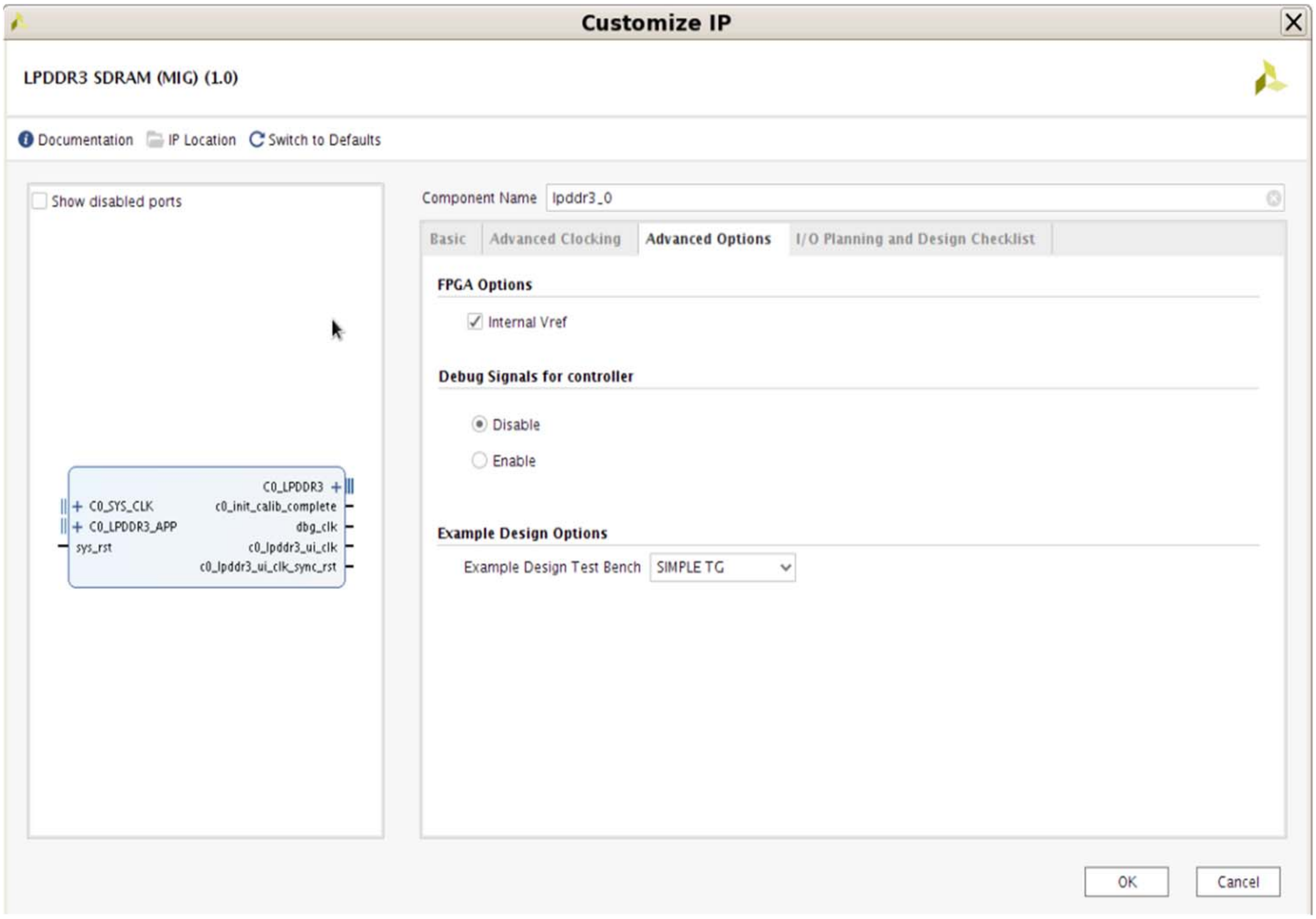
LPDDR3 can generate up to 4 additional clocks to be used in Fabric logic.This will be generated from the same MMCM which is used for generation of UI CLK. All the values in the additional clocks drop downs are calculated considering the selected MMCM VCO frequency in Mhz.
For complete details on clocking of LPDDR3, refer to LPDDR3 product Guide

Clock 1 (MHz): None D = 0
Clock 2 (MHz): None D = 0
Clock 3 (MHz): None D = 0
Clock 4 (MHz): None D = 0

Figure 12-2: **Vivado Customize IP Dialog Box for LPDDR3 – Advanced Clocking**

Advanced Options Tab

Figure 12-3 shows the next tab called **Advanced Options**. This displays the advanced memory options for the specific controller.



Component Name: lpddr3_0

FPGA Options

☑ Internal Vref

Debug Signals for controller

● Disable

○ Enable

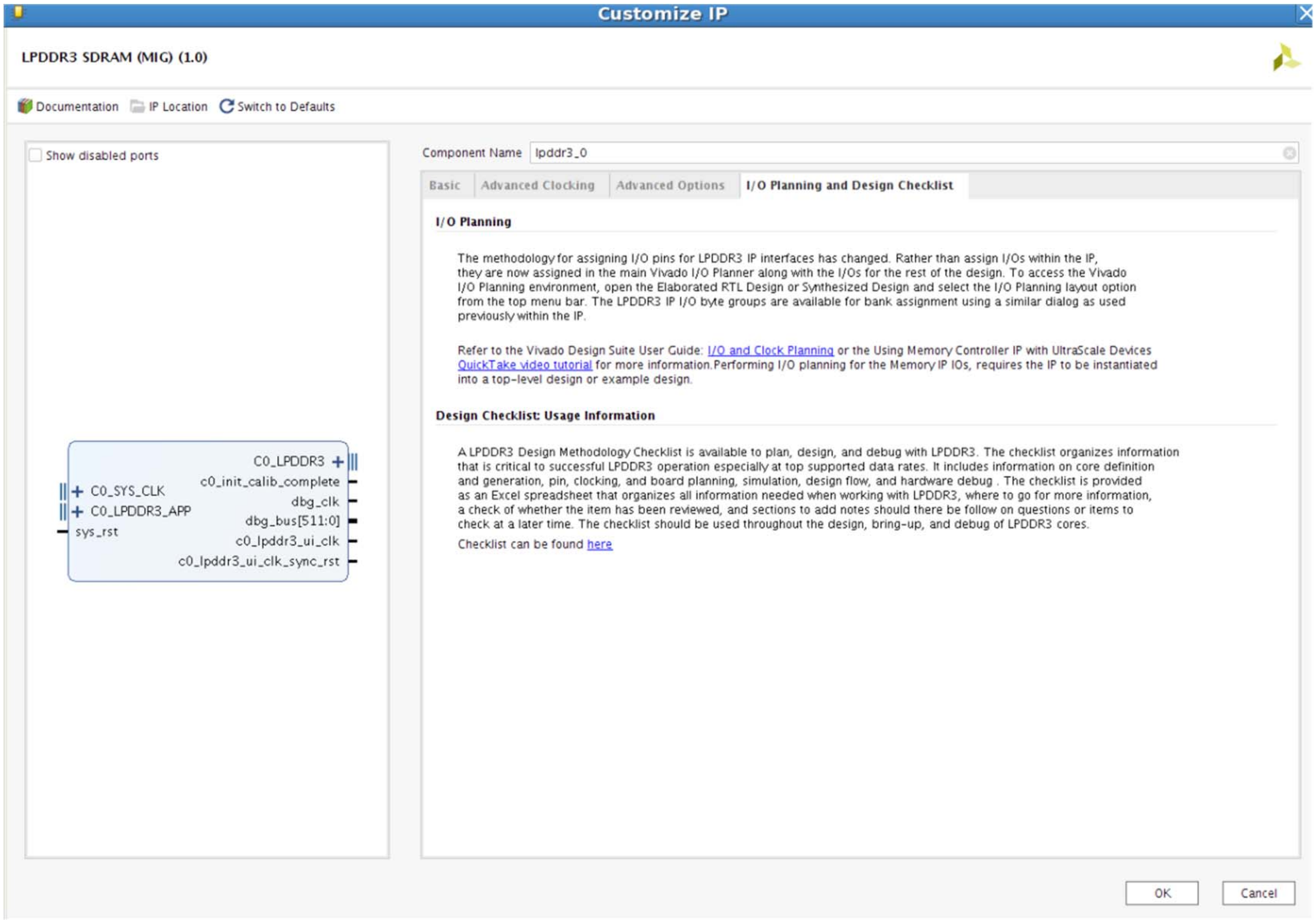
Example Design Options

Example Design Test Bench: SIMPLE TG

Figure 12-3: **Vivado Customize IP Dialog Box for LPDDR3 – Advanced Options**

LPDDR3 SDRAM I/O Planning and Design Checklist Tab

Figure 12-4 shows the **LPDDR SDRAM I/O Planning and Design Checklist** usage information.



Component Name lpddr3_0

I/O Planning

The methodology for assigning I/O pins for LPDDR3 IP interfaces has changed. Rather than assign I/Os within the IP, they are now assigned in the main Vivado I/O Planner along with the I/Os for the rest of the design. To access the Vivado I/O Planning environment, open the Elaborated RTL Design or Synthesized Design and select the I/O Planning layout option from the top menu bar. The LPDDR3 IP I/O byte groups are available for bank assignment using a similar dialog as used previously within the IP.

Refer to the Vivado Design Suite User Guide: I/O and Clock Planning or the Using Memory Controller IP with UltraScale Devices QuickTake video tutorial for more information.Performing I/O planning for the Memory IP IOs, requires the IP to be instantiated into a top-level design or example design.

Design Checklist: Usage Information

A LPDDR3 Design Methodology Checklist is available to plan, design, and debug with LPDDR3. The checklist organizes information that is critical to successful LPDDR3 operation especially at top supported data rates. It includes information on core definition and generation, pin, clocking, and board planning, simulation, design flow, and hardware debug . The checklist is provided as an Excel spreadsheet that organizes all information needed when working with LPDDR3, where to go for more information, a check of whether the item has been reviewed, and sections to add notes should there be follow on questions or items to check at a later time. The checklist should be used throughout the design, bring-up, and debug of LPDDR3 cores.

Checklist can be found here

Figure 12-4: **Vivado Customize IP Dialog Box – LPDDR3 SDRAM I/O Planning and Design Checklist**

User Parameters

Table 12-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 12-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value
System Clock Configuration	System_Clock	Differential
Internal V_{REF}	Internal_Vref	TRUE
DCI Cascade	DCI_Cascade	FALSE
Debug Signal for Controller	Debug_Signal	Disable
Clock 1 (MHz)	ADDN_UI_CLKOUT1_FREQ_HZ	None

Table 12-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value
Clock 2 (MHz)	ADDN_UI_CLKOUT2_FREQ_HZ	None
Clock 3 (MHz)	ADDN_UI_CLKOUT3_FREQ_HZ	None
Clock 4 (MHz)	ADDN_UI_CLKOUT4_FREQ_HZ	None
Enable System Ports	Enable_SysPorts	TRUE
Default Bank Selections	Default_Bank_Selections	FALSE
Reference Clock	Reference_Clock	FALSE
Enable System Ports	Enable_SysPorts	TRUE
Clock Period (ps)	C0.LPDDR3_TimePeriod	1,250
Input Clock Period (ps)	C0.LPDDR3_InputClockPeriod	14,000
General Interconnect to Memory Clock Ratio	C0.LPDDR3_PhyClockRatio	4:1
Configuration	C0.LPDDR3_MemoryType	Components
Memory Part	C0.LPDDR3_MemoryPart	MT52L256M32D1PF-107
Data Width	C0.LPDDR3_DataWidth	32
CAS Latency	C0.LPDDR3_CasLatency	12
CAS Write Latency	C0.LPDDR3_CasWriteLatency	6
Memory Address Map	C0.LPDDR3_Mem_Add_Map	ROW_BANK_COLUMN

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

I/O Planning

LPDDR3 SDRAM I/O pin planning is completed with the full design pin planning using the Vivado I/O Pin Planner. LPDDR3 SDRAM I/O pins can be selected through several Vivado I/O Pin Planner features including assignments using I/O Ports view, Package view, or Memory Bank/Byte Planner. Pin assignments can additionally be made through importing an XDC or modifying the existing XDC file.

These options are available for all LPDDR3 SDRAM designs and multiple LPDDR3 SDRAM IP instances can be completed in one setting. To learn more about the available Memory IP pin planning options, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 18].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

For LPDDR3 SDRAM Vivado IDE, you specify the pin location constraints. For more information on I/O standard and other constraints, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 18]. The location is chosen by the Vivado IDE according to the banks and byte lanes chosen for the design.

The I/O standard is chosen by the memory type selection and options in the Vivado IDE and by the pin type. A sample for `dq[0]` is shown here.

```
set_property PACKAGE_PIN AF20 [get_ports "c0_lpddr3_dq[0]"]
set_property IOSTANDARD POD12_DCI [get_ports "c0_lpddr3_dq[0]"]
```

The system clock must have the period set properly:

```
create_clock -name c0_sys_clk -period 10 [get_ports c0_sys_clk_p]
```

For HR banks, update the `output_impedance` of all the ports assigned to HR banks pins using the `reset_property` command. For more information, see AR: 63852.

IMPORTANT: *Do not alter these constraints. If the pin locations need to be altered, rerun the LPDDR3 SDRAM Vivado IDE to generate a new XDC file.*

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For more information on clocking, see Clocking, page 284.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The LPDDR3 SDRAM tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

IMPORTANT: *The `set_input_delay` and `set_output_delay` constraints are not needed on the external memory interface pins in this design due to the calibration process that automatically runs at start-up. Warnings seen during implementation for the pins can be ignored.*

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]. For more information on simulation, see Chapter 13, Example Design and Chapter 14, Test Bench.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite. Vivado supports Open IP Example Design flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in Figure 13-1 and select **Open IP Example Design**.

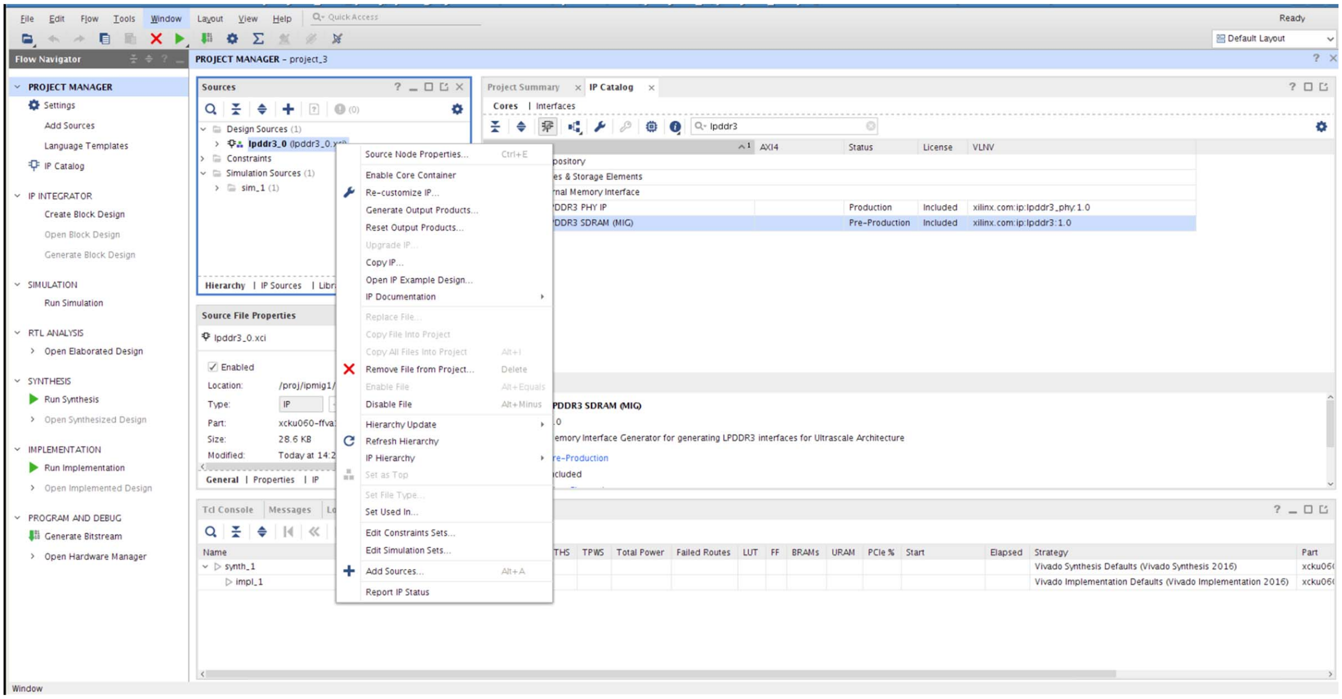


Figure 13-1: **LPDDR3 Open IP Example Design**

This option creates a new Vivado project. Upon selecting the menu, a dialog box to enter the directory information for the new design project opens.

Select a directory, or use the defaults, and click **OK**. This launches a new Vivado with all of the example design files and a copy of the IP.

Simulating the Example Design (Designs with Standard User Interface)

The example design provides a synthesizable test bench to generate a fixed simple data pattern. LPDDR3 SDRAM generates the Simple Traffic Generator (STG) module as **`example_tb`** for native interface. The STG native interface generates 100 writes and 100 reads.

The example design can be simulated using one of the methods in the following sections.

Project-Based Simulation

This method can be used to simulate the example design using the Vivado Integrated Design Environment (IDE). Memory IP delivers memory models for LPDDR3.

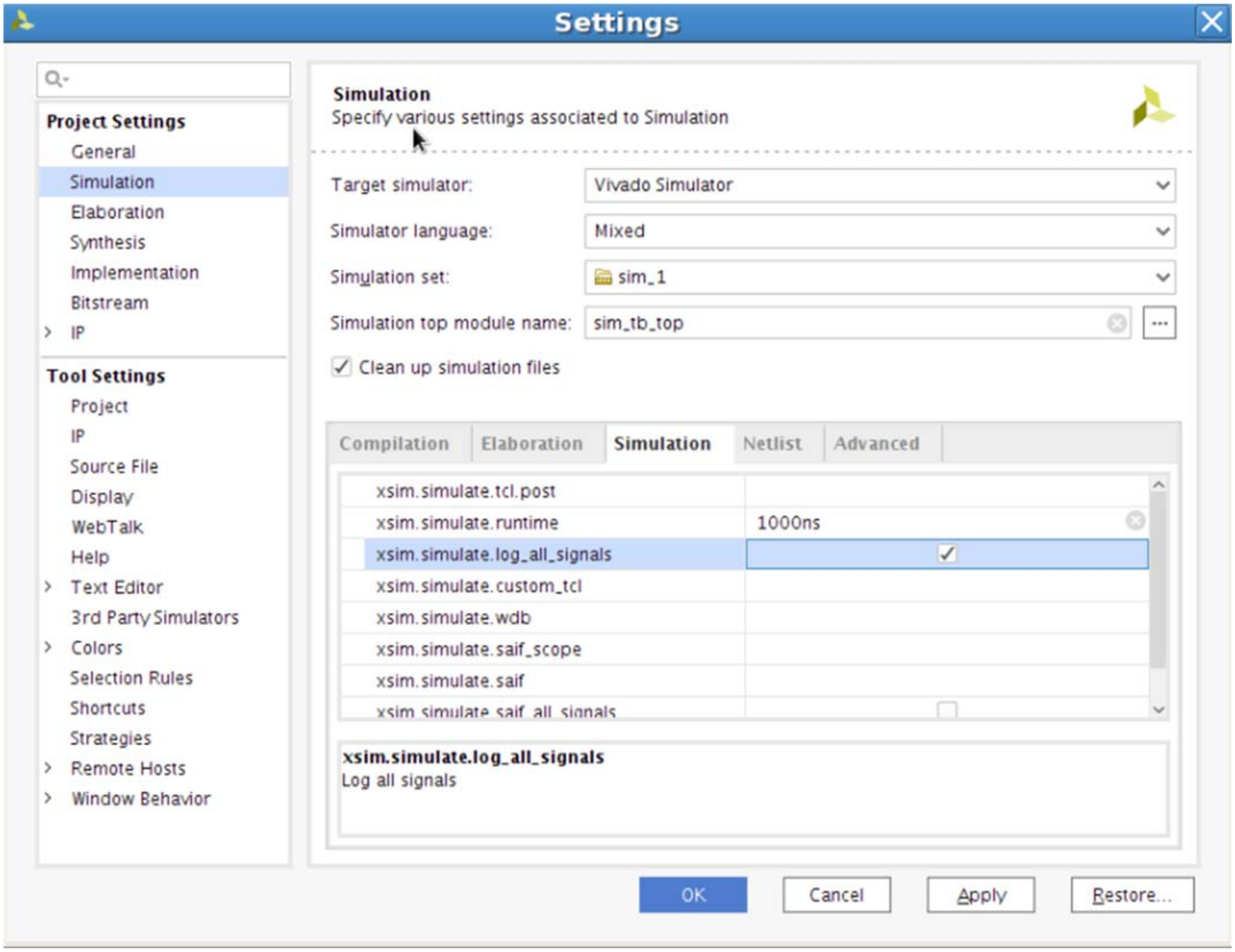
The Vivado simulator, Questa Advanced Simulator, IES, and VCS tools are used for LPDDR3 IP verification at each software release. The Vivado simulation tool is used for LPDDR3 IP verification from 2017.1 Vivado software release. The following subsections describe steps to run a project-based simulation using each supported simulator tool.

Project-Based Simulation Flow Using Vivado Simulator

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Vivado Simulator**.

   Under the **Simulation** tab, set the **`xsim.simulate.runtime`** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 13-2. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.

3. Set the **Simulation Language** to **Mixed**.
4. Apply the settings and select **OK**.



Simulation
Specify various settings associated to Simulation

Target simulator:	Vivado Simulator
Simulator language:	Mixed
Simulation set:	sim_1
Simulation top module name:	sim_tb_top

☑ Clean up simulation files

xsim.simulate.tcl.post	
xsim.simulate.runtime	1000ns
xsim.simulate.log_all_signals	☑
xsim.simulate.custom_tcl	
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐

xsim.simulate.log_all_signals
Log all signals

Figure 13-2: **Simulation with Vivado Simulator**

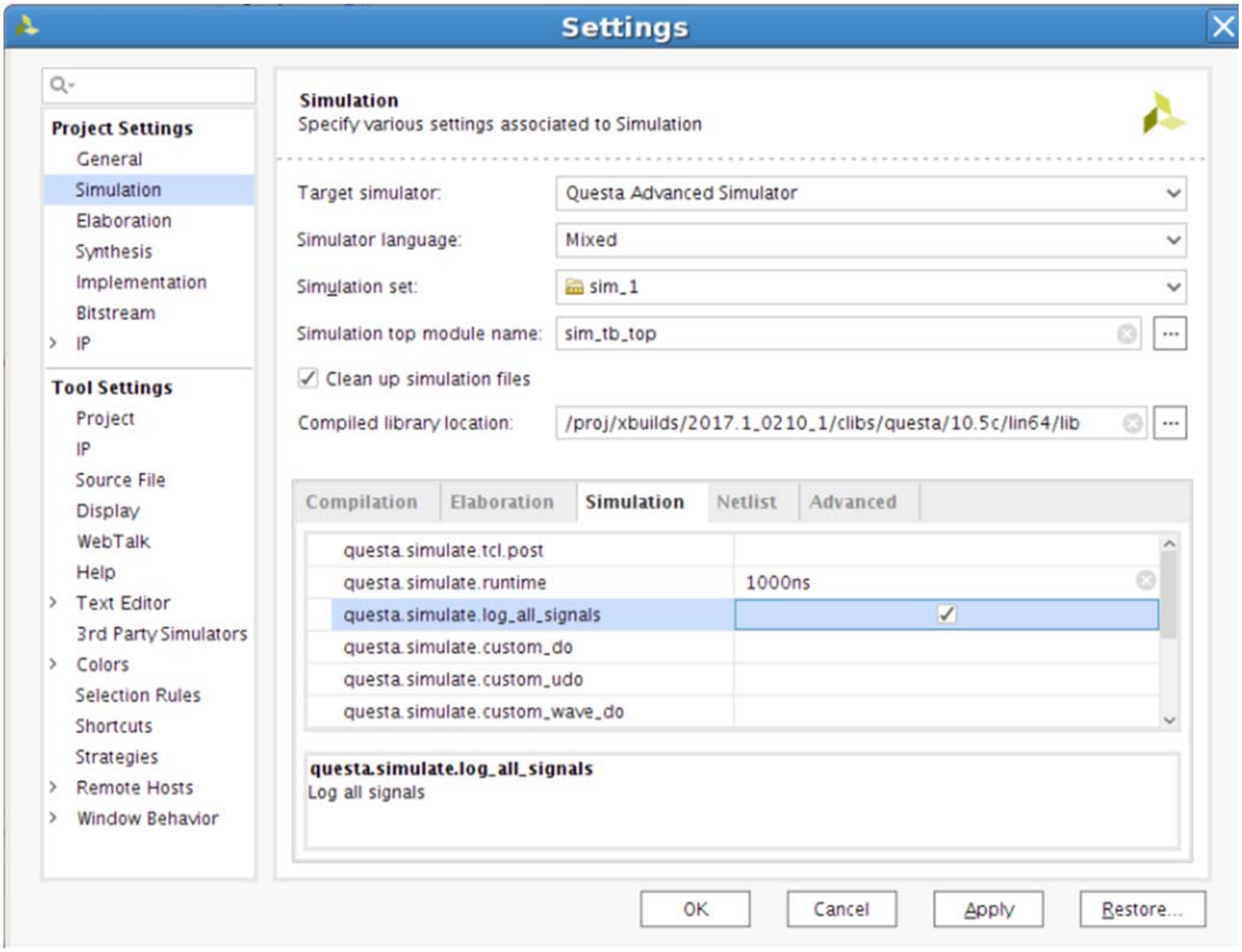
5. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 13-3.

Figure 13-3: **Run Behavioral Simulation**

6. Vivado invokes Vivado simulator and simulations are run in the Vivado simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using Questa Advanced Simulator

1. Open a LPDDR3 SDRAM example Vivado project (**Open IP Example Design**...), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as **Questa Advanced Simulator**.
   a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
   b. Under the **Simulation** tab, set the `modelsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 13-4. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



Simulation
Specify various settings associated to Simulation

Target simulator: Questa Advanced Simulator
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: sim_tb_top
☑ Clean up simulation files
Compiled library location: /proj/xbuilds/2017.1_0210_1/clibs/questa/10.5c/lin64/lib

questa.simulate.tcl.post	
questa.simulate.runtime	1000ns
questa.simulate.log_all_signals	☑
questa.simulate.custom_do	
questa.simulate.custom_udo	
questa.simulate.custom_wave_do	

questa.simulate.log_all_signals
Log all signals

Figure 13-4: **Simulation with Questa Advanced Simulator**

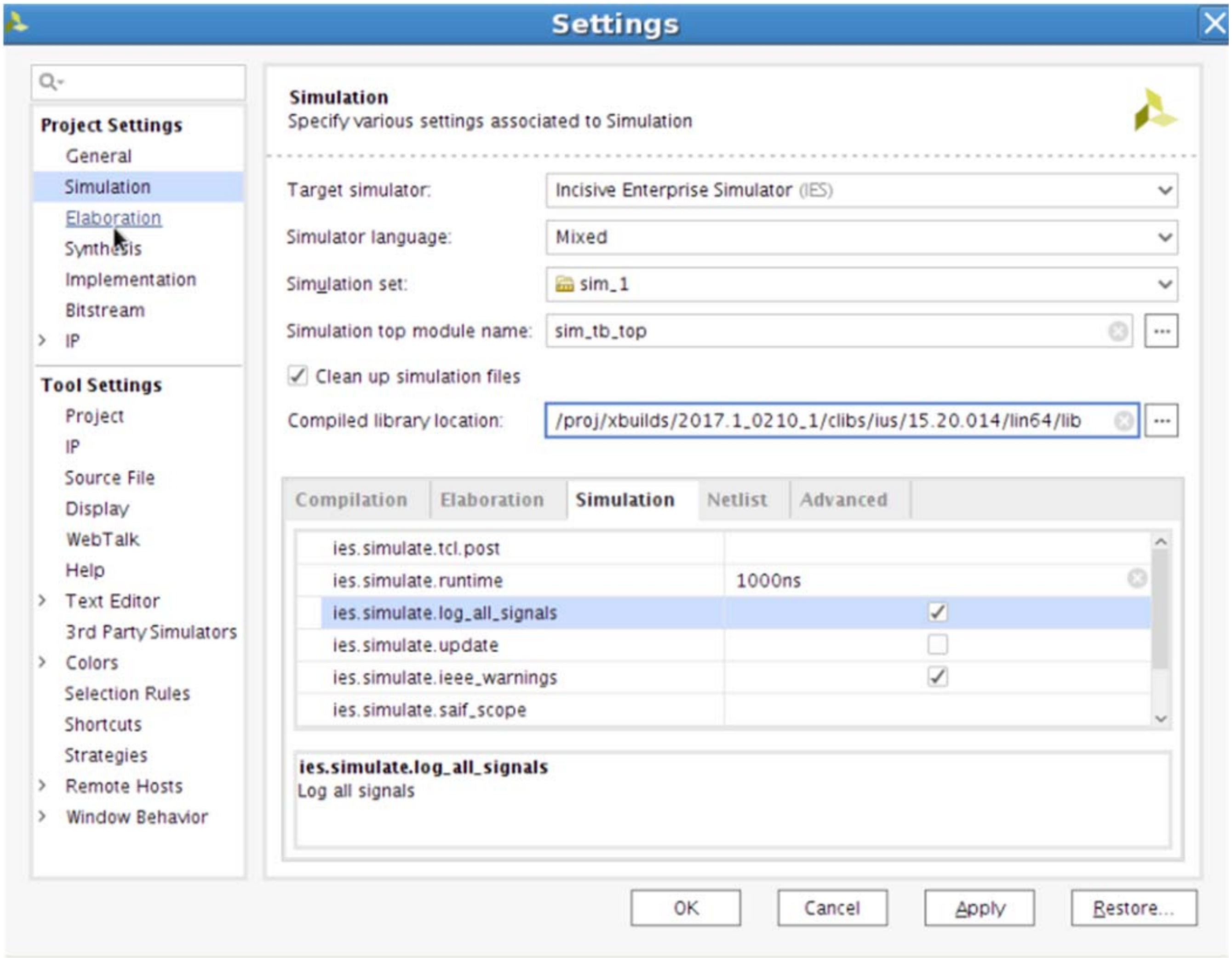
4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 13-5.

Figure 13-5: **Run Behavioral Simulation**

5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using IES

1. Open a LPDDR3 SDRAM example Vivado project (**Open IP Example Design**...), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `ies.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 13-6. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



Simulation
Specify various settings associated to Simulation

Target simulator: Incisive Enterprise Simulator (IES)
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: sim_tb_top
☑ Clean up simulation files
Compiled library location: /proj/xbuilds/2017.1_0210_1/clibs/ius/15.20.014/lin64/lib

ies.simulate.tcl.post	
ies.simulate.runtime	1000ns
ies.simulate.log_all_signals	☑
ies.simulate.update	☐
ies.simulate.ieee_warnings	☑
ies.simulate.saif_scope	

ies.simulate.log_all_signals
Log all signals

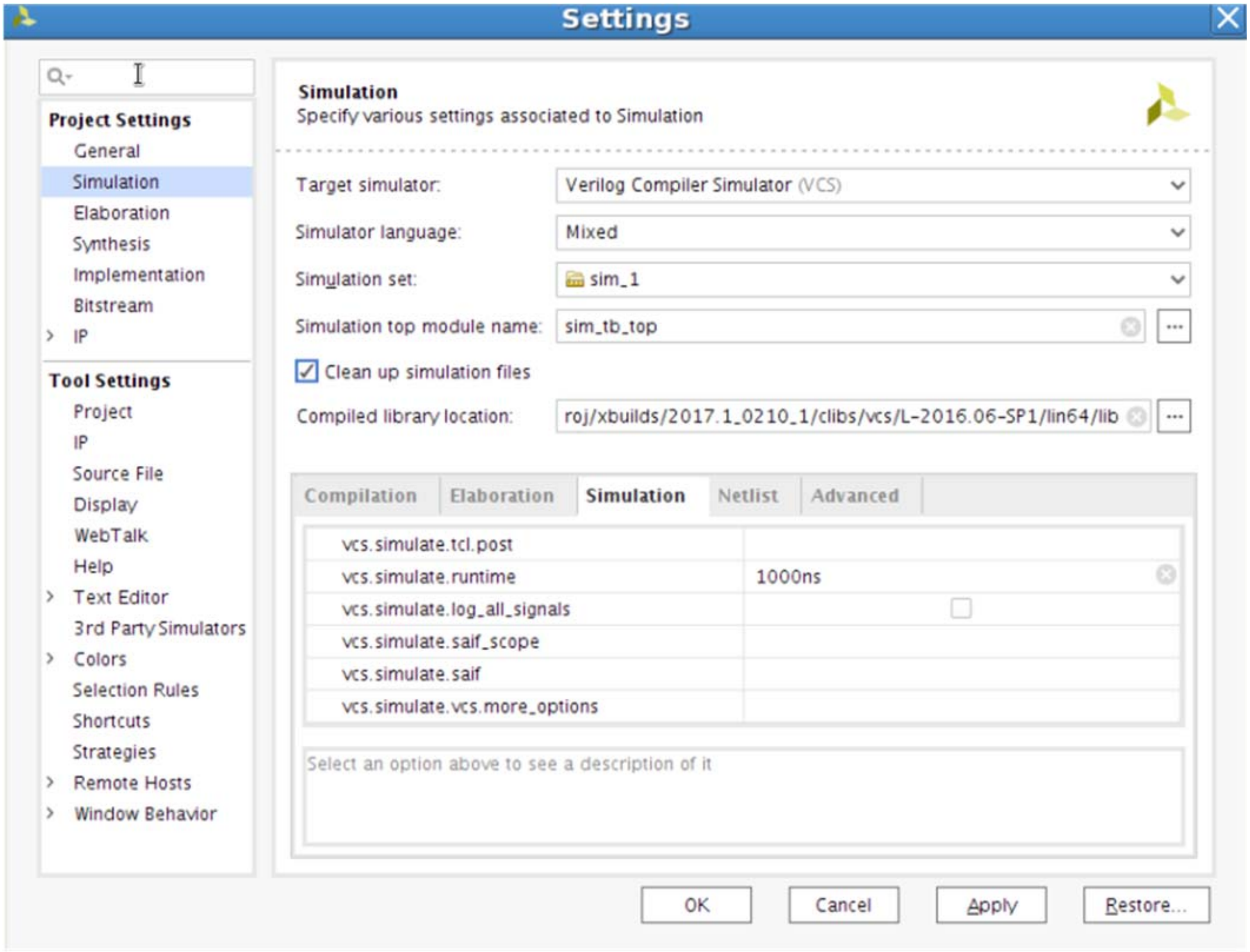
Figure 13-6: **Simulation with IES Simulator**

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 13-5.

5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using VCS

1. Open a LPDDR3 SDRAM example Vivado project (**Open IP Example Design...**), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as Verilog Compiler Simulator (VCS).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the **`vcs.simulate.runtime`** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 13-7. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



Simulation
Specify various settings associated to Simulation

Target simulator: Verilog Compiler Simulator (VCS)
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: sim_tb_top
☑ Clean up simulation files
Compiled library location: roj/xbuilds/2017.1_0210_1/clibs/vcs/L-2016.06-SP1/lin64/lib

vcs.simulate.tcl.post	
vcs.simulate.runtime	1000ns
vcs.simulate.log_all_signals	☐
vcs.simulate.saif_scope	
vcs.simulate.saif	
vcs.simulate.vcs.more_options	

Select an option above to see a description of it

Figure 13-7: **Simulation with VCS Simulator**

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 13-5.
5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation

If the GCIO pin and MMCM are not allocated in the same bank, the CLOCK_DEDICATED_ROUTE constraint must be set to BACKBONE. To use the BACKBONE route, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM input. LPDDR3 SDRAM manages these constraints for designs generated with the **Reference Input Clock** option selected as **Differential** (at **Advanced > FPGA Options > Reference Input**). Also, LPDDR3 SDRAM handles the IP and example design flows for all scenarios.

If the design is generated with the **Reference Input Clock** option selected as **No Buffer** (at **Advanced > FPGA Options > Reference Input**), the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation based on GCIO and MMCM allocation needs to be handled manually for the IP flow. LPDDR3 SDRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations for the IP flow.

For an example design flow with **No Buffer** configurations, LPDDR3 SDRAM generates the example design with differential buffer instantiation for system clock pins. LPDDR3 SDRAM generates clock constraints in the `example_design.xdc`. It also generates a CLOCK_DEDICATED_ROUTE constraint as the “BACKBONE” and instantiates BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM input if the GCIO and MMCM are not in same bank to provide a complete solution. This is done for the example design flow as a reference when it is generated for the first time.

If in the example design, the I/O pins of the system clock pins are changed to some other pins with the I/O pin planner, the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation need to be managed manually. A DRC error is reported for the same.

Chapter 14

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

The example design of the LPDDR3 Memory Controller generates either a simple test bench or an Advanced Traffic Generator based on the Example Design Test Bench input in the Vivado Integrated Design Environment wizard. For more information on the traffic generators, see Chapter 36, Traffic Generator.

SECTION IV: QDR II+ SRAM

Overview

IMPORTANT: *This document supports QDR II+ SRAM core v1.4.*

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Clocking
  - Resets
  - Protocol Description
  - Customizing and Generating the Core
  - Example Design

Core Overview

The Xilinx UltraScale™ architecture includes the QDR II+ SRAM core. This core provides solutions for interfacing with the QDR II+ SRAM memory type.

The QDR II+ SRAM core is a physical layer for interfacing Xilinx UltraScale FPGA user designs to the QDR II+ SRAM devices. QDR II+ SRAMs offer high-speed data transfers on separate read and write buses on the rising and falling edges of the clock. These memory devices are used in high-performance systems as temporary data storage, such as:

- Look-up tables in networking systems

- Packet buffers in network switches
- Cache memory in high-speed computing
- Data buffers in high-performance testers

The QDR II+ SRAM solutions core is a PHY that takes simple user commands, converts them to the QDR II+ protocol, and provides the converted commands to the memory. The design enables you to provide one read and one write request per cycle eliminating the need for a Memory Controller and the associated overhead, thereby reducing the latency through the core.

Figure 15-1 shows a high-level block diagram of the QDR II+ SRAM interface solution.

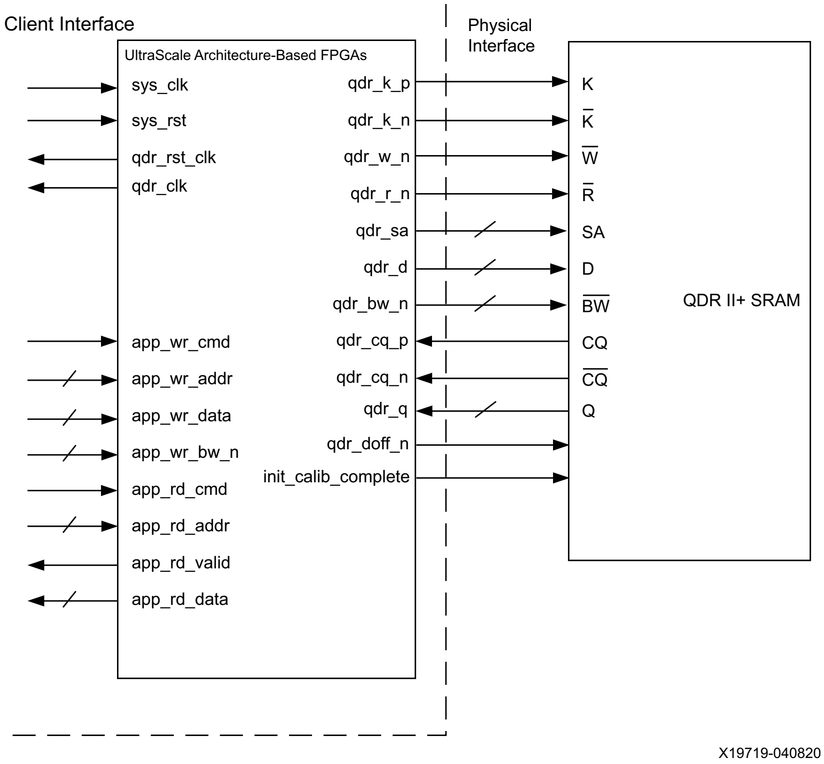


Figure 15-1: **High-Level Block Diagram of QDR II+ Interface Solution**

The physical layer includes the hard blocks inside the FPGA and the soft calibration logic necessary to ensure optimal timing of the hard blocks interfacing to the memory part.

The hard blocks include:

- Data serialization and transmission
- Data capture and deserialization
- High-speed clock generation and synchronization
- Coarse and fine delay elements per pin with voltage and temperature tracking

The soft blocks include:

- **Memory Initialization** – The calibration modules provide an initialization routine for the particular memory type. The delays in the initialization process can be bypassed to speed up simulation time if desired.

  The QDR II+ memories do not require an elaborate initialization procedure. However, you must ensure that the `Doff_n` signal is provided to the memory as required by the vendor. The QDR II+ SRAM interface design provided by the QDR II+ IP drives the `Doff_n` signal from the FPGA. After the internal MMCM has locked, the `Doff_n` signal is asserted High for 100 μs without issuing any commands to the memory device.

  For memory devices that require the `Doff_n` signal to be terminated at the memory and not be driven from the FPGA, you must perform the required initialization procedure.

- **Calibration** – The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance. Results of the calibration process is available through the Xilinx debug tools. After completion of calibration, the PHY layer presents raw interface to the memory part.

Feature Summary

- Component support for interface widths up to 36 bits
- x18 and x36 memory device support
- 4-word and 2-word burst support
- Only HSTL_I I/O standard support
- Cascaded data width support is available only for BL-4 designs
- Data rates up to 1,266 Mb/s for BL-4 designs
- Data rates up to 900 Mb/s for BL-2 designs
- Memory device support with 72 Mb density
- Support for 2.0 and 2.5 cycles of Read Latency
- Other densities for memory device support is available through custom part selection
- Source code delivery in Verilog and System Verilog
- 2:1 memory to FPGA logic interface clock ratio
- Interface calibration and training information available through the Vivado hardware manager

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Chapter 16

Product Specification

Standards

This core complies with the QDR II+ SRAM standard defined by the QDR Consortium. For more information on UltraScale™ architecture documents, see References, page 789.

Performance

Maximum Frequencies

For more information on the maximum frequencies, see the following documentation:

- *Kintex UltraScale FPGAs Data Sheet, DC and AC Switching Characteristics* (DS892) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6]
- *UltraScale Maximum Memory Performance Utility* (XTP414) [Ref 21]

Resource Utilization

For full details about performance and resource utilization, visit Performance and Resource Utilization.

Port Descriptions

There are three port categories at the top-level of the memory interface core called the "user design."

- The first category is the memory interface signals that directly interfaces with the memory part. These are defined by the QDR II+ SRAM specification.
- The second category is the application interface signals which is referred to as the "user interface." This is described in the Protocol Description, page 370.
- The third category includes other signals necessary for proper operation of the core. These include the clocks, reset, and status signals from the core. The clocking and reset signals are described in their respective sections.

  The active-High **`init_calib_complete`** signal indicates that the initialization and calibration are complete and that the interface is now ready to accept commands for the interface.

Core Architecture

This chapter describes the UltraScale™ architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Overview

The UltraScale architecture-based FPGAs Memory Interface Solutions is shown in Figure 17-1.

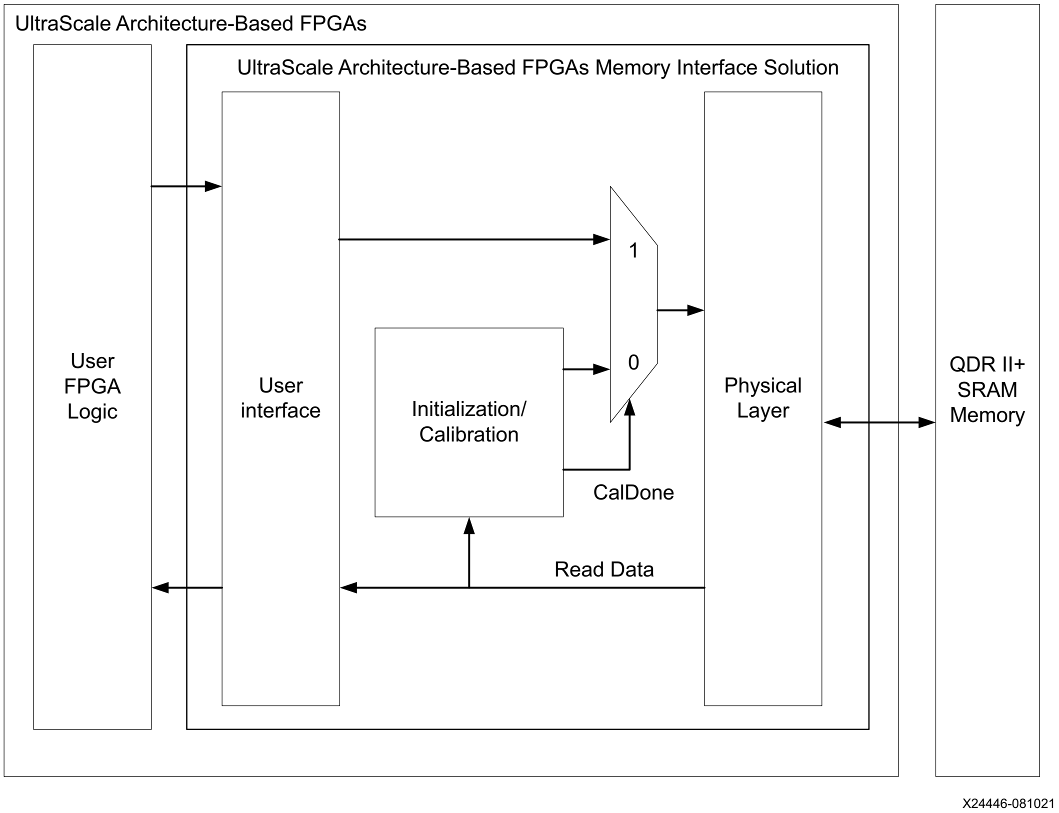


Figure 17-1: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core**

The user interface uses a simple protocol based entirely on SDR signals to make read and write requests. For more details describing this protocol, see User Interface in Chapter 18.

There is no requirement for the controller in QDR II+ SRAM protocol and thus, the Memory Controller contains only the physical interface. It takes commands from the user interface and adheres to the protocol requirements of the QDR II+ SRAM device. It is responsible to generate proper timing relationships and DDR signaling to communicate with the external memory device. For more details, see Memory Interface in Chapter 18.

PHY

The PHY is considered the low-level physical interface to an external QDR II+ SRAM device. It contains the entire calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

Overall PHY Architecture

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

The user interface and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is divided by 2. A more detailed block diagram of the PHY design is shown in Figure 17-2.

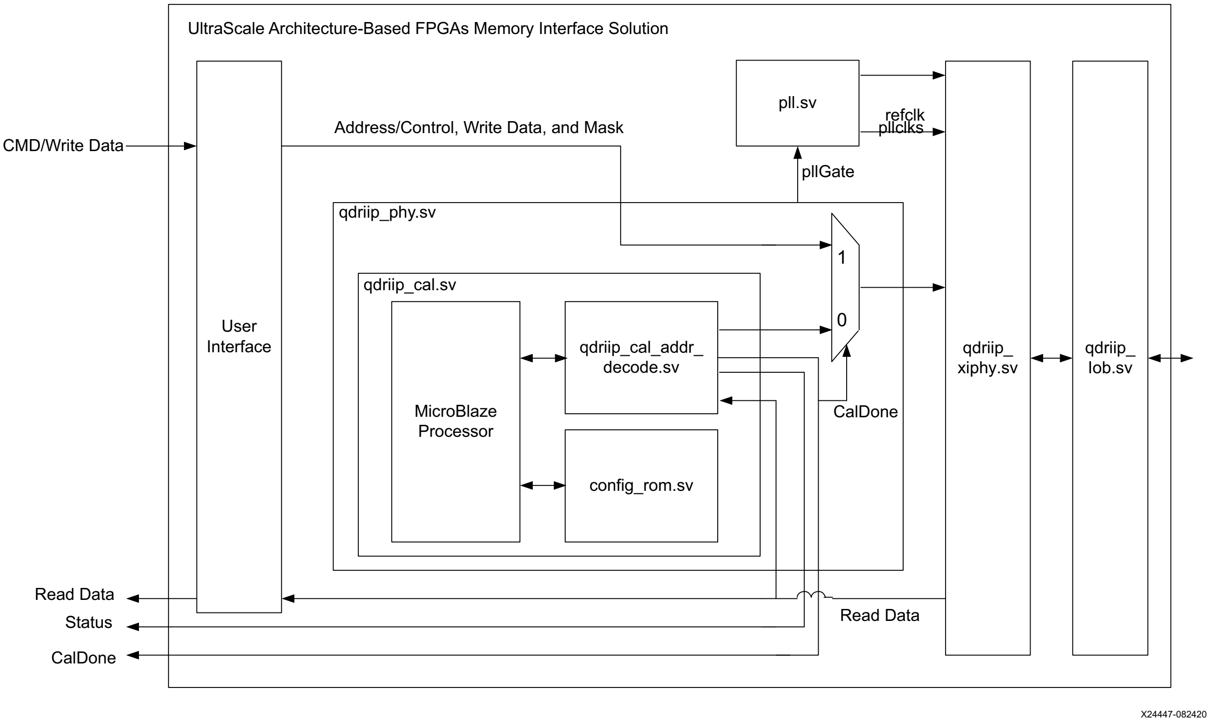


Figure 17-2: **PHY Block Diagram**

Table 17-1: **PHY Modules**

Module Name	Description
qdriip_phy.sv	PHY top of QDR II+ design
qdriip_phycal.sv	Contains the instances of XIPHY top and calibration top modules
qdriip_cal.sv	Calibration top module
qdriip_cal_addr_decode.sv	FPGA logic interface for the MicroBlaze processor
config_rom.sv	Configuration storage for calibration options
debug_microblaze.sv	MicroBlaze processor
qdriip_xiphy.sv	Contains the XIPHY instance
qdriip_iob.sv	Instantiates all byte IOB modules
qdriip_iob_byte.sv	Generates the I/O buffers for all the signals in a given byte lane
qdriip_rd_bit_slip.sv	Read bitslip

The PHY architecture encompasses all of the logic contained in **`qdriip_xiphy.sv`**. The PHY contains wrappers around dedicated hard blocks to build up the memory interface from smaller components. A byte lane contains all of the clocks, resets, and datapaths for a given subset of I/O. Multiple byte lanes are grouped together, along with dedicated clocking resources, to make up a single bank memory interface. For more information on the hard silicon physical layer architecture, see the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7].

The memory initialization and calibration are implemented in C programming on a small soft core processor. The MicroBlaze™ Controller System (MCS) is configured with an I/O Module and block RAM. The module **`qdriip_cal_adr_decode.sv`** module provides the interface for the processor to the rest of the system and implements helper logic. The **`config_rom.sv`** module stores settings that control the operation of initialization and calibration, providing run time options that can be adjusted without having to recompile the source code.

The address unit connects the MCS to the local register set and the PHY by performing address decode and control translation on the I/O module bus from spaces in the memory map and MUXing return data (**`qdriip_cal_adr_decode.sv`**). In addition, it provides address translation (also known as "mapping") from a logical conceptualization of the DRAM interface to the appropriate pinout-dependent location of the delay control in the PHY address space.

Although the calibration architecture presents a simple and organized address map for manipulating the delay elements for individual data, control and command bits, there is flexibility in how those I/O pins are placed. For a given I/O placement, the path to the FPGA logic is locked to a given pin. To enable a single binary software file to work with any memory interface pinout, a translation block converts the simplified Register Interface Unit (RIU) addressing into the pinout-specific RIU address for the target design. The specific address translation is written by QDR II+ SRAM after a pinout is selected. The code shows an example of the RTL structure that supports this.

```
Casez(io_address)// MicroBlaze I/O module address
  // … static address decoding skipped
  //========================================//
  //===========DQ ODELAYS===================//
  //========================================//
  //Byte0
  28`h0004100: begin //dq2
    riu_addr_cal =  /* QDR II+ SRAM Generated */ 6`hd;
    riu_nibble =    /* QDR II+ SRAM Generated */ 'h0;
  end
  // … additional dynamic addressing follows
```

In this example, **`DQ0`** is pinned out on Bit[0] of nibble 0 (nibble 0 according to instantiation order). The RIU address for the ODELAY for Bit[0] is **`0x0D`**. When **`DQ0`** is addressed — indicated by address **`0x000_4100`**), this snippet of code is active. It enables nibble 0 (decoded to one-hot downstream) and forwards the address **`0x0D`** to the RIU address bus.

The MicroBlaze I/O interface operates at much slower frequency, which is not fast enough for implementing all the functions required in calibration. A helper circuit implemented in `qdriip_cal_adr_decode.sv` is required to obtain commands from the registers and translate at least a portion into single-cycle accuracy for submission to the PHY. In addition, it supports command repetition to enable back-to-back read transactions and read data comparison.

Memory Initialization and Calibration Sequence

After deassertion of the system reset, the PHY performs some required internal calibration steps first.

1. The built-in self-check (BISC) of the PHY is run. It is used to compensate the internal skews among the data bits and the strobe on the read path.
2. After BISC completion, the required steps for the power-on initialization of the memory part starts.
3. It requires several stages of calibration for tuning the write and read datapath skews as mentioned in Figure 17-3.
4. After calibration is completed, PHY calculates internal offsets for the voltage and temperature tracking purpose by considering the taps used until the end of step 3.
5. When PHY indicates the calibration completion, the user interface command execution begins.

Figure 17-3 shows the overall flow of memory initialization and the different stages of calibration.

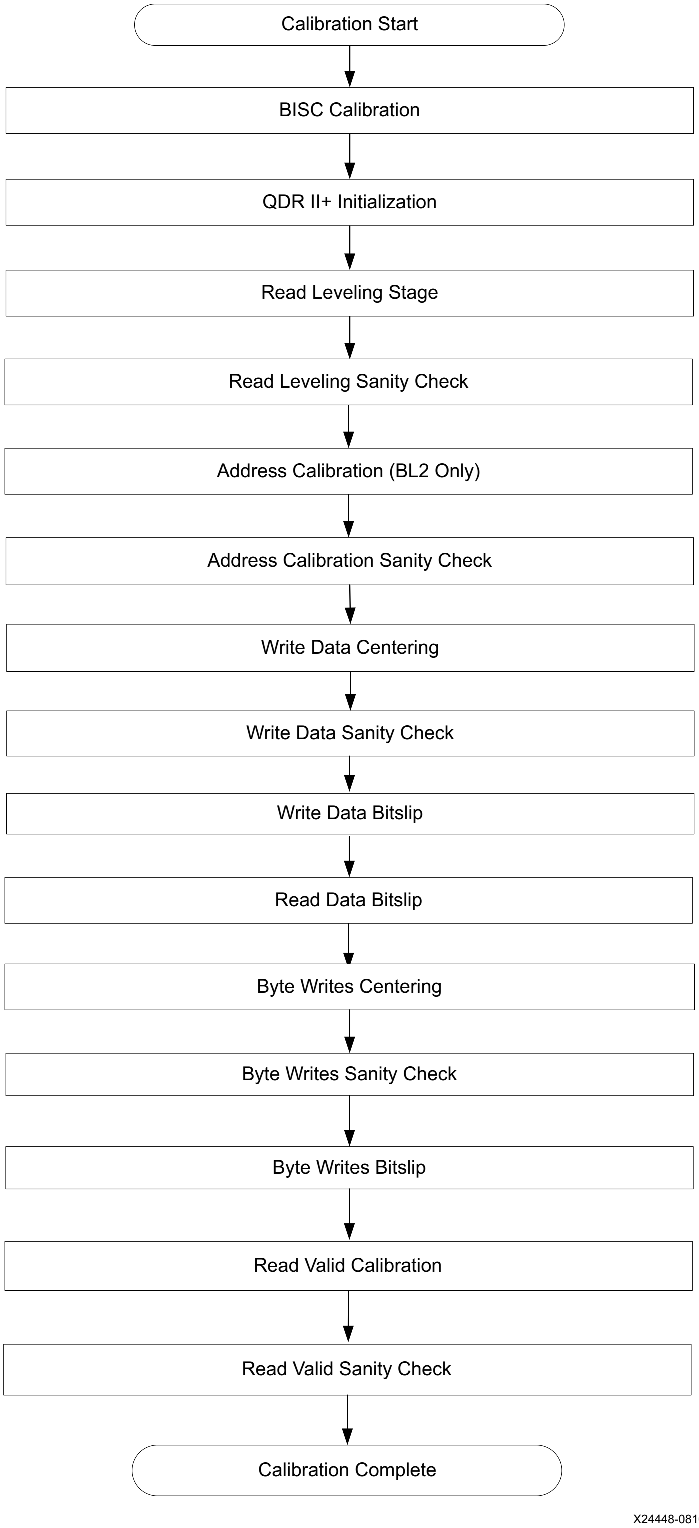


Figure 17-3: **PHY Overall Initialization and Calibration Sequence**

BISC Calibration

Built-in Self Calibration (BISC) is the first stage of calibration. BISC is enabled by configuring the SELF_CALIBRATE parameter to `2'b11` for all the byte lanes. BISC compensates the on chip delay variations among the read bits and to center align the read clock in the read data window (if enabled). BISC does not compensate the PCB delay variations and thus, the output of BISC gives a fine center alignment but not an accurate one.

Memory Initialization

The memory initialization sequence is done as per the vendor requirements.

Read Leveling

The aim of this stage is to deskew all read data bits in a nibble and then keep the rise and fall edges of the read strobe inside the valid window at an approximate 90° position.

After the completion of BISC, the capture clock position is within the valid window but not the center. Use this initial position to find the left and right edges of the valid window and then center align in it.

To create a clock pattern, write one burst of 1s and one burst of 0s into two address locations. Writing an entire burst of 1 or 0 eliminates toggles on the write data bits during the write transaction. Read leveling has to be done nibble wise as each nibble generates its own capture clock. You have to perform a back-to-back continuous reads from those two locations to find the two edges of the read data window. Here is the terminology used in read leveling algorithm:

- **PQTR** – It is the delay element on `CQ_p` capture clock. Its output is used to capture the rise data.
- **NQTR** – It is the delay element on `CQ_n` capture clock. Its output is used to capture the fall data,
- **IDELAY** – Delay element on each data bit,
- **INFIFO OUTPUT** – Read data to the user interface,

Read leveling is divided into two subsections:

Case 1: RL of 2

Aligning PQTR to Left Edge

The first step in the deskew process is to decrement PQTR and NQTR delays until one of them acquires a 0 value. After the decrement for deskew only, the P data for all the bits in the nibble are analyzed to find the left edge.

In Figure 17-4, if PQTR is in the window for all the bits in the nibble then increment IDELAY for each bit until they fail. This deskews all the bits in the nibble and PQTR is aligned at the left edge for all the bits.

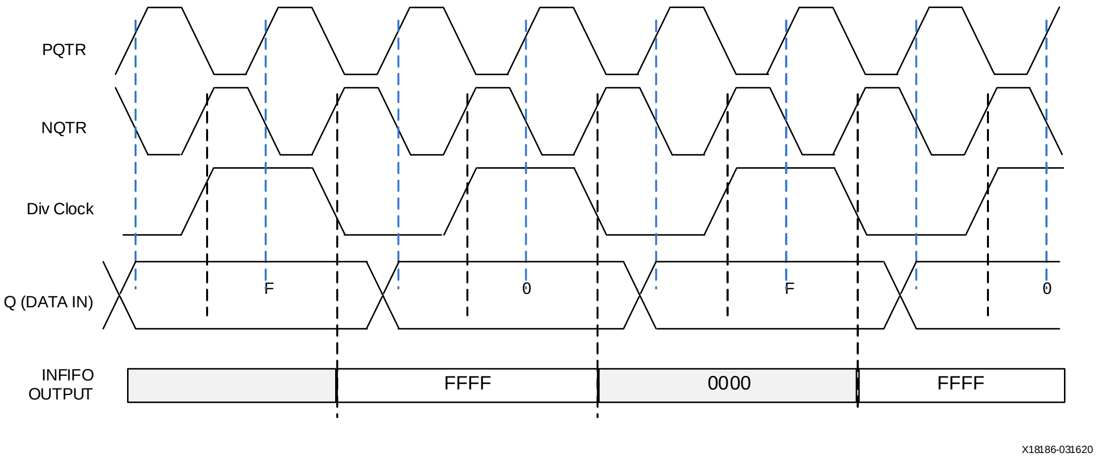


Figure 17-4: **P Data in Data Window**

For conditions in which the PQTR is outside the window for all the bits or any of the bits in the nibble, the PQTR/NQTR delays are incremented until there is a pass for all the bits. The next step would be to increment the IDELAY for each data bit in the nibble in Figure 17-5. This deskews all the bits in the nibble and PQTR is aligned at the left edge for all the bits.

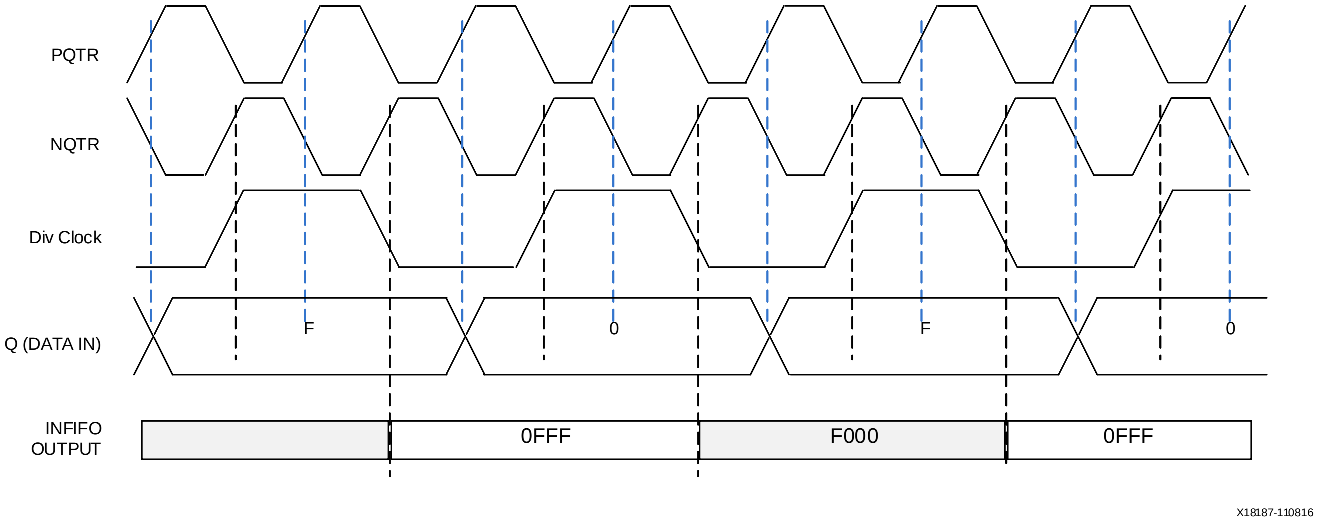


Figure 17-5: **PQTR Outside of Window**

Aligning NQTR to Right Edge

During this process both NQTR/PQTR delays are moved to find the right edge and only the N data is used for comparison. In this case since the deskew is already completed, the N data for any of the bits in a byte changes the right edge would be considered found.

Figure 17-6 shows the condition when the N data is aligned to the right edge.

The data is written into the INFIFO using the falling edge of the divided clock. The divided clock is derived from NQTR and it is being moved during the calibration stage.

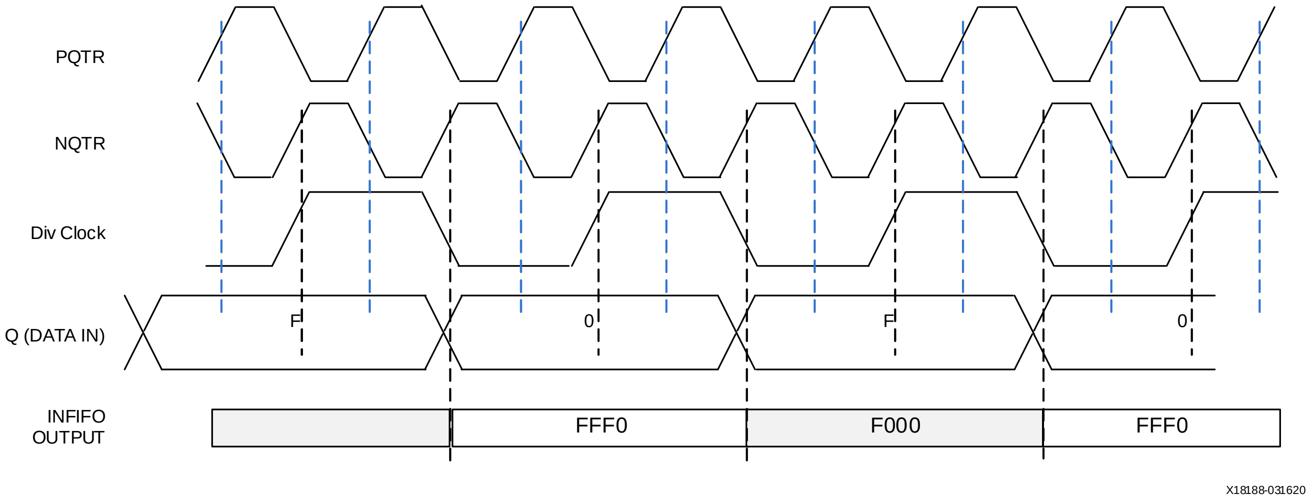


Figure 17-6: **NQTR Aligned to Right Edge**

Centering NQTR and PQTR

In this stage, the NQTR and PQTR values from the BISC calibration stage is used to center them in the data window. This is an initial calibration stage and the read leveling with complex data pattern is used after the write calibration.

> PQTR_90 = PQTR values after BISC calibration - PQTR_ALIGN gives the tap count needed for the 90° offset.
>
> NQTR_90 = NQTR values after BISC calibration - NQTR_ALIGN gives the tap count needed for the 90° offset.

These values are retained as soon as the calibration algorithm starts. Now PQTR is placed at the PQTR value at the end of "left edge alignment" + PQTR_90. NQTR is placed at NQTR value at the end of "right edge alignment" - NQTR_90.

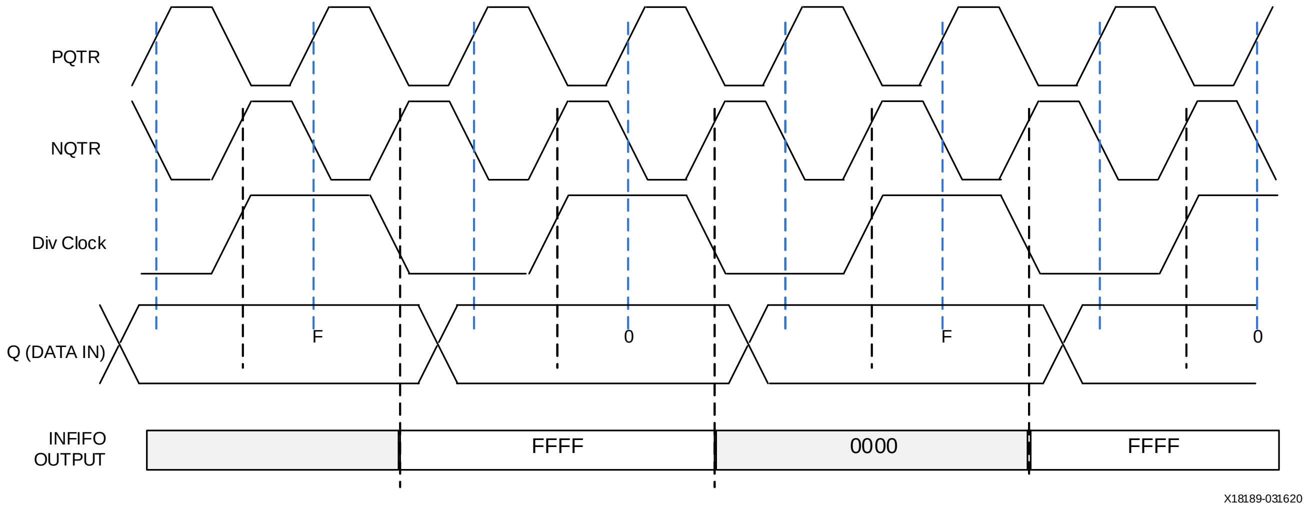


Figure 17-7: **NQTR and PQTR with 90° Offset from BISC**

Case 2: RL of 2.5

Aligning NQTR to Left Edge

With the latency of 2.5, the first data is aligned to CQ#.

The first step in the deskew process is to decrement PQTR and NQTR delays until one of them acquires a 0 value. After the decrement for deskew only, the N data for all the bits in the nibble are analyzed to find the left edge.

In Figure 17-8, if NQTR is in the window for all the bits in the nibble then increment IDELAY for each bit until they fail. This deskews all the bits in the nibble and NQTR is aligned at the left edge for all the bits.

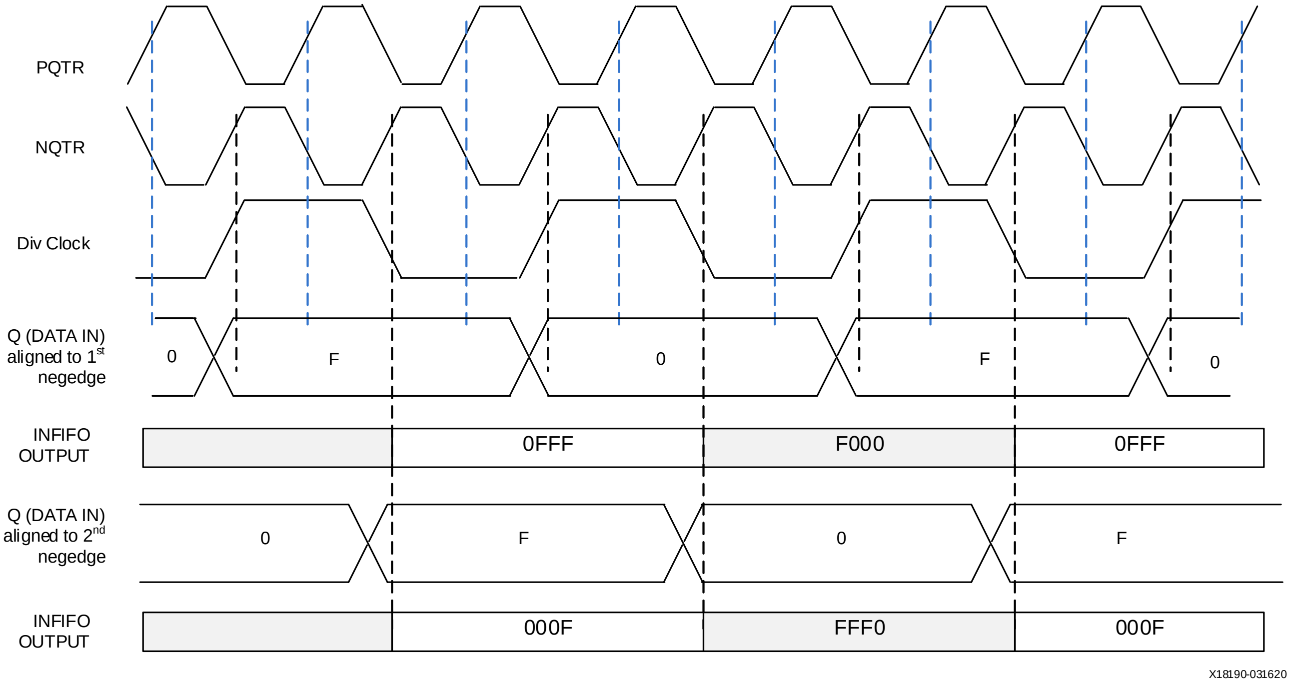


Figure 17-8: **N Data in Data Window**

For conditions in which the NQTR is outside the window for all the bits or any of the bits in the nibble, the PQTR/NQTR delays are incremented until there is a pass for all the bits. The next step would be to increment the IDELAY for each data bit in the nibble in Figure 17-9. This deskews all the bits in the nibble and NQTR will be aligned at the left edge for all the bits.

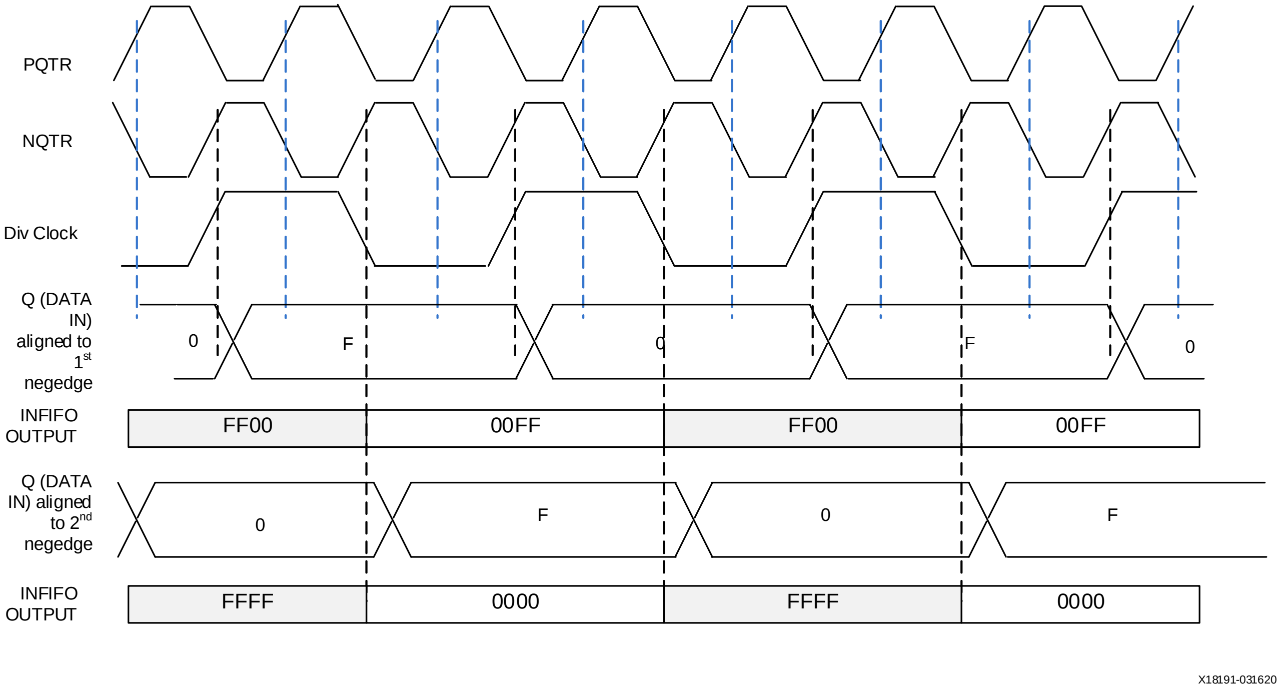


Figure 17-9: **NQTR Outside of Window**

Aligning PQTR to Right Edge

During this process both NQTR/PQTR delays are moved to find the right edge and only the P data is used for comparison. In this case since the deskew is already completed, the P data for any of the bits in a byte changes the right edge would be considered found.

Figure 17-10 shows the condition when the P data is aligned to the right edge.

The data is written into the INFIFO using the falling edge of the divided clock. The divided clock is derived from NQTR and it is being moved during the calibration stage.

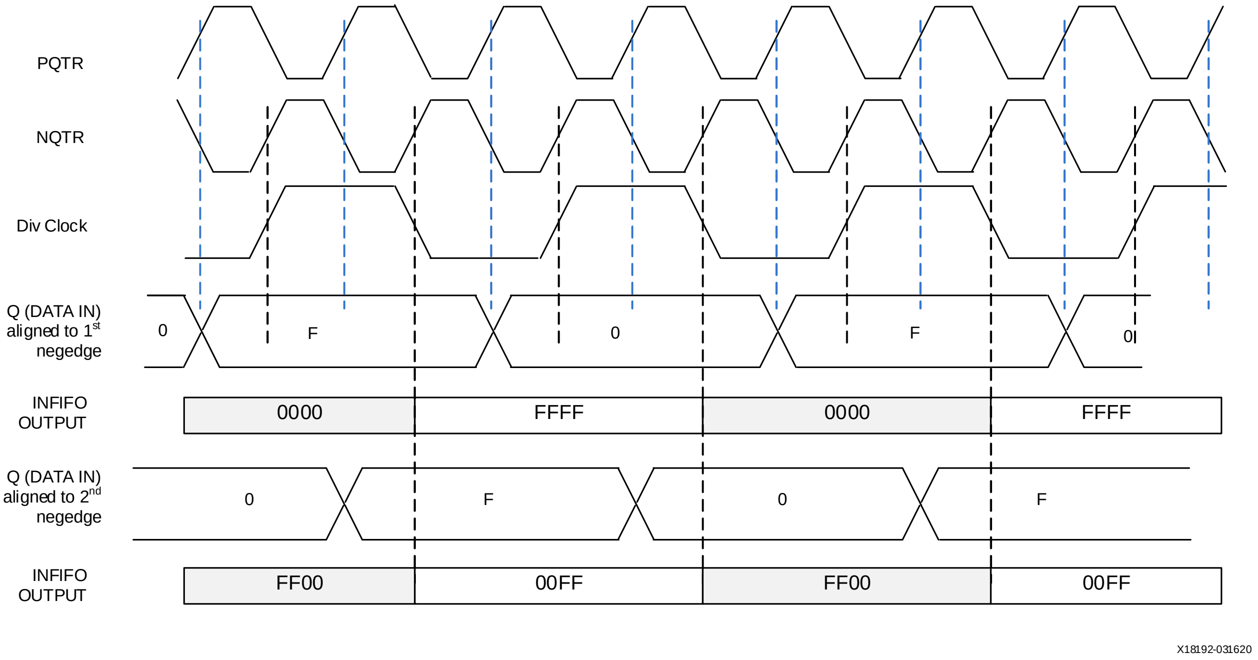


Figure 17-10: **PQTR Aligned to Right Edge**

Centering NQTR and PQTR

In this stage, the NQTR and PQTR values from the BISC calibration stage is used to center them in the data window. This is an initial calibration stage and read leveling with complex data pattern is used after the write calibration.

> PQTR_90 = PQTR values after BISC calibration - PQTR_ALIGN gives the tap count needed for the 90° offset
>
> NQTR_90 = NQTR values after BISC calibration - NQTR_ALIGN gives the tap count needed for the 90° offset

These values are retained as soon as the calibration algorithm starts. Now NQTR is placed at NQTR value at the end of "left edge alignment" + NQTR_90. PQTR is placed at PQTR value at the end of "right edge alignment" - PQTR_90.

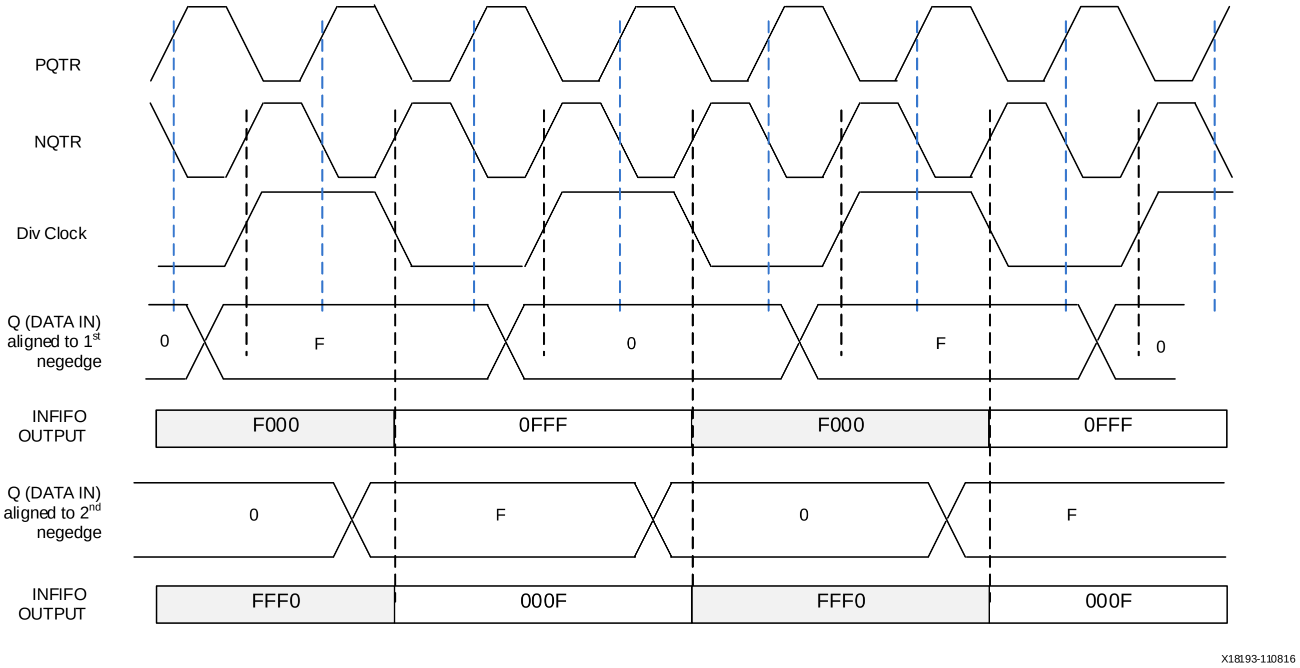


Figure 17-11: **NQTR and PQTR with 90° Offset from BISC**

Read Leveling Sanity Check

Sample write reads are performed to determine whether the read leveling calibration is successful.

Address Calibration (Enabled Only for BL2)

Address bits are DDR in BL2 SRAM parts while SDR in BL4 parts. It is done bitwise by only moving the ODELAY taps of the address bits. Memory clock K/K# is untouched throughout the calibration of QDR II+ IP. The calibration starts from A[0] until the last address bit. The algorithm is explained here by taking A[0] as an example:

1. To start, the algorithm writes one data burst each into two address locations.
   a. Writes `11111` (all 1s) into the address location `1111111` (all 1s).
   b. Writes `00000` (all 0s) into the address location `1111110` when calibrating A[0].
   c. When these two addresses are sent on the rise and fall edges of every memory clock cycle, it creates a clock pattern on A[0] and a constant 1 on all other address bits. The similar cycle repeats for all address bits.
   d. As you are writing the same data on all the edges of a write burst, you can keep the data free from any toggling. So, write calibration is not required at this stage.
   e. Continues read is started.

2. As there are no delay taps added on the address bits until this point of calibration, assume the initial relation between the memory clock and the address bit A[0] as shown in Figure 17-12.

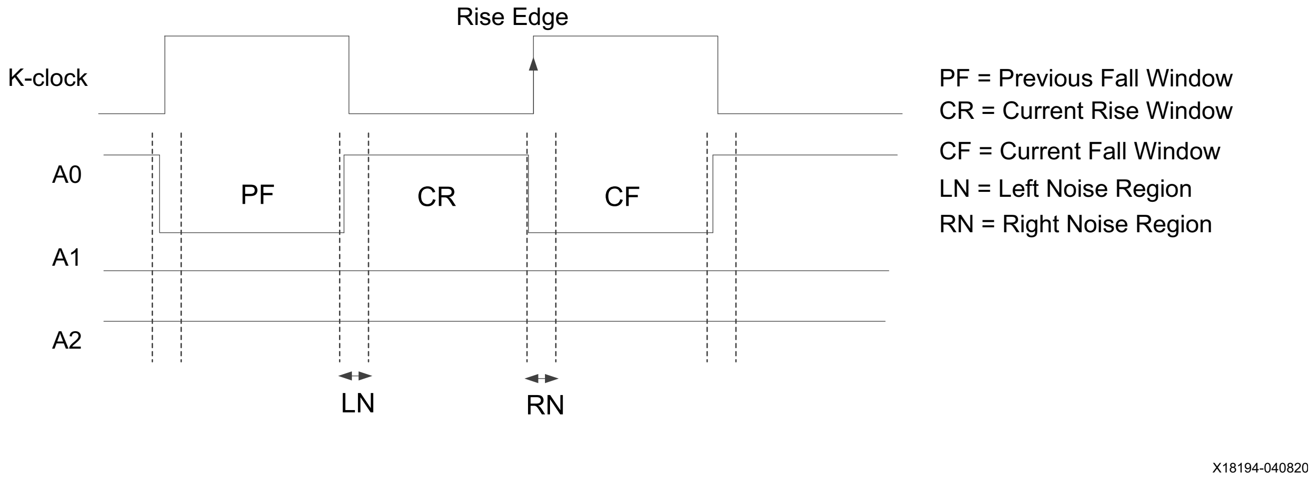


Figure 17-12: **Clock Approximately Edge Aligned with Address**

3. The position of the clock is found by reading the data pattern.
    a. As the address bit A[0] is continuously toggling, read commands are issued to either address `1111111` (all 1s) or `1111110` where there is a written known data pattern.
4. If the clock falls in the noise region or the previous fall window, first edge can be found. Else, the starting position is treated as the first edge. The same applies to the second edge as well.
5. The address bit is delayed by increasing its ODELAY taps until the next edge is found. At the end of this step, clock and address relation is shown in Figure 17-13.

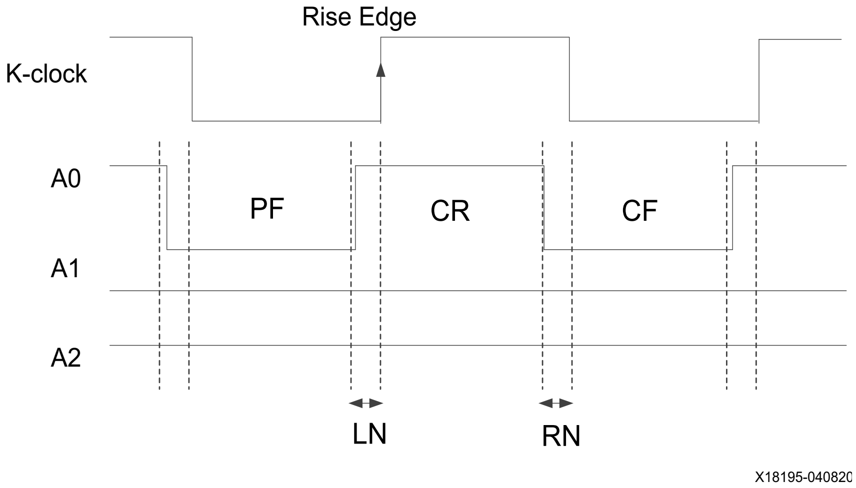


Figure 17-13: **Detecting Second Edge of Address Bit**

6. The first and second edge taps are noted and centered in the middle of them as shown in Figure 17-14. As the maximum frequency supported is 450 MHz, there are enough margin even if the algorithm is not able to find the first and second edges.

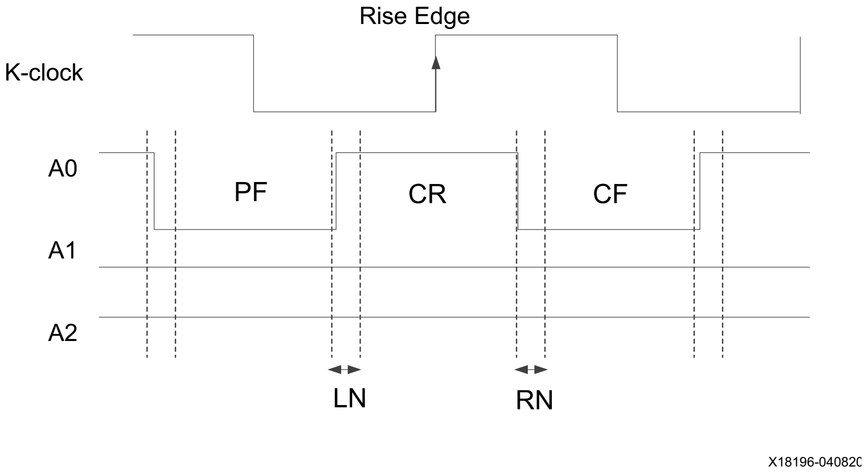


Figure 17-14: **Clock and Address Relation at End of A[0] Calibration**

Address Calibration Sanity Check (Enabled Only for BL2)

Sample write reads are performed to determine whether the address calibration is successful.

Write Data Centering

The purpose of this calibration stage is to center align the K-clock in the data window of every write data bit. Delay elements of the K-clock are untouched during the entire process of K/K# centering. Use only the ODELAYs of the write data bits. Only one bit is taken in every single iteration and it is delayed until two edges of a data window are found or taps elapse.

A static phase shift of 90° is applied on the K-clock all the time and thus, the initial position of the K-clock with respect to a write data bit is assumed to be one of the following three cases:

- **Case 1 –** Clock is aligned inside the valid window, which is termed in Figure 17-15 as Current Rise Window (CR) for a selected rise edge of the K-clock.

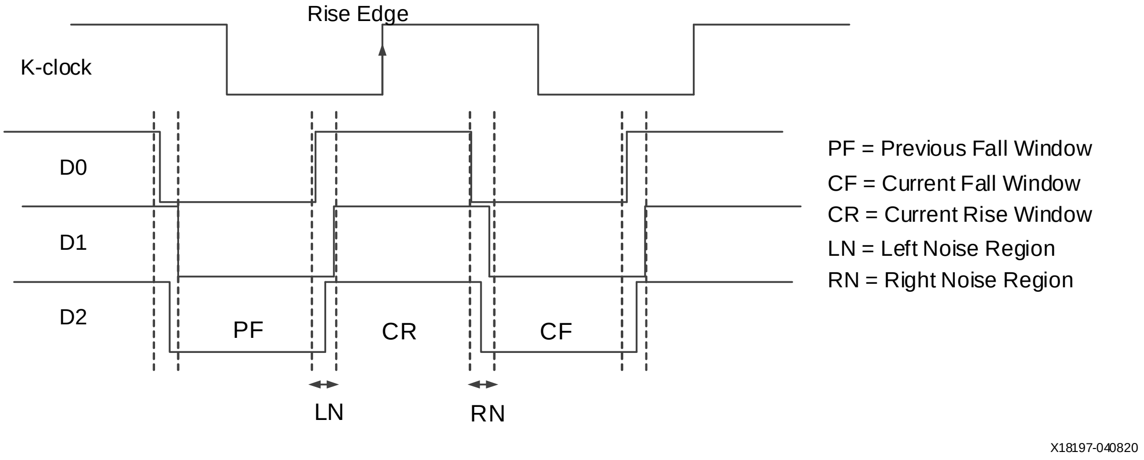


Figure 17-15: **Clock Aligned Inside Valid Window**

- **Case 2 –** Clock is aligned inside the Left Noise region (LN) as shown in Figure 17-16.

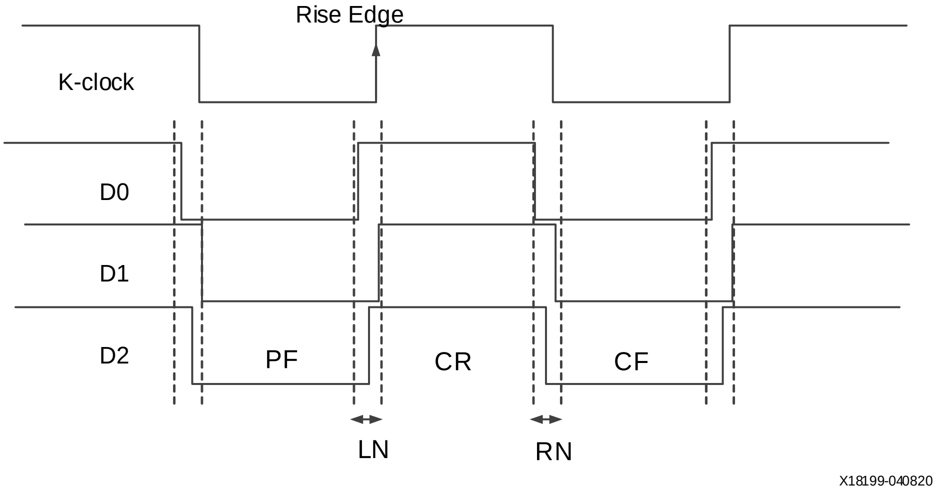
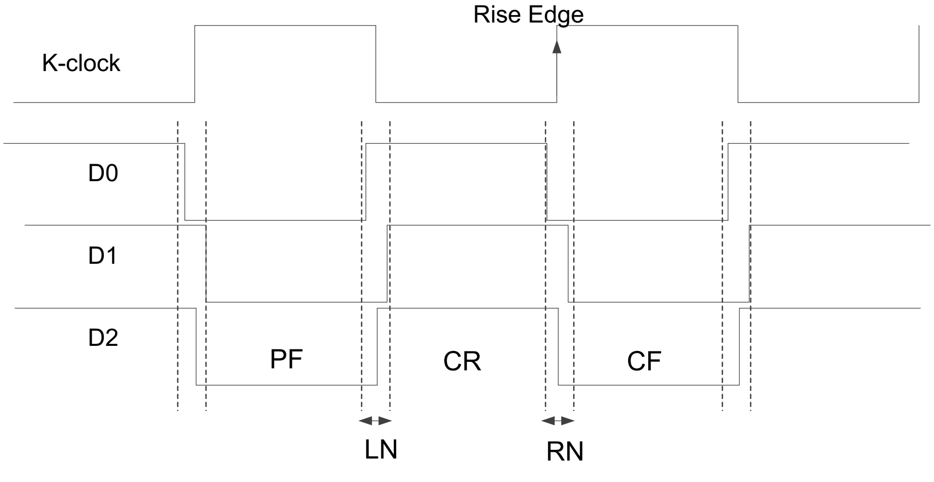


Figure 17-16: **Clock Aligned Inside Left Noise Region**

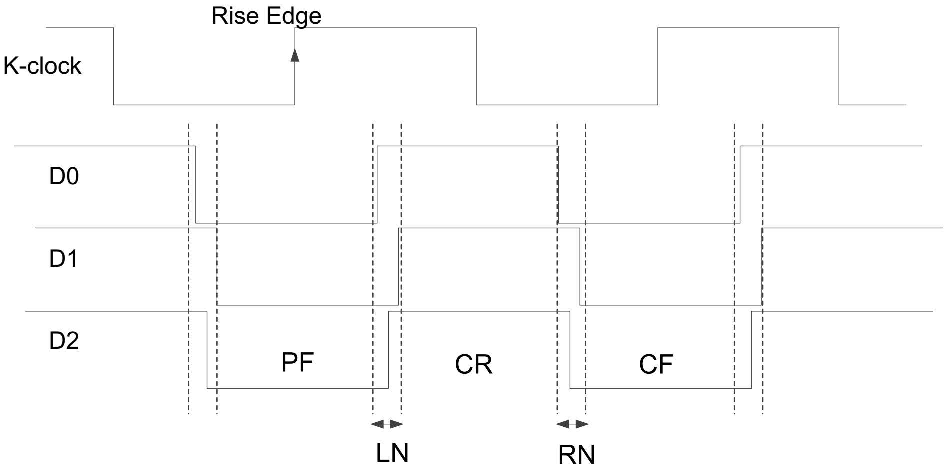
- **Case 3 –** Clock is aligned inside the Right Noise region (RN) as shown in Figure 17-17.



X18200-040820

Figure 17-17: **Clock Aligned Inside Right Noise Region**

When the initial placement of the clock with respect to a data bit is as mentioned in cases 1 and 2, you can only find the two edges of the previous fall window by moving the data delay taps. Therefore, you can only center in the previous fall window as shown in Figure 17-18. A separate bitslip stage is required to move the clock from previous fall to current rise window.



X18202-040820

Figure 17-18: **Clock Placement at End of K-Centering Stage in Cases 1 and 2**

However, the clock can be centered in proper data window (that is, current rise window) if the initial clock placement is as mentioned in case 3. The final placement is described in Figure 17-19. No bitslip is required in this scenario.

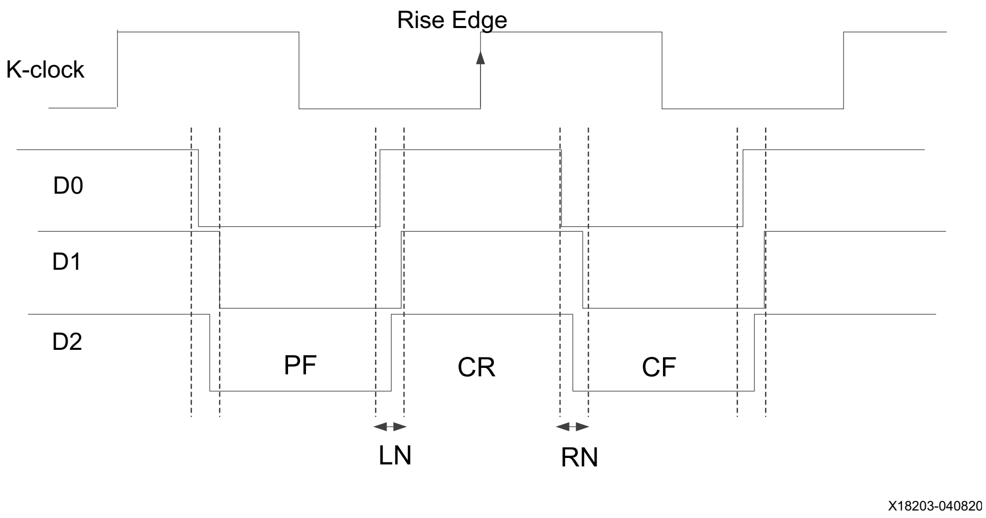


Figure 17-19: **Clock Placement at End of K-Centering Stage in Case 3**

The required immediate next step is to align the clock in the proper data window and bitslip calculation is done in the next stage of calibration.

Write Data Sanity Check

Sample write reads are performed to determine whether the write data calibration is successful.

Write Data Bitslip Calibration

Consider a scenario in which the write data bits stay in all the three categories that are explained previously in the K-centering stage, before the centering process starts. For example, take one bit from each category as shown in Figure 17-20.

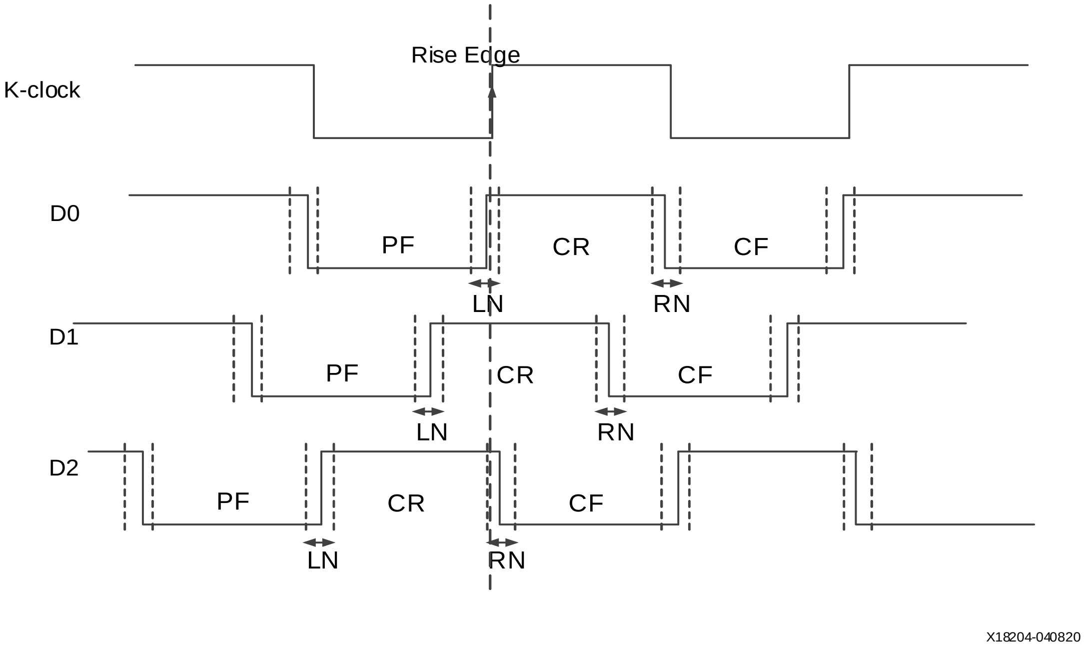


Figure 17-20: **Typical Clock Placement Inside Write Bus Before K-Centering**

After clock centering is completed, a few bits are centered in the previous fall window and the others, in the current rise window. Figure 17-21 shows how the data bits in Figure 17-20 is aligned after centering.

Figure 17-21 explains that clock placement for D2 is proper but it is improper for bits D0 and D1, which are delayed by one bit time. The only method to correct the clock alignment for D0 and D1 is to delay the address/control bits by the same number of bit times. However, one bit time cannot be added on address/control bits as they are SDR signals. Thus, delay the address/control bits by two bit times (one clock cycle) and delay D0 and D1 by one more bit time.

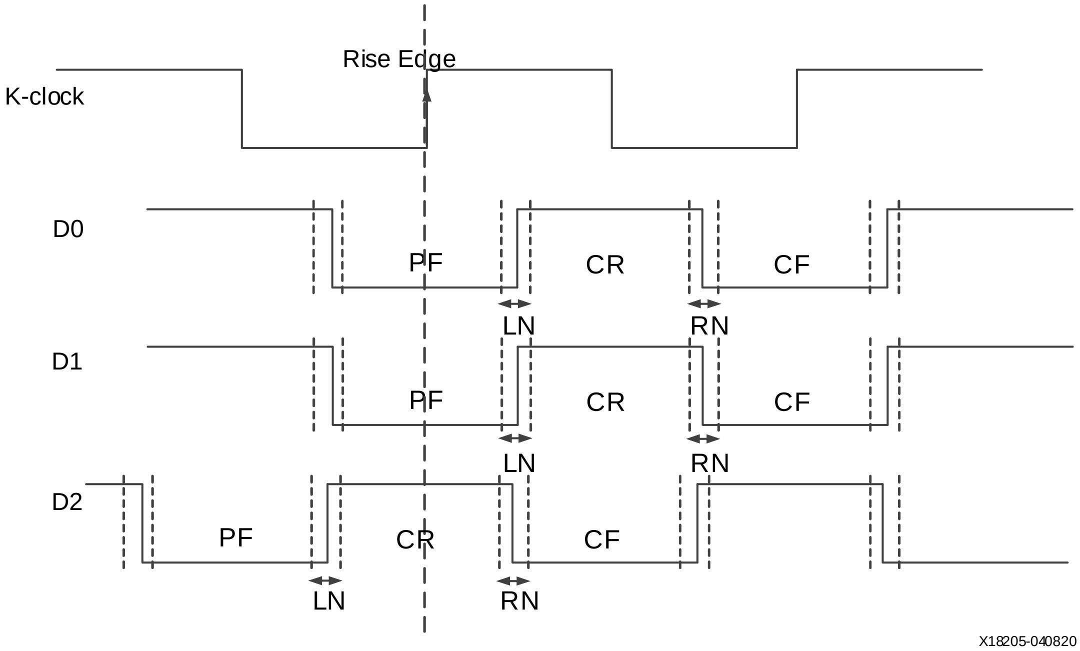


Figure 17-21: **Clock Placement After Centering**

The alignment in Figure 17-21 is modified to Figure 17-22 after adding one clock cycle delay.

Figure 17-22 explains that adding one bit time delay on D0 and D1 completes the clock alignment process. However, the data bit D2 is not done yet because it requires no bitslip before delaying the address/control bus. Therefore, it is required to add the same delay on D2 as that of address bus to complete its alignment. Figure 17-22 confirms the same as it takes two bit times (one clock cycle) for D2 to align the rise edge of the clock with CR (current rise window) of D2.

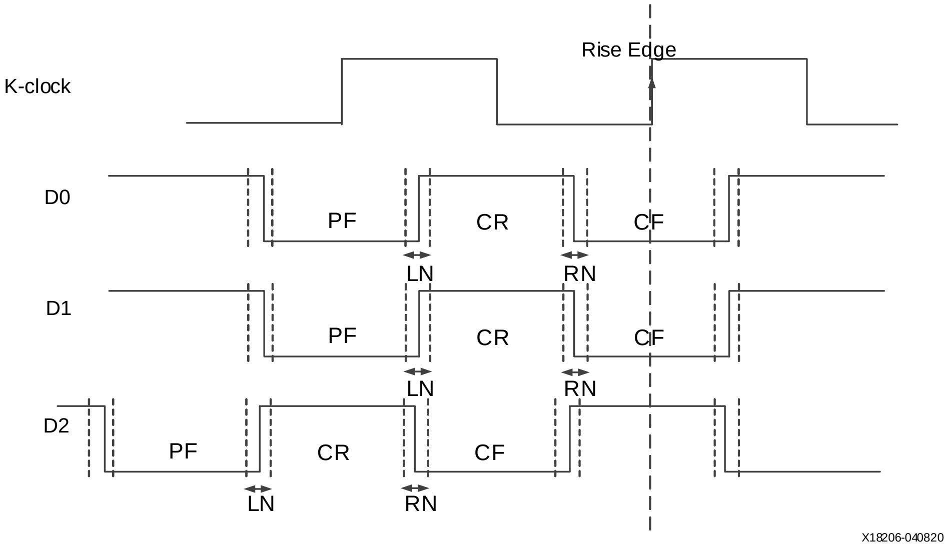


Figure 17-22: **Clock Placement After Adding One Clock Cycle Delay on Address/Control Bus**

Figure 17-23 shows the final alignment after adding corresponding delays on all data bits.

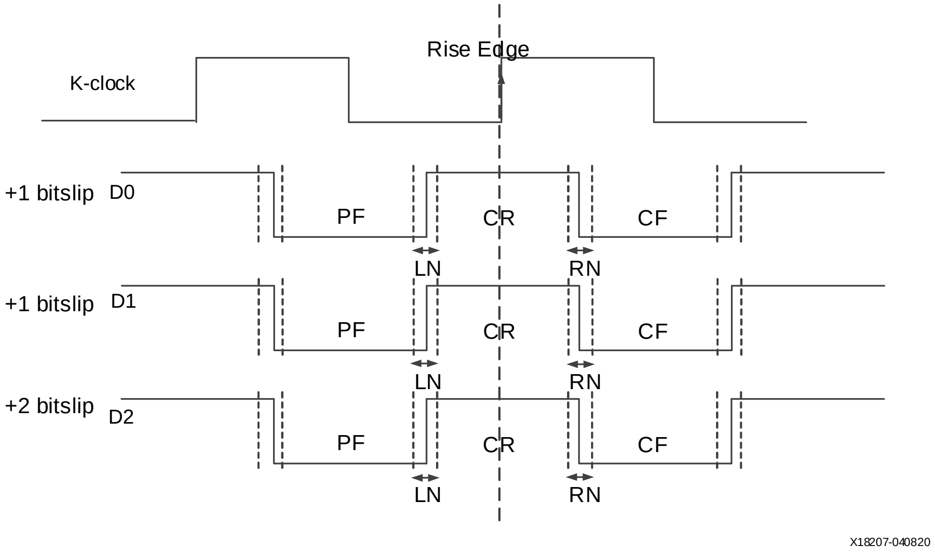


Figure 17-23: **Final Clock Placement at End of Write Bitslip Stage**

Read Bitslip Calibration

In this stage of calibration, specific data pattern is written to a set of address locations. The data is read back continuously to check for bitslip requirements. Bitslip on each read bit is incremented until it matches with the expected pattern. If the bitslip value on any bit is incremented to more than 2, an error is issued because maximum possible read bitslip is 2.

Byte Writes Centering

See the Write Data Centering. Clock pattern is used on write data bits in write data centering and the same can be used here for the byte write bits calibration as well.

Byte Writes Sanity Check

Sample write reads are performed to determine whether the byte writes centering is successful.

Byte Writes Bitslip

See the Write Data Bitslip Calibration.

Read Valid Calibration

In this stage of calibration, a specific data pattern is written to a set of address locations. The data is read back from the memory and the calibration algorithm looks for the expected data pattern to calculate the read latency. Based on the read latency the read valid signal is asserted for the read commands to validate the read data.

Read Valid Sanity Check

Sample write reads are performed to determine whether the read valid calibration is successful.

Reset Sequence

The `sys_rst` signal resets the entire memory design which includes general interconnect (fabric) logic which is driven by the MMCM clock (clkout0) and RIU logic. MicroBlaze™ and calibration logic are driven by the MMCM clock (clkout6). The `sys_rst` input signal is synchronized internally to create the `qdriip_rst_clk` signal. The `qdriip_rst_clk` reset signal is synchronously asserted and synchronously deasserted.

Figure 17-24 shows the `qdriip_rst_clk` (fabric reset) is synchronously asserted with a few clock delays after `sys_rst` is asserted. When `qdriip_rst_clk` is asserted, there are a few clocks before the clocks are shut off.

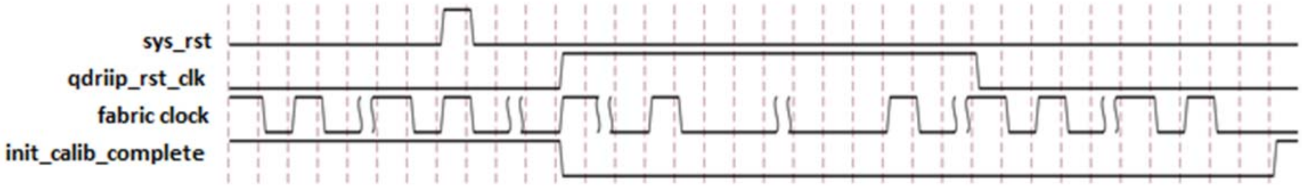


Figure 17-24: **Reset Sequence Waveform**

The following are the reset sequencing steps:

1. Reset to design is initiated after `qdriip_rst_clk` goes High.
2. `init_calib_complete` signal goes Low when `qdriip_rst_clk` is High.
3. Reset to design is deactivated after `qdriip_rst_clk` is Low.
4. After `qdriip_rst_clk` is deactivated, the `init_calib_complete` is asserted after calibration is completed.

MicroBlaze MCS ECC

The MicroBlaze MCS local memory provides an option to enable Error Correcting Code (ECC). Error correction corrects single bit errors and detects double bit errors. Two additional ports are added to indicate single bit errors (LMB_CE) and double bit errors (LMB_UE).

The MicroBlaze MCS ECC can be selected from the **MicroBlaze MCS ECC option** section in the **Advanced Options** tab. The block RAM size increases if the ECC option for MicroBlaze MCS is selected.

Chapter 18

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The memory interface requires one mixed-mode clock manager (MMCM), one TXPLL per I/O bank used by the memory interface, and two BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used.

Note: QDR II+ SRAM generates the appropriate clocking structure and no modifications to the RTL are supported.

The QDR II+ SRAM tool generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to MMCM (located in center bank of memory interface)
- MMCM to BUFG (located at center bank of memory interface) driving FPGA logic and all TXPLLs
- MMCM to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate FPGA logic
- Clocking pair of the interface must be in the same SLR of memory interface for the stacked silicon interconnect (SSI) technology devices

Requirements

GCIO

- Must use a differential I/O standard
- Must be in the same I/O column as the memory interface
- Must be in the same SLR of memory interface for the SSI technology devices
- The I/O standard and termination scheme are system dependent. For more information, consult the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 7].

MMCM

- MMCM is used to generate the FPGA logic system clock (1/2 of the memory clock)
- Must be located in the center bank of memory interface
- Must use internal feedback
- Input clock frequency divided by input divider must be ≥ 70 MHz (CLKINx / D ≥ 70 MHz)
- Must use integer multiply and output divide values

Input Clock Requirement

- The clock generator driving the GCIO should have jitter < 3 ps RMS.
- The input clock should always be clean and stable. The IP functionality is not guaranteed if this input system clock has a glitch, discontinuous, etc.
- No spread spectrum clock is allowed.

BUFGs and Clock Roots

- One BUFG is used to generate the system clock to FPGA logic and another BUFG is used to divide the system clock by two.
- BUFGs and clock roots must be located in center most bank of the memory interface.
    - For two bank systems, the bank with the higher number of bytes selected is chosen as the center bank. If the same number of bytes is selected in two banks, then the top bank is chosen as the center bank.
    - Both the BUFGs must be in the same bank.

TXPLL

- CLKOUTPHY from TXPLL drives XIPHY within its bank

- TXPLL must be set to use a CLKFBOUT phase shift of 90°
- TXPLL must be held in reset until the MMCM lock output goes High
- Must use internal feedback

Figure 18-1 shows an example of the clocking structure for a three bank memory interface. The GCIO drives the MMCM located at the center bank of the memory interface. MMCM drives both the BUFGs located in the same bank. The BUFG (which is used to generate system clock to FPGA logic) output drives the TXPLLs used in each bank of the interface.

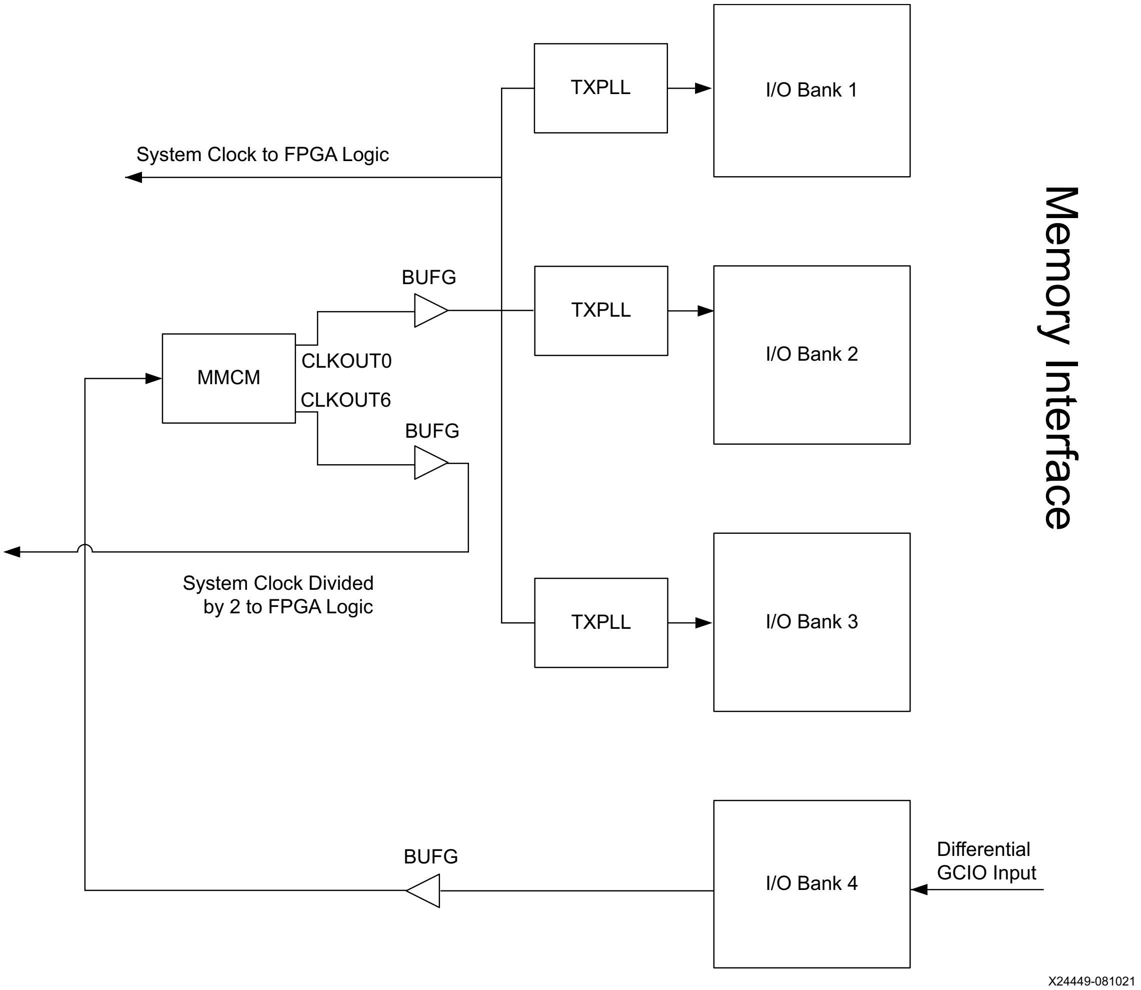


Figure 18-1: **Clocking Structure for Three Bank Memory Interface**

The MMCM is placed in the center bank of the memory interface.

- For two bank systems, MMCM is placed in a bank with the most number of bytes selected. If they both have the same number of bytes selected in two banks, then MMCM is placed in the top bank.
- For four bank systems, MMCM is placed in a second bank from the top.

For designs generated with System Clock configuration of **No Buffer**, MMCM must not be driven by another MMCM/PLL. Cascading clocking structures MMCM → BUFG → MMCM and PLL → BUFG → MMCM are not allowed.

If the MMCM is driven by the GCIO pin of the other bank, then the CLOCK_DEDICATED_ROUTE constraint with value "BACKBONE" must be set on the net that is driving MMCM or on the MMCM input. Setting up the CLOCK_DEDICATED_ROUTE constraint on the net is preferred. But when the same net is driving two MMCMs, the CLOCK_DEDICATED_ROUTE constraint must be managed by considering which MMCM needs the BACKBONE route.

In such cases, the CLOCK_DEDICATED_ROUTE constraint can be set on the MMCM input. To use the "BACKBONE" route, any clock buffer that exists in the same CMT tile as the GCIO must exist between the GCIO and MMCM input. The clock buffers that exists in the I/O CMT are BUFG, BUFGCE, BUFGCTRL, and BUFGCE_DIV. So QDR II+ SRAM instantiates BUFG between the GCIO and MMCM when the GCIO pins and MMCM are not in the same bank (see Figure 18-1).

If the GCIO pin and MMCM are allocated in different banks, QDR II+ SRAM generates CLOCK_DEDICATED_ROUTE constraints with value as "BACKBONE." If the GCIO pin and MMCM are allocated in the same bank, there is no need to set any constraints on the MMCM input.

Similarly when designs are generated with System Clock Configuration as a **No Buffer** option, you must take care of the "BACKBONE" constraint and the BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM if GCIO pin and MMCM are allocated in different banks. QDR II+ SRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations. For more information on clocking, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

XDC syntax for CLOCK_DEDICATED_ROUTE constraint is given here:

```
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hier -filter {NAME =~ */
u_qdriip_infrastructure/gen_mmcme*.u_mmcme_adv_inst/CLKIN1}]
```

For more information on the CLOCK_DEDICATED_ROUTE constraints, see the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 9].

Note: If two different GCIO pins are used for two QDR II+ SRAM IP cores in the same bank, center bank of the memory interface is different for each IP. QDR II+ SRAM generates MMCM LOC and CLOCK_DEDICATED_ROUTE constraints accordingly.

Sharing of Input Clock Source (sys_clk_p)

If the same GCIO pin must be used for two IP cores, generate the two IP cores with the same frequency value selected for option **Reference Input Clock Period (ps)** and **System Clock Configuration** option as **No Buffer**. Perform the following changes in the wrapper file in which both IPs are instantiated:

1. QDR II+ SRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) of both the IP cores.
2. System clock pins must be allocated within the same I/O column of the memory interface pins allocated. Add the pin LOC constraints for system clock pins and clock constraints in your top-level XDC.
3. You must add a "BACKBONE" constraint on the net that is driving the MMCM or on the MMCM input if GCIO pin and MMCM are not allocated in the same bank. Apart from this, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM to use the "BACKBONE" route.

Note:

- The UltraScale architecture includes an independent XIPHY power supply and TXPLL for each XIPHY. This results in clean, low jitter clocks for the memory system.
- Skew spanning across multiple BUFGs is not a concern because single point of contact exists between BUFG → TXPLL and the same BUFG → System Clock Logic.
- System input clock cannot span I/O columns because the longer the clock lines span, the more jitter is picked up.

TXPLL Usage

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used. One PLL per bank is used if a bank is used by a single memory interface. You can use a second PLL for other usage. To use a second PLL, you can perform the following steps:

1. Generate the design for the **System Clock Configuration** option as **No Buffer**.
2. QDR II+ SRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) and also to the input of PLL (PLL instance that you have in your design).
3. You can use the PLL output clocks.

Additional Clocks

You can produce up to four additional clocks which are created from the same MMCM that generates `ui_clk`. Additional clocks can be selected from the **Clock Options** section in the **Advanced Options** tab. The GUI lists the possible clock frequencies from MMCM and the frequencies for additional clocks vary based on selected memory frequency (**Memory Device Interface Speed (ps)** value in the **Basic** tab), selected FPGA, and FPGA speed grade.

Reduce System Noise during Calibration

The system design should be as quiet as possible during the calibration process. In particular, the Soft Error Mitigation (SEM) IP, if used, should be disabled during calibration. For calibration that occurs immediately after the configuration or reconfiguration of the FPGA, use the ICAP arbitration interface to hold off the SEM IP in the boot stage. For more information on the ICAP Arbitration Interface, see "ICAP Arbitration Interface" section in Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10].

For situations where the memory interface is reset and recalibrated without a reconfiguration of the FPGA, the SEM IP must be set into IDLE state to disable the memory scan and to send the SEM IP back into the scanning (Observation or Detect only) states afterwards. This can be done in two methods, through the "Command Interface" or "UART interface." See Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10] for more information.

Resets

An asynchronous reset (`sys_rst`) input is provided. This is an active-High reset and the `sys_rst` must assert for a minimum pulse width of 5 ns. The `sys_rst` can be an internal or external pin.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must have a common reset input.*

For more information on reset, see the Reset Sequence in Chapter 17, Core Architecture.

PCB Guidelines for QDR II+ SRAM

Strict adherence to all documented QDR II+ SRAM PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

Pin and Bank Rules

QDR II+ Pin Rules

This section describes the pin out rules for QDR II+ SRAM interface.

- Both HR and HP Banks are supported.
- All signal groups that are write data, read data, address/control, and system clock interfaces must be selected in a single column.
- All banks used must be adjacent. No skip banks allowed.

1. Write Data (D) and Byte Write (BW) Pins Allocation:
    a. The entire write data bus must be placed in a single bank regardless of the number of memory components.
    b. Only one write data byte is allowed per byte lane.
    c. All byte lanes that are used for the write data of a single component must be adjacent, no skip byte lanes are allowed.
    d. One of the write data bytes of a memory component should be allocated in the center byte lanes (byte lanes 1 and 2).
    e. Each byte write pin (BW) must be allocated in the corresponding write data byte lane.
2. Memory Clock (K/K#) Allocation:
    a. Memory Clock pair must be allocated in one of the byte lanes that are used for the write data of the corresponding memory component.
    b. Memory clock should come from one of the center byte lanes (byte lanes 1 and 2).
    c. K/K# can be allocated to any PN pair.
3. Read Data (Q) Allocation:
    a. The entire read data bus must be placed in a single bank irrespective of the number of memory components.
    b. All byte lanes that are used for the read data of a single component must be adjacent, no skip byte lanes are allowed.
    c. One of the read data bytes of a memory component should be allocated in the center byte lanes (byte lanes 1 and 2).
    d. If a byte lane is used for read data, Bit[0] and Bit[6] must be used. Read clock (CQ or CQ#) gets the first priority and data (Q) is the next.
    e. Read data buses of two components should not share a byte lane.

4. Read Clock (CQ/CQ#) Allocation:
   a. Read Clock pair must be allocated in one of the byte lanes that are used for the read data of the corresponding memory component.
   b. CQ/CQ# pair must be allocated in a single byte lane.
   c. CQ/CQ# must be allocated only in the byte lanes 1 and 2 because the other byte lanes cannot forward the clock out for read data capture.
   d. CQ/CQ# must be allocated in the center byte lane of all used byte lanes. If two byte lanes are used for the read data, either one of them can be used for CQ/CQ# allocation.
   e. CQ and CQ# must be allocated to either pin 0 or pin 6 of a byte lane. For example, if CQ is allocated to pin 0, CQ# should be allocated to pin 6 and vice versa.
5. For x36 and x18 component designs:

   All Read Data pins of a single component must not span more than three consecutive byte lanes and CQ/CQ# must always be allocated in center byte lane.
6. Address/Control (A/C) Pins Allocation:
   a. All address/control (A/C) bits must be allocated in a single bank.
   b. All A/C byte lanes should be contiguous and no skip byte lanes is allowed.
   c. The address/control bank should be the same or adjacent to that of the write data bank.
   d. There should not be any empty byte lane or read byte lane between A/C and write data byte lanes. This rule applies when A/C and write data share the same bank or allocated in adjacent banks.
   e. Address/control pins should not share a byte lane with the write data as well as read data.
   f. System clock pins (**`sys_clk_p`**/**`sys_clk_n`**) must be placed on any GCIO pin pair in the same column as that of the memory interface. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements and MMCM requirements should be considered). For more information, see Clocking, page 359.
7. All I/O banks used by the memory interface must be in the same SLR of the column for the SSI technology devices.
8. One **`vrp`** pin per bank is used and DCI is required for the interfaces. A **`vrp`** pin is required in I/O banks containing inputs as well as output only banks. It is required in output only banks because address/control signals use HSTL_I_DCI to enable usage of controlled output impedance. DCI cascade is allowed. When DCI cascade is selected, **`vrp`** pin can be used as a normal I/O. All rules for the DCI in the *UltraScale™ Device FPGAs SelectIO™ Resources User Guide* (UG571) [Ref 7] must be followed.

RECOMMENDED: *Xilinx strongly recommends that the DCIUpdateMode option is kept with the default value of ASREQUIRED so that the DCI circuitry is allowed to operate normally.*

9. There are dedicated V_{REF} pins (not included in the rules above). Either internal or external V_{REF} is permitted. If an external V_{REF} is not used, the V_{REF} pins must be pulled to ground by a resistor value specified in the *UltraScale™ Device FPGAs SelectIO™ Resources User Guide* (UG571) [Ref 7]. These pins must be connected appropriately for the standard in use.
10. The system reset pin (`sys_rst_n`) must not be allocated to Pins N0 and N6 if the byte is used for the memory I/Os.

Pin Swapping

- Pins can swap freely within each Write Data byte group.
- Pins can swap freely within each Read Data byte group, except CQ/CQ# pins. Pins can swap freely within and between their corresponding byte groups, but should not violate above mentioned Read Data pin/byte lane allocation rules.
- Pins can swap freely withing each Address/Control byte group. Pins can swap freely within and between their corresponding byte groups, but should not violate above mentioned Address/Control pin/byte lane allocation rules.
- Write Data Byte groups can swap easily with each other, but should not violate above mentioned Write Data pin/byte lane allocation rules.
- Read Data Byte groups can swap easily with each other, but should not violate above mentioned Read Data pin/byte lane allocation rules.
- Address/Control Byte groups can swap easily with each other, but should not violate above mentioned Address/Control pin/byte lane allocation rules.
- No other pin swapping is permitted.

QDR II+ Pinout Examples

IMPORTANT: *Due to the calibration stage, there is no need for* `set_input_delay/ set_output_delay` *on the QDR II+ SRAM. Ignore the unconstrained inputs and outputs for QDR II+ SRAM and the signals which are calibrated.*

Table 18-1 shows an example of an 18-bit QDR II+ SRAM interface contained within two banks.

Table 18-1: **18-Bit QDR II+ Interface Contained in Two Banks**

Bank	Signal Name	Byte Group	I/O Type
1	–	T1U_12	–
1	sys_clk_p	T1U_11	N

Table 18-1: **18-Bit QDR II+ Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
1	sys_clk_n	T1U_10	P
1	–	T1U_9	N
1	q17	T1U_8	P
1	q16	T1U_7	N
1	cq_p	T1U_6	P
1	q15	T1L_5	N
1	q14	T1L_4	P
1	q13	T1L_3	N
1	q12	T1L_2	P
1	q11	T1L_1	N
1	cq_n	T1L_0	P
1	vrp	T0U_12	–
1	–	T0U_11	N
1	q10	T0U_10	P
1	q9	T0U_9	N
1	q8	T0U_8	P
1	q7	T0U_7	N
1	q6	T0U_6	P
1	q5	T0L_5	N
1	q4	T0L_4	P
1	q3	T0L_3	N
1	q2	T0L_2	P
1	q1	T0L_1	N
1	q0	T0L_0	P
0	–	T3U_12	–
0	–	T3U_11	N
0	–	T3U_10	P
0	d17	T3U_9	N
0	d16	T3U_8	P
0	d15	T3U_7	N
0	d14	T3U_6	P
0	d13	T3L_5	N
0	d12	T3L_4	P

Table 18-1: **18-Bit QDR II+ Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
0	d11	T3L_3	N
0	d10	T3L_2	P
0	bwsn1	T3L_1	N
0	d9	T3L_0	P
0	–	T2U_12	–
0	d8	T2U_11	N
0	d7	T2U_10	P
0	d6	T2U_9	N
0	d5	T2U_8	P
0	k_n	T2U_7	N
0	k_p	T2U_6	P
0	d4	T2L_5	N
0	d3	T2L_4	P
0	d2	T2L_3	N
0	d1	T2L_2	P
0	bwsn0	T2L_1	N
0	d0	T2L_0	P
0	doff	T1U_12	–
0	a21	T1U_11	N
0	a20	T1U_10	P
0	a19	T1U_9	N
0	a18	T1U_8	P
0	a17	T1U_7	N
0	a16	T1U_6	P
0	a15	T1L_5	N
0	a14	T1L_4	P
0	a13	T1L_3	N
0	a12	T1L_2	P
0	rpsn	T1L_1	N
0	a11	T1L_0	P
0	vrp	T0U_12	–
0	a10	T0U_11	N

Table 18-1: **18-Bit QDR II+ Interface Contained in Two Banks *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type
0	a9	T0U_10	P
0	a8	T0U_9	N
0	a7	T0U_8	P
0	a6	T0U_7	N
0	a5	T0U_6	P
0	a4	T0L_5	N
0	a3	T0L_4	P
0	a2	T0L_3	N
0	a1	T0L_2	P
0	wpsn	T0L_1	N
0	a0	T0L_0	P

Protocol Description

This core has the following interfaces:

- User Interface
- Memory Interface

User Interface

The user interface connects an FPGA user design to the QDR II+ SRAM solutions core to simplify interactions between the user logic and the external memory device. The user interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in Table 18-2.

Table 18-2: **User Interface Signals**

Signal	I/O	Description
app_rd_addr0[ADDR_WIDTH – 1:0]	I	Read Address. This bus provides the address to use for a read request. It is valid when app_rd_cmd0 is asserted.
app_rd_cmd0	I	Read Command. This signal is used to issue a read request and indicates that the address on port0 is valid.
app_rd_data0[DBITS × BURST_LEN – 1:0]	O	Read Data. This bus carries the data read back from the read command issued on app_rd_cmd0

Table 18-2: **User Interface Signals *(Cont'd)***

Signal	I/O	Description
app_rd_valid0	O	Read Valid. This signal indicates that data read back from memory is now available on app_rd_data0 and should be sampled.
app_wr_addr0[ADDR_WIDTH – 1:0]	I	Write Address. This bus provides the address for a write request. It is valid when app_wr_cmd0 is asserted.
app_wr_bw_n0[(DBITS/9) × BURST_LEN – 1:0]	I	Byte Writes. This bus provides the byte writes for a write request and indicates which bytes need to be written into the SRAM. It is valid when app_wr_cmd0 is asserted and is active-Low.
app_wr_cmd0	I	Write Command. This signal is used to issue a write request and indicates that the corresponding sideband signals on write port0 are valid.
app_wr_data0[DBITS × BURST_LEN – 1:0]	I	Write Data. This bus provides the data to use for a write request. It is valid when app_wr_cmd0 is asserted.
app_rd_addr1[ADDR_WIDTH – 1:0](1)	I	Read Address. This bus provides the address to use for a read request. It is valid when app_rd_cmd1 is asserted.
app_rd_cmd1(1)	I	Read Command. This signal is used to issue a read request and indicates that the address on port1 is valid.
app_rd_data1[DBITS × BURST_LEN – 1:0](1)	O	Read Data. This bus carries the data read back from the read command issued on app_rd_cmd1.
app_rd_valid1(1)	O	Read Valid. This signal indicates that data read back from memory is now available on app_rd_data1 and should be sampled.
app_wr_addr1[ADDR_WIDTH – 1:0](1)	I	Write Address. This bus provides the address for a write request. It is valid when app_wr_cmd1 is asserted.
app_wr_bw_n1[(DBITS/9) × BURST_LEN – 1:0](1)	I	Byte Writes. This bus provides the byte writes for a write request and indicates which bytes need to be written into the SRAM. It is valid when app_wr_cmd1 is asserted and is active-Low.
app_wr_cmd1(1)	I	Write Command. This signal is used to issue a write request and indicates that the corresponding sideband signals on write port1 are valid.
app_wr_data1[DBITS × BURST_LEN – 1:0](1)	I	Write Data. This bus provides the data to use for a write request. It is valid when app_wr_cmd1 is asserted.
clk	O	User Interface clock.
rst_clk	O	Reset signal synchronized by the User Interface clock.
Init_calib_complete	O	Calibration Done. This signal indicates to the user design that read calibration is complete and the user can now initiate read and write requests from the client interface.
sys_rst	I	Asynchronous system reset input.
sys_clk_p/n	I	System clock to the Memory Controller.

Table 18-2: **User Interface Signals *(Cont'd)***

Signal	I/O	Description
dbg_clk	O	Debug Clock. Do not connect any signals to dbg_clk and keep the port open during instantiation.
dbg_bus	O	Reserved. Do not connect any signals to dbg_bus and keep the port open during instantiation.

Notes:

1. These ports are available and valid only in BL2 configuration. For BL4 configuration, these ports are not available or if available, no need to be driven.

Interfacing with the Core through the User Interface

Figure 18-2 shows the user interface protocol.

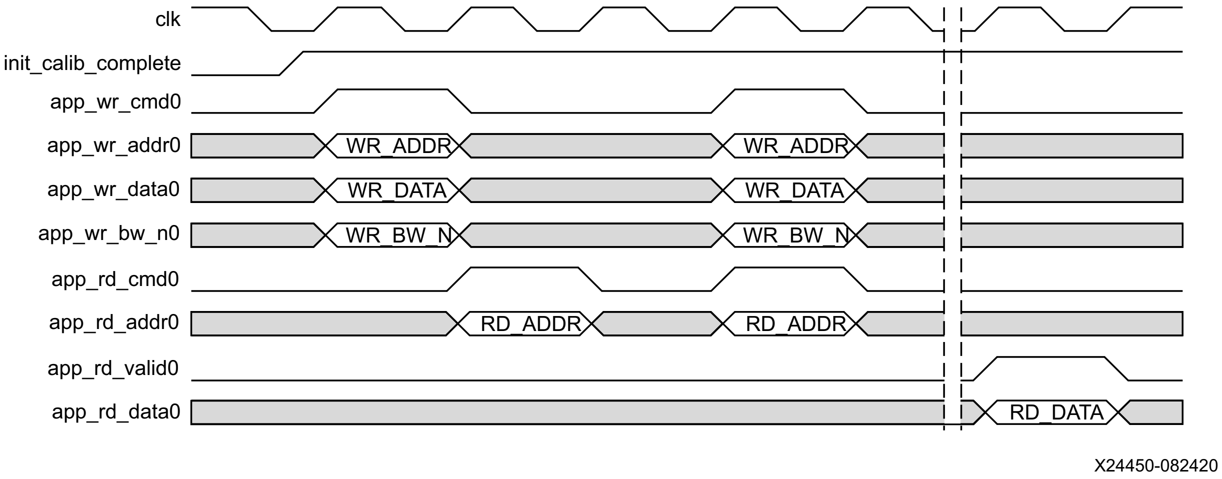


Figure 18-2: **User Interface Write/Read Timing Diagram**

Before any requests can be made, the **`init_calib_complete`** signal must be asserted High, as shown in Figure 18-2, no read or write requests can take place, and the assertion of **`app_wr_cmd0`** or **`app_rd_cmd0`** on the client interface is ignored. A write request is issued by asserting **`app_wr_cmd0`** as a single cycle pulse. At this time, the **`app_wr_addr0`**, **`app_wr_data0`**, and **`app_wr_bw_n0`** signals must be valid.

On the following cycle, a read request is issued by asserting **`app_rd_cmd0`** for a single cycle pulse. At this time, **`app_rd_addr0`** must be valid. After one cycle of idle time, a read and write request are both asserted on the same clock cycle. In this case, the read to the memory occurs first, followed by the write. The write and read commands can be applied in any order at the user interface, two examples are shown in the Figure 18-2.

Also, Figure 18-2 shows data returning from the memory device to the user design. The **`app_rd_valid0`** signal is asserted, indicating that **`app_rd_data0`** is now valid. This should be sampled on the same cycle when **`app_rd_valid0`** is asserted because the core does not buffer returning data.

In case of BL2, the same protocol should be followed on two independent ports: port-0 and port-1. Figure 18-2 shows the user interface signals on port-0 only.

Memory Interface

Memory interface is a connection from the FPGA memory solution to an external QDR II+ SRAM device. The I/O signals for this interface are defined in Table 18-3. These signals can be directly connected to the corresponding signals on the memory device.

Table 18-3: **Memory Interface Signals**

Signal	I/O	Description
qdriip_cq_n	I	QDR CQ#. This is the echo clock returned from the memory derived from qdr_k_n.
qdriip_cq_p	I	QDR CQ. This is the echo clock returned from the memory derived from qdr_k_p.
qdriip_d	O	QDR Data. This is the write data from the PHY to the QDR II+ memory device.
qdriip_doff_n	O	QDR DLL Off. This signal turns off the DLL in the memory device.
qdriip_bw_n	O	QDR Byte Write. This is the byte write signal from the PHY to the QDR II+ SRAM device.
qdriip_k_n	O	QDR Clock K#. This is the inverted input clock to the memory device.
qdriip_k_p	O	QDR Clock K. This is the input clock to the memory device.
qdriip_q	I	QDR Data Q. This is the data returned from reads to memory.
qdriip_qvld	I	QDR Q Valid. This signal indicates that the data on qdriip_q is valid. However, the QDR II+ core is not using this port to validate the data from the SRAM device. Instead, the core calibrates read latency for valid read data.
qdriip_sa	O	QDR Address. This is the address supplied for memory operations.
qdriip_w_n	O	QDR Write. This is the write command to memory.
qdriip_r_n	O	QDR Read. This is the read command to memory.

Figure 18-3 shows the timing diagram for the sample write and read operations at the memory interface of a BL4 QDR II+ SRAM device and Figure 18-4 is that of a BL2 device.

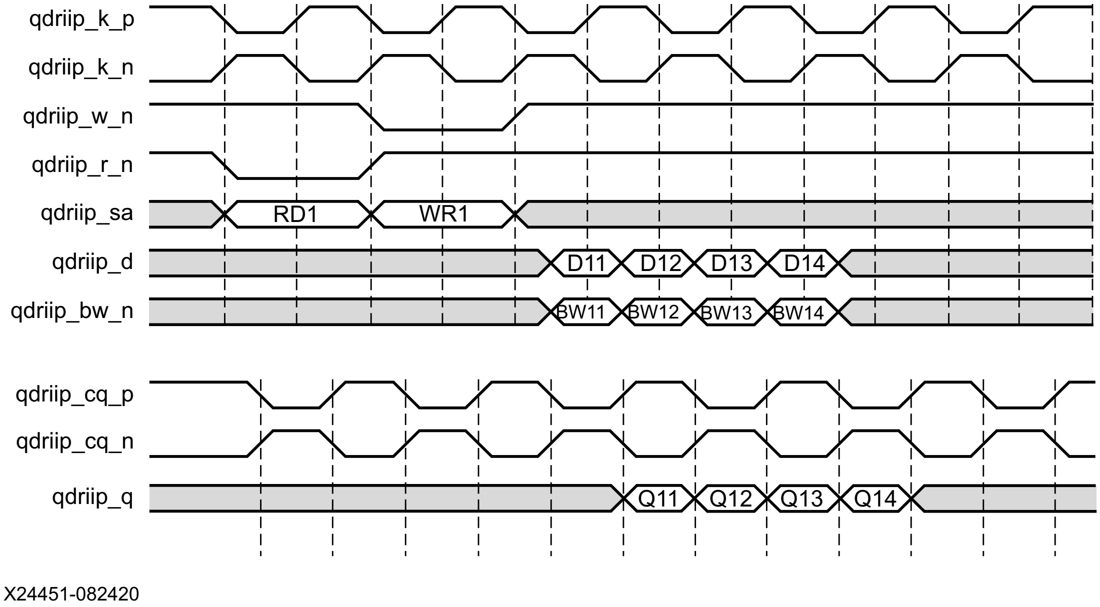


Figure 18-3: **Interfacing with a Four-Word Burst Length Memory Device**

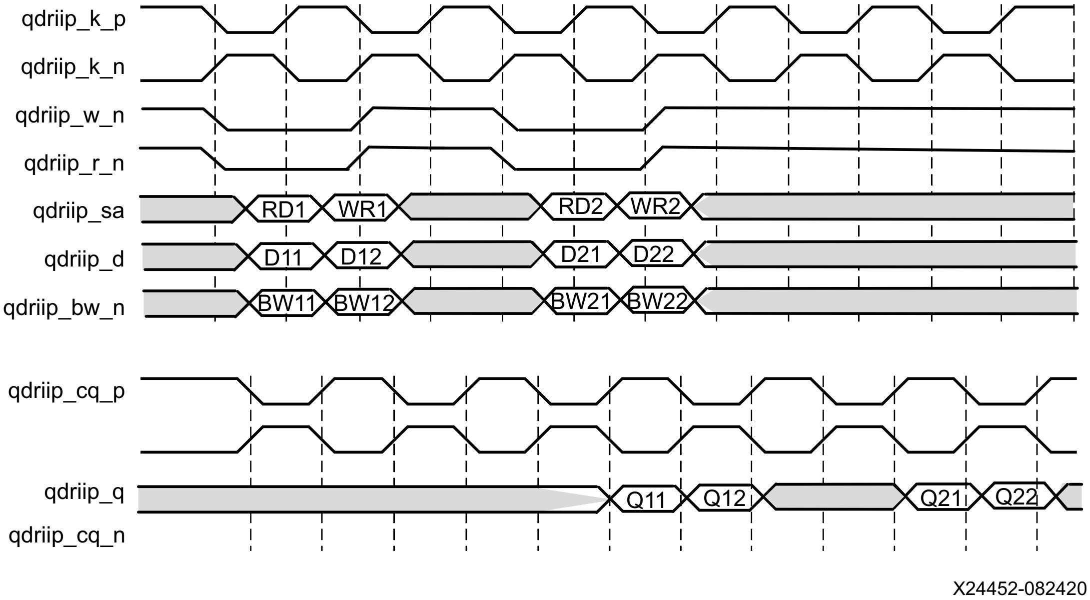


Figure 18-4: **Interfacing with a Two-Word Burst Length Memory Device**

M and D Support for Reference Input Clock Speed

Memory IPs provide two possibilities to select the Reference Input Clock Speed. Value allowed for Reference Input Clock Speed (ps) is always ≥ Memory Device Interface Speed (ps).

- Memory IP lists the possible Reference Input Clock Speed values based on the targeted memory frequency (based on selected Memory Device Interface Speed).
- Otherwise, select **M and D Options** and target for desired Reference Input Clock Speed which is calculated based on selected CLKFBOUT_MULT (M), DIVCLK_DIVIDE (D), and CLKOUT0_DIVIDE (D0) values in the Advanced Clocking Tab.

The required Reference Input Clock Speed is calculated from the M, D, and D0 values entered in the GUI using the following formulas:

- MMCM_CLKOUT (MHz) = tCK / Phy_Clock_Ratio

  Where tCK is the Memory Device Interface Speed selected in the Basic tab.

- CLKIN (MHz) = (MMCM_CLKOUT (MHz) × D × D0) / M

  CLKIN (MHz) is the calculated Reference Input Clock Speed.

- VCO (MHz) = (CLKIN (MHz)) / D

  VCO (MHz) is the calculated VCO frequency.

- PFD (MHz) = CLKIN (MHz) / D

  PFD (MHz) is the calculated PFD frequency.

Calculated Reference Input Clock Speed from M, D, and D0 values are validated as per clocking guidelines. For more information on clocking rules, see Clocking.

Apart from the memory specific clocking rules, validation of the possible MMCM input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values are completed for M, D, and D0 in the GUI.

For UltraScale devices, see *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2] and *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For UltraScale+ devices, see *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4], *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5], and *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For possible M, D, and D0 values and detailed information on clocking and the MMCM, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]

Customizing and Generating the Core

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

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

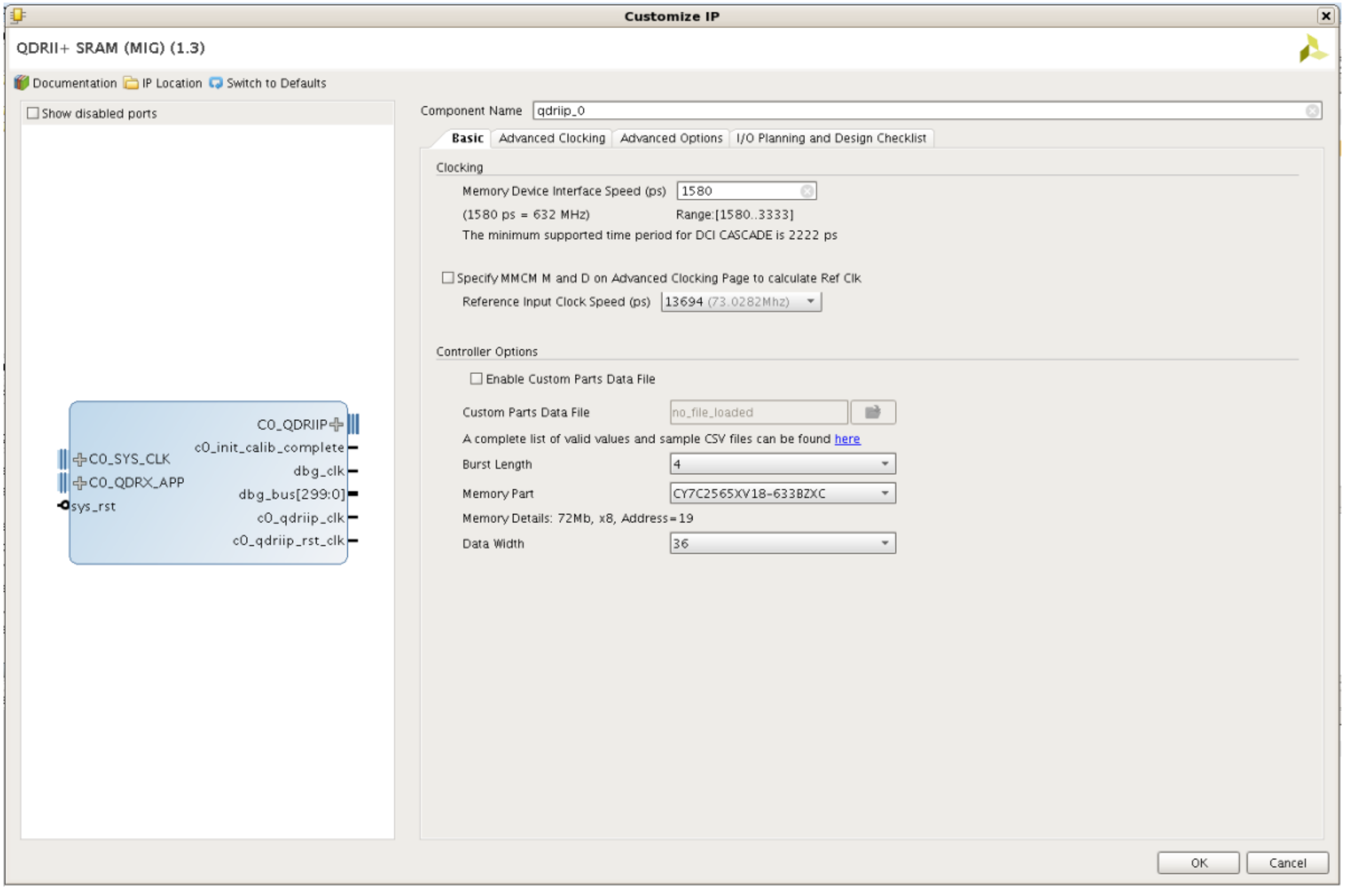
1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For more information about generating the core in Vivado, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

Basic Tab

Figure 19-1 shows the **Basic** tab when you start up the QDR II+ SRAM.



Component Name: qdriip_0

Clocking

Memory Device Interface Speed (ps): 1580

(1580 ps = 632 MHz) Range:[1580..3333]

The minimum supported time period for DCI CASCADE is 2222 ps

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps): 13694 (73.0282Mhz)

Controller Options

☐ Enable Custom Parts Data File

Custom Parts Data File: no_file_loaded

A complete list of valid values and sample CSV files can be found here

Burst Length: 4

Memory Part: CY7C2565XV18-633BZXC

Memory Details: 72Mb, x8, Address=19

Data Width: 36

Figure 19-1: **Vivado Customize IP Dialog Box – Basic**

IMPORTANT: *All parameters shown in the controller options dialog box are limited selection options in this release.*

For the Vivado IDE, all controllers (DDR3, DDR4, LPDDR3, QDR II+, QDR-IV, and RLDRAM 3) can be created and available for instantiation.

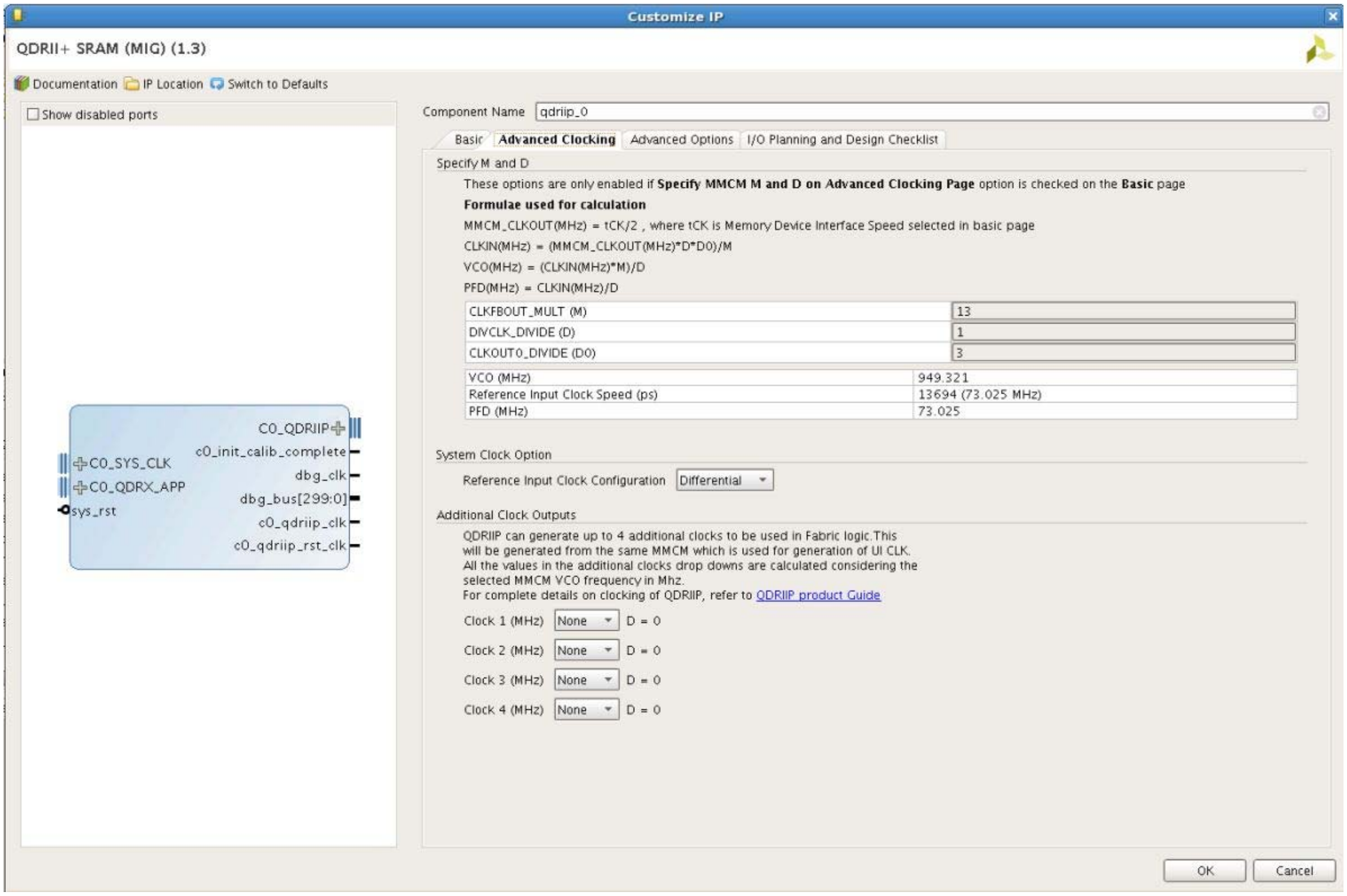
1. Select the settings in the **Clocking** and **Controller Options**.

In **Clocking**, the **Memory Device Interface Speed** sets the speed of the interface. The speed entered drives the available **Reference Input Clock Speeds**. For more information on the clocking structure, see the Clocking, page 359.

2. To use memory parts which are not available by default through the QDR II+ SRAM Vivado IDE, you can create a custom parts CSV file, as specified in the AR: 63462. This CSV file has to be provided after enabling the **Custom Parts Data File** option. After selecting this option. you are able to see the custom memory parts along with the default memory parts. Note that, simulations are not supported for the custom part. Custom part simulations require manually adding the memory model to the simulation and might require modifying the test bench instantiation.

Advanced Clocking Tab

Figure 19-2 shows the next tab called **Advanced Clocking**. This displays the settings for **Specify M and D** value, **System Clock Options**, and **Additional Clock Outputs** for the specific controller.



Component Name: qdriip_0

Specify M and D

These options are only enabled if **Specify MMCM M and D on Advanced Clocking Page** option is checked on the **Basic** page

Formulae used for calculation

MMCM_CLKOUT(MHz) = tCK/2 , where tCK is Memory Device Interface Speed selected in basic page

CLKIN(MHz) = (MMCM_CLKOUT(MHz)*D*D0)/M

VCO(MHz) = (CLKIN(MHz)*M)/D

PFD(MHz) = CLKIN(MHz)/D

CLKFBOUT_MULT (M)	13
DIVCLK_DIVIDE (D)	1
CLKOUT0_DIVIDE (D0)	3

VCO (MHz)	949.321
Reference Input Clock Speed (ps)	13694 (73.025 MHz)
PFD (MHz)	73.025

System Clock Option

Reference Input Clock Configuration: Differential

Additional Clock Outputs

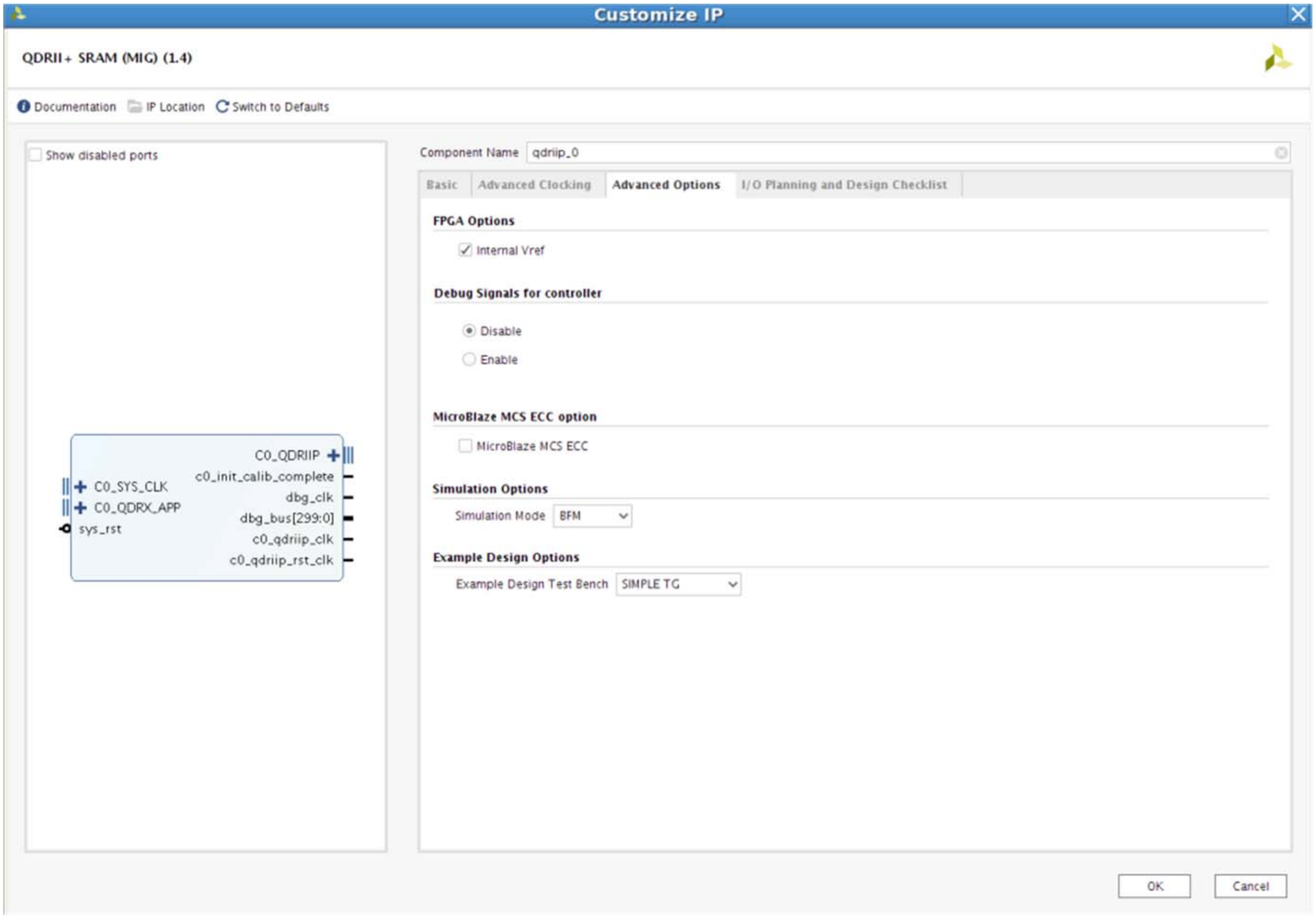
QDRIIP can generate up to 4 additional clocks to be used in Fabric logic.This will be generated from the same MMCM which is used for generation of UI CLK. All the values in the additional clocks drop downs are calculated considering the selected MMCM VCO frequency in Mhz. For complete details on clocking of QDRIIP, refer to QDRIIP product Guide

Clock 1 (MHz): None D = 0
Clock 2 (MHz): None D = 0
Clock 3 (MHz): None D = 0
Clock 4 (MHz): None D = 0

Figure 19-2: **Vivado Customize IP Dialog Box – Advanced Clocking**

Advanced Options Tab

Figure 19-3 shows the next tab called **Advanced Options**. This displays the advanced memory options settings for the specific controller.



Component Name qdriip_0

FPGA Options

☑ Internal Vref

Debug Signals for controller

● Disable

○ Enable

MicroBlaze MCS ECC option

☐ MicroBlaze MCS ECC

Simulation Options

Simulation Mode BFM

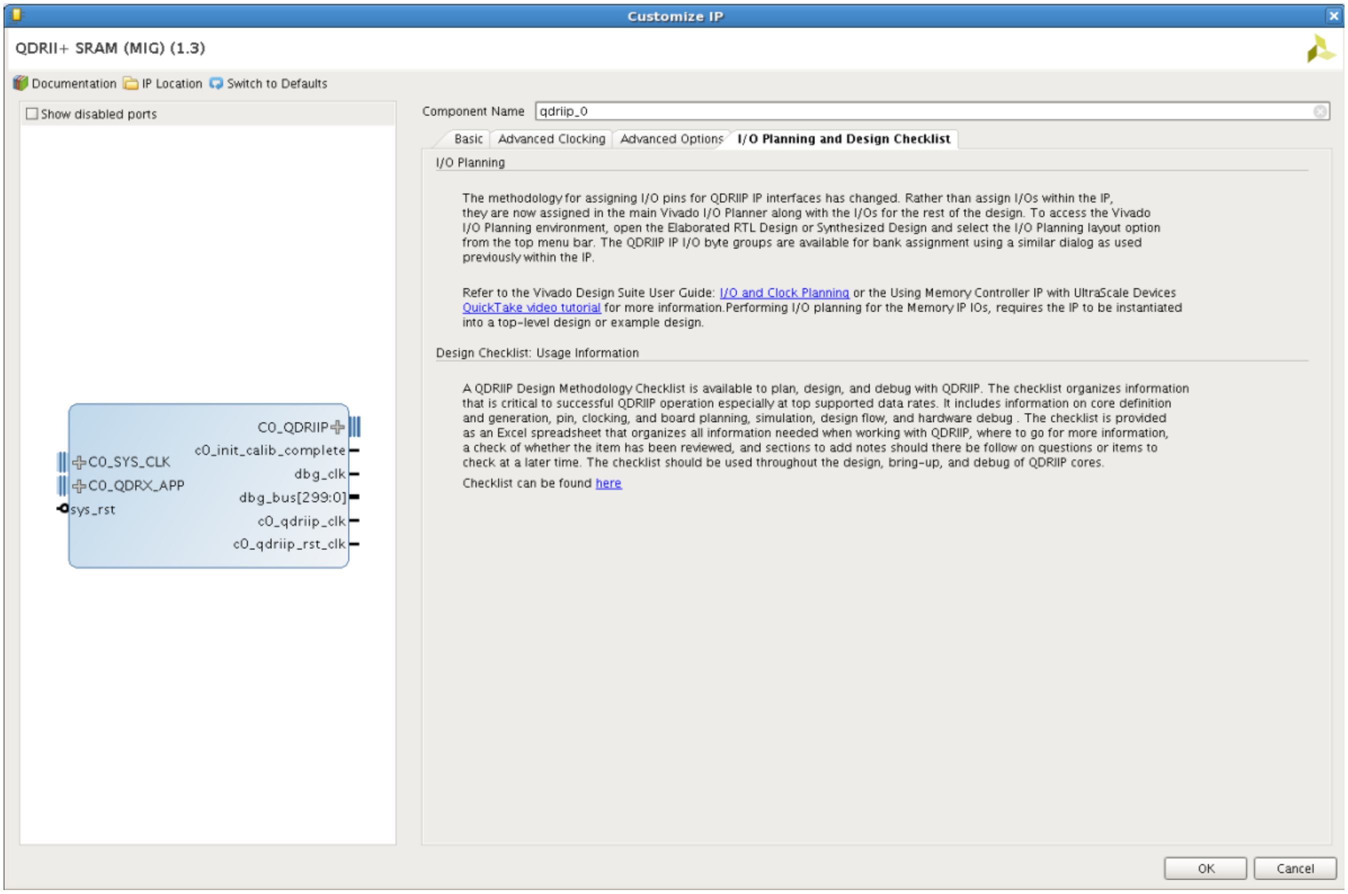
Example Design Options

Example Design Test Bench SIMPLE TG

Figure 19-3: **Vivado Customize IP Dialog Box – Advanced Options**

QDR II+ SRAM I/O Planning and Design Checklist Tab

Figure 19-4 shows the **QDR II+ SRAM I/O Planning and Design Checklist** usage information.



Component Name qdriip_0

I/O Planning

The methodology for assigning I/O pins for QDRIIP IP interfaces has changed. Rather than assign I/Os within the IP, they are now assigned in the main Vivado I/O Planner along with the I/Os for the rest of the design. To access the Vivado I/O Planning environment, open the Elaborated RTL Design or Synthesized Design and select the I/O Planning layout option from the top menu bar. The QDRIIP IP I/O byte groups are available for bank assignment using a similar dialog as used previously within the IP.

Refer to the Vivado Design Suite User Guide: I/O and Clock Planning or the Using Memory Controller IP with UltraScale Devices QuickTake video tutorial for more information.Performing I/O planning for the Memory IP IOs, requires the IP to be instantiated into a top-level design or example design.

Design Checklist: Usage Information

A QDRIIP Design Methodology Checklist is available to plan, design, and debug with QDRIIP. The checklist organizes information that is critical to successful QDRIIP operation especially at top supported data rates. It includes information on core definition and generation, pin, clocking, and board planning, simulation, design flow, and hardware debug . The checklist is provided as an Excel spreadsheet that organizes all information needed when working with QDRIIP, where to go for more information, a check of whether the item has been reviewed, and sections to add notes should there be follow on questions or items to check at a later time. The checklist should be used throughout the design, bring-up, and debug of QDRIIP cores.

Checklist can be found here

Figure 19-4: **Vivado Customize IP Dialog Box – I/O Planning and Design Checklist**

User Parameters

Table 19-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 19-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value
System Clock Configuration	System_Clock	Differential
Internal V_{REF}	Internal_Vref	TRUE
DCI Cascade	DCI_Cascade	FALSE
Debug Signal for Controller	Debug_Signal	Disable
Clock 1 (MHz)	ADDN_UI_CLKOUT1_FREQ_HZ	None
Clock 2 (MHz)	ADDN_UI_CLKOUT2_FREQ_HZ	None

Table 19-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value(1)	User Parameter/Value(1)	Default Value
Clock 3 (MHz)	ADDN_UI_CLKOUT3_FREQ_HZ	None
Clock 4 (MHz)	ADDN_UI_CLKOUT4_FREQ_HZ	None
Enable System Ports	Enable_SysPorts	TRUE
Default Bank Selections	Default_Bank_Selections	FALSE
Reference Clock	Reference_Clock	FALSE
Clock Period (ps)	C0.QDRIIP_TimePeriod	1,819
Input Clock Period (ps)	C0.QDRIIP_InputClockPeriod	13,637
Configuration	C0.QDRIIP_MemoryType	Components
Memory Part	C0.QDRIIP_MemoryPart	CY7C2565XV18-633BZXC
Data Width	C0.QDRIIP_DataWidth	36
Burst Length	C0.QDRIIP_BurstLen	4
Memory Name	C0.QDRIIP_MemoryName	Main Memory

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

I/O Planning

For details on I/O planning, see I/O Planning, page 235.

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

The QDR II+ SRAM Vivado IDE generates the required constraints. A location constraint and an I/O standard constraint are added for each external pin in the design. The location is chosen by the Vivado IDE according to the banks and byte lanes chosen for the design.

The I/O standard is chosen by the memory type selection and options in the Vivado IDE and by the pin type. A sample for `qdriip_d[0]` is shown here.

```
set_property LOC AP25 [get_ports {c0_qdriip_d[0]}]
set_property IOSTANDARD HSTL_I [get_ports {c0_qdriip_d[0]}]
```

The system clock must have the period set properly:

```
create_clock -name c0_sys_clk -period 1.818 [get_ports c0_sys_clk_p]
```

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For more information on clocking, see Clocking, page 359.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The QDR II+ SRAM tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

IMPORTANT: *The* `set_input_delay` *and* `set_output_delay` *constraints are not needed on the external memory interface pins in this design due to the calibration process that automatically runs at start-up. Warnings seen during implementation for the pins can be ignored.*

Simulation

This section contains information about simulating the QDR II+ SRAM generated IP. Vivado simulator, Questa Advanced Simulator, IES, and VCS simulation tools are used for verification of the QDR II+ SRAM IP at each software release. Vivado simulator is not supported yet. For more information on simulation, see Chapter 20, Example Design and Chapter 21, Test Bench.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite. Vivado supports Open IP Example Design flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in Figure 20-1 and select **Open IP Example Design**.

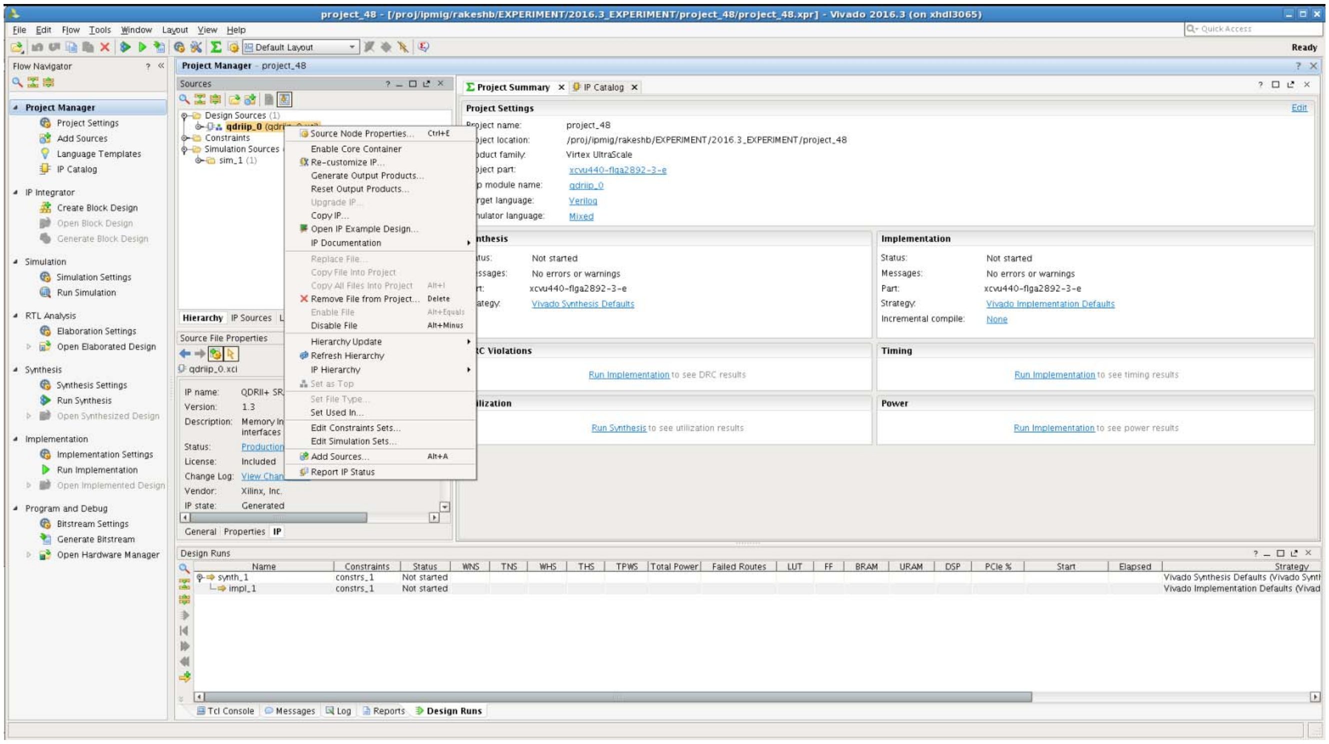


Figure 20-1: **Open IP Example Design**

This option creates a new Vivado project. Upon selecting the menu, a dialog box to enter the directory information for the new design project opens.

Select a directory, or use the defaults, and click **OK**. This launches a new Vivado with all of the example design files and a copy of the IP.

Simulating the Example Design (Designs with Standard User Interface)

The example design provides a synthesizable test bench to generate a fixed simple data pattern to the Memory Controller. This test bench consists of an IP wrapper and an **`example_tb`** that generates 100 writes and 100 reads. QDR II+ SRAM does not deliver the QDR II+ memory models. The memory model required for the simulation must be downloaded from the memory vendor's website.

The example design can be simulated using one of the methods in the following sections.

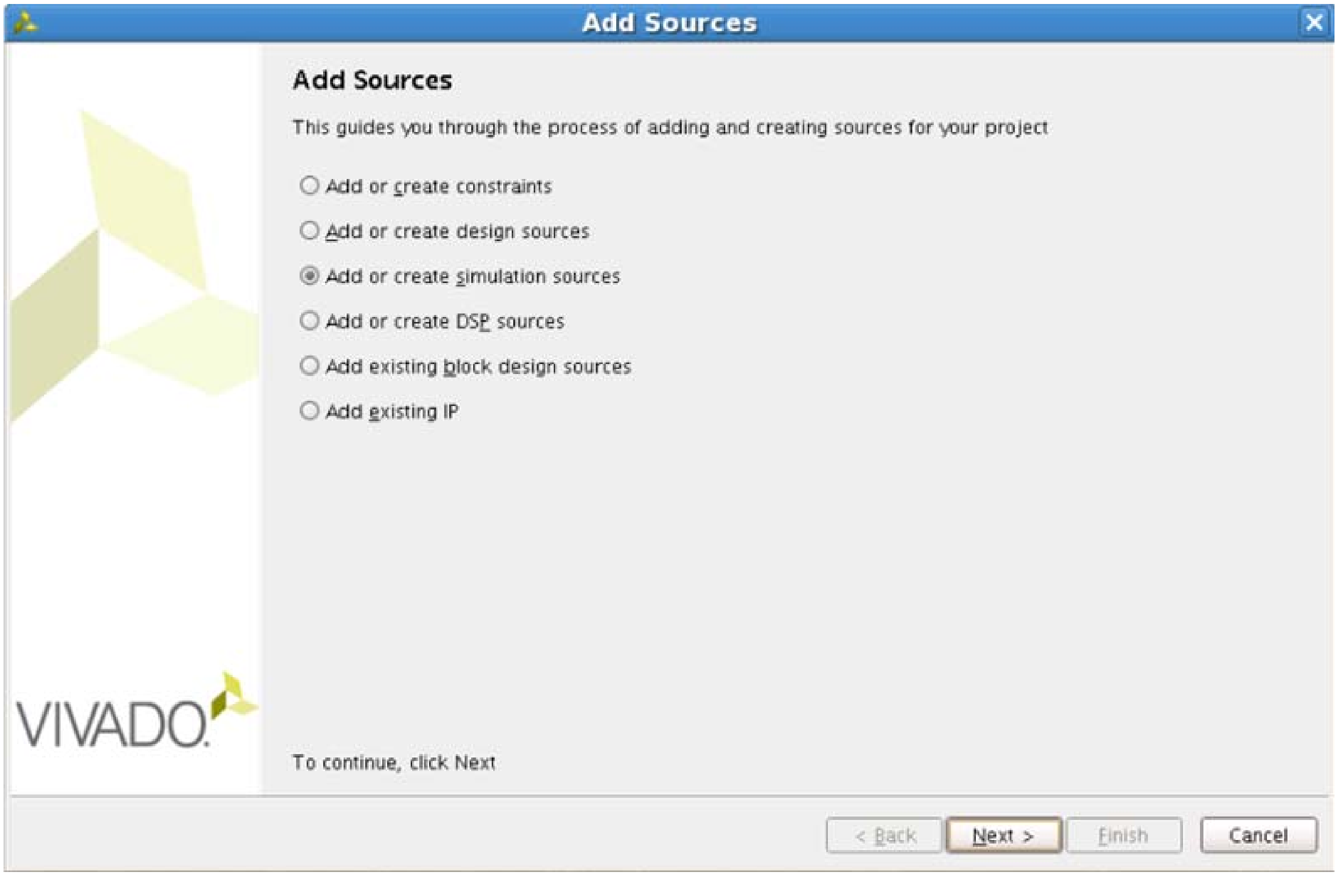
Project-Based Simulation

This method can be used to simulate the example design using the Vivado Integrated Design Environment (IDE). Memory IP does not deliver the QDR II+ memory models. The memory model required for the simulation must be downloaded from the memory vendor's website. The memory model file must be added in the example design using **Add Sources** option to run simulation.

The Vivado simulator, Questa Advanced Simulator, IES, and VCS tools are used for QDR II+ IP verification at each software release. The Vivado simulation tool is used for QDR II+ IP verification from 2015.1 Vivado software release. The following subsections describe steps to run a project-based simulation using each supported simulator tool.

Project-Based Simulation Flow Using Vivado Simulator

1. In the **Open IP Example Design** Vivado project, under **Add sources** option, select the **Add or create simulation sources** option, and click **Next** as shown in Figure 20-2.



Add Sources

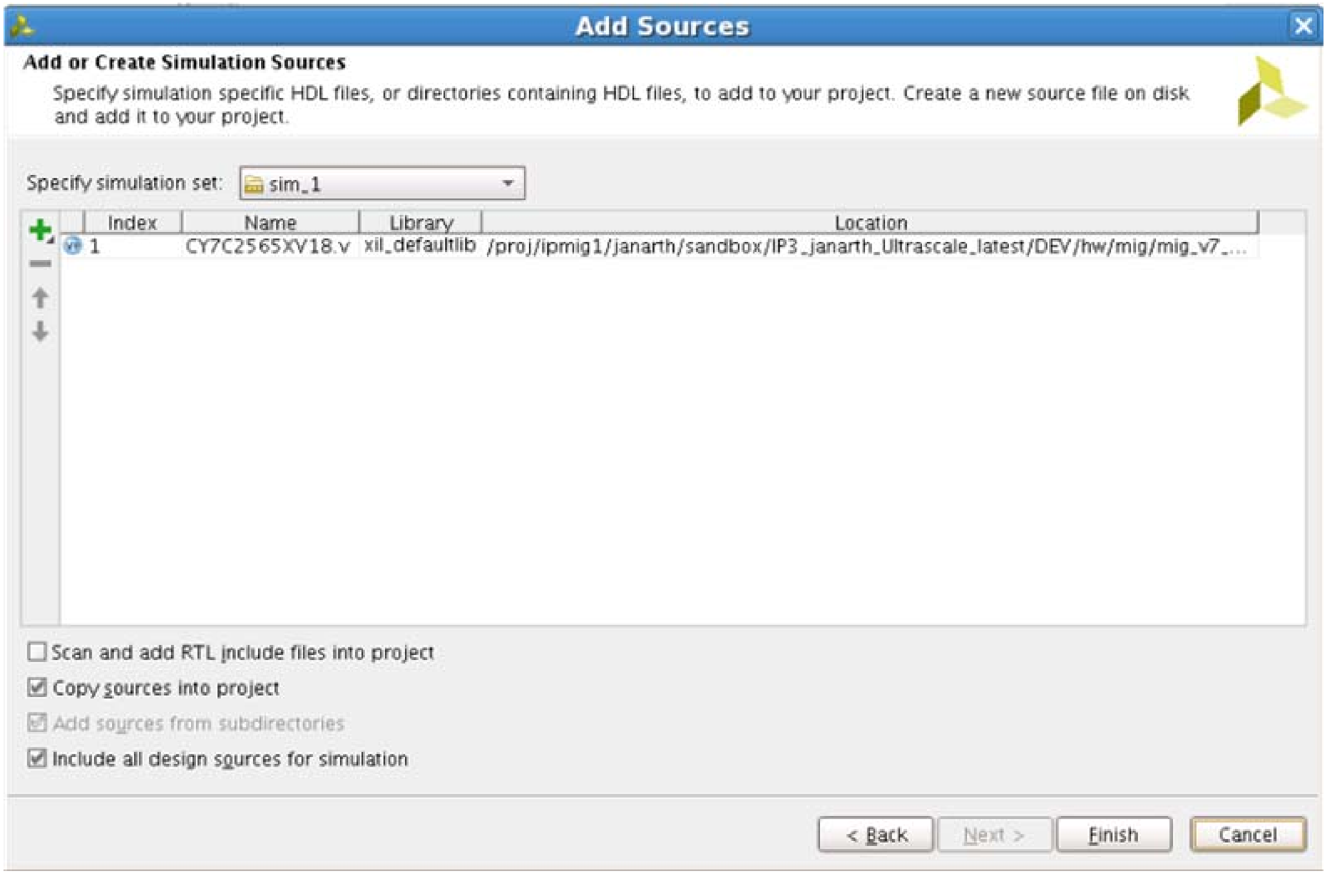
This guides you through the process of adding and creating sources for your project

○ Add or create constraints
○ Add or create design sources
● Add or create simulation sources
○ Add or create DSP sources
○ Add existing block design sources
○ Add existing IP

To continue, click Next

Figure 20-2: **Add Source Option in Vivado**

2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 20-3.



Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location
1	CY7C2565XV18.v	xil_defaultlib	/proj/ipmig1/janarth/sandbox/IP3_janarth_Ultrascale_latest/DEV/hw/mig/mig_v7_...

☐ Scan and add RTL include files into project

☑ Copy sources into project

☑ Add sources from subdirectories

☑ Include all design sources for simulation

Figure 20-3: **Add or Create Simulation Sources in Vivado**

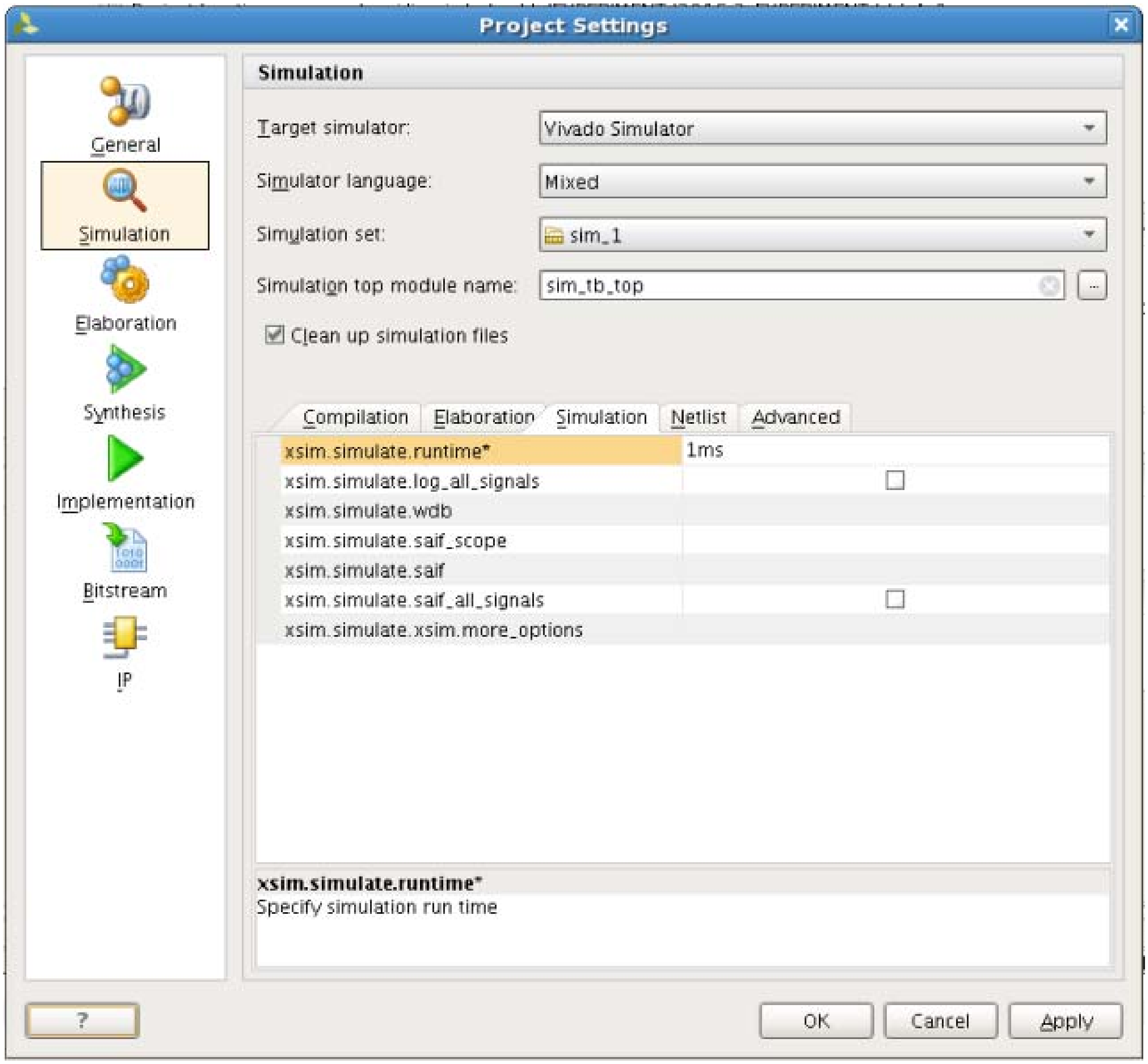
3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**

4. Select **Target simulator** as **Vivado Simulator**.

   Under the **Simulation** tab, set the `xsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 20-4. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.

5. Set the **Simulation Language** to **Mixed**.

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



Simulation

Target simulator: Vivado Simulator

Simulator language: Mixed

Simulation set: sim_1

Simulation top module name: sim_tb_top

☑ Clean up simulation files

xsim.simulate.runtime*	1ms
xsim.simulate.log_all_signals	☐
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐
xsim.simulate.xsim.more_options	

xsim.simulate.runtime*
Specify simulation run time

Figure 20-4: **Simulation with Vivado Simulator**

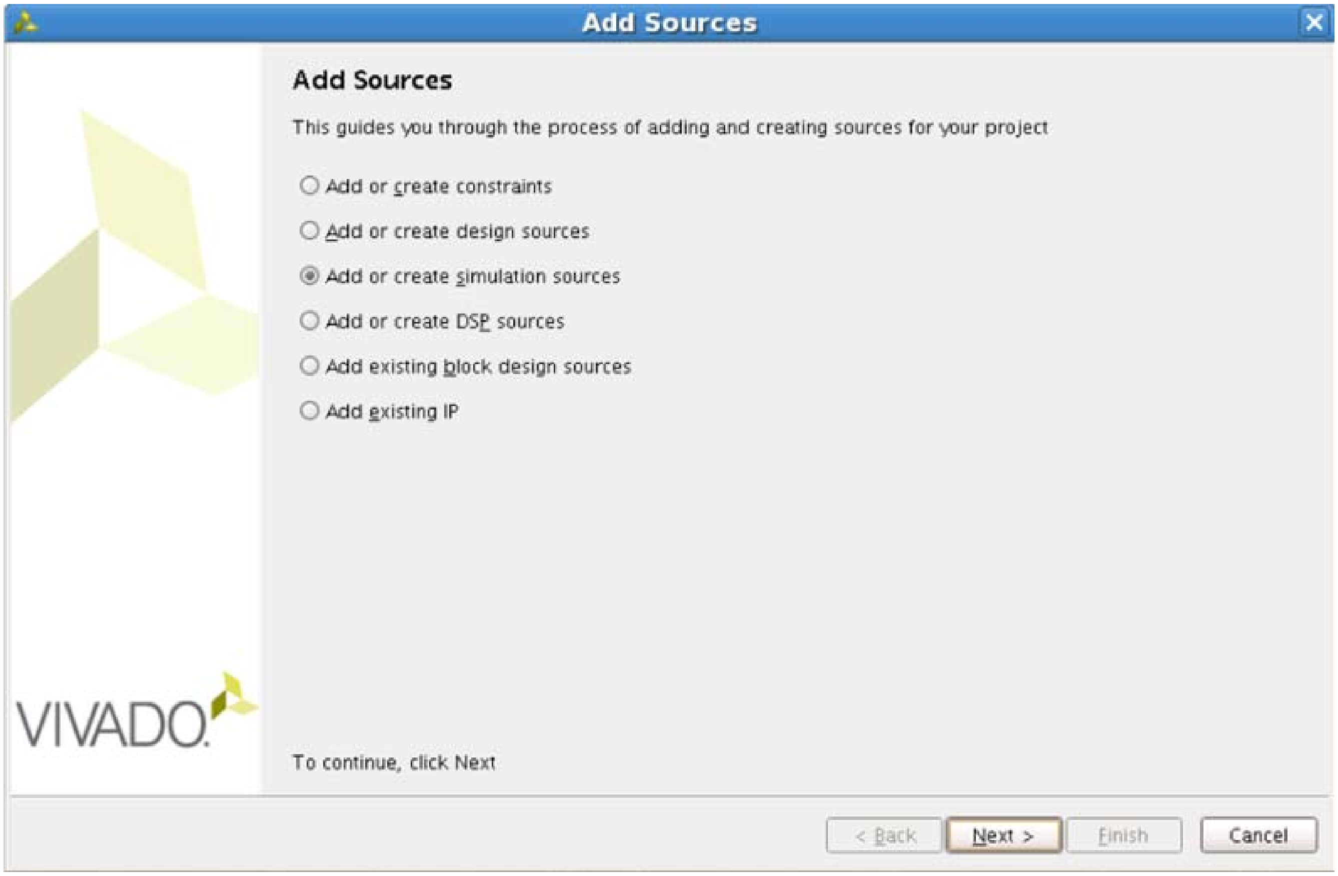
7. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 20-5.

Figure 20-5: **Run Behavioral Simulation**

8. Vivado invokes Vivado simulator and simulations are run in the Vivado simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using Questa Advanced Simulator

1. Open a QDR II+ SRAM example Vivado project (**Open IP Example Design**...), then under **Add sources** option, select the **Add or create simulation sources** option, and click **Next** as shown in Figure 20-6.



Add Sources

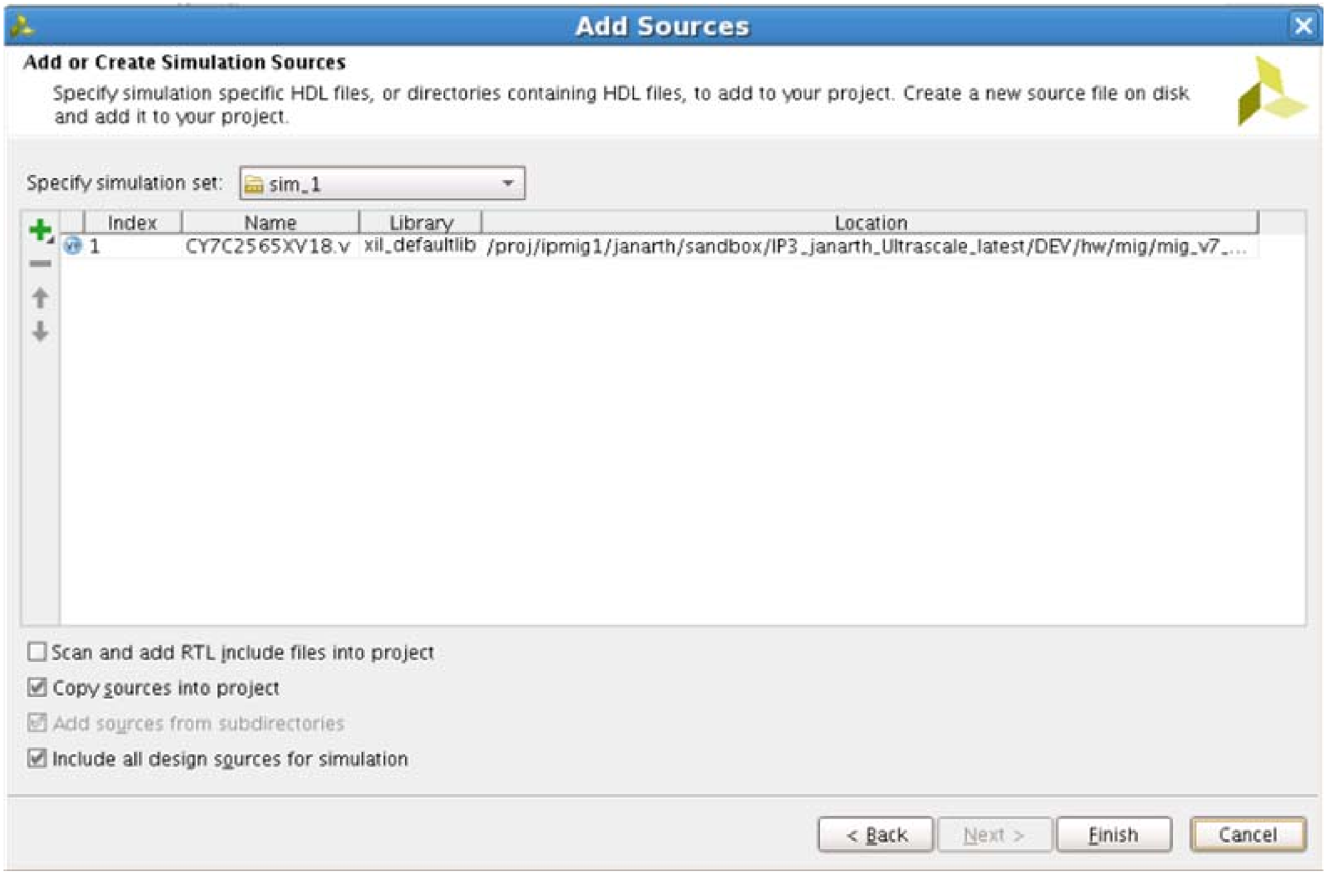
This guides you through the process of adding and creating sources for your project

○ Add or create constraints
○ Add or create design sources
● Add or create simulation sources
○ Add or create DSP sources
○ Add existing block design sources
○ Add existing IP

To continue, click Next

Figure 20-6: **Add Source Option in Vivado**

2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 20-7.



Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location
1	CY7C2565XV18.v	xil_defaultlib	/proj/ipmig1/janarth/sandbox/IP3_janarth_Ultrascale_latest/DEV/hw/mig/mig_v7_...

☐ Scan and add RTL include files into project

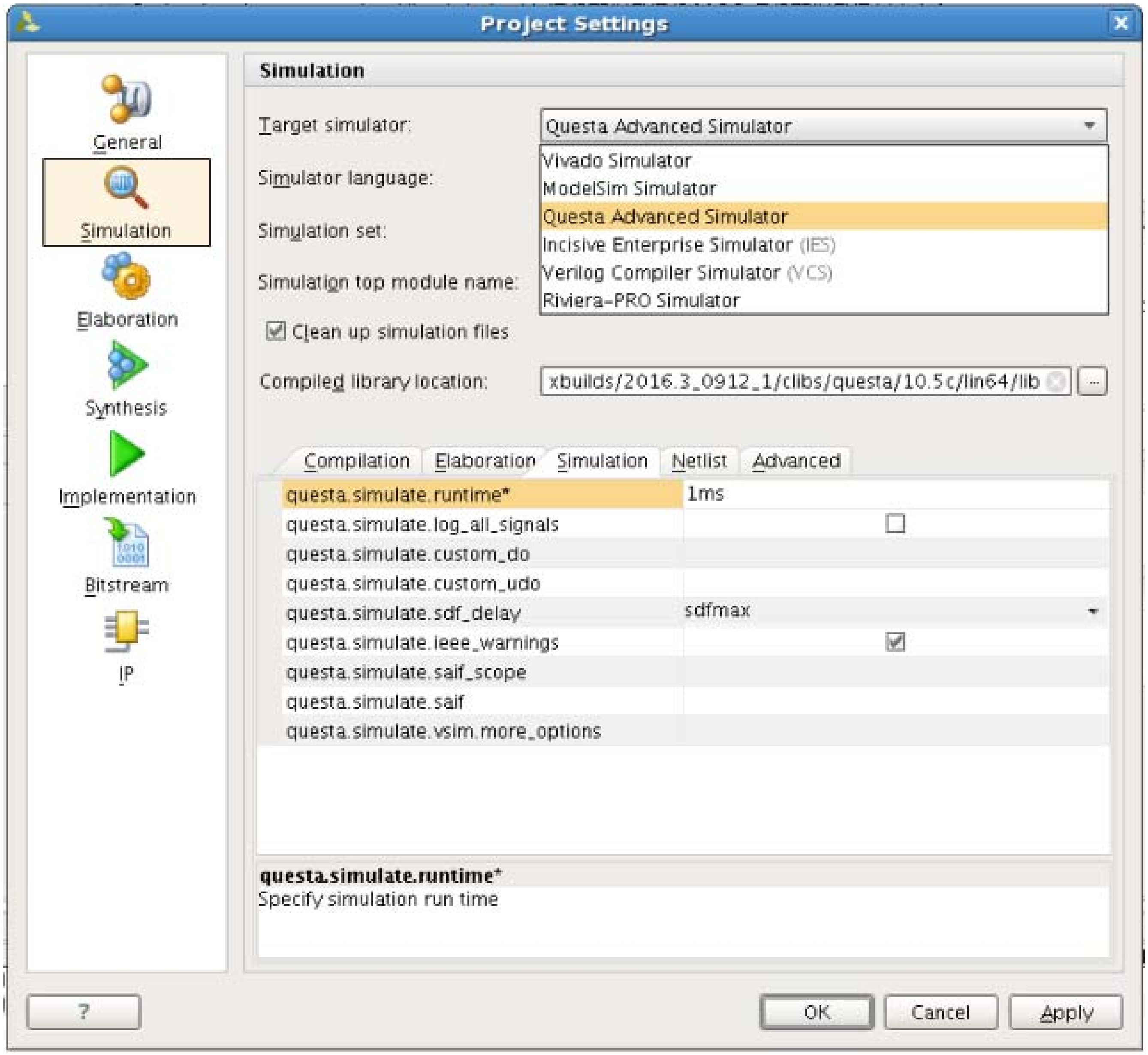
☑ Copy sources into project

☑ Add sources from subdirectories

☑ Include all design sources for simulation

Figure 20-7: **Add or Create Simulation Sources in Vivado**

3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**
4. Select **Target simulator** as **Questa Advanced Simulator**.
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `modelsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 20-8. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
5. Apply the settings and select **OK**.



Simulation

Target simulator: Questa Advanced Simulator

- Vivado Simulator
- ModelSim Simulator
- Questa Advanced Simulator
- Incisive Enterprise Simulator (IES)
- Verilog Compiler Simulator (VCS)
- Riviera-PRO Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: xbuilds/2016.3_0912_1/clibs/questa/10.5c/lin64/lib

Compilation | Elaboration | Simulation | Netlist | Advanced

questa.simulate.runtime*	1ms
questa.simulate.log_all_signals	☐
questa.simulate.custom_do	
questa.simulate.custom_udo	
questa.simulate.sdf_delay	sdfmax
questa.simulate.ieee_warnings	☑
questa.simulate.saif_scope	
questa.simulate.saif	
questa.simulate.vsim.more_options	

questa.simulate.runtime*
Specify simulation run time

Figure 20-8: **Simulation with Questa Advanced Simulator**

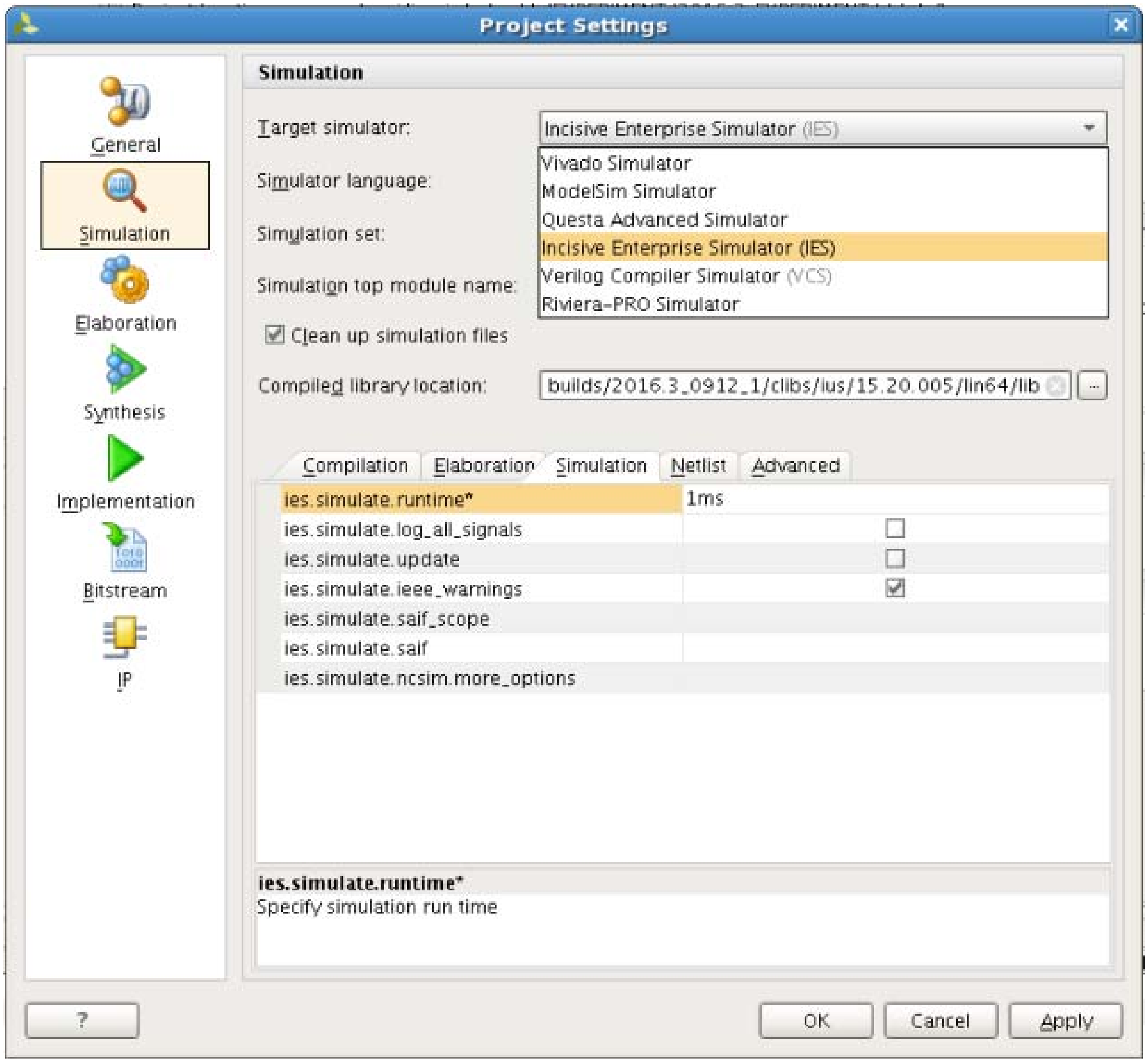
6. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 20-9.

Figure 20-9: **Run Behavioral Simulation**

7. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using IES

1. Open a QDR II+ SRAM example Vivado project (**Open IP Example Design**...), then under **Add sources option**, select the **Add or create simulation sources** option and click **Next** as shown in Figure 20-6.
2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 20-7.
3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**
4. Select **Target simulator** as Incisive Enterprise Simulator (IES).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `ies.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 20-10. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
5. Apply the settings and select **OK**.



Simulation

Target simulator: Incisive Enterprise Simulator (IES)

- Vivado Simulator
- ModelSim Simulator
- Questa Advanced Simulator
- Incisive Enterprise Simulator (IES)
- Verilog Compiler Simulator (VCS)
- Riviera-PRO Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: builds/2016.3_0912_1/clibs/ius/15.20.005/lin64/lib

Compilation | Elaboration | Simulation | Netlist | Advanced

ies.simulate.runtime*	1ms
ies.simulate.log_all_signals	☐
ies.simulate.update	☐
ies.simulate.ieee_warnings	☑
ies.simulate.saif_scope	
ies.simulate.saif	
ies.simulate.ncsim.more_options	

ies.simulate.runtime*

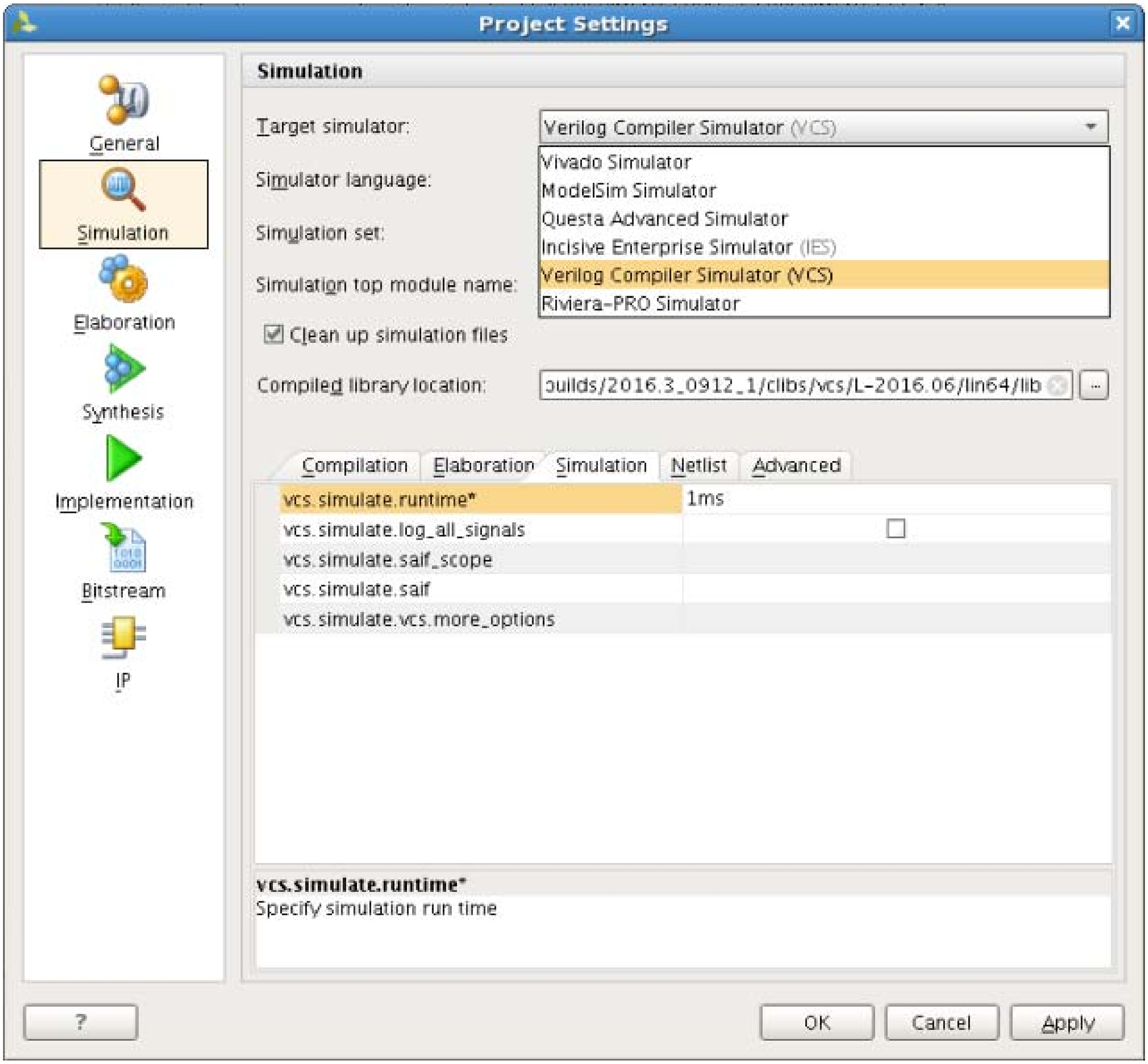
Specify simulation run time

Figure 20-10: **Simulation with IES Simulator**

6. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 20-9.
7. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using VCS

1. Open a QDR II+ SRAM example Vivado project (**Open IP Example Design**...), then under **Add sources option**, select the **Add or create simulation sources** option and click **Next** as shown in Figure 20-6.
2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 20-7.
3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**
4. Select **Target simulator** as Verilog Compiler Simulator (VCS).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `vcs.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 20-11. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
5. Apply the settings and select **OK**.



Target simulator: Verilog Compiler Simulator (VCS)

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: builds/2016.3_0912_1/clibs/vcs/L-2016.06/lin64/lib

vcs.simulate.runtime*	1ms
vcs.simulate.log_all_signals	☐
vcs.simulate.saif_scope	
vcs.simulate.saif	
vcs.simulate.vcs.more_options	

vcs.simulate.runtime*
Specify simulation run time

Figure 20-11: **Simulation with VCS Simulator**

6. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 20-9.
7. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Simulation Speed

QDR II+ SRAM provides a Vivado IDE option to reduce the simulation speed by selecting behavioral XIPHY model instead of UNISIM XIPHY model. Behavioral XIPHY model simulation is a default option for QDR II+ SRAM designs. To select the simulation mode, click the **Advanced Options** tab and find the **Simulation Options** as shown in Figure 19-3.

The SIM_MODE parameter in the RTL is given a different value based on the Vivado IDE selection.

- **SIM_MODE = BFM** – If fast mode is selected in the Vivado IDE, the RTL parameter reflects this value for the SIM_MODE parameter. This is the default option.
- **SIM_MODE = FULL** – If UNISIM mode is selected in the Vivado IDE, XIPHY UNISIMs are selected and the parameter value in the RTL is FULL.

IMPORTANT: *QDR II+ memory models from Cypress® Semiconductor need to be modified with the following two timing parameter values to run the simulations successfully:*
`define tcqd #0
`define tcqdoh #0.15

Using Xilinx IP with Third-Party Synthesis Tools

For more information on how to use Xilinx IP with third-party synthesis tools, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation

If the GCIO pin and MMCM are not allocated in the same bank, the CLOCK_DEDICATED_ROUTE constraint must be set to BACKBONE. To use the BACKBONE route, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM input. QDR II+ SRAM manages these constraints for designs generated with the **Reference Input Clock** option selected as **Differential** (at **Advanced > FPGA Options > Reference Input**). Also, QDR II+ SRAM handles the IP and example design flows for all scenarios.

If the design is generated with the **Reference Input Clock** option selected as **No Buffer** (at **Advanced > FPGA Options > Reference Input**), the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation based on GCIO and

MMCM allocation needs to be handled manually for the IP flow. QDR II+ SRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations for the IP flow.

For an example design flow with **No Buffer** configurations, QDR II+ SRAM generates the example design with differential buffer instantiation for system clock pins. QDR II+ SRAM generates clock constraints in the **`example_design.xdc`**. It also generates a CLOCK_DEDICATED_ROUTE constraint as the “BACKBONE” and instantiates BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM input if the GCIO and MMCM are not in same bank to provide a complete solution. This is done for the example design flow as a reference when it is generated for the first time.

If in the example design, the I/O pins of the system clock pins are changed to some other pins with the I/O pin planner, the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation need to be managed manually. A DRC error is reported for the same.

Chapter 21

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

The Memory Controller is generated along with a simple test bench to verify the basic read and write operations. The stimulus contains 10 consecutive writes followed by 10 consecutive reads for data integrity check.

SECTION V: QDR-IV SRAM

Overview

IMPORTANT: *This document supports QDR-IV SRAM core v2.0.*

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
    - Clocking
    - Resets
    - Protocol Description
    - Customizing and Generating the Core
    - Example Design

Core Overview

The Xilinx UltraScale™ architecture includes the QDR-IV SRAM core. This core provides solutions for interfacing with the QDR-IV SRAM memory type.

The QDR-IV SRAM is a high-performance memory device optimized to maximize the number of random transactions per second by the use of two independent bidirectional data ports.

The QDR-IV SRAM core is a physical layer with a controller for interfacing Xilinx UltraScale FPGA user designs to the QDR-IV SRAM devices. QDR-IV SRAMs offer high-speed data transfers on separate read and write buses on the rising and falling edges of the clock.

These memory devices are used in high-performance systems as temporary data storage, such as:

- Look-up tables in networking systems
- Packet buffers in network switches
- Cache memory in high-speed computing
- Data buffers in high-performance testers

The QDR-IV SRAM solutions core includes a PHY and the controller that takes user commands, processes them to make them compatible to the QDR-IV protocol, and provides the converted commands to the QDR-IV memory. The controller inside the core enables you to provide four commands per cycle simultaneously.

Figure 22-1 shows a high-level block diagram of the QDR-IV SRAM interface solution.

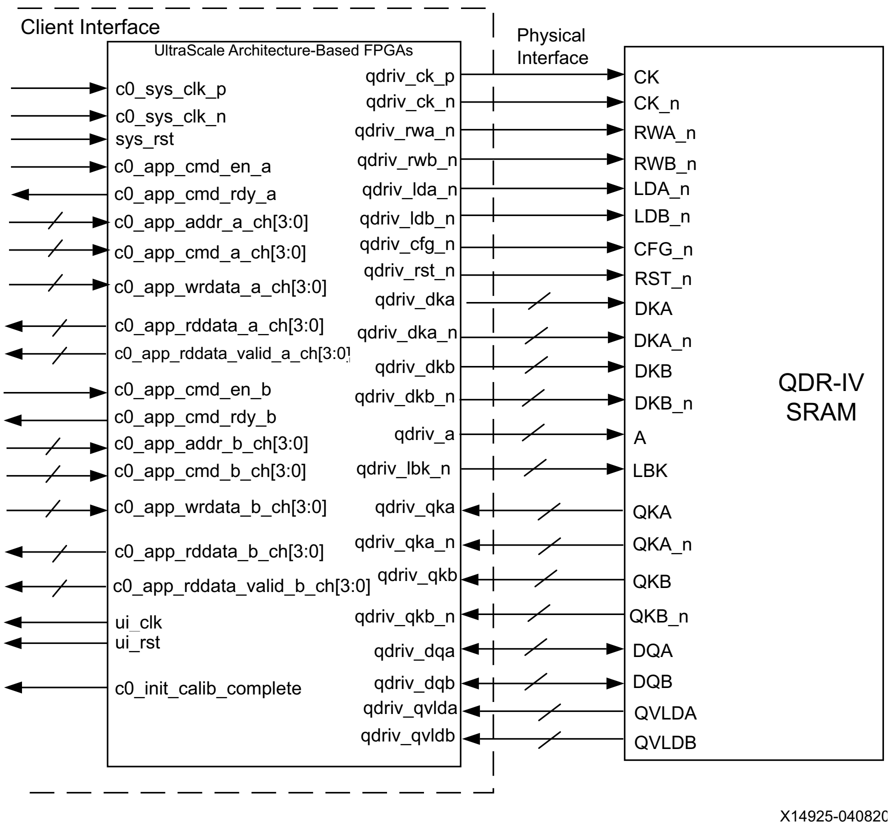


Figure 22-1: **High-Level Block Diagram of QDR-IV Interface Solution**

The QDR-IV core includes the hard blocks inside the FPGA and the soft calibration logic necessary to ensure optimal timing of the hard blocks interfacing to the memory part.

The hard blocks include:

- Data serialization and transmission

- Data capture and deserialization
- High-speed clock generation and synchronization
- Coarse and fine delay elements per pin with voltage and temperature tracking

The soft blocks include:

- **Memory Initialization** – The calibration modules provide an initialization routine and a reset sequence for the particular memory type. The delays in the initialization process can be bypassed to speed up simulation time if desired.

  The QDR-IV SRAM must be initialized before it can operate in the normal functional mode. Initialization uses four special pins:

  - `RST_n` pin to reset the device.
  - `CFG_n` pin to program the configuration registers
  - `LBK0_n` and `LBK1_n` pins for the loopback function.

  The following steps should be followed to initialize the QDR-IV memory:

  a. Apply power to the QDR-IV SRAM. Follow instructions described in the power-up sequence section in the memory data sheet.

  b. Apply reset to the QDR-IV SRAM. Follow reset sequence instruction in the memory data sheet.

  c. Assert Config (`CFG_n` = 0) and program the impedance control register.

  d. Because the input impedance is updated, allow the PLL time (`tPLL`) to lock to the input clock. See the memory data sheet for `tPLL` value.

- **Calibration** – The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance. Results of the calibration process are available through the Xilinx debug tools. After completion of calibration, the PHY layer presents the raw interface to the memory part.

Feature Summary

- Component support for interface widths up to 36 bits
- Single component interface with x18 and x36 memory device support
- 2-word burst support (BL2 only)
- Only POD12 standard support
- Memory device support with 72 Mb density and 144 Mb density

- Other densities for memory device support is available through custom part selection
- Support for 5 (for HP memory part) and 8 (for XP memory part) cycles of read latency
- Support for 3 (for HP memory part) and 5 (for XP memory part) cycles of write latency
- Source code delivery in Verilog and SystemVerilog
- 4:1 memory to FPGA logic interface clock ratio
- Interface calibration and training information available through the Vivado hardware manager
- Programmable On-die Termination (ODT) support for address, clock, and data

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

Standards

This core complies with the QDR-IV SRAM standard defined by the QDR Consortium. For more information on UltraScale™ architecture documents, see References, page 789.

Performance

Maximum Frequencies

For more information on the maximum frequencies, see the following documentation:

- *Kintex UltraScale FPGAs Data Sheet, DC and AC Switching Characteristics* (DS892) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6]
- *UltraScale Maximum Memory Performance Utility* (XTP414) [Ref 21]

Resource Utilization

For full details about performance and resource utilization, visit Performance and Resource Utilization.

Port Descriptions

There are three port categories at the top-level of the memory interface core called the "user design."

- The first category is the memory interface signals that directly interfaces with the memory part. These are defined by the QDR-IV SRAM specification.
- The second category is the application interface signals which is referred to as the "user interface." This is described in the Protocol Description, page 436.
- The third category includes other signals necessary for proper operation of the core. These include the clocks, reset, and status signals from the core. The clocking and reset signals are described in their respective sections.

  The active-High **`c0_init_calib_complete`** signal indicates that initialization and calibration are complete and that the interface is now ready to accept commands for the interface.

  Ensure that the commands are issued only after **`c0_init_calib_complete`** is High. Any commands issued before **`c0_init_calib_complete`** signal is High will be lost.

Core Architecture

This chapter describes the UltraScale™ architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Overview

The UltraScale architecture-based FPGAs Memory Interface Solutions is shown in Figure 24-1.

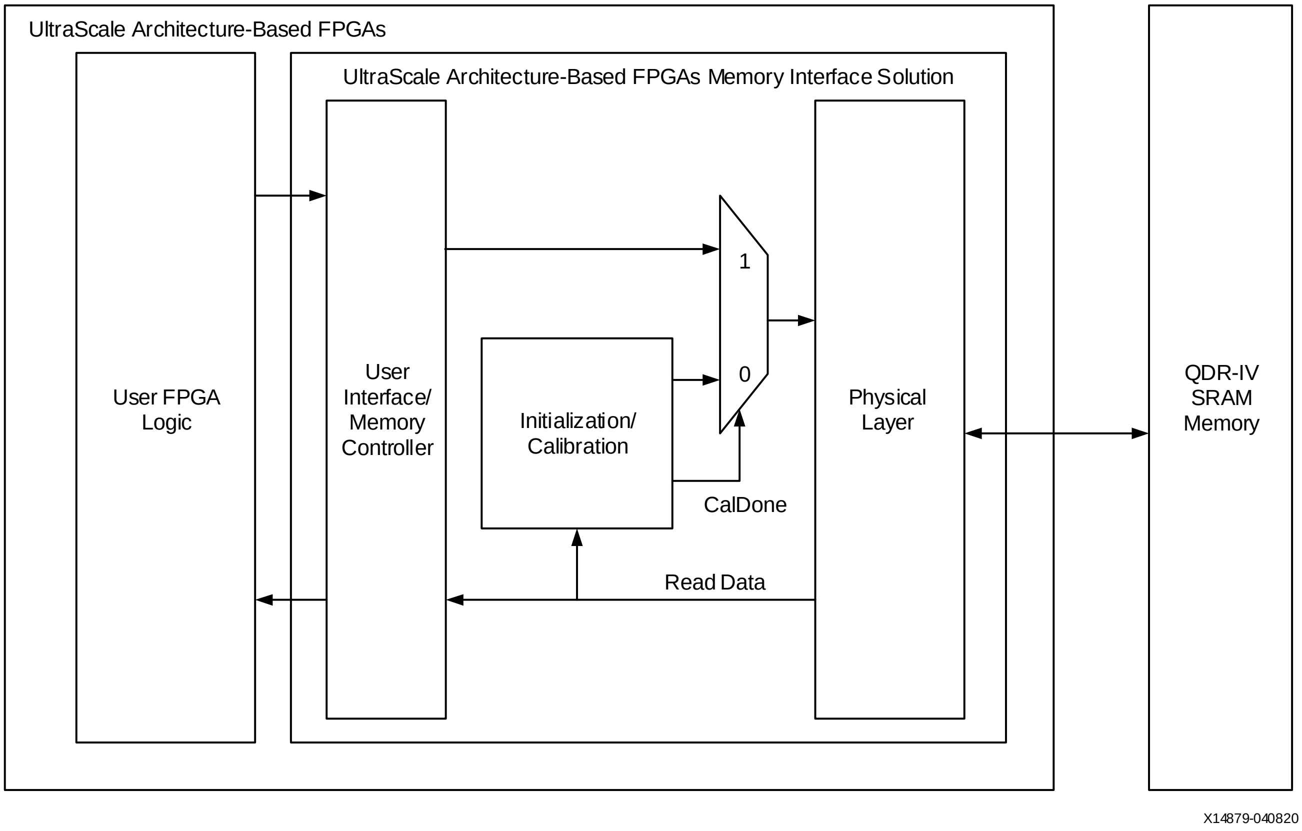


Figure 24-1: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core**

The user interface uses a simple protocol based entirely on SDR signals to make read and write requests. For more details describing this protocol, see User Interface in Chapter 25.

PHY

The PHY is considered the low-level physical interface to an external QDR-IV SRAM device. It contains the entire calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the QDR-IV SRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

Overall PHY Architecture

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

The user interface/controller and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is divided by 4. A more detailed block diagram of the PHY design is shown in Figure 24-2.

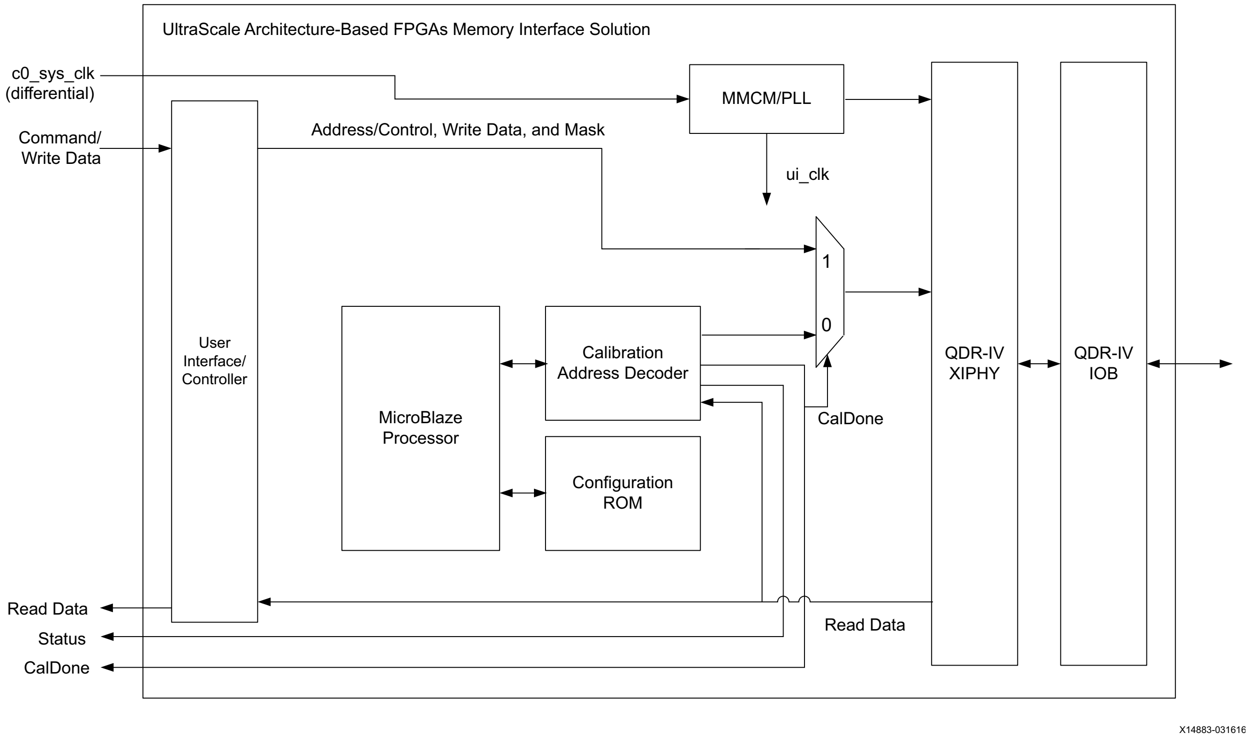


Figure 24-2: **PHY Block Diagram**

Table 24-1: **PHY Modules**

Module Name	Description
QDR-IV PHY	PHY top of QDR-IV design
QDR-IV Calibration	Calibration top module
QDR-IV Calibration Address Decoder	FPGA logic interface for the MicroBlaze processor
QDR-IV Configuration ROM	Configuration storage for calibration options
MicroBlaze MCS	MicroBlaze processor
QDR-IV XIPHY	Contains the XIPHY instance
QDR-IV IOB Byte	Instantiates all byte IOB modules
QDR-IV IOB	QDR-IV

The PHY architecture encompasses all of the logic contained in QDR-IV XIPHY module. The PHY contains wrappers around dedicated hard blocks to build up the memory interface from smaller components. A byte lane contains all of the clocks, resets, and datapaths for a given subset of I/O. Multiple byte lanes are grouped together, along with dedicated clocking resources, to make up a single bank memory interface. For more information on the hard silicon physical layer architecture, see the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7].

The memory initialization and calibration are implemented in C programming on a small soft core processor. The MicroBlaze™ Controller System (MCS) is configured with an I/O Module, MicroBlaze Debug Module (MDM), and block RAM. The module QDR-IV Calibration Address Decoder provides the interface for the processor to the rest of the system and implements helper logic. The QDR-IV Configuration ROM module stores settings that control the operation of initialization and calibration, providing run time options that can be adjusted without having to recompile the source code.

The address unit connects the MCS to the local register set and the PHY by performing address decode and control translation on the I/O module bus from spaces in the memory map and MUXing return data (QDR-IV Calibration Address Decoder). In addition, it provides address translation (also known as “mapping”) from a logical conceptualization of the SRAM interface to the appropriate pinout-dependent location of the delay control in the PHY address space.

Although the calibration architecture presents a simple and organized address map for manipulating the delay elements for individual data, control and command bits, there is flexibility in how those I/O pins are placed. For a given I/O placement, the path to the FPGA logic is locked to a given pin. To enable a single binary software file to work with any memory interface pinout, a translation block converts the simplified Register Interface Unit (RIU) addressing into the pinout-specific RIU address for the target design. The specific address translation is written by QDR-IV SRAM after a pinout is selected. The code shows an example of the RTL structure that supports this.

```
Casez(io_address)// MicroBlaze I/O module address
  // … static address decoding skipped
  //=======================================//
  //===========DQ ODELAYS==================//
  //=======================================//
  //Byte0
  28`h0004100: begin //dq0
    riu_addr_cal =  /* QDR-IV SRAM Generated */ 6`hd;
    riu_nibble =    /* QDR-IV SRAM Generated */ 'h13;
  end
  // … additional dynamic addressing follows
```

In this example, `DQ0` is pinned out on Bit[0] of nibble 0 (nibble 0 according to instantiation order). The RIU address for the ODELAY for Bit[0] is `0x0D`. When `DQ0` is addressed — indicated by address `0x000_4100`), this snippet of code is active. It enables nibble 0 (decoded to one-hot downstream) and forwards the address `0x0D` to the RIU address bus.

The MicroBlaze I/O module interface updates at a maximum rate of once every three clock cycles, which is not always fast enough for implementing all of the functions required in calibration. A helper circuit implemented in Calibration Address Decoder module is required to obtain commands from the registers and translate at least a portion into single-cycle accuracy for submission to the PHY. In addition, it supports command repetition to enable back-to-back read transactions and read data comparison.

Memory Initialization and Calibration Sequence

After deassertion of the system reset, the PHY performs some required internal calibration steps first.

1. The built-in self-check (BISC) of the PHY is run. It is used to compensate the internal skews among the data bits and the strobe on the read path. The computed skews are used in the voltage and temperature tracking after calibration is completed.
2. After BISC completion, calibration logic performs the required power-on initialization sequence for the memory. This is followed by several stages of timing calibration for the write and read datapaths.
3. After calibration is completed, PHY calculates internal offsets to be used in voltage and temperature tracking.
4. When PHY indicates the calibration completion, the user interface command execution begins.

Figure 24-3 shows the overall flow of memory initialization and the different stages of calibration.

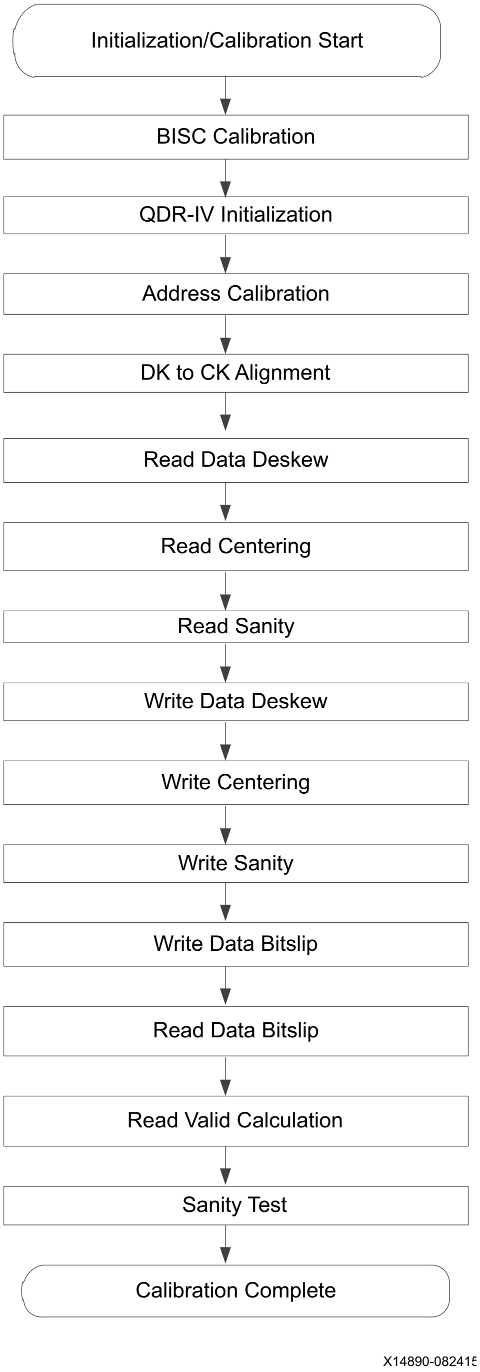


Figure 24-3: **PHY Overall Initialization and Calibration Sequence**

Reset Sequence

The **`sys_rst`** signal resets the entire memory design which includes general interconnect (fabric) logic which is driven by the MMCM clock (clkout0) and RIU logic. MicroBlaze™ and calibration logic are driven by the MMCM clock (clkout6). The **`sys_rst`** input signal is synchronized internally to create the **`qdriv_rst_clk`** signal. The **`qdriv_rst_clk`** reset signal is synchronously asserted and synchronously deasserted.

Figure 24-4 shows the **`qdriv_rst_clk`** (fabric reset) is synchronously asserted with a few clock delays after **`sys_rst`** is asserted. When **`qdriv_rst_clk`** is asserted, there are a few clocks before the clocks are shut off.

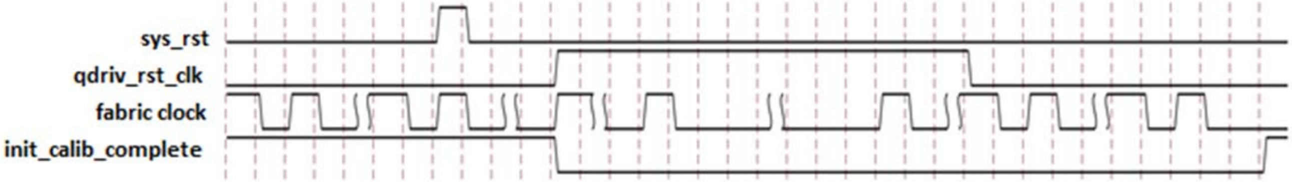


Figure 24-4: **Reset Sequence Waveform**

The following are the reset sequencing steps:

1. Reset to design is initiated after **`qdriv_rst_clk`** goes High.
2. **`init_calib_complete`** signal goes Low when **`qdriv_rst_clk`** is High.
3. Reset to design is deactivated after **`qdriv_rst_clk`** is Low.
4. After **`qdriv_rst_clk`** is deactivated, the **`init_calib_complete`** is asserted after calibration is completed.

MicroBlaze MCS ECC

The MicroBlaze MCS local memory provides an option to enable Error Correcting Code (ECC). Error correction corrects single bit errors and detects double bit errors. Two additional ports are added to indicate single bit errors (LMB_CE) and double bit errors (LMB_UE).

The MicroBlaze MCS ECC can be selected from the **MicroBlaze MCS ECC option** section in the **Advanced Options** tab. The block RAM size increases if the ECC option for MicroBlaze MCS is selected.

Chapter 25

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The memory interface requires one mixed-mode clock manager (MMCM), one TXPLL per I/O bank used by the memory interface and two BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used.

The QDR-IV IP generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to MMCM (located in center bank of memory interface)
- MMCM to BUFG (located at center bank of memory interface) driving FPGA logic and all TXPLLs
- MMCM to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate FPGA logic
- Clocking pair of the interface must be in the same SLR of memory interface for the stacked silicon interconnect (SSI) technology devices

Requirements

GCIO

- Must use a differential I/O standard
- Must be in the same I/O column as the memory interface
- The I/O standard and termination scheme are system dependent. For more information, consult the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 7].

MMCM

- MMCM is used to generate the FPGA logic system clock (1/2 of the memory clock)
- Must be located in the center bank of memory interface
- Must use internal feedback
- Input clock frequency divided by input divider must be ≥ 70 MHz (CLKINx / D ≥ 70 MHz)
- Must use integer multiply and output divide values

BUFGs and Clock Roots

- One BUFG is used to generate the system clock to FPGA logic and another BUFG is used to divide the system clock by two.
- BUFG and clock roots must be located in center most bank of the memory interface.
    - For two bank systems, the bank with the higher number of bytes selected is chosen as the center bank. If the same number of bytes is selected in two banks, then the top bank is chosen as the center bank.
    - Both the BUFGs must be in the same bank

TXPLL

- CLKOUTPHY from TXPLL drives XIPHY within its bank
- TXPLL must be set to use a CLKFBOUT phase shift of 90°
- TXPLL must be held in reset until the MMCM lock output goes High
- Must use internal feedback

Figure 25-1 shows an example of the clocking structure for a three bank memory interface. The GCIO drives the MMCM located at the center bank of the memory interface. MMCM drives both the BUFGs located in the same bank. The BUFG (which is used to generate system clock to FPGA logic) output drives the TXPLLs used in each bank of the interface.

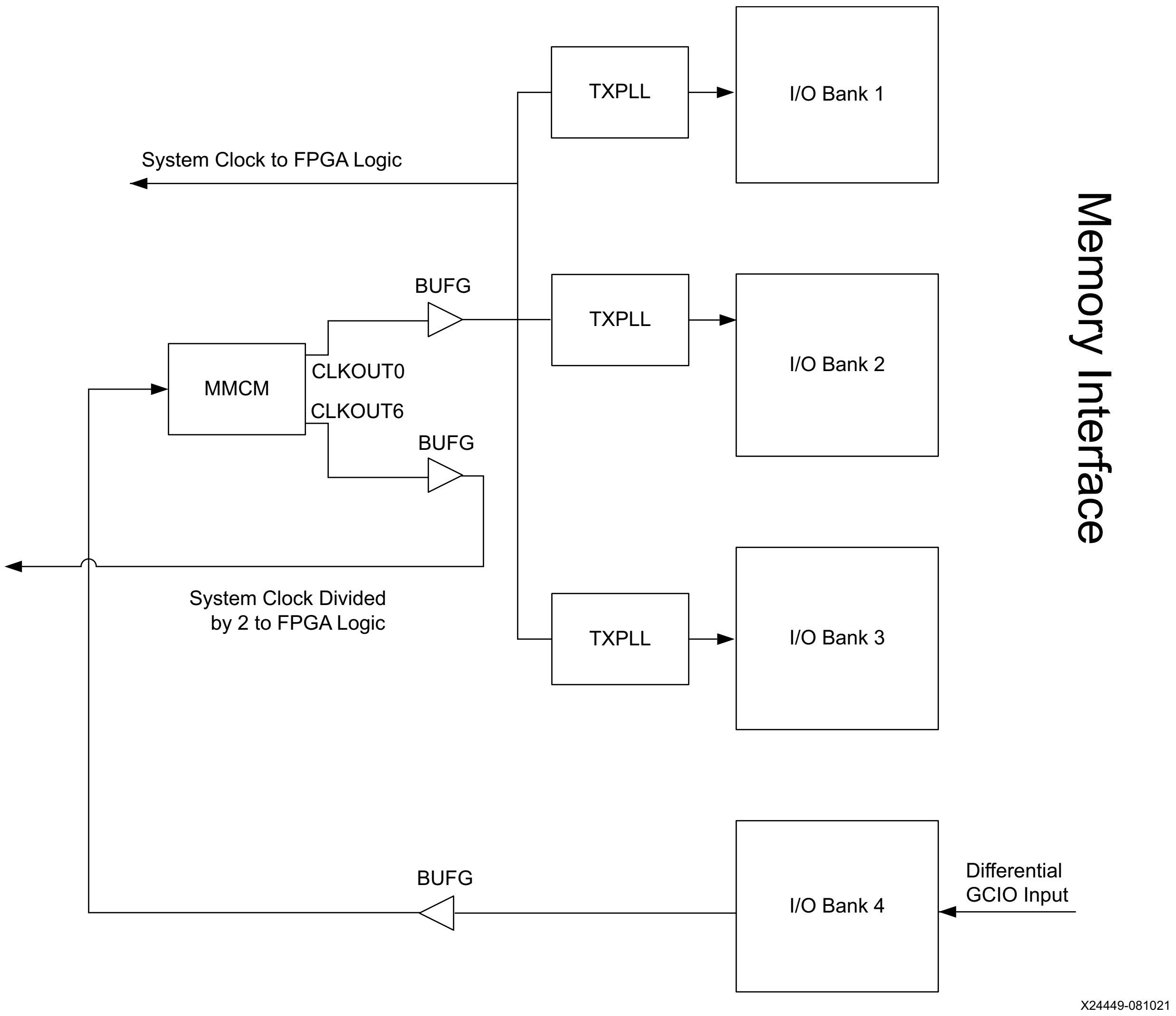


Figure 25-1: **Clocking Architecture Inside a QDR-IV Design**

The MMCM is placed in the center bank of the memory interface.

- For two bank systems, MMCM is placed in a bank with the most number of bytes selected. If they both have the same number of bytes selected in two banks, then MMCM is placed in the top bank.
- For four bank systems, MMCM is placed in a second bank from the top.

For designs generated with System Clock configuration of **No Buffer**, MMCM must not be driven by another MMCM/PLL. Cascading clocking structures MMCM → BUFG → MMCM and PLL → BUFG → MMCM are not allowed.

If the MMCM is driven by the GCIO pin of the other bank, then the CLOCK_DEDICATED_ROUTE constraint with value "BACKBONE" must be set on the net that is driving MMCM or on the MMCM input. Setting up the CLOCK_DEDICATED_ROUTE constraint on the net is preferred. But when the same net is driving two MMCMs, the CLOCK_DEDICATED_ROUTE constraint must be managed by considering which MMCM needs the BACKBONE route.

In such cases, the CLOCK_DEDICATED_ROUTE constraint can be set on the MMCM input. To use the "BACKBONE" route, any clock buffer that exists in the same CMT tile as the GCIO must exist between the GCIO and MMCM input. The clock buffers that exists in the I/O CMT are BUFG, BUFGCE, BUFGCTRL, and BUFGCE_DIV. So QDR-IV SRAM instantiates BUFG between the GCIO and MMCM when the GCIO pins and MMCM are not in the same bank (see Figure 25-1).

If the GCIO pin and MMCM are allocated in different banks, QDR-IV SRAM generates CLOCK_DEDICATED_ROUTE constraints with value as "BACKBONE." If the GCIO pin and MMCM are allocated in the same bank, there is no need to set any constraints on the MMCM input.

Similarly when designs are generated with System Clock Configuration as a **No Buffer** option, you must take care of the "BACKBONE" constraint and the BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM if GCIO pin and MMCM are allocated in different banks. QDR-IV SRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations. For more information on clocking, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

XDC syntax for CLOCK_DEDICATED_ROUTE constraint is given here:

```
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hier -filter {NAME =~ */
u_qdriv_infrastructure/gen_mmcme*.u_mmcme_adv_inst/CLKIN1}]
```

For more information on the CLOCK_DEDICATED_ROUTE constraints, see the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 9].

Note: If two different GCIO pins are used for two QDR-IV SRAM IP cores in the same bank, center bank of the memory interface is different for each IP. QDR-IV SRAM generates MMCM LOC and CLOCK_DEDICATED_ROUTE constraints accordingly.

Sharing of Input Clock Source (sys_clk_p)

If the same GCIO pin must be used for two IP cores, generate the two IP cores with the same frequency value selected for option **Reference Input Clock Period (ps)** and **System Clock Configuration** option as **No Buffer**. Perform the following changes in the wrapper file in which both IPs are instantiated:

1. QDR-IV SRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) of both the IP cores.
2. System clock pins must be allocated within the same I/O column of the memory interface pins allocated. Add the pin LOC constraints for system clock pins and clock constraints in your top-level XDC.
3. You must add a "BACKBONE" constraint on the net that is driving the MMCM or on the MMCM input if GCIO pin and MMCM are not allocated in the same bank. Apart from this, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM to use the "BACKBONE" route.

Note:

- The UltraScale architecture includes an independent XIPHY power supply and TXPLL for each XIPHY. This results in clean, low jitter clocks for the memory system.
- Skew spanning across multiple BUFGs is not a concern because single point of contact exists between BUFG → TXPLL and the same BUFG → System Clock Logic.
- System input clock cannot span I/O columns because the longer the clock lines span, the more jitter is picked up.

TXPLL Usage

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used. One PLL per bank is used if a bank is used by a single memory interface. You can use a second PLL for other usage. To use a second PLL, you can perform the following steps:

1. Generate the design for the **System Clock Configuration** option as **No Buffer**.
2. QDR-IV SRAM generates a single-ended input for system clock pins, such as **`sys_clk_i`**. Connect the differential buffer output to the single-ended system clock inputs (**`sys_clk_i`**) and also to the input of PLL (PLL instance that you have in your design).
3. You can use the PLL output clocks.

Additional Clocks

You can produce up to four additional clocks which are created from the same MMCM that generates `ui_clk`. Additional clocks can be selected from the **Clock Options** section in the **Advanced Options** tab. The GUI lists the possible clock frequencies from MMCM and the frequencies for additional clocks vary based on selected memory frequency (**Memory Device Interface Speed (ps)** value in the **Basic** tab), selected FPGA, and FPGA speed grade.

Reduce System Noise during Calibration

The system design should be as quiet as possible during the calibration process. In particular, the Soft Error Mitigation (SEM) IP, if used, should be disabled during calibration. For calibration that occurs immediately after the configuration or reconfiguration of the FPGA, use the ICAP arbitration interface to hold off the SEM IP in the boot stage. For more information on the ICAP Arbitration Interface, see "ICAP Arbitration Interface" section in Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10].

For situations where the memory interface is reset and recalibrated without a reconfiguration of the FPGA, the SEM IP must be set into IDLE state to disable the memory scan and to send the SEM IP back into the scanning (Observation or Detect only) states afterwards. This can be done in two methods, through the "Command Interface" or "UART interface." See Chapter 3 of the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 10] for more information.

Resets

An asynchronous reset (`sys_rst`) input is provided. This active-High reset must assert for a minimum of 20 cycles of the FPGA logic clock.

For more information on reset, see the Reset Sequence in Chapter 24, Core Architecture.

PCB Guidelines for QDR-IV SRAM

Strict adherence to all documented QDR-IV SRAM PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

Pin and Bank Rules

QDR-IV Pin Rules

This section describes the pin out rules for QDR-IV SRAM (XP and HP) interface.

1. Only HP banks of the FPGA device are supported.
2. Data Group Definition: DQ pins, associated QK/QK# pins, associated DK/DK# pins, QVLD pin of corresponding port.
    a. Association of DQ, QK/QK#, DK/DK#, and QVLD pins are as per the QDR-IV data sheet (defined by Cypress® Semiconductor).
    b. For x18 component, PORT A, DQA[8:0], QKA[0]/QKA#[0], and DKA[0]/DKA#[0] associate a Data group; DQA[17:9], QKA[1]/QKA#[1]. DKA[1]/DKA#[1] associate with another Data group. Similar association is followed for PORT B.
    c. For x36 component, PORT B, DQB[17:0], QKB[0]/QKB#[0], and DKB[0]/DKB#[0] associate a Data group; DQB[35:18], QKB[1]/QKB#[1]. DKB[1]/DKB#[1] associate with another Data group. Similar association is followed for PORT A.
3. Address/Control Group Definition: A, CK/CK#, AP, PE#, AINV, LDA#, LDB#, RWA#, RWB#, CFG#, RST#, LBK0#, and LBK1# pins of a single memory component.
4. All signal groups of the memory interface (that is, Data group, Address/Control group, and system clock) must be selected in a single column of banks.
5. All of the Address/Control group and Data group pins of a given memory interface design must be allocated within three consecutive banks, no skip banks allowed.
6. Address/Control group must be allocated in the center bank.
7. Pin association within Data and Address groups is strictly followed.
8. Data Group (x18 Component):
    a. All the data groups of a single PORT must be allocated in a single bank.
    b. All the pins of a single data group must be allocated within two consecutive byte lanes in a given bank.
    c. DQ and associated QK/QK# and QVLD of a single data group must be allocated in a single byte lane.
    d. DQ pins allocation:

        All the DQ pins of a single data group can be allocated to any I/O pin except pin 1, pin 7, and pin 12 of the given byte lane.

    e. QK/QK# pin allocation:

All the QK/QK# pair of a single data group must be allocated to pin 0/pin 1 pair or pin 6/pin7 pair of the given byte lane.

f. DK/DK# pin allocation:
   - All the DK/DK# pair of a single data group can be allocated to any differential pin pair.
   - All the DK/DK# pair of a single data group can be allocated in the same or consecutive byte lane of the DQ pins (of the same data group) allocated byte lane.

g. QVLD pin allocation:
   - QVLD of a single data group can be allocated to any I/O pin.
   - QVLD is not utilized in the current design. It is reserved for future use.

h. Only DK/DK# pins of different data groups from a single PORT of a single component can share byte lanes. This rule should be used in conjunction with the above mentioned step a and step b.

i. Data groups of PORT A and PORT B of a single component cannot share byte lanes.

j. See the QDR-IV Pinout Examples, page 424.

9. Data Group (x36 Component):

a. All the data groups of a single PORT must be allocated in a single bank.

b. All the pins of a single data group must be allocated within two consecutive byte lanes in a given bank.

c. DQ pins allocation:
   - All the DQ pins of a single data group can be allocated to any I/O pin except pin 1, pin 7, and pin 12 of any given byte lane.
   - DQ 0 to 8 pins of a single data group must be allocated in a single byte lane. Similarly, DQ 9 to 17 pins of the same data group must be allocated in the next consecutive byte lane of the same bank. Likewise, DQ 18 to 26 and DQ 27 to 35 must be allocated in consecutive byte lanes, respectively.

d. QK/QK# pin allocation:

   All the QK/QK# pair of a single data group must be allocated to pin 0/pin 1 pair or pin 6/pin7 pair of byte lanes 1 or 2.

e. DK/DK# pin allocation:

   All the DK/DK# pair of a single data group can be allocated to any differential pin pair.

f. QVLD pin allocation:

    - QVLD of a single data group can be allocated to any I/O pin.
    - QVLD is not utilized in the current design. It is reserved for future use.

  g. Data groups of PORT A and PORT B of a single component cannot share byte lanes.

  h. See the QDR-IV Pinout Examples, page 424.

10. Address/Control Group:

  a. All the address/control group pins must be allocated in a single bank.

  b. Pins A, AP, PE#, AINV, LDA#, LDB#, RWA#, RWB#, CFG#, RST#, LBK0#, and LBK1# of the design can be allocated to any I/O pin.

  c. CK/CK# pin allocation:

    - CK/CK# pair must be allocated only in byte lanes 1 or 2 (whichever is center to the A, AP, and AINV allocated byte lanes) of a given bank.
    - CK/CK# pair can be allocated to any I/O differential pin pair.

  d. Pins A, AP, and AINV must be allocated in consecutive 3-byte lanes (only) in a given bank.

11. System clock pins (**`sys_clk_p`**/**`sys_clk_n`**) must be placed on any GCCIO pin pair in the same column and same SLR as that of the memory interface. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements and MMCM requirements should be considered).

12. One **`vrp`** pin per bank is used and DCI is required for the interfaces. A **`vrp`** pin is required in I/O banks containing inputs as well as in output only banks. It is required in output only banks because address/control signals use POD12_DCI to enable usage of controlled output impedance. DCI cascade is allowed. When DCI cascade is selected, **`vrp`** pin can be used as a normal I/O. All rules for the DCI in the *UltraScale™ Device FPGAs SelectIO™ Resources User Guide* (UG571) [Ref 7] must be followed.

RECOMMENDED: *Xilinx strongly recommends that the DCIUpdateMode option is kept with the default value of ASREQUIRED so that the DCI circuitry is allowed to operate normally.*

---

13. There are dedicated V_{REF} pins (not included in the rules above). Either internal or external V_{REF} is permitted. If an external V_{REF} is not used, the V_{REF} pins must be pulled to ground by a resistor value specified in the *UltraScale™ Device FPGAs SelectIO™ Resources User Guide* (UG571) [Ref 7]. These pins must be connected appropriately for the standard in use.

14. The system reset pin (**`sys_rst_n`**) must not be allocated to Pins N0 and N6 if the byte is used for the memory I/Os.

IMPORTANT: *QDR-IV IP does not support data inversion. Contact your memory vendor for terminating DINVA and DINVB at memory.*

QDR-IV Pinout Examples

Table 25-1 shows an example of an 36-bit QDR-IV SRAM interface contained within three banks.

Table 25-1: **36-Bit QDR-IV Interface Contained in Three Banks**

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
44	DQB6	T0	N0	–	BC31
	–		N1	–	BD31
	DQB1		N2	–	BB29
	DQB3		N3	–	BC29
	DQB2		N4	–	BB31
	DQB4		N5	–	BC32
	DKB0_P		N6	–	BD29
	DKB0_N		N7	–	BD30
	DQB0		N8	–	BA32
	DQB8		N9	–	BB32
	DQB7		N10	–	BA30
	DQB5		N11	–	BB30
	–		N12	VRP	BA29
44	DQB12	T1	N0	–	AY30
	QVLDB0		N1	–	AY31
	DQB9		N2	–	AW31
	DQB11		N3	–	AY32
	DQB15		N4	–	AW29
	DQB10		N5	–	AW30
	QKB0_P		N6	–	AV33
	QKB0_N		N7	–	AW33
	DQB17		N8	GCIO_P_3	AU30
	DQB13		N9	GCIO_N_3	AU31
	DQB14		N10	GCIO_P_4	AV31
	DQB16		N11	GCIO_N_4	AV32
	–		N12	–	AV29

Table 25-1: **36-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
44	DQB24	T2	N0	GCIO_P_1	AT32
	–		N1	GCIO_N_1	AU32
	DQB19		N2	GCIO_P_2	AT29
	DQB20		N3	GCIO_N_2	AU29
	DQB22		N4	–	AR32
	DQB21		N5	–	AR33
	QKB1_P		N6	–	AR28
	QKB1_N		N7	–	AT28
	DQB23		N8	–	AP30
	DQB26		N9	–	AR31
	DQB25		N10	–	AR30
	DQB18		N11	–	AT30
	–		N12	–	AT33
44	DQB27	T3	N0	–	AN31
	QVLDB1		N1	–	AP31
	DQB29		N2	–	AN32
	DQB33		N3	–	AP33
	DQB32		N4	–	AP28
	DQB34		N5	–	AP29
	DKB1_P		N6	–	AM30
	DKB1_N		N7	–	AM31
	DQB31		N8	–	AM29
	DQB35		N9	–	AN29
	DQB28		N10	–	AL29
	DQB30		N11	–	AL30
	–		N12	–	AN28

Table 25-1: **36-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
45	–	T0	N0	–	BC28
	–		N1	–	BD28
	–		N2	–	BD25
	–		N3	–	BD26
	–		N4	–	BB27
	–		N5	–	BC27
	–		N6	–	BC24
	–		N7	–	BD24
	–		N8	–	BB26
	–		N9	–	BC26
	–		N10	–	BB24
	–		N11	–	BB25
	–		N12	VRP	BA28
45	–	T1	N0	–	AW28
	PE_N		N1	–	AY28
	RWB_N		N2	–	BA24
	LDB_N		N3	–	BA25
	RWA_N		N4	–	AY27
	LDA_N		N5	–	BA27
	LBK1_N		N6	–	AW25
	LBK0_N		N7	–	AY25
	–		N8	GCIO_P_4	AV26
	–		N9	GCIO_N_4	AV27
	RST_N		N10	GCIO_P_3	AW26
	CFG_N		N11	GCIO_N_3	AY26
	–		N12	–	AV28

Table 25-1: **36-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
45	A18	T2	N0	GCIO_P_2	AU25
	A20		N1	GCIO_N_2	AU26
	A17		N2	GCIO_P_1	AR25
	A16		N3	GCIO_N_1	AT25
	A15		N4	–	AT27
	A14		N5	–	AU27
	CK_P		N6	–	AP25
	CK_N		N7	–	AP26
	A13		N8	–	AR26
	A12		N9	–	AR27
	A11		N10	–	AN24
	A10		N11	–	AP24
	AP		N12	–	AT24
45	A9	T3	N0	–	AN26
	A19		N1	–	AN27
	A8		N2	–	AK25
	A7		N3	–	AL25
	A6		N4	–	AM26
	A5		N5	–	AM27
	A4		N6	–	AK27
	AINV		N7	–	AL27
	A3		N8	–	AM24
	A2		N9	–	AM25
	A1		N10	–	AJ26
	A0		N11	–	AK26
	–		N12	–	AL24

Table 25-1: **36-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
46	DQA12	T0	N0	–	BA33
	–		N1	–	BB34
	DQA15		N2	–	BC33
	DQA11		N3	–	BD33
	DQA10		N4	–	BA34
	DQA16		N5	–	BA35
	DKA0_P		N6	–	BC34
	DKA0_N		N7	–	BD34
	DQA9		N8	–	BB35
	DQA17		N9	–	BC36
	DQA14		N10	–	BD35
	DQA13		N11	–	BD36
	–		N12	VRP	AY33
46	DQA7	T1	N0	–	BC39
	QVLDA0		N1	–	BD39
	DQA5		N2	–	BB36
	DQA3		N3	–	BC37
	DQA8		N4	–	BA38
	DQA1		N5	–	BB39
	QKA0_P		N6	–	BC38
	QKA0_N		N7	–	BD38
	DQA0		N8	GCIO_P_1	AY37
	DQA4		N9	GCIO_N_1	AY38
	DQA2		N10	GCIO_P_2	BA37
	DQA6		N11	GCIO_N_2	BB37
	–		N12	–	BA39

Table 25-1: **36-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
46	DQA19	T2	N0	GCIO_P_4	AV36
	–		N1	GCIO_N_4	AW36
	DQA18		N2	GCIO_P_3	AY35
	DQA26		N3	GCIO_N_3	AY36
	DQA25		N4	–	AW38
	DQA20		N5	–	AW39
	QKA1_P		N6	–	AW34
	QKA1_N		N7	–	AW35
	DQA24		N8	–	AU39
	DQA23		N9	–	AV39
	DQA22		N10	–	AV37
	DQA21		N11	–	AV38
	–		N12	–	AV34
46	DQA32	T3	N0	–	AT34
	QVLDA1		N1	–	AT35
	DQA35		N2	–	AU34
	DQA31		N3	–	AU35
	DQA30		N4	–	AR39
	DQA27		N5	–	AT39
	DKA1_P		N6	–	AU36
	DKA1_N		N7	–	AU37
	DQA33		N8	–	AR37
	DQA29		N9	–	AR38
	DQA34		N10	–	AT37
	DQA28		N11	–	AT38
	–		N12	–	AR36

Table 25-2 shows an example of an 18-bit QDR-IV SRAM interface contained within three banks.

Table 25-2: **18-Bit QDR-IV Interface Contained in Three Banks**

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
49	–	T0	N0	–	L38
	–		N1	–	L39
	–		N2	–	K38
	–		N3	–	J38
	–		N4	–	J39
	–		N5	–	H39
	–		N6	–	H37
	–		N7	–	G37
	–		N8	–	H38
	–		N9	–	G39
	DKB1_P		N10	–	F38
	DKB1_N		N11	–	F39
	–		N12	VRP	K37
49	QKB1_P	T1	N0	–	K36
	QKB1_N		N1	–	J36
	DQB9		N2	–	J33
	DQB13		N3	–	H34
	DQB12		N4	–	J35
	DQB15		N5	–	H36
	DQB17		N6	–	H33
	QVLDB1		N7	–	G34
	DQB16		N8	GCIO_P_1	F34
	DQB10		N9	GCIO_N_1	F35
	DQB14		N10	GCIO_P_2	G35
	DQB11		N11	GCIO_N_2	G36
	–		N12	–	F33

Table 25-2: **18-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
49	QKB0_P	T2	N0	GCIO_P_3	F37
	QKB0_N		N1	GCIO_N_3	E38
	DQB6		N2	GCIO_P_4	E36
	DQB8		N3	GCIO_P_3	E37
	DQB5		N4	–	D39
	DQB7		N5	–	C39
	DQB4		N6	–	D38
	QVLDB0		N7	–	C38
	DQB0		N8	–	D36
	DQB3		N9	–	C36
	DQB1		N10	–	B37
	DQB2		N11	–	A37
	–		N12	–	C37
49	DKB0_P	T3	N0	–	E35
	DKB0_N		N1	–	D35
	–		N2	–	D34
	–		N3	–	C34
	–		N4	–	D33
	–		N5	–	C33
	–		N6	–	B35
	–		N7	–	B36
	–		N8	–	B34
	–		N9	–	A35
	–		N10	–	A33
	–		N11	–	A34
	–		N12	–	E33

Table 25-2: **18-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
50	–	T0	N0	–	R27
	–		N1	–	R28
	–		N2	–	M30
	–		N3	–	L30
	–		N4	–	P28
	–		N5	–	P29
	–		N6	–	N29
	–		N7	–	M29
	–		N8	–	N27
	–		N9	–	N28
	–		N10	–	L28
	–		N11	–	L29
	–		N12	VRP	T28
50	–	T1	N0	–	K30
	PE_N		N1	–	J30
	RWB_N		N2	–	K31
	LDB_N		N3	–	J31
	RWA_N		N4	–	J28
	LDA_N		N5	–	J29
	LBK1_N		N6	–	H32
	LBK0_N		N7	–	G32
	–		N8	GCIO_P_4	H28
	–		N9	GCIO_N_4	H29
	RST_N		N10	GCIO_P_3	H31
	CFG_N		N11	GCIO_N_3	G31
	–		N12	–	K28

Table 25-2: **18-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
50	A18	T2	N0	GCIO_P_2	G30
	A20		N1	GCIO_N_2	F30
	A17		N2	GCIO_P_1	G29
	A16		N3	GCIO_N_1	F29
	A15		N4	–	F32
	A14		N5	–	E32
	CK_P		N6	–	E30
	CK_N		N7	–	D30
	A13		N8	–	E31
	A12		N9	–	D31
	A11		N10	–	F28
	A10		N11	–	E28
	AP		N12	–	D29
50	A9	T3	N0	–	C31
	A19		N1	–	C32
	A8		N2	–	B30
	A7		N3	–	B31
	A6		N4	–	B32
	A5		N5	–	A32
	A4		N6	–	A29
	AINV		N7	–	A30
	A3		N8	–	C29
	A2		N9	–	B29
	A1		N10	–	D28
	A0		N11	–	C28
	–		N12	–	A28

Table 25-2: **18-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
51	QKA0_P	T0	N0	–	T25
	QKA0_N		N1	–	R25
	DQA1		N2	–	T23
	DQA8		N3	–	R23
	DQA4		N4	–	R26
	DQA6		N5	–	P26
	DQA0		N6	–	P24
	QVLDA0		N7	–	N24
	DQA7		N8	–	P25
	DQA5		N9	–	N26
	DQA2		N10	–	P23
	DQA3		N11	–	N23
	–		N12	VRP	M25
51	DKA0_P	T1	N0	–	M26
	DKA0_N		N1	–	M27
	–		N2	–	M24
	–		N3	–	L25
	–		N4	–	L27
	–		N5	–	K27
	–		N6	–	L23
	–		N7	–	L24
	–		N8	GCIO_P_4	K25
	–		N9	GCIO_N_4	J25
	DKA1_P		N10	GCIO_P_2	K26
	DKA1_N		N11	GCIO_N_2	J26
	–		N12	–	J24

Table 25-2: **18-Bit QDR-IV Interface Contained in Three Banks *(Cont'd)***

Bank	Signal Name	Byte Group	Byte Group I/O Number	Special Designation	Pin Number
51	QKA1_P	T2	N0	GCIO_P_3	G25
	QKA1_N		N1	GCIO_N_3	F25
	DQA12		N2	GCIO_P_1	H26
	DQA9		N3	GCIO_N_1	G26
	DQA10		N4	–	F27
	DQA13		N5	–	E27
	DQA15		N6	–	H27
	QVLDA1		N7	–	G27
	DQA16		N8	–	E25
	DQA17		N9	–	E26
	DQA11		N10	–	G24
	DQA14		N11	–	F24
	–		N12	–	H24
51	–	T3	N0	–	D26
	–		N1	–	C27
	–		N2	–	B27
	–		N3	–	A27
	–		N4	–	C26
	–		N5	–	B26
	–		N6	–	B25
	–		N7	–	A25
	–		N8	–	D24
	–		N9	–	D25
	–		N10	–	C24
	–		N11	–	B24
	–		N12	–	A24

Protocol Description

This core has the following interfaces:

- Memory Interface
- User Interface
- Physical Interface

Memory Interface

The QDR-IV SRAM core is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top-level of the core.

User Interface

The user interface connects to an FPGA user design to the QDR-IV SRAM core to simplify interactions between the user design and the external memory device. The user interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in Table 25-3.

Parameters and values:

- **CH_ADDR_WIDTH** – Address width of a channel. It depends on the QDR-IV SRAM. memory part.
- **CH_DATA_WIDTH** – Data bus width of a channel. It is twice the memory data width. For example, 36 for 18-bit memory interface and 72 for 36-bit memory interface.
- **CH_NUM** – Number of channels. The value is fixed to be 4.
- **CH_CMD_WIDTH** – Number of bits required to represent a command. The value is fixed to be 2.

Table 25-3: **User Interface Signals**

Signal	I/O	Description
sys_clk_p	I	P clock of input differential clock for internal MMCM.
sys_clk_n	I	N clock of input differential clock for internal MMCM.
sys_rst	I	System Reset. Brings the design to initial state.
ui_clk	O	FPGA logic clock to generate traffic for the IP.
ui_rst	O	Reset coming out from the design.
init_calib_complete	O	Calibration Completion status.

Table 25-3: **User Interface Signals *(Cont'd)***

Signal	I/O	Description
app_addr_a_ch0[ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT A corresponding to channel 0.
app_addr_a_ch1[ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT A corresponding to channel 1.
app_addr_a_ch2[ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT A corresponding to channel 2.
app_addr_a_ch3[ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT A corresponding to channel 3.
app_cmd_a_ch0[CH_CMD_WIDTH–1:0]	I	Write, Read, or NOP command to the UI for PORT A corresponding to channel 0.
app_cmd_a_ch1[CH_CMD_WIDTH–1:0]	I	Write, Read, or NOP command to the UI for PORT A corresponding to channel 1.
app_cmd_a_ch2[CH_CMD_WIDTH–1:0]	I	Write, Read, or NOP command to the UI for PORT A corresponding to channel 2.
app_cmd_a_ch3[CH_CMD_WIDTH–1:0]	I	Write, Read, or NOP command to the UI for PORT A corresponding to channel 3.
app_cmd_en_a	I	This signal is used to indicate to the User Interface that the input commands for PORT A are valid provided app_cmd_rdy_a is High. If app_cmd_rdy_a is Low, PORT A commands are ignored.
app_wrdata_a_ch0[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT A corresponding to channel 0.
app_wrdata_a_ch1[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT A corresponding to channel 1.
app_wrdata_a_ch2[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT A corresponding to channel 2.
app_wrdata_a_ch3[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT A corresponding to channel 3.
app_cmd_rdy_a	O	This signal is used to indicate the user that UI is ready to accept new commands. ***Note:*** PORT A commands are not processed by the User Interface when app_cmd_rdy_a signal is Low.
app_rddata_a_ch0[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT A corresponding to channel 0.
app_rddata_a_ch1[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT A corresponding to channel 1.
app_rddata_a_ch2[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT A corresponding to channel 2.
app_rddata_a_ch3[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT A corresponding to channel 3.
app_rddata_valid_a[CH_NUM–1:0]	O	Read data valid of PORT A for four channels where Bit[0] corresponds to channel 0 and etc.

Table 25-3: **User Interface Signals *(Cont'd)***

Signal	I/O	Description
app_addr_b_ch0[CH_ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT B corresponding to channel 0.
app_addr_b_ch1[CH_ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT B corresponding to channel 1.
app_addr_b_ch2[CH_ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT B corresponding to channel 2.
app_addr_b_ch3[CH_ADDR_WIDTH–1:0]	I	Address of write/read commands data of PORT B corresponding to channel 3.
app_cmd_b_ch0[CH_CMD_WIDTH–1:0]	I	Write, Read or NOP command to the UI for PORT B corresponding to channel 0.
app_cmd_b_ch1[CH_CMD_WIDTH–1:0]	I	Write, Read or NOP command to the UI for PORT B corresponding to channel 1.
app_cmd_b_ch2[CH_CMD_WIDTH–1:0]	I	Write, Read or NOP command to the UI for PORT B corresponding to channel 2.
app_cmd_b_ch3[CH_CMD_WIDTH–1:0]	I	Write, Read or NOP command to the UI for PORT B corresponding to channel 3.
app_cmd_en_b	I	This signal is used to indicate to the User Interface that the input commands for PORT B are valid provided app_cmd_rdy_b is High. If app_cmd_rdy_b is Low, PORT B commands are ignored.
app_wrdata_b_ch0[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT B corresponding to channel 0.
app_wrdata_b_ch1[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT B corresponding to channel 1.
app_wrdata_b_ch2[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT B corresponding to channel 2.
app_wrdata_b_ch3[CH_DATA_WIDTH–1:0]	I	Write Data for write commands of PORT B corresponding to channel 3.
app_cmd_rdy_b	O	This signal is used to indicate the user that UI is ready to accept new commands. ***Note:*** No commands for PORT B are processed by the UI when this signal is Low.
app_rddata_b_ch0[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT B corresponding to channel 0.
app_rddata_b_ch1[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT B corresponding to channel 1.
app_rddata_b_ch2[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT B corresponding to channel 2.
app_rddata_b_ch3[CH_DATA_WIDTH–1:0]	O	Read Data of Read commands of PORT B corresponding to channel 3.
app_rddata_valid_b[CH_NUM–1:0]	O	Read data valid of PORT B for four channels where Bit[0] corresponds to channel 0 and etc.

Controller Features

The QDR-IV SRAM memory controller is designed to take read and write commands from the user interface and converts them so that they become compatible to the QDR-IV SRAM memory protocol. Also, it ensures that the commands to the memory are handled with low latencies meeting all the QDR-IV SRAM memory timing requirements.

The best efficiency from the controller is achieved when there is unidirectional traffic on each port, without any bank collision in them, without the command switch from read to write, or vice-versa. When there are alternate read/write commands, the efficiency is lost because the bidirectional QDR-IV SRAM data bus needs to be turned around. Also when there is bank collision, the controller has to add up latencies to avoid collision at the memory interface which reduces efficiency. Because there are four channels per port, which can be used for sending the command to the memory, you should know the command order and priorities. The following sections describe these in detail.

Command Order to the Memory

Figure 25-2 shows the command order when there is no command switch from read to write (vice-versa) and no bank collision. PORT A is called first by the controller followed by PORT B. This gets repeated in the same method.

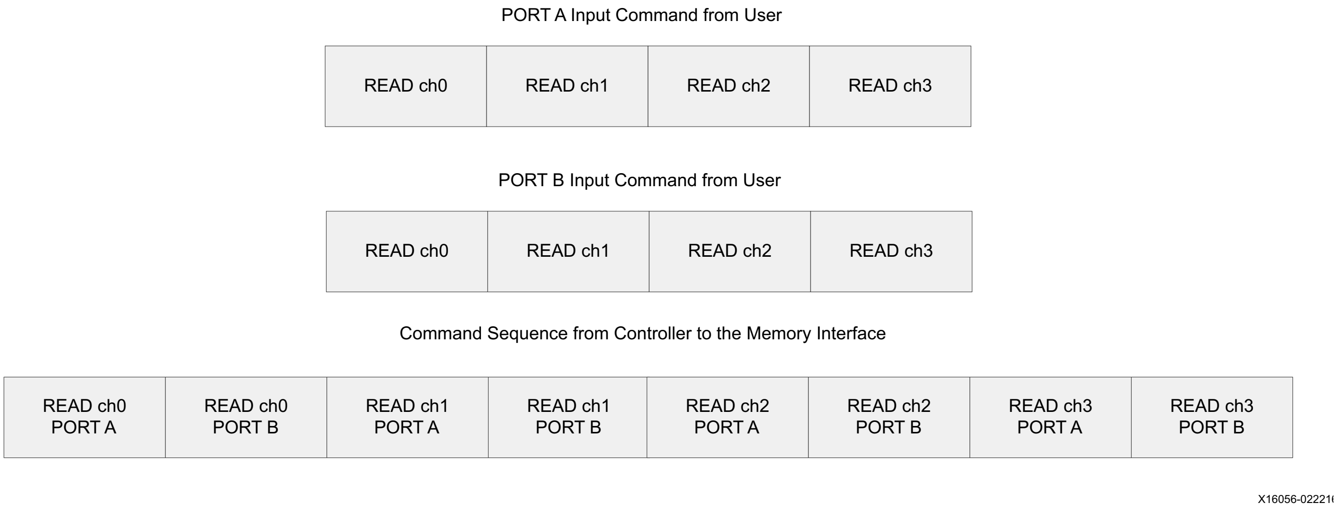


Figure 25-2: **Command Order**

Bank Collision

Note: HP memory devices do not have bank access restriction so bank collision does not apply when you are dealing with HP memory devices.

The last three bits of the address denote which bank out of the eight available banks in the memory device is being accessed. The rule as per the memory access for XP part is that PORT B cannot access the same bank in the same clock cycle as PORT A. Because there are four channels, the bank comparison for collision is done on a per-channel basis. If a collision is found on any of the four channels, all of the four channels of the corresponding ports are affected as explained in the later sections.

For detecting whether there is a collision, the last three bits of the channel address are compared. The following conditions are checked for collision detection:

1. Comparison is done channel-wise. That is, PORT A channel 0 is compared with PORT B channel 0. PORT A channel 0 is never compared with PORT B channel 1 or any other channel.
2. Only the last three bits of the channel address is compared. They all should match for collision to be detected.
3. Restriction for accessing the same bank in the same cycle only lies with PORT B and not on PORT A. This means that for detecting the collision, the last three bits of channel 0 PORT A is compared with the last three bits of channel 0 of PORT B.

   PORT B channel 0 is not compared with PORT A channel 1 for collision detection. This is illustrated in Figure 25-3.

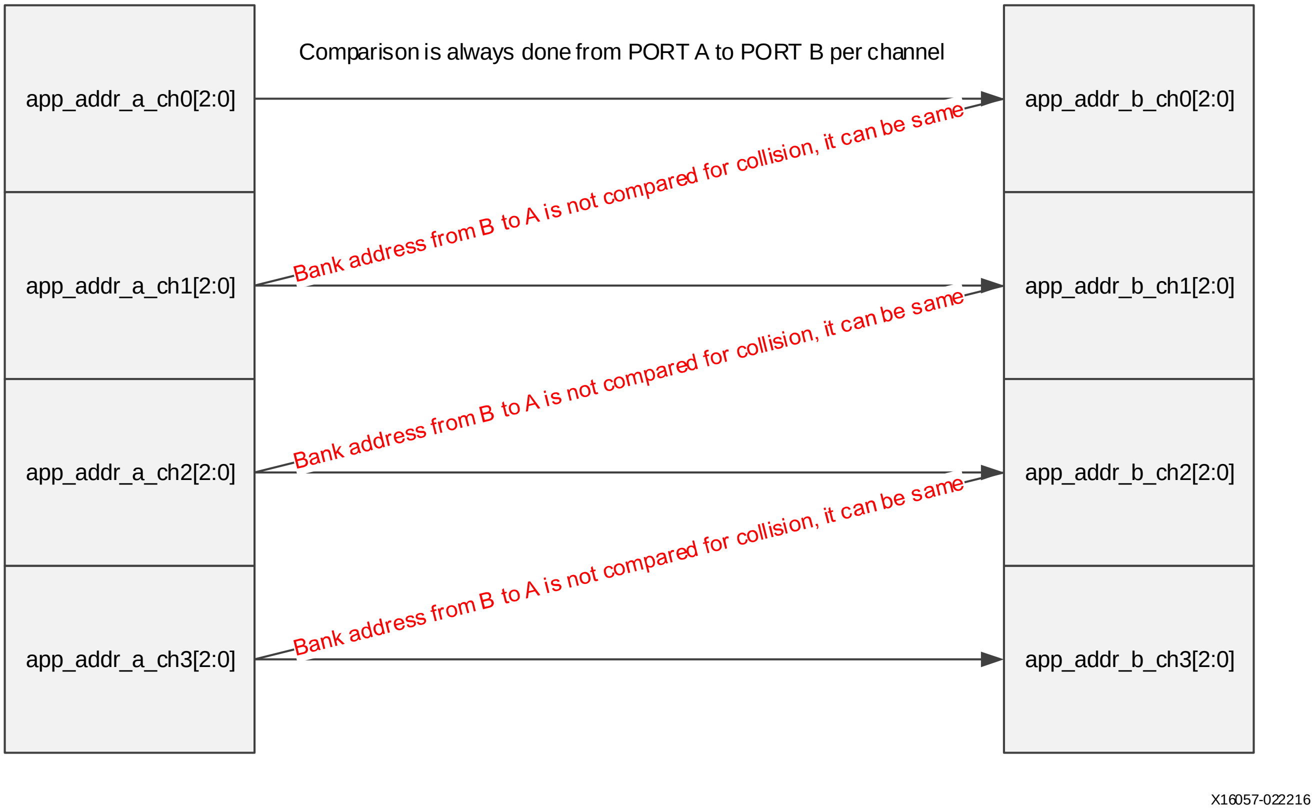


Figure 25-3: **Bank Address Comparison for Collision**

Because of the four channels per port sending commands in every clock, there is an order in which the command is called by the controller towards the memory.

Figure 25-4 shows how the bank collision signals for PORT A and PORT B are asserted. To begin, the priority remains with PORT A for accessing any bank. If PORT B tries to access (read command or write command) the same bank in the same clock, it is considered a PORT B collision. Therefore, **`Bank_collision_B`** is asserted and controller delays the processing of PORT B command by one user clock.

If PORT A accesses the same bank again after the pending PORT B command is serviced, it is considered a PORT A collision. PORT A command processing is delayed by one user clock. This is done to provide equal opportunity to both the ports in case they are trying to access the same bank back-to-back.

Figure 25-4 takes channel 0 as an example and is true for all the channels.

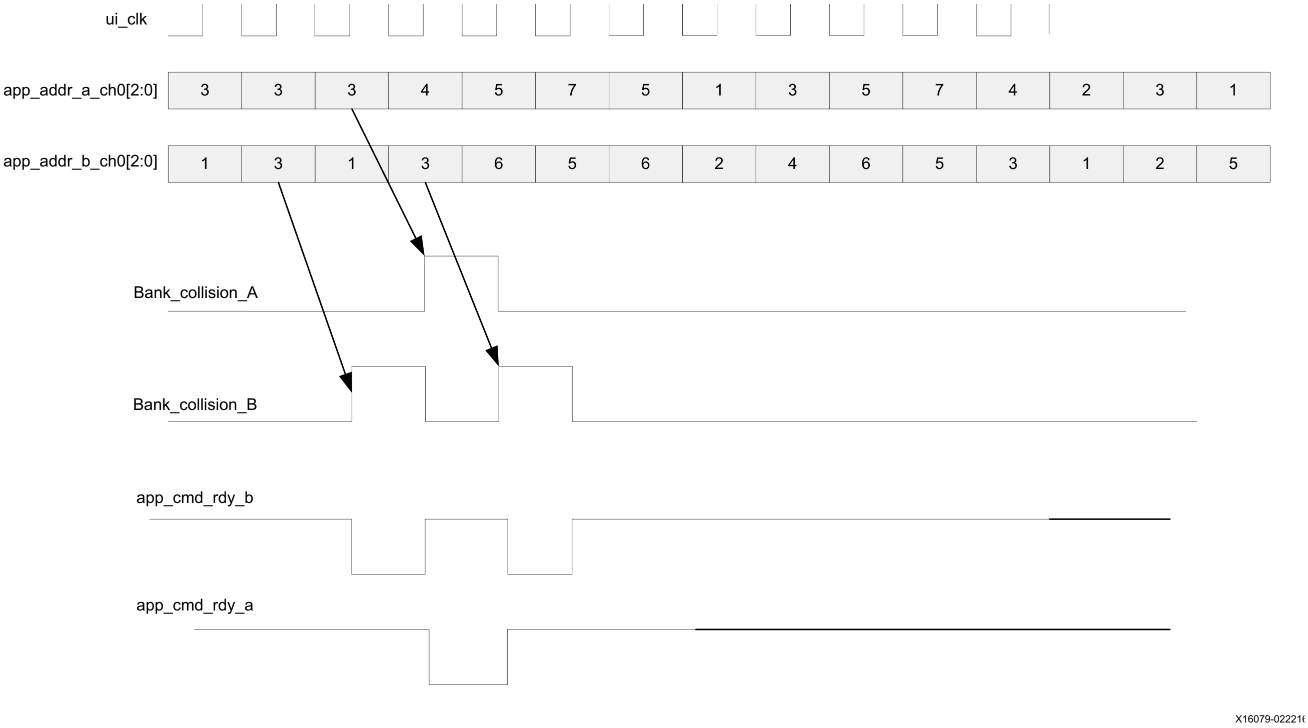


Figure 25-4: **Bank Collision Signals for PORT A and PORT B**

From Figure 25-4, there is a bank collision on PORT A or B and a corresponding **`ready`** is deasserted. It is your responsibility to ensure that it should not issue any command after it samples a **`ready`** to be Low (that is, hold its next command transaction until the **`ready`** gets asserted back). If you are not sure, the command issued while **`ready`** is Low is lost for all the four channels corresponding to the port as **`app_cmd_rdy_a`** and **`app_cmd_rdy_b`** are common for all four channels.

Note: Because there are four channels per port, collision on one channel delays the command processing for all four channels for that port. The following simulation snapshot explains this scenario (Figure 25-5).

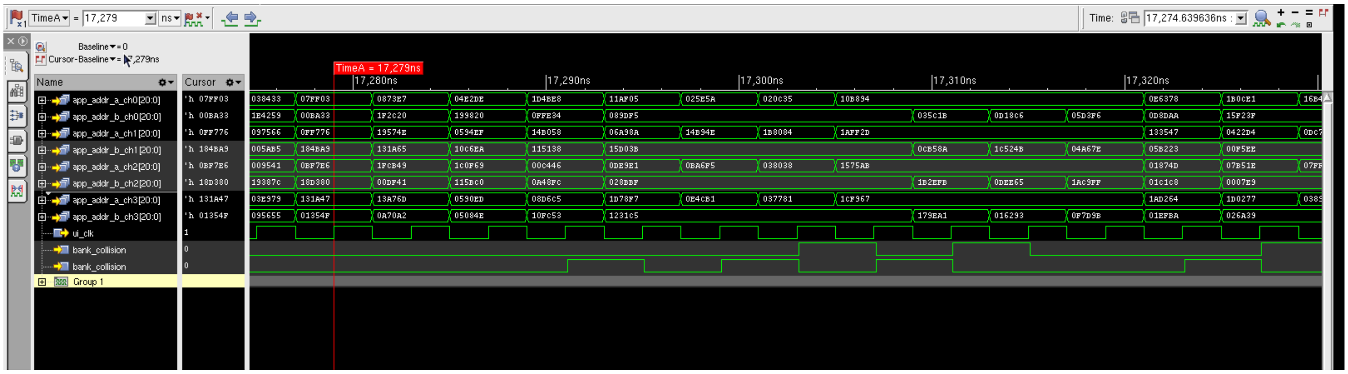


Figure 25-5: **QDR-IV Bank Collision Simulation**

For example, the first collision occurs on PORT B channel 0 because PORT B wants to access the same bank in the same user clock. Then, the controller stores PORT B commands for all four channels and deasserts the PORT B `ready` signal. It is your responsibility to hold the next set of commands for PORT B until the user interface asserts the `ready` signal. If you issue another set of commands when the `ready` is Low, those commands for all four channels are lost and it does not go to the memory interface.

Channel Wise Command Order to the Memory

Figure 25-6 shows the channel wise command order when there is no command switch from read to write (vice-versa) and no collision. Channel 0 PORT A is sent to the memory interface first, followed by channel 0 PORT B, followed by channel 1 PORT A, followed by channel 1 PORT B, etc.

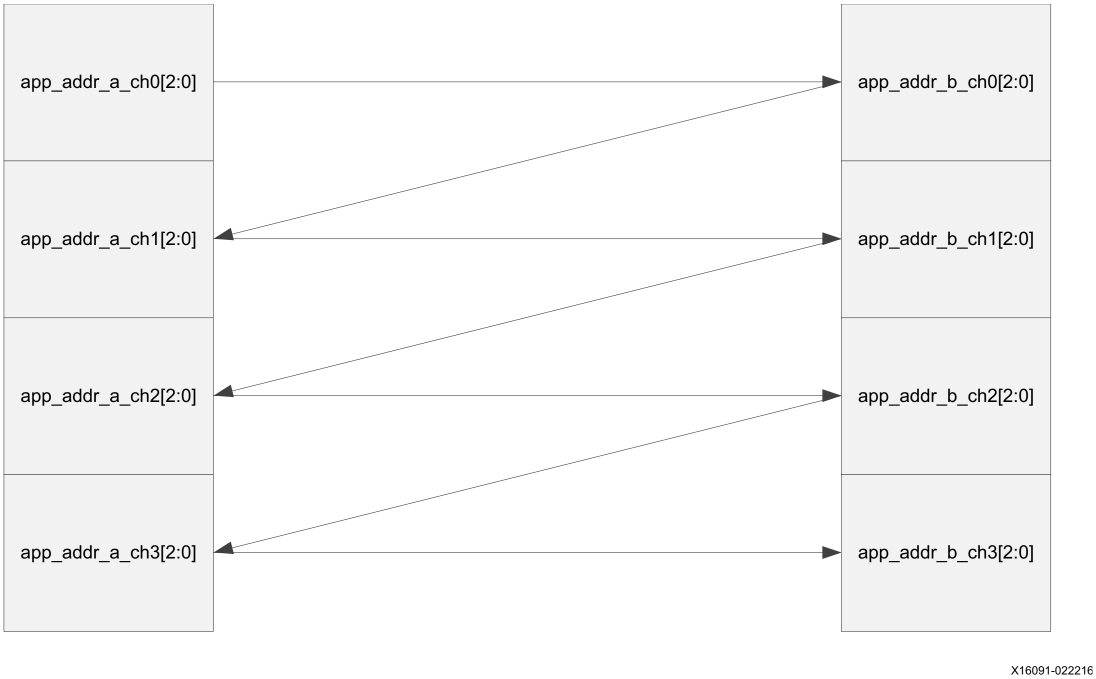


Figure 25-6: **Command Order to the Memory**

Examples

The following section focuses on your command sequence and the controller after handling the command switching/collision processes and distributing to the memory interface.

In the first case, the command sequence is located at the input and output of the controller when there is no collision. The PORT A command has switched from read to write. As explained in the earlier section, the data bus is common for read and write and it has to switch direction. Therefore, it has to wait for the read command to be completed.

The controller introduces eight NOPs on PORT A to avoid the bus contention at the memory interface (see Figure 25-7). Because the write latency of the memory device is less than the read latency when the command switches from write to read, the controller inserts four NOPs between the commands. All PORT A commands are sent on the rising edge of the memory clock (CK clock shown in Figure 25-7) and all PORT B commands are sent on the falling edge.

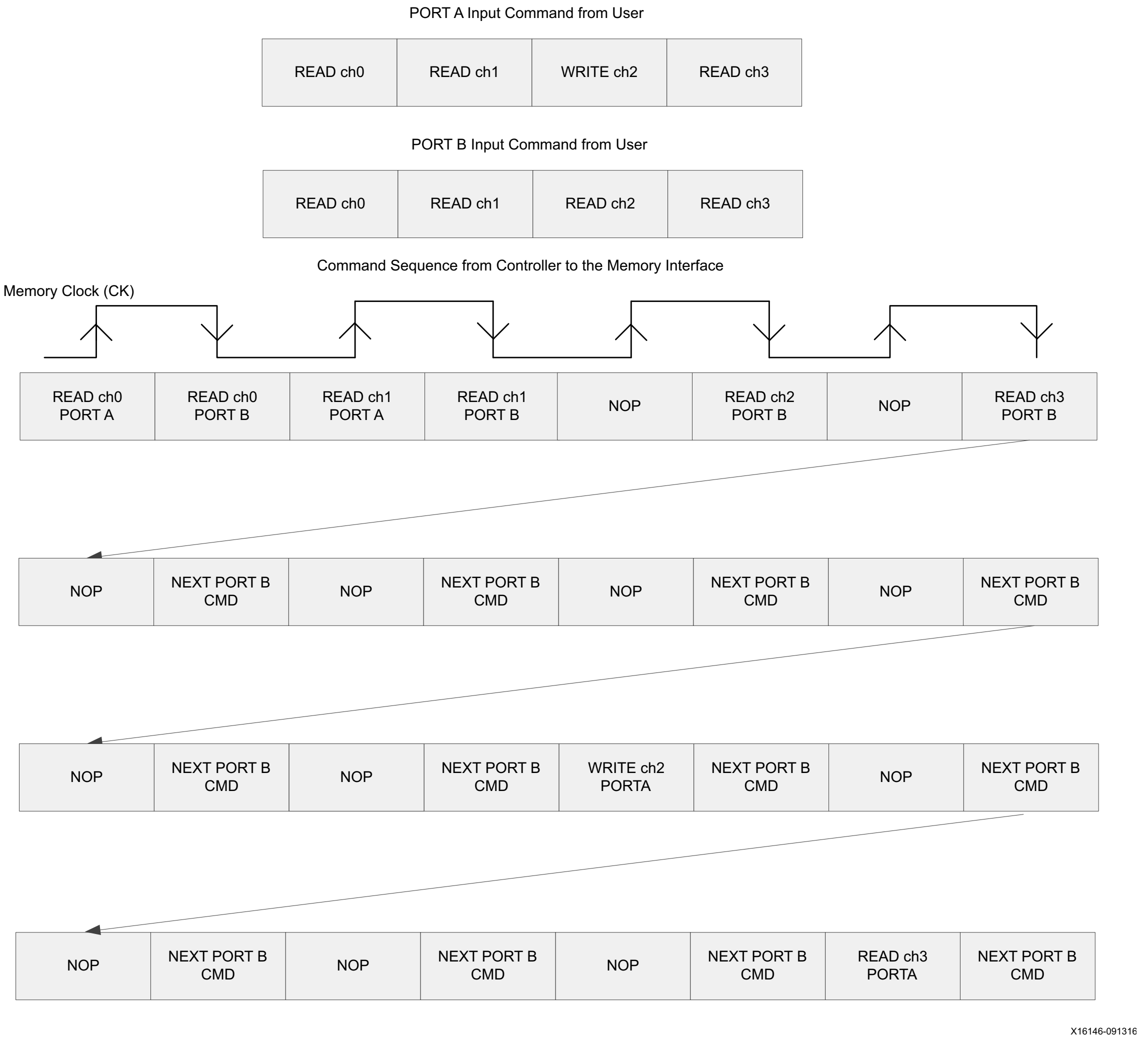


Figure 25-7: **Case 1: No Bank Collision, Read to Write Command Switching**

In the second case, there is a collision between the channel 0 of PORT A and PORT B, but there is no command switching. The collision on PORT B results in its command processing getting delayed by one user clock. The controller now serves the next PORT A command to avoid bank rule violation at the memory. This is shown in Figure 25-8 where the rising edge on all PORT A commands are sent to the memory, but on the falling edge. Four NOPs are inserted first and then pending PORT B commands.

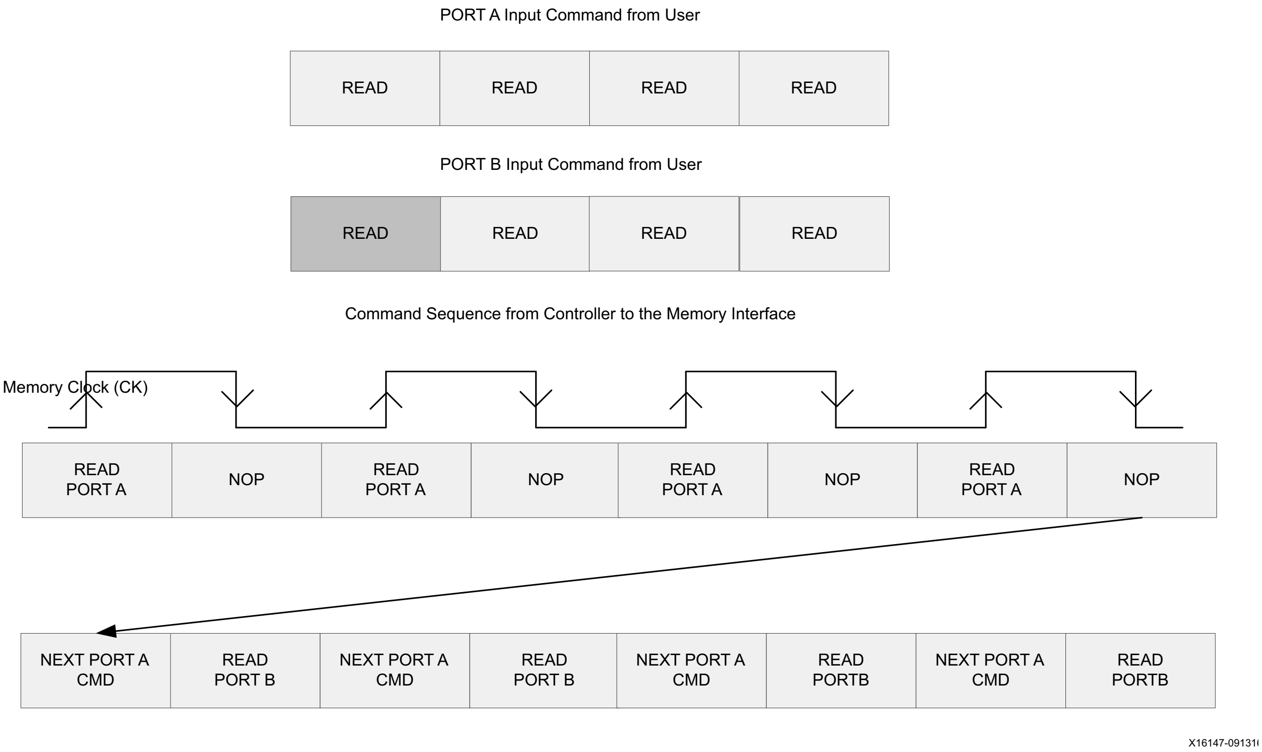


Figure 25-8: **Case 2: Bank Collision, No Command Switching**

Finally, the next and worst case is when there are bank collision and command switching. First, PORT B has a collision and its execution is delayed by one clock. After one clock when the controller serves the PORT B command, the next command on PORT A is a write which is a command switch. This is seen in Figure 25-9 where four NOPs are inserted on PORT B because of the collision and eight NOPs are inserted on PORT A for read to write command switching.

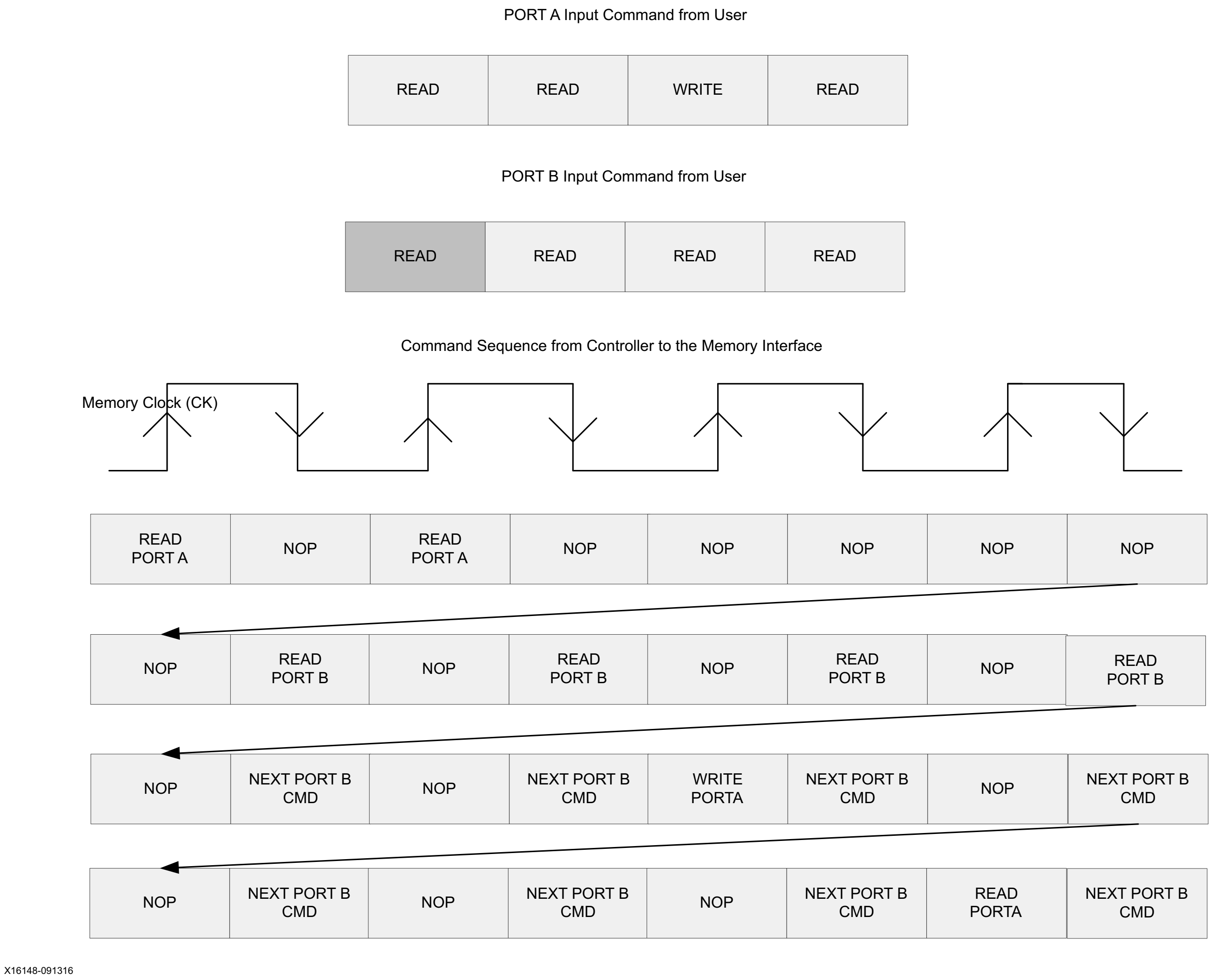


Figure 25-9: **Case 3: Bank Collision, Read to Write Command Switching**

Command Table

There are three types of commands supported by the controller. Table 25-4 lists the command encoding.

Table 25-4: **Command Encoding**

Command Value	Command Type
00	NOP
01	Reserved
10	Read
11	Write

Support for Mixed Command Assertion in Per User Clock

The QDR-IV device has a write latency of five and a read latency of eight. Also, there is a bus turnaround time of one clock. There must be five NOP commands between the read and write command whenever the write command follows a read command.

If you assert write and read commands in the same user clock, the controller takes care of asserting the NOP command before asserting the write command after a read command. Then, it asserts a busy signal to stop you from sending any further commands until it completes execution of the all accepted commands.

Taking Care of Bank Access Collision of PORT A and PORT B

There is a limitation for PORT B to access a bank at next edge after the bank is accessed by PORT A. There must be 1.5 memory clock gaps between the access of a bank by PORT B and PORT A if the bank access is followed by PORT B.

If you assert the same bank for PORT A and PORT B, the controller delays the command from PORT B by 1.5 memory clock cycles and asserts a busy until the execution of all the commands get completed.

Command Sequence

Because the read and write latencies are different for the given memory, the user interface ensures that the read and write command sequence issued at the memory are in correct order. If this is not guaranteed by the user interface, the device might be defective as the data bus in case of QDR-IV memory is bidirectional. Suppose for a given memory device, the read latency is eight and write latency is five.

For example, if you issued one read command followed by three write commands on the four channels. This means that from the time when a read command is executed, eight memory clocks are required to retrieve the read data. On the eighth clock, memory drives the data bus. Because there are following write commands, the FPGA tries to drive the data bus on the sixth, seventh, and eighth cycle.

On the eighth cycle, there is a case where both the FPGA and the memory tries to drive the same bus. This might damage the device and hence it is taken care of by the user interface inserting NOPs. This is explained further in Table 25-5 (W stands for a write command, R stands for a read command, NOP stands for No Operation).

Table 25-5: **Command Sequence**

Input Command from User Interface CH0, CH1, CH2,C H3	Output Command from User Interface to Memory Interface
WR-WR-WR-WR	WR-WR-WR-WR
RD-RD-RD-RD	RD-RD-RD-RD
WR-RD-WR-RD	WR-NP-NP-NP-NP-RD-NP-NP-NP-NP-NP-NP-NP-NP-WR-NP-NP-NP-NP-RD
RD-WR-RD-WR	RD-NP-NP-NP-NP-NP-NP-NP-NP-WR-NP-NP-NP-NP-RD-NP-NP-NP-NP-NP-NP-NP-NP-WR

Order of Command Processing

Because the controller can handle four commands, you can issue four commands in parallel on four channels. The command issued on the first channel (CH0) is processed first, followed by the command issued on the second channel (CH1), followed by the command issued on the third channel (CH2), and followed by the command issued on the fourth channel (CH3). Expect the same order at the output of the controller.

Interfacing with the Core through the User Interface

The user interface protocol is shown in the following Figure 25-10.

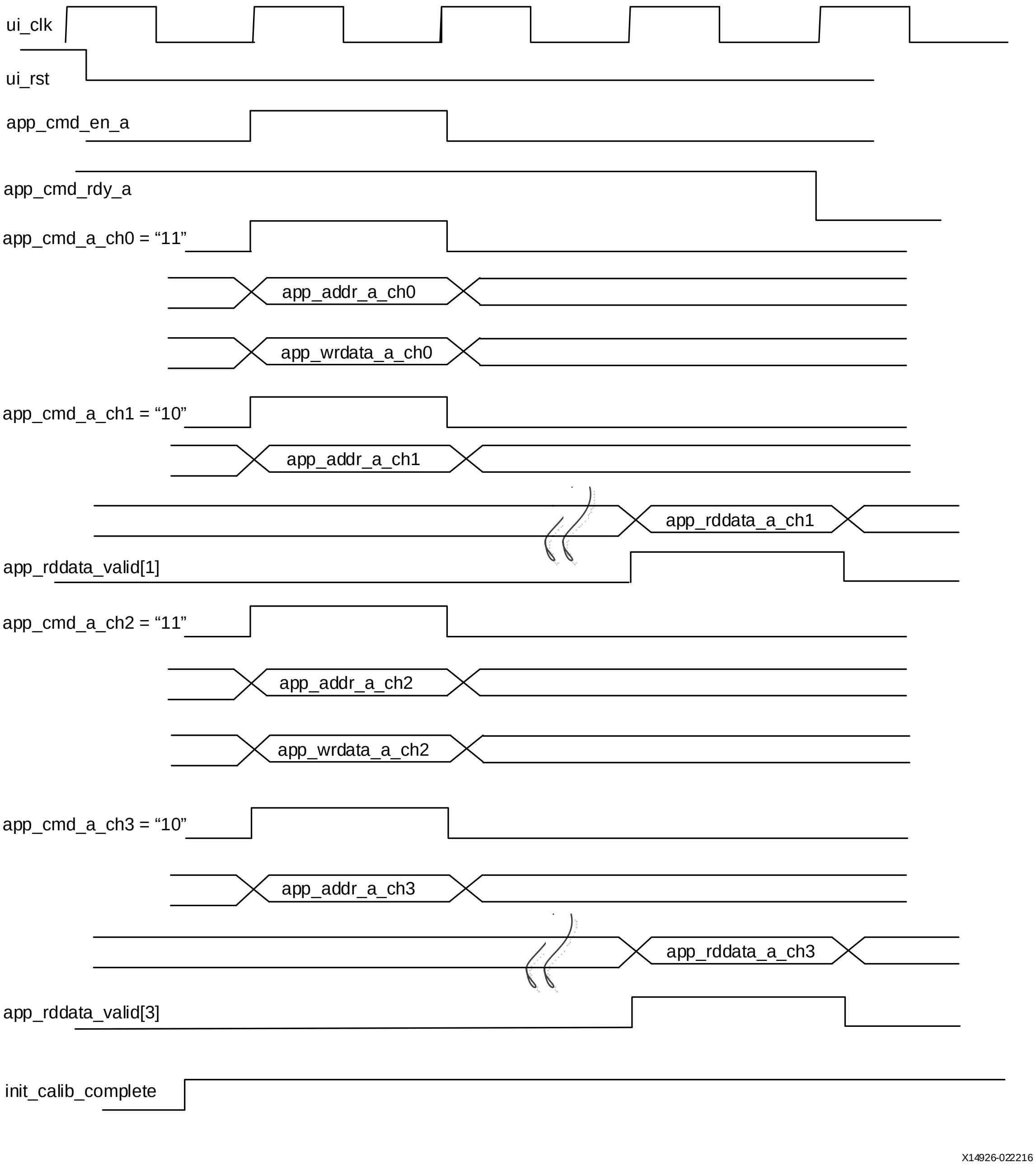


Figure 25-10: **User Interface Write/Read Timing Diagram**

Wait until the `init_calib_complete` signal is asserted High before sending any command as shown in Figure 25-10. No read or write requests are processed (that is, `app_wr_cmd` or `app_rd_cmd` on the client interface is ignored before `init_calib_complete` is High).

Figure 25-10 shows various commands being issued for different channels from you. For channel 0 and 2, it is the write command and for 1 and 3 it is the read command. For details, see the command table (Table 25-4).

For the write commands, write address and write data has to be valid in the same clock cycle as the write command. This means that for channel 0, **`app_wrdata_a_ch0`** gets written at location **`app_addr_a_ch0`**. For channel 1 also, it occurs the same way.

For the read commands, read address has to be present at the time of read commands assertion. The read data is available after a few clock cycles along with read valid signal. For Figure 25-10, for channel 1, **`app_rddata_a_ch1`** becomes available with the **`app_rddara_valid[1]`** signal.

Physical Interface

The physical interface is the connection from the FPGA memory interface solution to an external QDR-IV SRAM device. The I/O signals for this interface are defined in Table 25-6. These signals can be directly connected to the corresponding signals on the memory device.

Table 25-6: **Physical Interface Signals**

Signal	I/O	Description
ck	I	Address/Command Input Clock. CK is differential clock input. All control and address input signals are sampled on both the rising and falling edges of CK. The rising edge of CK samples the control and address inputs for PORT A, while the falling edge of CK samples the control and address inputs for PORT B.
ck_n	I	CK# is 180° out of phase with CK.

A[x:0]	I	Address Inputs. Sampled on the rising edge of both CK and CK# clocks during active read and write operations. These address inputs are used for read and write operations on both ports.
		For (×36) data width, Address inputs A[20:0] are used and A[24:21] are reserved.
		For (×18) data width, Address inputs A[21:0] are used and A[24:22] are reserved. The reserved address inputs are No Connects and might be tied High, Low, or left floating.

AP	I	Address Parity Input. Used to provide even parity across the address pins.
		For (×36) data width, AP covers address inputs A[20:0].
		For (×18) data width, AP covers address inputs A[21:0].

PE_n	O	Address Parity Error Flag. Asserted Low when address parity error is detected. After asserted, PE# remains Low until cleared by a Configuration register command.

AINV	I	Address Inversion Pin for Address and Address Parity Inputs.
		For (x36) data width, AINV covers address inputs A[20:0] and the address parity input (AP).
		For (x18) data width, AINV covers address inputs A[21:0] and the address parity input (AP).

Table 25-6: **Physical Interface Signals *(Cont'd)***

Signal	I/O	Description
DKA[1:0], DKA_n[1:0]	I	DKA[0]/DKA#[0] controls the DQA[17:0] inputs for x36 configuration and DQA[8:0] inputs for x18 configuration, respectively. DKA[1]/DKA#[1] controls the DQA[35:18] inputs for x36 configuration and DQA[17:9] inputs for x18 configuration, respectively.
DKB[1:0], DKB_n[1:0]	I	DKB[0]/DKB#[0] controls the DQB[17:0] inputs for x36 configuration and DQB[8:0] inputs for x18 configuration, respectively. DKB[1]/DKB#[1] controls the DQB[35:18] inputs for x36 configuration and DQB[17:9] inputs for x18 configuration, respectively.
QKA[1:0], QKA_n[1:0]	O	Data Output Clock. QKA[0]/QKA#[0] controls the DQA[17:0] outputs for x36 configuration and DQA[8:0] outputs for x18 configuration, respectively. QKA[1]/QKA#[1] controls the DQA[35:18] outputs for x36 configuration and DQA[17:9] outputs for x18 configuration, respectively.
QKB[1:0], QKB_n[1:0]	O	QKB[0]/QKB#[0] controls the DQB[17:0] outputs for x36 configuration and DQB[8:0] outputs for x18 configuration, respectively. QKB[1]/QKB#[1] controls the DQB[35:18] outputs for x36 configuration and DQB[17:9] outputs for x18 configuration, respectively.
LDA_n	I	Synchronous Load Input. LDA_n is sampled on the rising edge of the CK clock. LDA_n enables commands for data PORT A. LDA_n enables the commands when LDA_n is Low and disables the commands when LDA_n is High. When the command is disabled, new commands are ignored, but internal operations continue.
LDB_n	I	Synchronous Load Input. LDB_n is sampled on the falling edge of the CK clock. LDB_n enables commands for data PORT B. LDB_n enables the commands when LDB_n is Low and disables the commands when LDB_n is High. When the command is disabled, new commands are ignored, but internal operations continue.
RWA_n	I	Synchronous Read/Write Input. RWA_n input is sampled on the rising edge of the CK clock. The RWA_n input is used in conjunction with the LDA_n input to select a read or write operation.
RWB_n	I	RWB_n input is sampled on the falling edge of the CK clock. The RWB_n input is used in conjunction with the LDB_n input to select a read or write operation. The RWB_n input is used in conjunction with the LDB_n input to select a Read or Write operation.
QVLDA[1:0]	O	Output Data Valid Indicator. The QVLDA pin indicates valid output data. QVLD is edge-aligned with QKA. For example, QVLDA[0] is edge-aligned with QKA[1:0] and QVLDA[1] is edge-aligned with QKA_n[1:0].
QVLDB[1:0]	O	Output Data Valid Indicator. The QVLDB pin indicates valid output data. QVLD is edge-aligned with QKB. For example, QVLDB[0] is edge-aligned with QKB[1:0] and QVLDB[1] is edge-aligned with QKB_n[1:0].
CFG_n	I	Configuration bit. This pin is used to configure different mode registers.

Table 25-6: **Physical Interface Signals *(Cont'd)***

Signal	I/O	Description
RST_n	I	Active-Low Asynchronous RST. This pin is active when RST# is Low and inactive when RST# is High. The RST# pin has an internal pull-down resistor.
LBK0_n, LBK1_n	I	Loopback mode for control and address/command/clock deskewing.

Figure 25-11 shows the timing diagram for the sample write and read operations at the memory interface with write latency of three clock cycles and read latency of five clock cycles, respectively.

The command is detected by the memory only when `LDA_n` and `LDB_n` are Low for PORT A and PORT B, respectively. When `RWA_n` is Low, it is write command and when it is High, it is a read command. This is true for PORT B as well. Address is DDR and hence on the rising edge of `CK`, address is considered to be valid for PORT A and on the falling edge it is considered for PORT B.

In Figure 25-11, the cursor position is pointing to PORT A write command. Write address is `0x050EE8`. The DDR data is written into the memory as `0xC_6B7*` and `0x0_57B*` with the write latency at three clock cycles.

Following falling edge is a PORT B write command at address `0x0A7BC4` and the DDR data which is written to this memory address at PORT B is `0xF_754*` and `0x7_7B2`.

Next, the `CK` rising edge is a PORT A read command at address `0x0E6741` and corresponding data becomes available at the `DQA` data bus after five `CK` clock cycles aligned to the rising edge of `QK` clock edge because the read latency is five. The DDR read data is `0xC_818*` and `0xA_150*`. The `qvlda` is also asserted along with the data. For more information on read and write timing, see the QDR-IV memory specification.

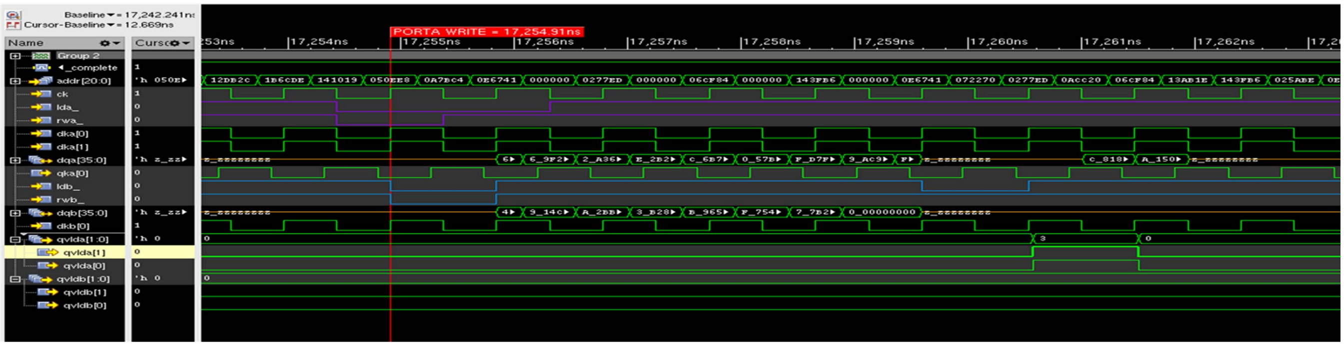


Figure 25-11: **QDR-IV Memory Read Write Timing**

M and D Support for Reference Input Clock Speed

Memory IPs provide two possibilities to select the Reference Input Clock Speed. The value allowed for Reference Input Clock Speed (ps) is always ≥ Memory Device Interface Speed (ps).

- Memory IP lists the possible Reference Input Clock Speed values based on the targeted memory frequency (based on selected Memory Device Interface Speed).
- Otherwise, select **M and D Options** and target for desired Reference Input Clock Speed which is calculated based on selected CLKFBOUT_MULT (M), DIVCLK_DIVIDE (D), and CLKOUT0_DIVIDE (D0) values in the Advanced Clocking Tab.

The required Reference Input Clock Speed is calculated from the M, D, and D0 values entered in the GUI using the following formulas:

- MMCM_CLKOUT (MHz) = tCK / Phy_Clock_Ratio

  Where tCK is the Memory Device Interface Speed selected in the Basic tab.

- CLKIN (MHz) = (MMCM_CLKOUT (MHz) × D × D0) / M

  CLKIN (MHz) is the calculated Reference Input Clock Speed.

- VCO (MHz) = (CLKIN (MHz)) / D

  VCO (MHz) is the calculated VCO frequency.

- PFD (MHz) = CLKIN (MHz) / D

  PFD (MHz) is the calculated PFD frequency.

Calculated Reference Input Clock Speed from M, D, and D0 values are validated as per clocking guidelines. For more information on clocking rules, see Clocking.

Apart from the memory specific clocking rules, validation of the possible MMCM input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values are completed for M, D, and D0 in the GUI.

For UltraScale devices, see *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2] and *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For UltraScale+ devices, see *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4], *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5], and *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For possible M, D, and D0 values and detailed information on clocking and the MMCM, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

Chapter 26

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]

Customizing and Generating the Core

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

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

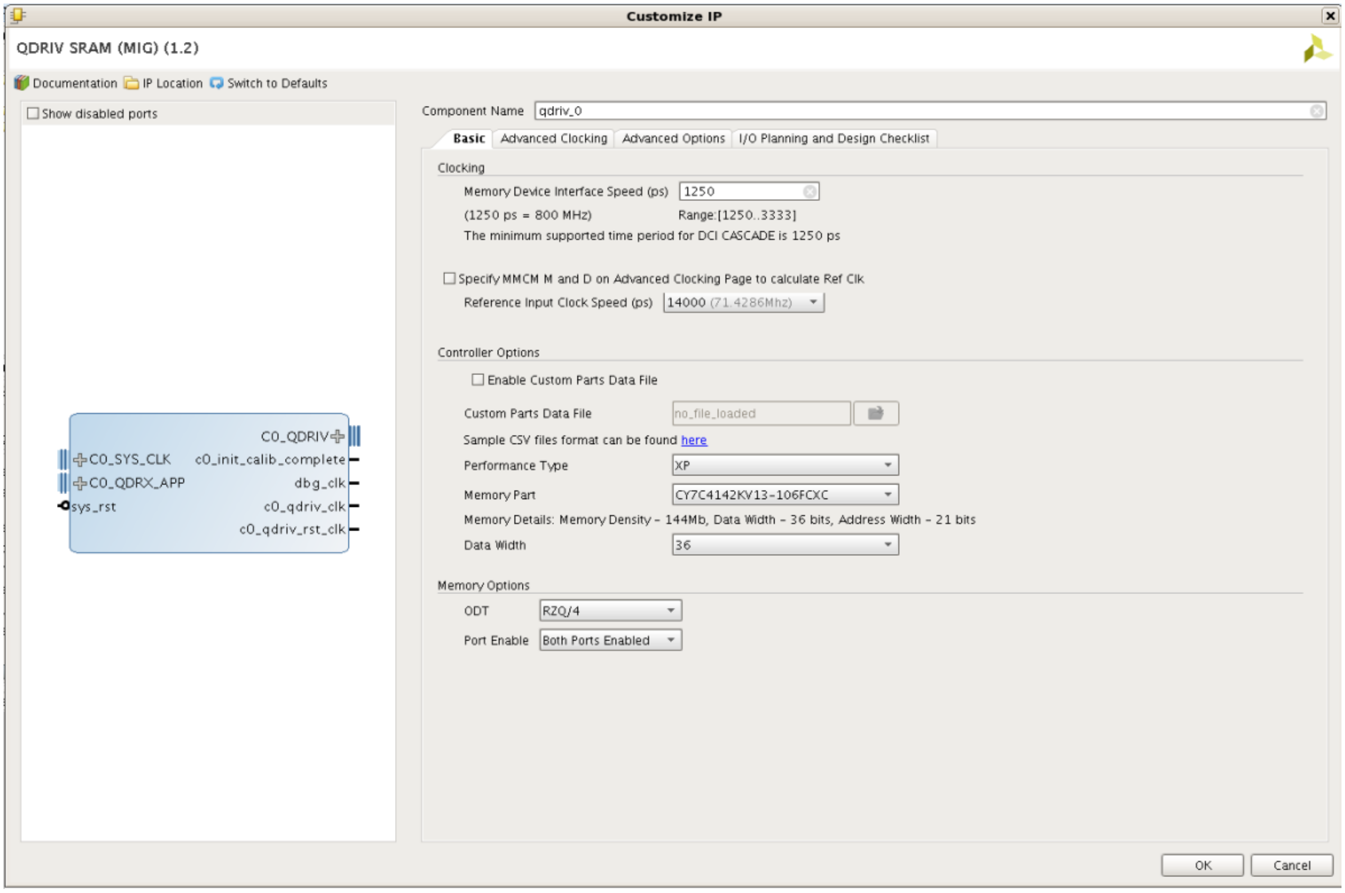
1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For more information about generating the core in Vivado, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

Basic Tab

Figure 26-1 shows the **Basic** tab when you start up the QDR-IV SRAM.



Component Name: qdriv_0

Clocking

Memory Device Interface Speed (ps): 1250

(1250 ps = 800 MHz) Range:[1250..3333]

The minimum supported time period for DCI CASCADE is 1250 ps

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps): 14000 (71.4286Mhz)

Controller Options

☐ Enable Custom Parts Data File

Custom Parts Data File: no_file_loaded

Sample CSV files format can be found here

Performance Type: XP

Memory Part: CY7C4142KV13-106FCXC

Memory Details: Memory Density - 144Mb, Data Width - 36 bits, Address Width - 21 bits

Data Width: 36

Memory Options

ODT: RZQ/4

Port Enable: Both Ports Enabled

Figure 26-1: **Vivado Customize IP Dialog Box – Basic**

IMPORTANT: *All parameters shown in the controller options dialog box are limited selection options in this release.*

For the Vivado IDE, all controllers (DDR3, DDR4, LPDDR3, QDR II+, QDR-IV, and RLDRAM 3) can be created and available for instantiation.

1. Select the settings in the **Clocking**, **Controller Options**, and **Memory Options**.

In **Clocking**, the **Memory Device Interface Speed** sets the speed of the interface. The speed entered drives the available **Reference Input Clock Speeds**. For more information on the clocking structure, see the Clocking, page 415.

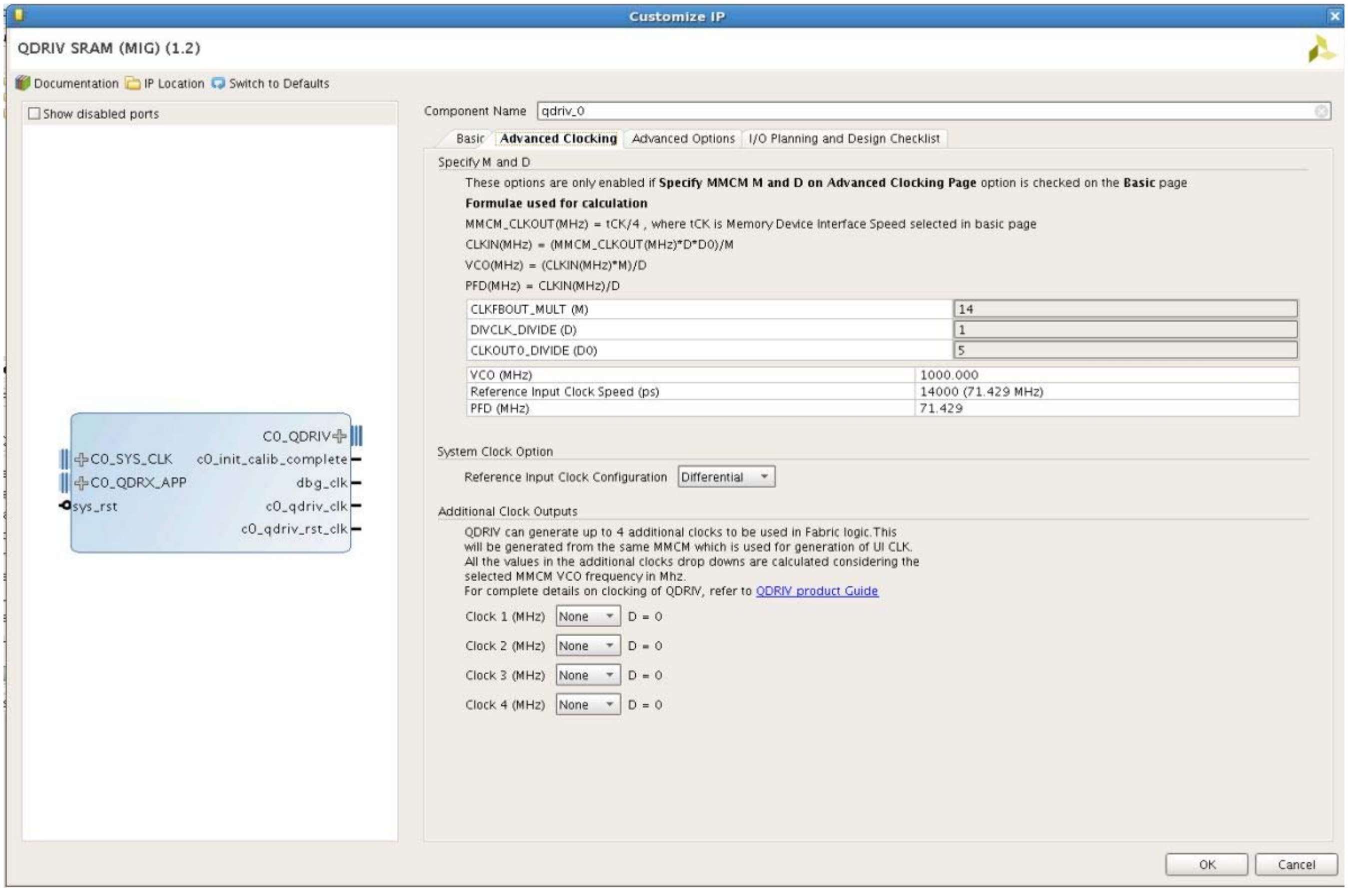
2. To use memory parts which are not available by default through the QDR-IV SRAM Vivado IDE, you can create a custom parts CSV file, as specified in the AR: 63462. This CSV file has to be provided after enabling the **Custom Parts Data File** option. After selecting this option. you are able to see the custom memory parts along with the default memory parts. Note that, simulations are not supported for the custom part. Custom part simulations require manually adding the memory model to the simulation and might require modifying the test bench instantiation.

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

For 72-bit interfaces, ECC is enabled and DM is deselected and grayed out for 72-bit designs. If DM is enabled for 72-bit designs, computing ECC is not compatible, therefore DM is disabled for 72-bit designs.

Advanced Clocking Tab

Figure 26-2 shows the next tab called **Advanced Clocking**. This displays the settings for **Specify M and D** value, **System Clock Options**, and **Additional Clock Outputs** for the specific controller.



Component Name: qdriv_0

Specify M and D

These options are only enabled if **Specify MMCM M and D on Advanced Clocking Page** option is checked on the **Basic** page

Formulae used for calculation

MMCM_CLKOUT(MHz) = tCK/4 , where tCK is Memory Device Interface Speed selected in basic page

CLKIN(MHz) = (MMCM_CLKOUT(MHz)*D*D0)/M

VCO(MHz) = (CLKIN(MHz)*M)/D

PFD(MHz) = CLKIN(MHz)/D

CLKFBOUT_MULT (M)	14
DIVCLK_DIVIDE (D)	1
CLKOUT0_DIVIDE (D0)	5

VCO (MHz)	1000.000
Reference Input Clock Speed (ps)	14000 (71.429 MHz)
PFD (MHz)	71.429

System Clock Option

Reference Input Clock Configuration: Differential

Additional Clock Outputs

QDRIV can generate up to 4 additional clocks to be used in Fabric logic.This will be generated from the same MMCM which is used for generation of UI CLK. All the values in the additional clocks drop downs are calculated considering the selected MMCM VCO frequency in Mhz. For complete details on clocking of QDRIV, refer to QDRIV product Guide

Clock 1 (MHz): None D = 0

Clock 2 (MHz): None D = 0

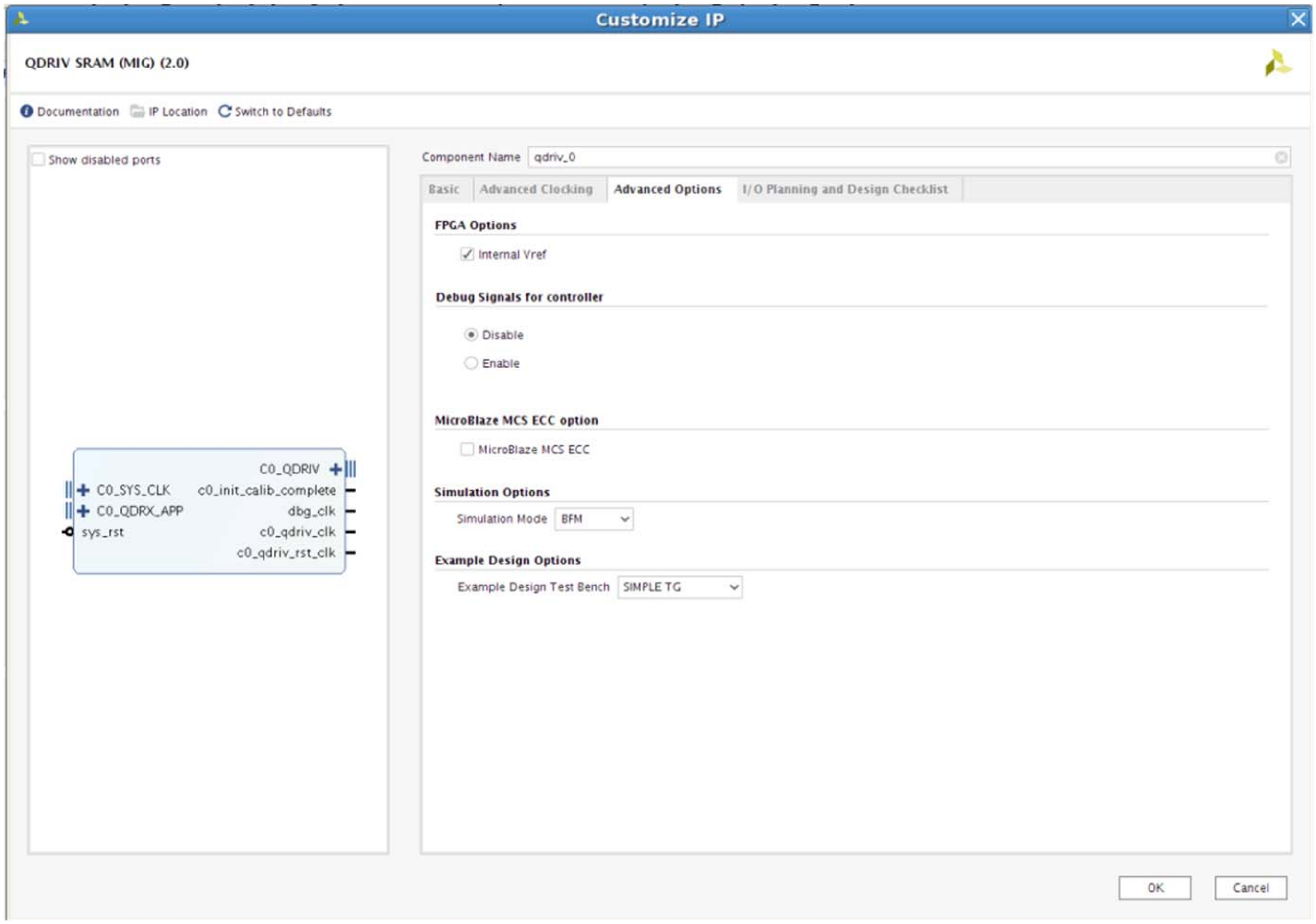
Clock 3 (MHz): None D = 0

Clock 4 (MHz): None D = 0

Figure 26-2: **Vivado Customize IP Dialog Box – Advanced Clocking**

Advanced Options Tab

Figure 26-3 shows the next tab called **Advanced Options**. This displays the advanced memory options settings for the specific controller.



☐ Show disabled ports

Component Name qdriv_0

FPGA Options

☑ Internal Vref

Debug Signals for controller

● Disable

○ Enable

MicroBlaze MCS ECC option

☐ MicroBlaze MCS ECC

Simulation Options

Simulation Mode BFM

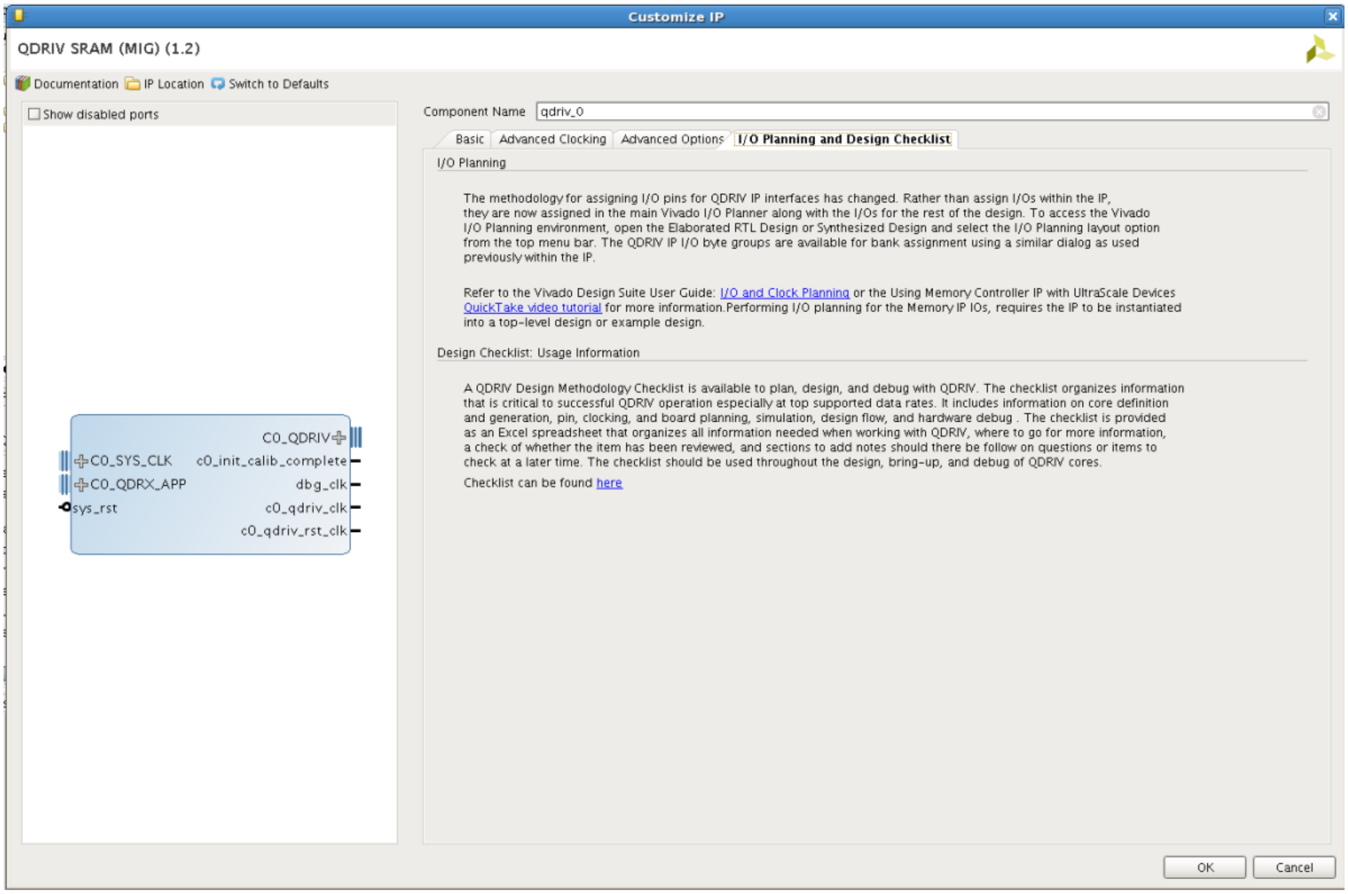
Example Design Options

Example Design Test Bench SIMPLE TG

Figure 26-3: **Vivado Customize IP Dialog Box – Advanced Options**

QDR-IV SRAM I/O Planning and Design Checklist Tab

Figure 26-4 shows the **QDR-IV SRAM I/O Planning and Design Checklist** usage information.



Component Name: qdriv_0

I/O Planning

The methodology for assigning I/O pins for QDRIV IP interfaces has changed. Rather than assign I/Os within the IP, they are now assigned in the main Vivado I/O Planner along with the I/Os for the rest of the design. To access the Vivado I/O Planning environment, open the Elaborated RTL Design or Synthesized Design and select the I/O Planning layout option from the top menu bar. The QDRIV IP I/O byte groups are available for bank assignment using a similar dialog as used previously within the IP.

Refer to the Vivado Design Suite User Guide: I/O and Clock Planning or the Using Memory Controller IP with UltraScale Devices QuickTake video tutorial for more information.Performing I/O planning for the Memory IP IOs, requires the IP to be instantiated into a top-level design or example design.

Design Checklist: Usage Information

A QDRIV Design Methodology Checklist is available to plan, design, and debug with QDRIV. The checklist organizes information that is critical to successful QDRIV operation especially at top supported data rates. It includes information on core definition and generation, pin, clocking, and board planning, simulation, design flow, and hardware debug . The checklist is provided as an Excel spreadsheet that organizes all information needed when working with QDRIV, where to go for more information, a check of whether the item has been reviewed, and sections to add notes should there be follow on questions or items to check at a later time. The checklist should be used throughout the design, bring-up, and debug of QDRIV cores.

Checklist can be found here

Figure 26-4: **Vivado Customize IP Dialog Box – I/O Planning and Design Checklist**

User Parameters

Table 26-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 26-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value(1)	User Parameter/Value(1)	Default Value
System Clock Configuration	System_Clock	Differential
Internal V_{REF}	Internal_Vref	TRUE
DCI Cascade	DCI_Cascade	FALSE
Debug Signal for Controller	Debug_Signal	Disable
Clock 1 (MHz)	ADDN_UI_CLKOUT1_FREQ_HZ	None
Clock 2 (MHz)	ADDN_UI_CLKOUT2_FREQ_HZ	None

Table 26-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[(1)]	User Parameter/Value[(1)]	Default Value
Clock 3 (MHz)	ADDN_UI_CLKOUT3_FREQ_HZ	None
Clock 4 (MHz)	ADDN_UI_CLKOUT4_FREQ_HZ	None
I/O Power Reduction	IOPowerReduction	OFF
Enable System Ports	Enable_SysPorts	TRUE
Default Bank Selections	Default_Bank_Selections	FALSE
Reference Clock	Reference_Clock	FALSE
Clock Period (ps)	C0.QDRIV_TimePeriod	1,250
Input Clock Period (ps)	C0.QDRIV_InputClockPeriod	13,000
PORT_ENABLE	C0.QDRIV_PORT_ENABLE	EN_BOTH
Configuration	C0.QDRIV_MemoryType	Components
TERMINATION_REG_VAL	C0.QDRIV_ODT_VAL	RZQ/4
Memory Part	C0.QDRIV_MemoryPart	CY7C4142KV13-106FCXC
Data Width	C0.QDRIV_DataWidth	36
Performance Type	C0.QDRIV_PerformanceType	XP
Burst Length	C0.QDRIV_BurstLen	2
Memory Name	C0.QDRIV_MemoryName	Main Memory

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

I/O Planning

For details on I/O planning, see I/O Planning, page 235.

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

The QDR-IV SRAM Vivado IDE generates the required constraints. A location constraint and an I/O standard constraint are added for each external pin in the design. The location is chosen by the Vivado IDE according to the banks and byte lanes chosen for the design.

The I/O standard is chosen by the memory type selection and options in the Vivado IDE and by the pin type. A sample for `qdriv_a[0]` is shown here.

```
set_property PACKAGE_PIN AK26 [get_ports {a[0]}]
set_property IOSTANDARD POD12_DCI [get_ports {a[0]}]
```

The system clock must have the period set properly:

```
create_clock -name sys_clk_i -period 2.000 [get_ports sys_clk_p]
```

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For more information on clocking, see Clocking, page 359.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The QDR-IV SRAM tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

IMPORTANT: *The `set_input_delay` and `set_output_delay` constraints are not needed on the external memory interface pins in this design due to the calibration process that automatically runs at start-up. Warnings seen during implementation for the pins can be ignored.*

Simulation

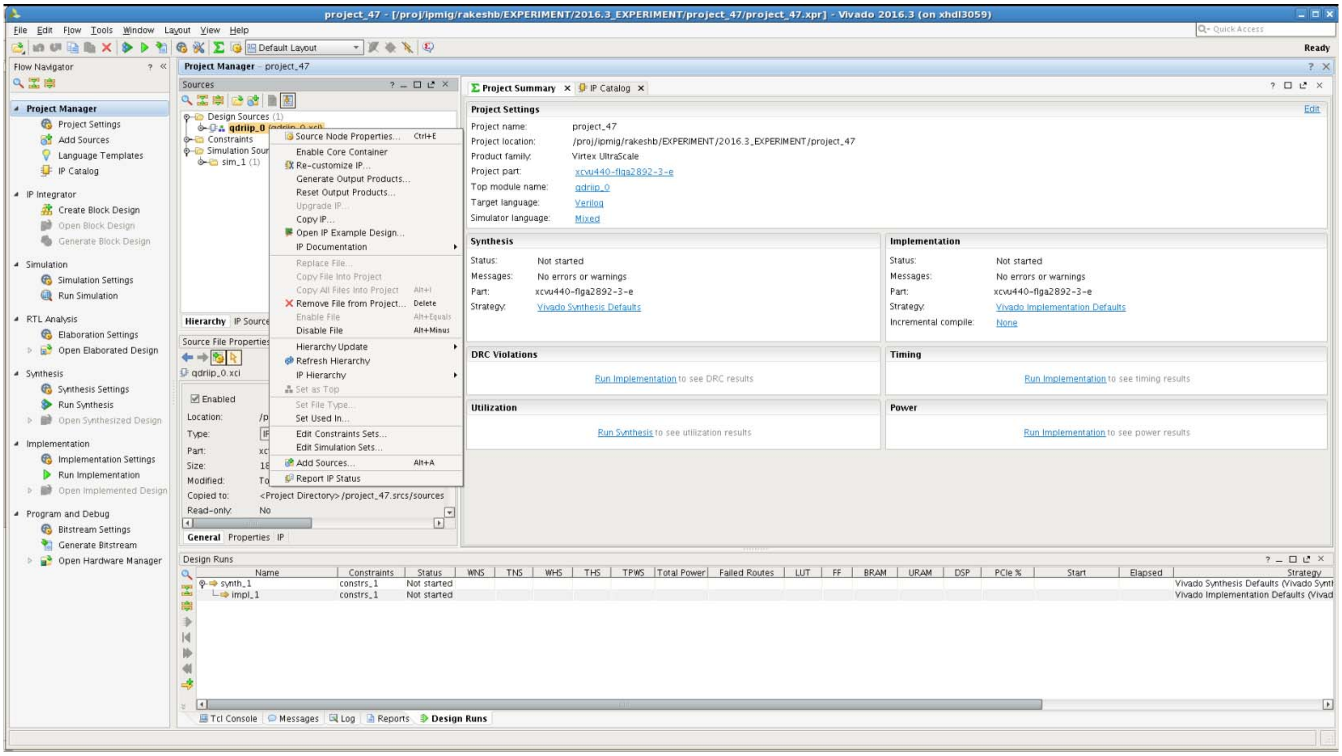
This section contains information about simulating the QDR-IV SRAM generated IP. Vivado simulator, Questa Advanced Simulator, IES, and VCS simulation tools are used for verification of the QDR-IV SRAM IP at each software release. For more information on simulation, see Chapter 27, Example Design and Chapter 28, Test Bench.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite. Vivado supports Open IP Example Design flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in Figure 27-1 and select **Open IP Example Design**.



qdriip_0.xci
☑ Enabled
Location: /p
Type: IP
Part: xc
Size: 18
Modified: To
Copied to: <Project Directory>/project_47.srcs/sources
Read-only: No

Project Settings
Project name: project_47
Project location: /proj/ipmig/rakeshb/EXPERIMENT/2016.3_EXPERIMENT/project_47
Product family: Virtex UltraScale
Project part: xcvu440-flga2892-3-e
Top module name: qdriip_0
Target language: Verilog
Simulator language: Mixed

Synthesis
Status: Not started
Messages: No errors or warnings
Part: xcvu440-flga2892-3-e
Strategy: Vivado Synthesis Defaults

Implementation
Status: Not started
Messages: No errors or warnings
Part: xcvu440-flga2892-3-e
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations
Run Implementation to see DRC results

Timing
Run Implementation to see timing results

Utilization
Run Synthesis to see utilization results

Power
Run Implementation to see power results

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	PCIe %	Start	Elapsed	Strategy
synth_1	constrs_1	Not started																Vivado Synthesis Defaults (Vivado Synth
impl_1	constrs_1	Not started																Vivado Implementation Defaults (Vivad

Figure 27-1: **Open IP Example Design**

This option creates a new Vivado project. Upon selecting the menu, a dialog box to enter the directory information for the new design project opens.

Select a directory, or use the defaults, and click **OK**. This launches a new Vivado with all of the example design files and a copy of the IP.

Simulating the Example Design (Designs with Standard User Interface)

The example design provides a synthesizable test bench to generate a fixed simple data pattern to the Memory Controller. This test bench consists of an IP wrapper and an **`example_tb`** that generates 16 writes and 16 reads. QDR-IV SRAM does not deliver the QDR-IV memory models. The memory model required for the simulation must be downloaded from the memory vendor's website.

The example design can be simulated using one of the methods in the following sections.

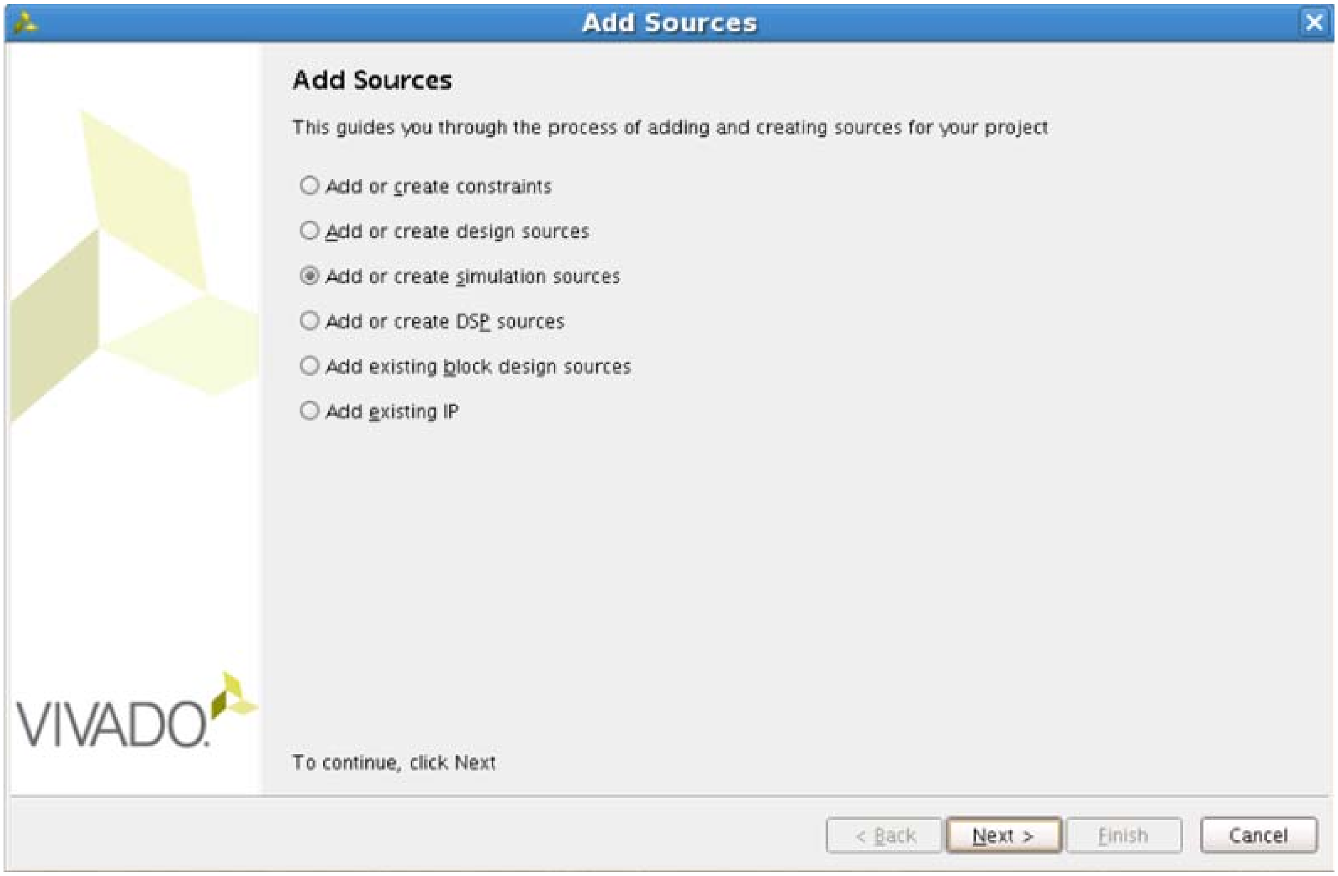
Project-Based Simulation

This method can be used to simulate the example design using the Vivado Integrated Design Environment (IDE). Memory IP does not deliver the QDR-IV memory models. The memory model required for the simulation must be downloaded from the memory vendor website. The memory model file must be added in the example design using **Add Sources** option to run simulation.

The Vivado simulator, Questa Advanced Simulator, IES, and VCS tools are used for QDR-IV IP verification at each software release. The Vivado simulation tool is used for QDR-IV IP verification from 2015.1 Vivado software release. The following subsections describe steps to run a project-based simulation using each supported simulator tool.

Project-Based Simulation Flow Using Vivado Simulator

1. In the **Open IP Example Design** Vivado project, under **Add sources** option, select the **Add or create simulation sources** option, and click **Next** as shown in Figure 27-2.



Add Sources

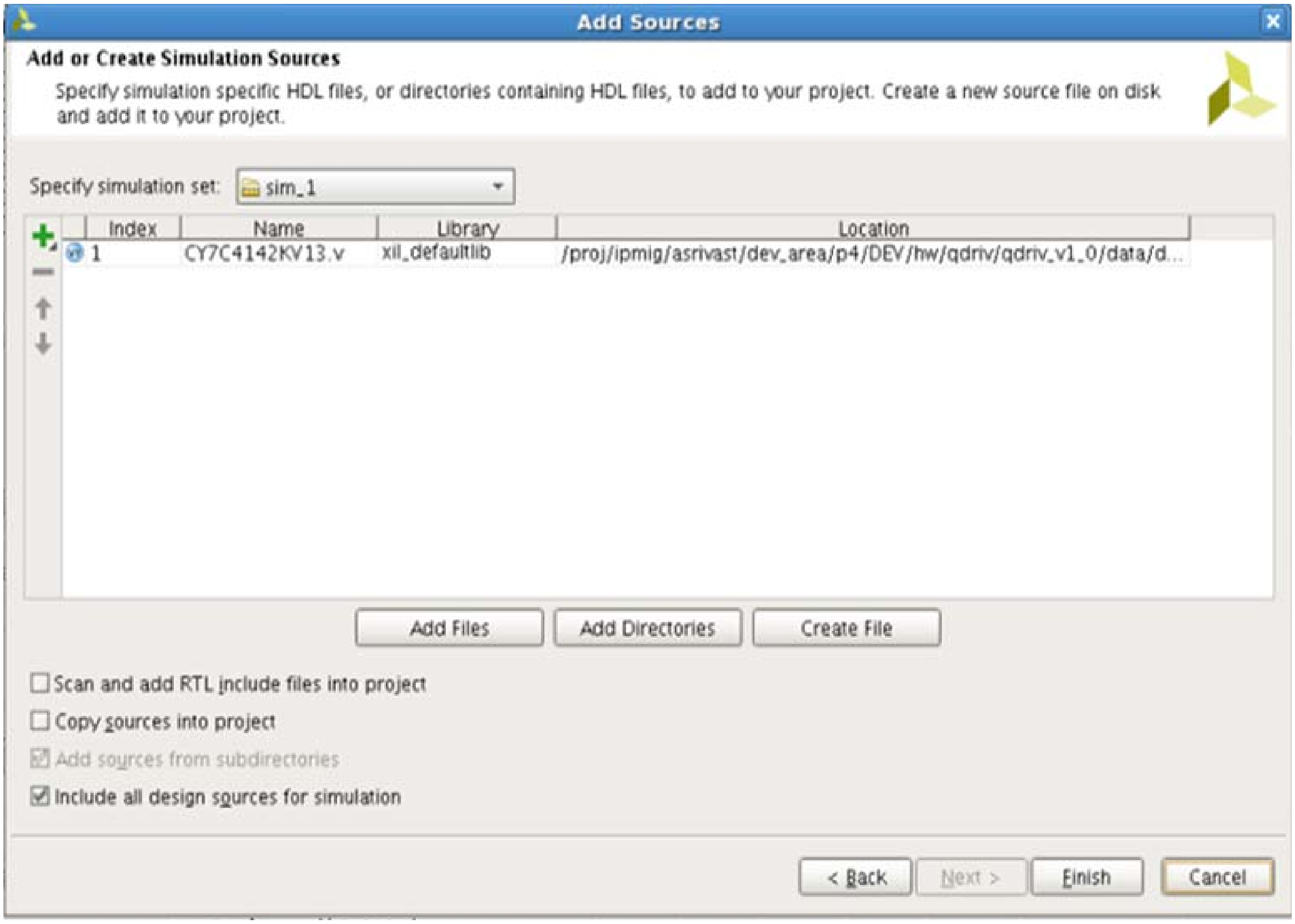
This guides you through the process of adding and creating sources for your project

○ Add or create constraints
○ Add or create design sources
● Add or create simulation sources
○ Add or create DSP sources
○ Add existing block design sources
○ Add existing IP

To continue, click Next

Figure 27-2: **Add Source Option in Vivado**

2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 27-3.



Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location
1	CY7C4142KV13.v	xil_defaultlib	/proj/ipmig/asrivast/dev_area/p4/DEV/hw/qdriv/qdriv_v1_0/data/d...

☐ Scan and add RTL include files into project

☐ Copy sources into project

☑ Add sources from subdirectories

☑ Include all design sources for simulation

Figure 27-3: **Add or Create Simulation Sources in Vivado**

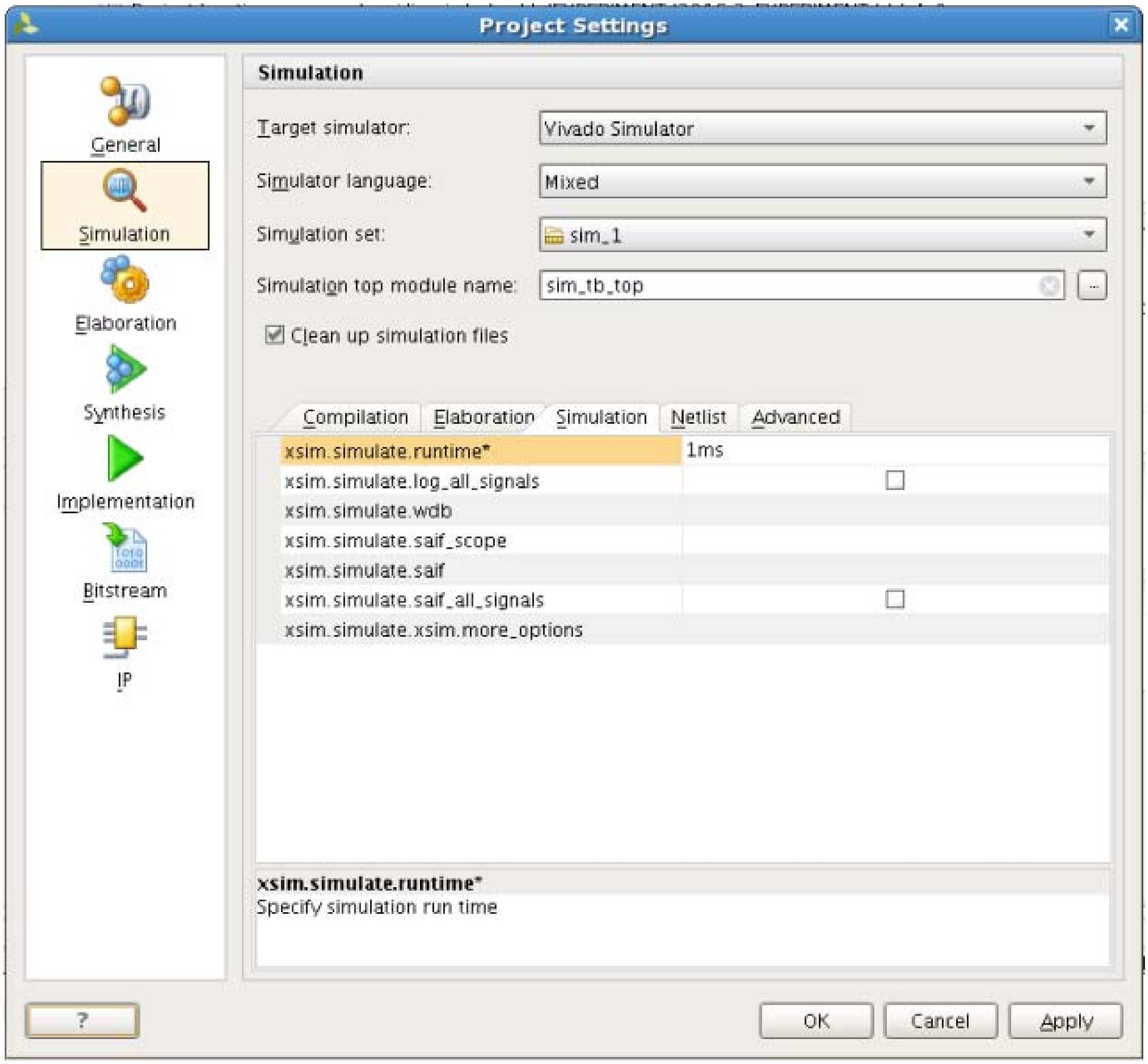
3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**

4. Select **Target simulator** as **Vivado Simulator**.

   Under the **Simulation** tab, set the `xsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 27-4. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.

5. Set the **Simulation Language** to **Mixed**.

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



Simulation

Target simulator: Vivado Simulator

Simulator language: Mixed

Simulation set: sim_1

Simulation top module name: sim_tb_top

☑ Clean up simulation files

xsim.simulate.runtime*	1ms
xsim.simulate.log_all_signals	☐
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐
xsim.simulate.xsim.more_options	

xsim.simulate.runtime*
Specify simulation run time

Figure 27-4: **Simulation with Vivado Simulator**

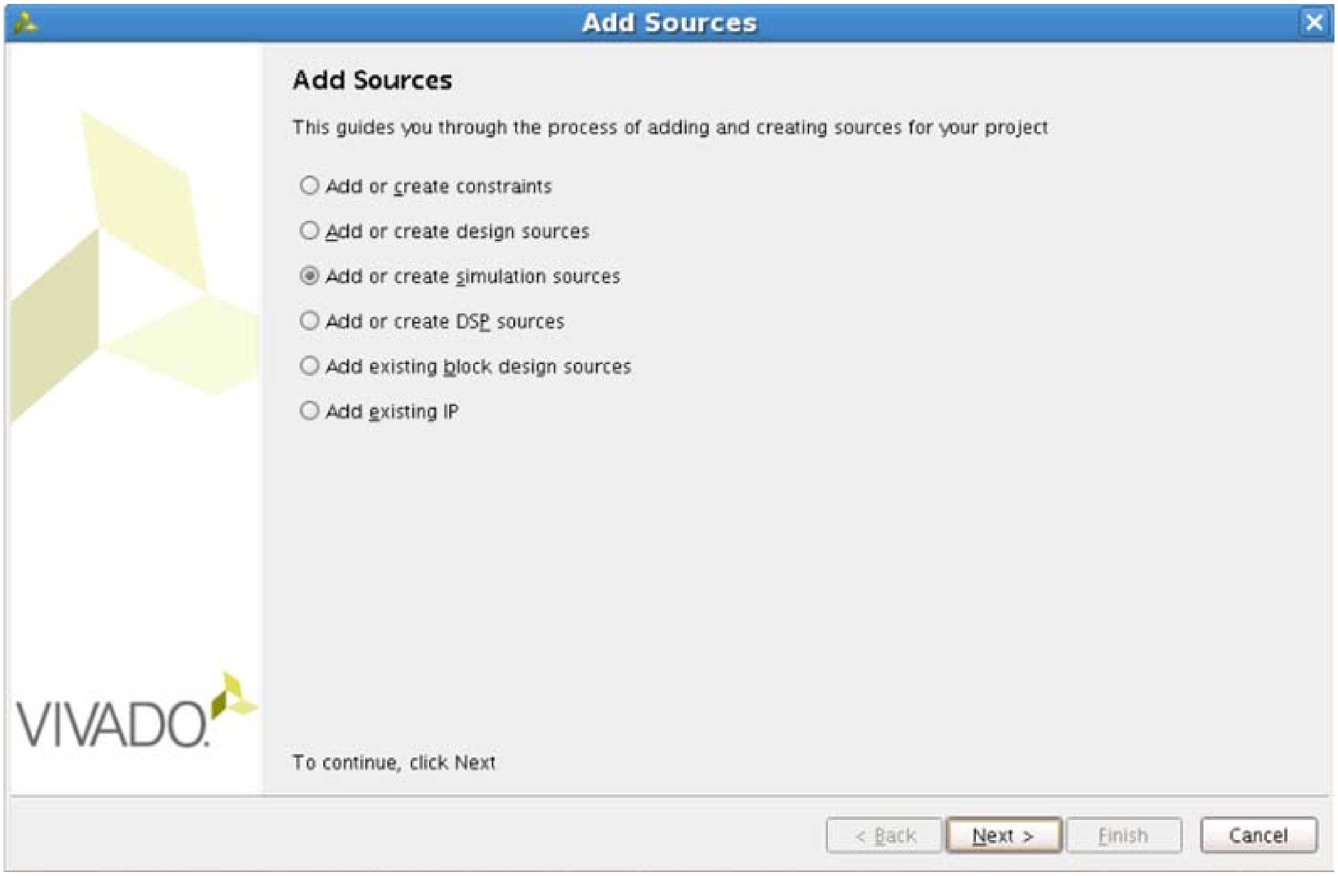
7. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 27-5.

Figure 27-5: **Run Behavioral Simulation**

8. Vivado invokes Vivado simulator and simulations are run in the Vivado simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using Questa Advanced Simulator

1. Open a QDR-IV SRAM example Vivado project (**Open IP Example Design**...), then under **Add sources** option, select the **Add or create simulation sources** option, and click **Next** as shown in Figure 27-6.



Add Sources

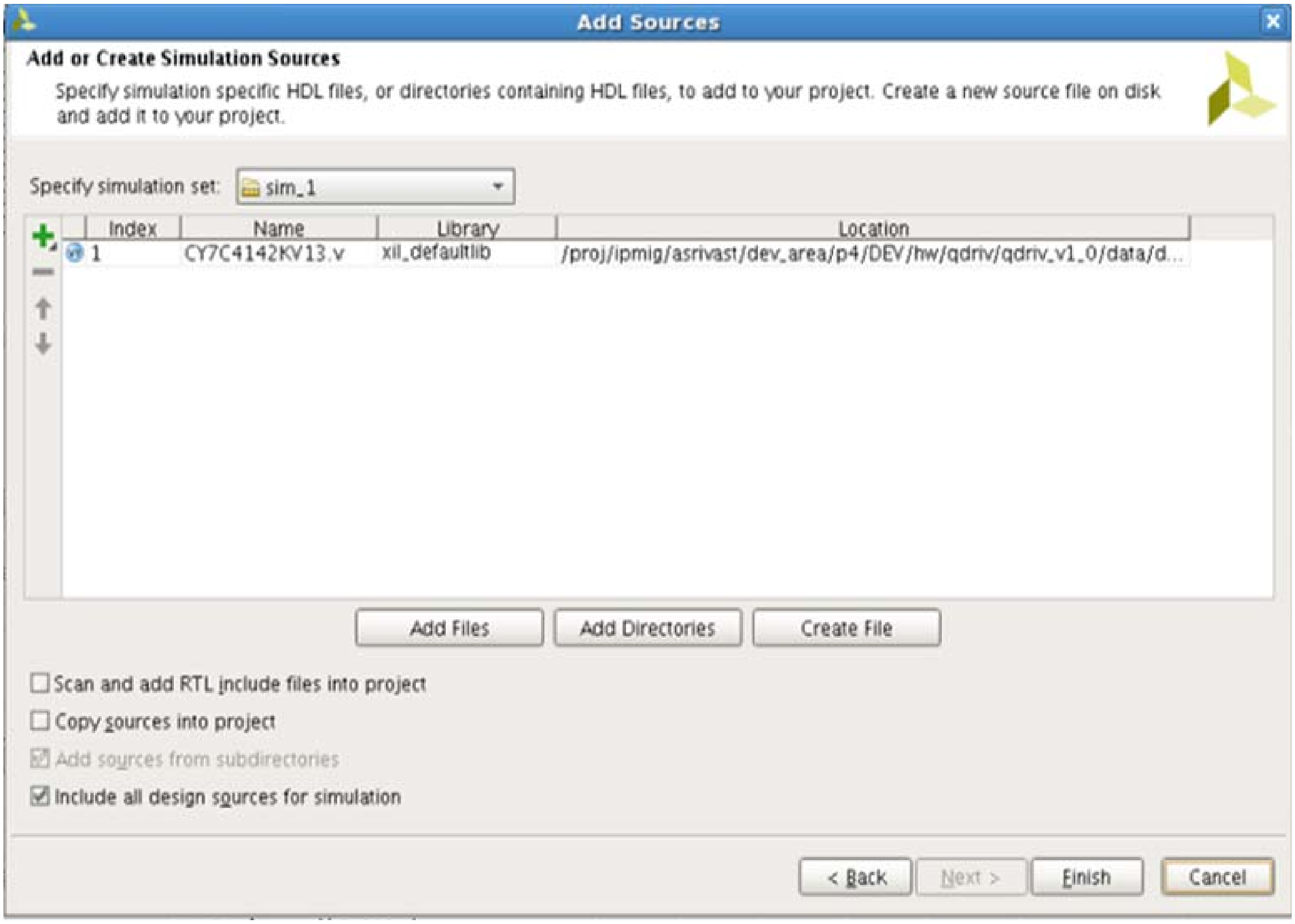
This guides you through the process of adding and creating sources for your project

○ Add or create constraints
○ Add or create design sources
● Add or create simulation sources
○ Add or create DSP sources
○ Add existing block design sources
○ Add existing IP

To continue, click Next

Figure 27-6: **Add Source Option in Vivado**

2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 27-7.



Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location
1	CY7C4142KV13.v	xil_defaultlib	/proj/ipmig/asrivast/dev_area/p4/DEV/hw/qdriv/qdriv_v1_0/data/d...

☐ Scan and add RTL include files into project

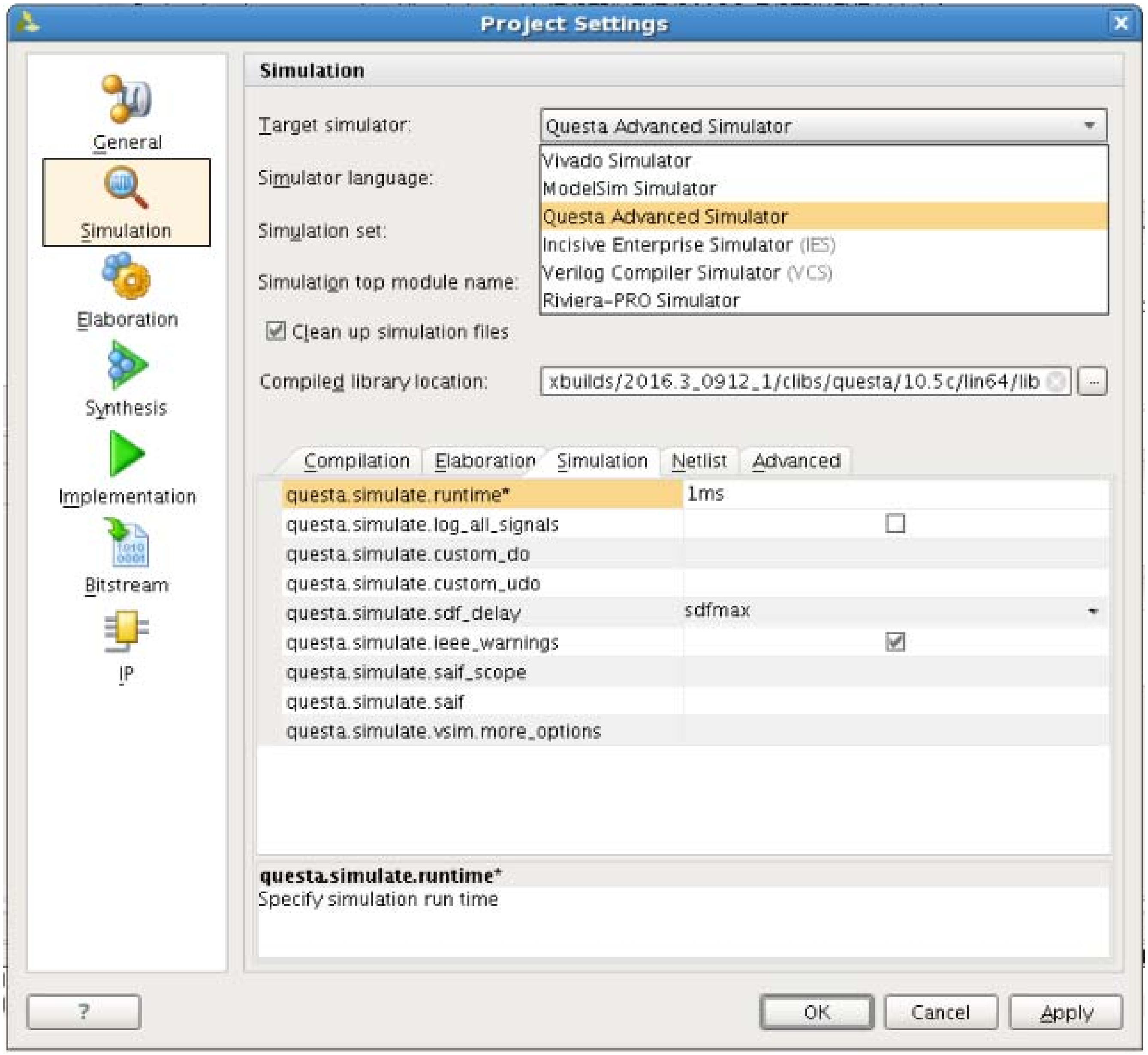
☐ Copy sources into project

☑ Add sources from subdirectories

☑ Include all design sources for simulation

Figure 27-7: **Add or Create Simulation Sources in Vivado**

3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**
4. Select **Target simulator** as **Questa Advanced Simulator**.
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `modelsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 27-8. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
5. Apply the settings and select **OK**.



Target simulator: Questa Advanced Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: xbuilds/2016.3_0912_1/clibs/questa/10.5c/lin64/lib

questa.simulate.runtime*	1ms
questa.simulate.log_all_signals	☐
questa.simulate.custom_do	
questa.simulate.custom_udo	
questa.simulate.sdf_delay	sdfmax
questa.simulate.ieee_warnings	☑
questa.simulate.saif_scope	
questa.simulate.saif	
questa.simulate.vsim.more_options	

questa.simulate.runtime*
Specify simulation run time

Figure 27-8: **Simulation with Questa Advanced Simulator**

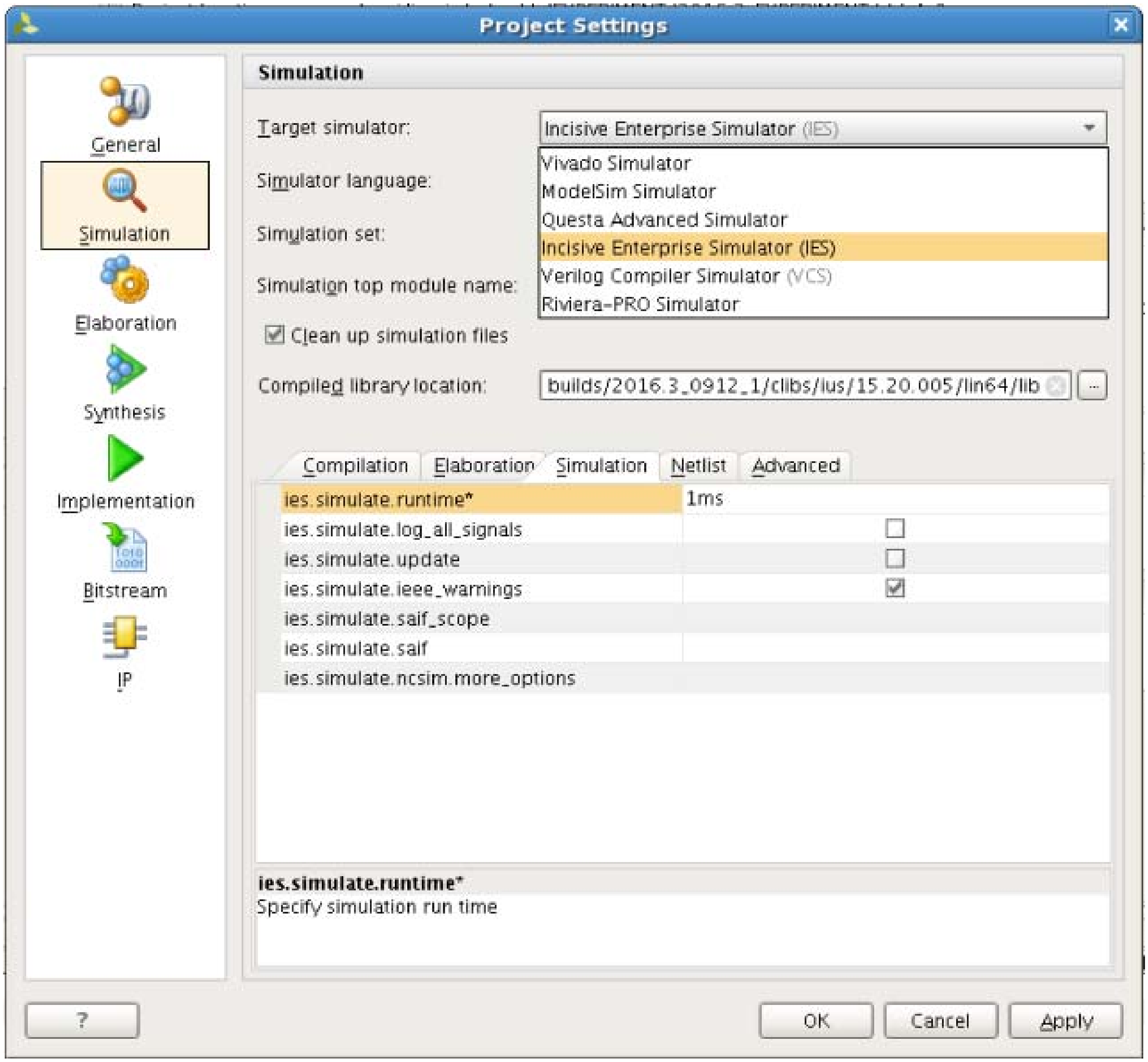
6. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 27-9.

Figure 27-9: **Run Behavioral Simulation**

7. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using IES

1. Open a QDR-IV SRAM example Vivado project (**Open IP Example Design**...), then under **Add sources option**, select the **Add or create simulation sources** option and click **Next** as shown in Figure 27-6.
2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 27-7.
3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**
4. Select **Target simulator** as Incisive Enterprise Simulator (IES).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `ies.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 27-10. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
5. Apply the settings and select **OK**.



Target simulator: Incisive Enterprise Simulator (IES)

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: builds/2016.3_0912_1/clibs/ius/15.20.005/lin64/lib

ies.simulate.runtime*	1ms
ies.simulate.log_all_signals	☐
ies.simulate.update	☐
ies.simulate.ieee_warnings	☑
ies.simulate.saif_scope	
ies.simulate.saif	
ies.simulate.ncsim.more_options	

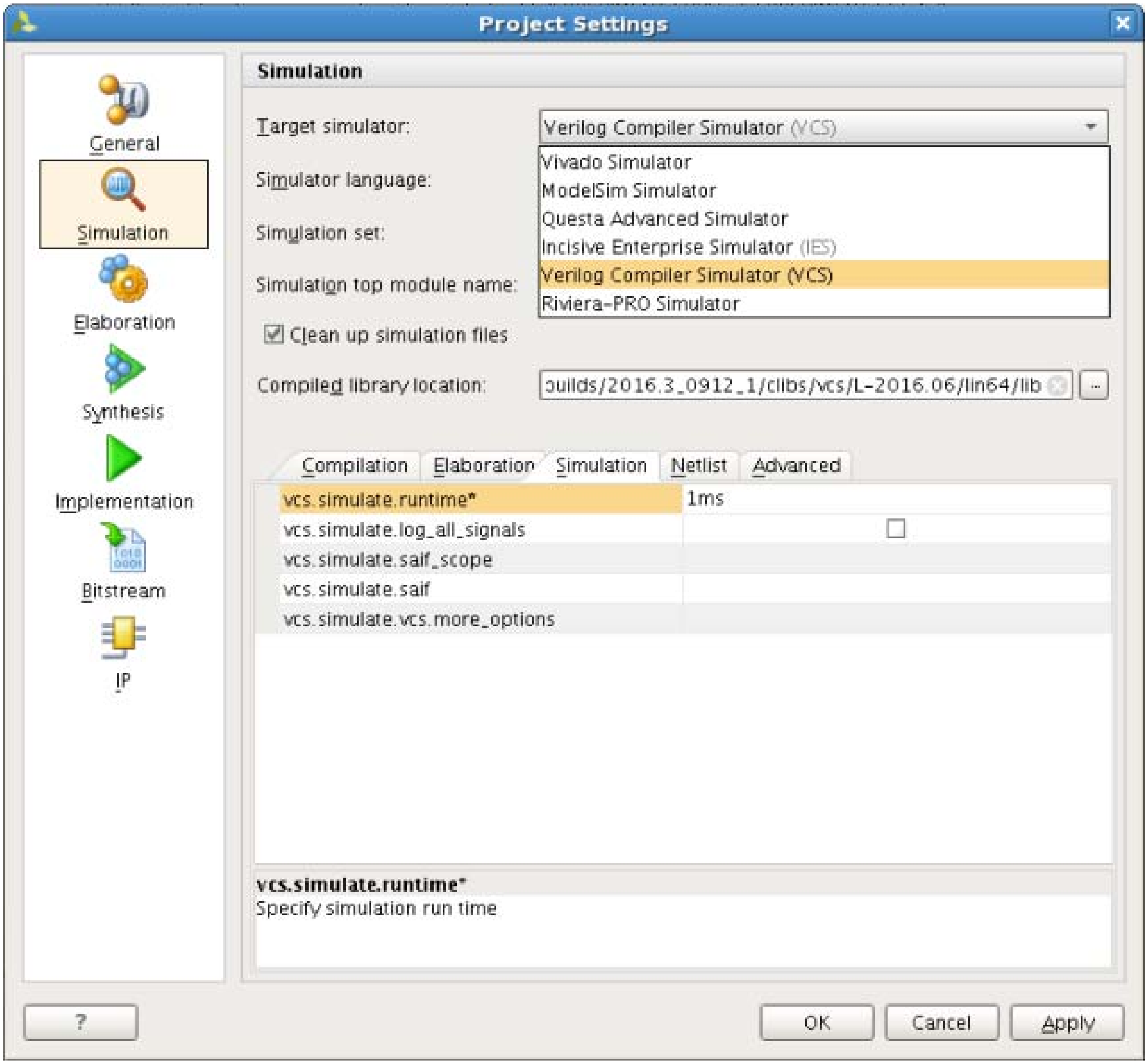
ies.simulate.runtime*
Specify simulation run time

Figure 27-10: **Simulation with IES Simulator**

6. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 27-9.
7. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using VCS

1. Open a QDR-IV SRAM example Vivado project (**Open IP Example Design**...), then under **Add sources option**, select the **Add or create simulation sources** option and click **Next** as shown in Figure 27-6.
2. Add the memory model in the **Add or create simulation sources** page and click **Finish** as shown in Figure 27-7.
3. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings.**
4. Select **Target simulator** as Verilog Compiler Simulator (VCS).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `vcs.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 27-11. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
5. Apply the settings and select **OK**.



Simulation

Target simulator: Verilog Compiler Simulator (VCS)

☑ Clean up simulation files

Compiled library location: ɔuilds/2016.3_0912_1/clibs/vcs/L-2016.06/lin64/lib

vcs.simulate.runtime*	1ms
vcs.simulate.log_all_signals	☐
vcs.simulate.saif_scope	
vcs.simulate.saif	
vcs.simulate.vcs.more_options	

vcs.simulate.runtime*

Specify simulation run time

Figure 27-11: **Simulation with VCS Simulator**

6. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 27-9.

7. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Simulation Speed

QDR-IV SRAM provides a Vivado IDE option to reduce the simulation speed by selecting behavioral XIPHY model instead of UNISIM XIPHY model. Behavioral XIPHY model simulation is a default option for QDR-IV SRAM designs. To select the simulation mode, click the **Advanced Options** tab and find the **Simulation Options** as shown in Figure 26-3.

The SIM_MODE parameter in the RTL is given a different value based on the Vivado IDE selection.

- **SIM_MODE = BFM** – If fast mode is selected in the Vivado IDE, the RTL parameter reflects this value for the SIM_MODE parameter. This is the default option.
- **SIM_MODE = FULL** – If UNISIM mode is selected in the Vivado IDE, XIPHY UNISIMs are selected and the parameter value in the RTL is FULL.

IMPORTANT: *QDR-IV memory models from Cypress® Semiconductor need to be modified with the following two timing parameters values to run the simulations successfully:*
`define tcqd #0
`define tcqdoh #0.15

Using Xilinx IP with Third-Party Synthesis Tools

For more information on how to use Xilinx IP with third-party synthesis tools, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation

If the GCIO pin and MMCM are not allocated in the same bank, the CLOCK_DEDICATED_ROUTE constraint must be set to BACKBONE. To use the BACKBONE route, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM input. QDR-IV SRAM manages these constraints for designs generated with the **Reference Input Clock** option selected as **Differential** (at **Advanced > FPGA Options > Reference Input**). Also, QDR-IV SRAM handles the IP and example design flows for all scenarios.

If the design is generated with the **Reference Input Clock** option selected as **No Buffer** (at **Advanced > FPGA Options > Reference Input**), the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation based on GCIO and

MMCM allocation needs to be handled manually for the IP flow. QDR-IV SRAM does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations for the IP flow.

For an example design flow with **No Buffer** configurations, QDR-IV SRAM generates the example design with differential buffer instantiation for system clock pins. QDR-IV SRAM generates clock constraints in the `example_design.xdc`. It also generates a CLOCK_DEDICATED_ROUTE constraint as the "BACKBONE" and instantiates BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM input if the GCIO and MMCM are not in same bank to provide a complete solution. This is done for the example design flow as a reference when it is generated for the first time.

If in the example design, the I/O pins of the system clock pins are changed to some other pins with the I/O pin planner, the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation need to be managed manually. A DRC error is reported for the same.

Chapter 28

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

The Memory Controller is generated along with a simple test bench to verify the basic read and write operations. The stimulus contains 16 consecutive writes followed by 16 consecutive reads for data integrity check.

SECTION VI: RLDRAM 3

Overview

IMPORTANT: *Contact Xilinx Support if the overall system design includes the SEM IP prior to attempting to use the RLDRAM 3 memory interface.*

Xilinx does not recommend using the RLD3 IP with an interface rate of 800 MHz or higher when the SEM IP is enabled.

There is a risk of post-calibration data errors with RLD3 designs that span multiple FPGA banks when the SEM IP is enabled. For RLD3 designs with an 18-bit data bus and address multiplexing enabled, it is possible to fit the entire interface in one FPGA bank. Other configurations will not be able to fit in a single FPGA bank and are at risk when the SEM IP is enabled.

You should always disable the SEM IP during RLD3 calibration.

IMPORTANT: *This document supports RLDRAM 3 core v1.4.*

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Clocking
  - Resets
  - Protocol Description
  - Customizing and Generating the Core
  - Example Design

Core Overview

The Xilinx UltraScale™ architecture includes the RLDRAM 3 core. This core provides solutions for interfacing with these DRAM memory types. The UltraScale architecture for the RLDRAM 3 core is organized in the following high-level blocks:

- **Controller** – The controller accepts burst transactions from the User Interface and generates transactions to and from the RLDRAM 3. The controller takes care of the DRAM timing parameters and refresh.
- **Physical Layer** – The physical layer provides a high-speed interface to the DRAM. This layer includes the hard blocks inside the FPGA and the soft blocks calibration logic necessary to ensure optimal timing of the hard blocks interfacing to the DRAM.

The new hard blocks in the UltraScale architecture allow interface rates of up to 2,133 Mb/s to be achieved.

  - These hard blocks include:
    - Data serialization and transmission
    - Data capture and deserialization
    - High-speed clock generation and synchronization
    - Fine delay elements per pin with voltage and temperature tracking
  - The soft blocks include:
    - **Memory Initialization** – The calibration modules provide an initialization routine for RLDRAM 3. The delays in the initialization process are bypassed to speed up simulation time.
    - **Calibration** – The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance. Results of the calibration process are available through the Xilinx debug tools. After completion of calibration, the PHY layer presents raw interface to the DRAM.

- **Application Interface** – The "User Interface" layer provides a simple FIFO interface to the application. Data is buffered and read data is presented in request order.

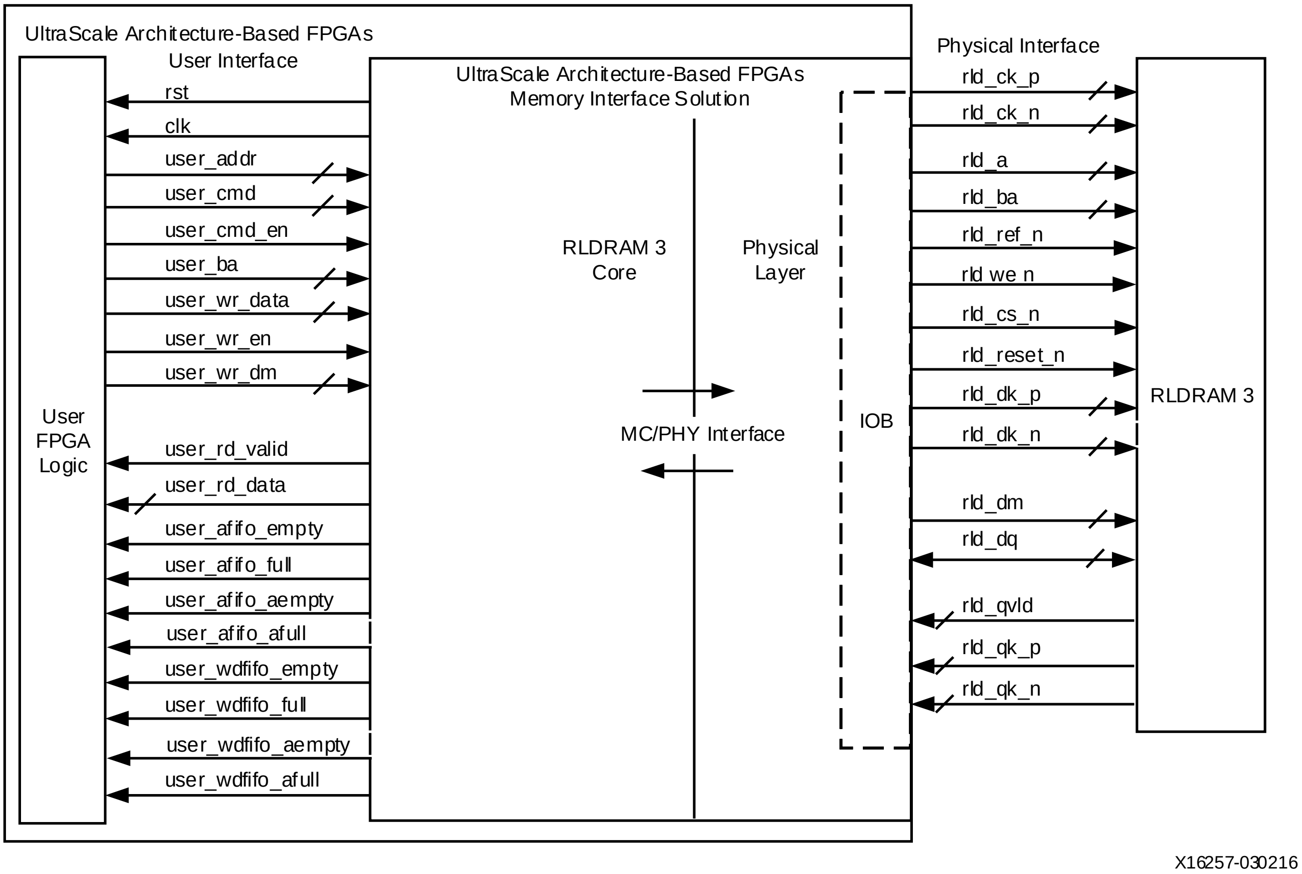


Figure 29-1: **UltraScale Architecture-Based FPGAs Memory Interface Solution**

Feature Summary

- Component support for interface widths of 18, 36, and 72 bits

Table 29-1: **Supported Configurations**

Interface Width	Burst Length	Number of Device
36	BL2, BL4	1, 2
18	BL2, BL4, BL8	1, 2
36 with address multiplexing	BL4	1, 2
18 with address multiplexing	BL4, BL8	1, 2

- ODT support
- Memory device support with 576 Mb and 1.125 Gb densities
- RLDRAM 3 initialization support

- Source code delivery in Verilog
- 4:1 memory to FPGA logic interface clock ratio
- Interface calibration and training information available through the Vivado hardware manager

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

Standards

For more information on UltraScale™ architecture documents, see References, page 789.

Performance

Maximum Frequencies

For more information on the maximum frequencies, see the following documentation:

- *Kintex UltraScale FPGAs Data Sheet, DC and AC Switching Characteristics* (DS892) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6]
- *UltraScale Maximum Memory Performance Utility* (XTP414) [Ref 21]

Resource Utilization

For full details about performance and resource utilization, visit Performance and Resource Utilization.

Port Descriptions

There are three port categories at the top-level of the memory interface core called the "user design."

- The first category is the memory interface signals that directly interfaces with the RLDRAM. These are defined by the Micron® RLDRAM 3 specification.
- The second category is the application interface signals which is the "user interface." These are described in the Protocol Description, page 511.
- The third category includes other signals necessary for proper operation of the core. These include the clocks, reset, and status signals from the core. The clocking and reset signals are described in their respective sections.

  The active-High **`init_calib_complete`** signal indicates that the initialization and calibration are complete and that the interface is now ready to accept commands for the interface.

Core Architecture

This chapter describes the UltraScale™ architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Overview

Figure 31-1 shows the UltraScale architecture-based FPGAs Memory Interface Solutions diagram.

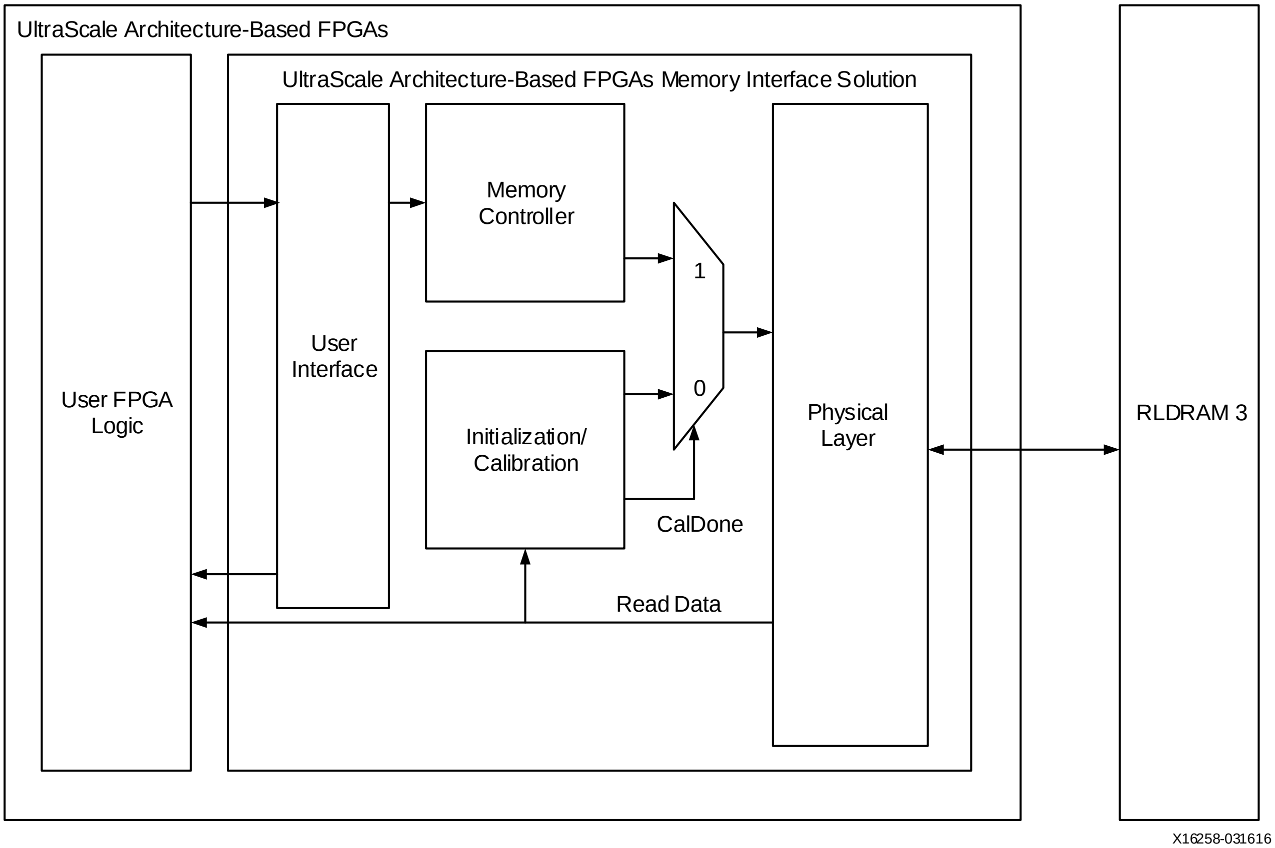


Figure 31-1: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core**

The user interface uses a simple protocol based entirely on SDR signals to make read and write requests. See User Interface in Chapter 32 for more details describing this protocol.

The Memory Controller takes commands from the user interface and adheres to the protocol requirements of the RLDRAM 3 device. See Memory Controller for more details.

The physical interface generates the proper timing relationships and DDR signaling to communicate with the external memory device, while conforming to the RLDRAM 3 protocol and timing requirements. See Physical Interface in Chapter 32 for more details.

Memory Controller

The Memory Controller (MC) enforces the RLDRAM 3 access requirements and interfaces with the PHY. The controller processes read and write commands in order for BL4 and BL8, so the commands presented to the controller is the order in which they are presented to the memory device. For BL2, the read commands are processed in order but the write commands are rearranged to increase the throughput.

The MC first receives commands from the user interface and determines if the command can be processed immediately or needs to wait. When all requirements are met, the command is placed on the PHY interface. For a write command, the controller generates a signal for the user interface to provide the write data to the PHY. This signal is generated based on the memory configuration to ensure the proper command-to-data relationship. Auto-refresh commands are inserted into the command flow by the controller to meet the memory device refresh requirements.

The data bus is shared for read and write data in RLDRAM 3. Switching from read commands to write commands and vice versa introduces gaps in the command stream due to switching the bus. For better throughput, changes in the command bus should be minimized when possible.

CMD_PER_CLK is a top-level parameter used to determine how many memory commands are provided to the controller per FPGA logic clock cycle. It depends on nCK_PER_CLK and the burst length. For example if nCK_PER_CLK = 4, the CMD_PER_CLK is set to 1 for burst length = 8 and CMD_PER_CLK is set to 2 for burst length = 4 and CMD_PER_CLK is set to 4 for burst length = 2.

PHY

The PHY is considered the low-level physical interface to an external RLDRAM 3 device as well as all calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

Overall PHY Architecture

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

The MC and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is divided by 4. A more detailed block diagram of the PHY design is shown in Figure 31-1.

The MC is designed to separate out the command processing from the low-level PHY requirements to ensure a clean separation between the controller and physical layer. The command processing can be replaced with custom logic if desired, while the logic for interacting with the PHY stays the same and can still be used by the calibration logic.

Table 31-1: **PHY Modules**

Module Name	Description
rld3_phy.sv	Contains infrastructure (infrastructure.sv), rld_cal.sv, rld_xiphy.sv, and MUXes between the calibration and the Memory Controller.
rld_iob.sv	Instantiates all byte IOB modules
rld_iob_byte.sv	Generates the I/O buffers for all the signals in a given byte lane.
rld_addr_mux.sv	Address MUX
rld_rd_bit_slip.sv	Read bitslip
rld_wr_lat.sv	Write latency
rld_xiphy.sv	Top-level XIPHY module

The PHY architecture encompasses all of the logic contained in `rld_xiphy.sv`. The PHY contains wrappers around dedicated hard blocks to build up the memory interface from smaller components. A byte lane contains all of the clocks, resets, and datapaths for a given subset of I/O. Multiple byte lanes are grouped together, along with dedicated clocking resources, to make up a single bank memory interface. For more information on the hard silicon physical layer architecture, see the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7].

Memory Initialization and Calibration Sequence

Immediately after power-up and on deassertion of system reset, built-in self-check (BISC) which is a PHY routine, is run to compensate the internal skews of the read data bits and the read capture clock. RLDRAM 3 power-up initialization routine, which is run through a RTL state machine, is triggered after successful completion of the BISC routine. When both the routines have run, the control is transferred to MicroBlaze™, which is a soft processor that calibrates the timing of the write and read data paths. At the end of calibration, BISC recalculates the ratio of the offsets between read/write data and their corresponding strobe clocks to track them over voltage and temperature.

Figure 31-2 shows the overall flow of memory initialization and the different stages of calibration.

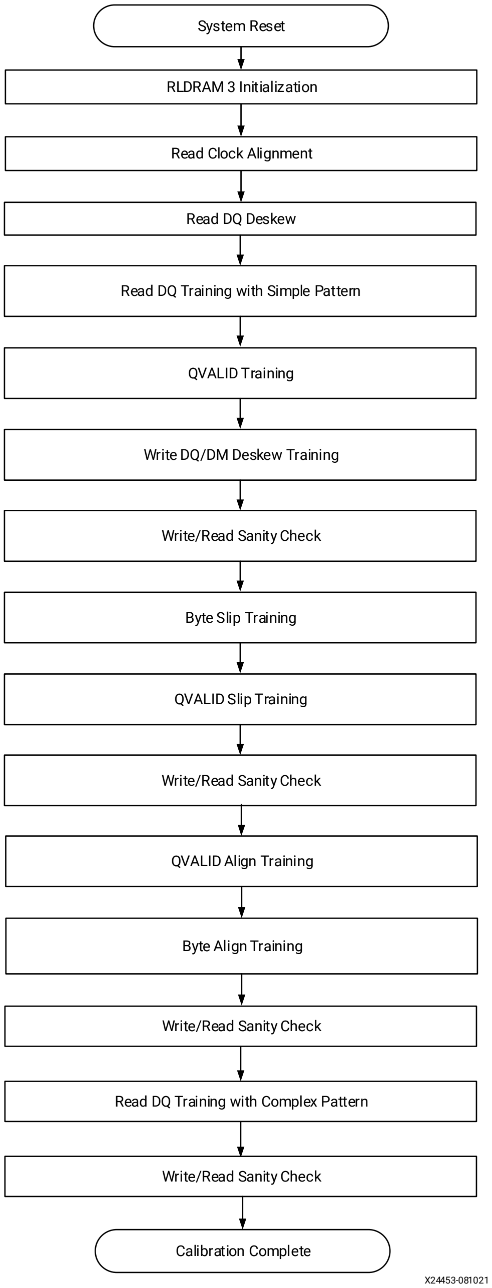


Figure 31-2: **Initialization and Calibration Sequence**

When simulating the RLDRAM 3 example design, the calibration process is bypassed to allow for quick traffic generation to and from the RLDRAM 3 device. Calibration is always enabled when running the example design in hardware. The hardware manager GUI provides information on the status of each calibration step or description of error in case of calibration failure.

If the hardware manager GUI is not used, the first step in determining the calibration status is to check the status of `init_calib_complete` and `calib_error` signals. The `init_calib_complete` only asserts if calibration passes successfully, otherwise `calib_error` is asserted. Calibration halts on the very first error encountered. There are three status registers, `dbg_pre_cal_status`, `dbg_cal_status`, and `dbg_post_cal_status` that provide information on the failing calibration stage. Each bit of the `dbg_cal_status` register represents a successful start/end of a calibration step while that for `dbg_pre_cal_status` and `dbg_post_cal_status` represent the successful completion of certain events during and after calibration. Not all bits are assigned and some bits might be reserved. Table 31-2 lists the pre-calibration status signal description.

Table 31-2: **Pre-Calibration XSDB Status Signal Description**

XSDB Status Register	XSDB Bits [8:0]	Description
RLD3_PRE_ CAL_STATUS	0	MicroBlaze started up successfully
	1	All PLLs in the interface have locked successfully
	2	BISC successfully completed initial calibration
	3	RLDRAM 3 initialization completed
	4	XSDB block RAM register setup complete
	5	Reserved
	6	Reserved
	7	Reserved
	8	Reserved

Table 31-3: **XSDB Status Signal Description**

XSDB Status Register	XSDB Bits[8:0]	dbg_cal_status Port Bits[31:0]	Status	Calibration Stage Name	Calibration Stage Number
RLD3_CAL_ STATUS_RANK0_0	0	0	Start	Read Clock Alignment	1
	1	1	Done		
	2	2	Start	Read DQ Deskew	2
	3	3	Done		
	4	4	Start	Read DQ Training (Simple)	3
	5	5	Done		
	6	6	Start	Read QVLD Training	4
	7	7	Done		
	8	8	Start	Write DQ/ DM Deskew	5
RLD3_CAL_ STATUS_RANK0_1	0	9	Done		
	1	10	Start	Write/ Read Sanity Check	6
	2	11	Done		
	3	12	Start	Byte Slip Training	7
	4	13	Done		
	5	14	Start	QVLD Slip Training	8
	6	15	Done		
	7	16	Start	Write/ Read Sanity Check	9
	8	17	Done		
RLD3_CAL_ STATUS_RANK0_2	0	18	Start	QVLD Align Training	10
	1	19	Done		
	2	20	Start	Byte Align Training	11
	3	21	Done		
	4	22	Start	Write/ Read Sanity Check	12
	5	23	Done		
	6	24	Start	Read DQ Training (Complex)	13
	7	25	Done		
	8	26	Start	Write/ Read Sanity Check	14
RLD3_CAL_ STATUS_RANK0_3	0	27	Done		
	1	28		Reserved	
	2	29		Reserved	
	3	30		Reserved	
	4	31		Reserved	

Table 31-4: **XSDB Post-Calibration XSDB Status Signal Description**

XSDB Status Register	XSDB Bits [8:0]	Description
RLD3_POST_ CAL_STATUS	0	PHY ready failed to assert
	1	Read margin started (stays asserted while running)
	2	Write margin started (stays asserted while running)
	3	Read margin failed
	4	Write margin failed
	5	Reserved
	6	Reserved
	7	Reserved
	8	Reserved

Read Clock Alignment

QK clock is required to be gated internally during various stages of RLDRAM 3 calibration in order to make delay adjustments. The internal gating signal needs to be aligned with the QK clock to prevent glitches from getting generating when the gating signals is released. Read clock alignment routine aligns the gating signal with the rising edge of the QK clock. Because QK is a free running clock, no write/read commands are issues and the internal gate delay adjustments are done based on coarse and fine gate delay taps (RL_DLY_COARSE and RL_DLY_FINE).

Read DQ Deskew

Read deskew routine helps to eliminate any delay variation within the DQ bits of a byte, which in turn improves the read DQ window size. During this stage of calibration, all DQ bits within a byte are deskewed by aligning them to the internal capture clock belonging to the same byte. The internal capture clock is a delayed version of QK and/or PQTR/NQTR delay taps of the capture clocks. The alignment is done by changing the IDELAY taps of individual DQs and/or of the capture clocks until all the bits in a byte are aligned.

A pattern of all `0s` and all `1s` is written to various locations in the RLDRAM 3 device. The write is done one location at a time with the data available on the memory bus four memory clock cycles ahead of the actual BL4 write data transaction and stays on the bus for two more memory clock cycles. Because the datapath is not calibrated at this point, this eliminates any critical timing between DQ and DK clock and ensures correct data is getting registered in the RLDRAM 3 device. The data read back appears as all `0s` and all `1s` over alternate general interconnect cycles. As an example, read data for a single DQ bit appears as a continuous stream of `00000000_11111111_00000000_11111111` over several memory clock cycles. Eight `0s` represent data over four memory clock cycles (one general interconnect clock cycle).

Read DQ Training (Simple)

Read DQ training is done to center the delayed version of QK capture clock within the read DQ window. This is done on a per nibble basis. Read Training Register (RTR) mode is enabled in the RLDRAM 3 device through the MRS2 command, which generates a continuous stream of `0101010101` pattern whenever a read transaction is issued. This provides a pattern for calibrating the internal clock by adjusting its PQTR/NQTR delays without having to write any pattern to the RLDRAM 3 device.

The routine initially searches for the left edge and when successful, looks for the right edge. This is done by moving the PQTR/NQTR delays of the capture clock. When both left and right edges of the read DQ window have been found, the routine centers the capture clock.

Read QVLD Training

This calibration step aligns the incoming **`qvalid`** signal to the negative edge of the internal capture clock. This gives the maximum margin when capturing the value of **`qvalid`** on the positive edge of capture clock, which in turn is presented at the User Interface.

Initially the **`qvalid`** signal is assigned the same IDELAY value as the corresponding QK clock of the byte it resides in. The **`qvalid`** IDELAY taps are then either incremented or decremented to align it to the negative edge of internal capture clock.

Write DQ/DM Deskew

Similar to read deskew routine, this routine aligns all the bits of the write data spanning either a single byte or two bytes depending on the number of bytes per DK clock. This is done by changing the ODELAY tap values of individual DQ bits and/ or the DK clocks until all bits are deskewed.

DQ bits associated with each DK clock are initially phase shifted by 90° to roughly align them with the DK clock. A repetitive pattern of `10101010` is written and read back from memory. Each DQ ODELAY tap is changed to fine tune the alignment with DK clock. When all DQs are edge aligned to DK clock, the 90° phase shift on DQs is removed, leaving the DK clock center aligned in the write data window.

The same 90° shift is done on the DM bit during DM calibration. To deskew DM, certain bits of the original pattern are masked and the pattern is changed to all `0s`. Alignment is achieved when the data bits with value 1 fail to get masked and are overwritten by value 0. The 90° phase shift on DM bits is removed at the end of alignment. DM deskew calibration is only performed when it is enabled at the time of RLDRAM 3 IP generation in Vivado Integrated Design Environment (IDE).

Write/Read Sanity Check Post Write DQ/DM Deskew

Since the write DQ/DM deskew alignment involves phase shifts and movement of DQ/DM and DK signals, a write/read check is performed to check the integrity of the write data path between the FPGA and RLDRAM 3 device. This is done by writing and reading the same `10101010` pattern as write DQ/DM deskew stage.

Byte Slip Training

Calibration algorithm treats `qvalid` signal similar to data, that is, it does not rely on `qvalid` signal to capture the incoming data. Instead, the read data along with `qvalid` is continuously captured and presented at the User Interface. Based on the initial synchronization between the memory clock domain and the general interconnect clock domain, read data inside the general interconnect clock domain might not align in the same phase as that sent to the memory device.

As an example, data written for a single bit as `00000000-11111111-00000000` might be seen in the general interconnect domain as `11110000-00001111-11110000`. To correct this, the read data is "slipped" by the required number of memory clock cycles, in half-memory cycle increments. In the above example, slip value of 4 is assigned by the algorithm to align the data in the general interconnect domain. The slip values are always an even number and range from 2 to 6.

QVLD Slip Training

Since the DQ bytes are "slipped" to correct the phase alignment in the earlier stage, the same slip must be applied to the `qvalid` signal to align them to the DQ bits. To account for delay variation, the slip calculation for `qvalid` is done independent of the DQ byte slip calculation. This allows for more calibration flexibility and accommodates wider range of delay variation between bytes and `qvalid` signal. The slip values are always an even number and range from 2 to 6.

Write/Read Sanity Check Post Byte/QVLD Slip Training

Since DQ byte and `qvalid` slip calibration are done independently, a write/read check is performed at the end of it to ensure the assigned slip values have aligned the read data correctly in the general interconnect. This is done by repeating the data pattern used during byte slip training and checking against the expected data pattern in the general interconnect clock domain.

QVLD Align Training/Byte Align Training

Slip training in the previous calibration step is done on a per byte basis. Depending on delay variation between bytes as well as the synchronization of individual QK clocks to the general interconnect clock domain, the read data for each byte might appear on different general interconnect cycles at the end of Byte/QVLD slip training. As an example, consider two bytes with the following data prior to slip calibration.

Table 31-5: **QVLD Align Training/Byte Align Training for Two Bytes**

Memory Clock Cycle	Byte 0			Byte 1		
	General Interconnect Cycle 0	General Interconnect Cycle 1	General Interconnect Cycle 2	General Interconnect Cycle 0	General Interconnect Cycle 1	General Interconnect Cycle 2
Rise 0	0x1	0x9	0x17	0xx	0x7	0x15
Fall 0	0x2	0x10	0x18	0xx	0x8	0x16
Rise 1	0x3	0x11	0x19	0x1	0x9	0x17
Fall 1	0x4	0x12	0x20	0x2	0x10	0x18
Rise 2	0x5	0x13	0x21	0x3	0x11	0x19
Fall 2	0x6	0x14	0x22	0x4	0x12	0x20
Rise 3	0x7	0x15	0x23	0x5	0x13	0x21
Fall 3	0x8	0x16	0x24	0x6	0x14	0x22

Byte slip calibration assigns slip value of 0 to Byte 0 and value of 6 to Byte 1. As a result, data in both bytes is offset by 1 general interconnect cycle. `qvalid` and byte align training is used to align the data between bytes. This is done by analyzing the spatial location of specific data within a byte relative to all other bytes in the general interconnect domain and adding an additional slip value of eight on top of the slip value from previous step to the bytes arriving one general interconnect cycle ahead of the other bytes.

Similar to the Slip stage, `qvalid` and byte align calibration are done independently as one `qvalid` spans two bytes in certain configurations and assigning slip value of one byte to it might cause the other byte to go out of sync.

Write/Read Sanity Check Post QVLD/Byte Align Training

Write/Read check is performed by reading back single general interconnect cycle worth of data to ensure all bytes and their corresponding `qvalid` signals are in alignment and appear at the same time at the User Interface.

Read DQ Training (Complex)

The final stage of read capture clock centering is done when most of the other calibration stages have successfully completed. This stage is similar to read training (Simple) calibration but instead of simple clock pattern, more complex data patterns are written and read from the memory device to fine tune the centering of read capture clock. The patterns attempt to induce SI effects such as ISI and noise to emulate traffic running in an actual system and centers the capture strobe based on the reduced DQ read window size. This provides better margin when running system traffic.

Final Write/Read Sanity Check

A final write/read check is done to ensure previous stages of calibration did not inadvertently leave the write or read path in a failing state. All the calibration steps done prior to this are done in burst length four mode and the RLDRAM 3 device is updated with the default burst length at the start of this stage of calibration. A single general interconnect cycle worth of write/read transaction is performed and checked against expected data.

When all calibration stages are completed, the `calib_complete` signal is asserted at the User Interface and the control of the write/read datapath through the XIPHY gets transferred from calibration module to the User Interface.

Reset Sequence

The `sys_rst` signal resets the entire memory design which includes general interconnect (fabric) logic which is driven by the MMCM clock (clkout0) and RIU logic. MicroBlaze™ and calibration logic are driven by the MMCM clock (clkout6). The `sys_rst` input signal is synchronized internally to create the `ui_clk_sync_rst` signal. The `ui_clk_sync_rst` reset signal is synchronously asserted and synchronously deasserted.

Figure 31-3 shows the `ui_clk_sync_rst` (fabric reset) is synchronously asserted with a few clock delays after `sys_rst` is asserted. When `ui_clk_sync_rst` is asserted, there are a few clocks before the clocks are shut off.

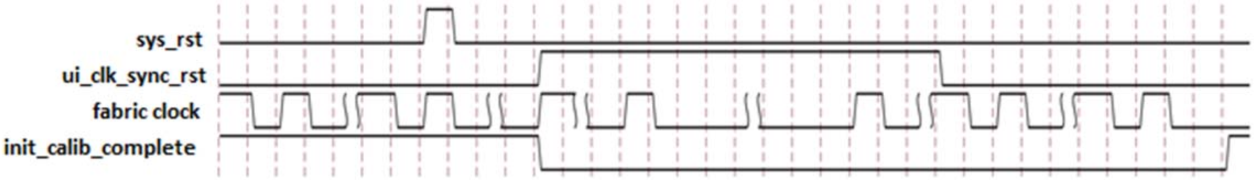


Figure 31-3: **Reset Sequence Waveform**

The following are the reset sequencing steps:

1. Reset to design is initiated after `ui_clk_sync_rst` goes High.
2. `init_calib_complete` signal goes Low when `ui_clk_sync_rst` is High.
3. Reset to design is deactivated after `ui_clk_sync_rst` is Low.
4. After `ui_clk_sync_rst` is deactivated, the `init_calib_complete` is asserted after calibration is completed.

MicroBlaze MCS ECC

The MicroBlaze MCS local memory provides an option to enable Error Correcting Code (ECC). Error correction corrects single bit errors and detects double bit errors. Two additional ports are added to indicate single bit errors (LMB_CE) and double bit errors (LMB_UE).

The MicroBlaze MCS ECC can be selected from the **MicroBlaze MCS ECC option** section in the **Advanced Options** tab. The block RAM size increases if the ECC option for MicroBlaze MCS is selected.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

Clocking

The memory interface requires one mixed-mode clock manager (MMCM), one TXPLL per I/O bank used by the memory interface, and two BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used.

Note: RLDRAM 3 generates the appropriate clocking structure and no modifications to the RTL are supported.

The RLDRAM 3 tool generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to MMCM (located in center bank of memory interface)
- MMCM to BUFG (located at center bank of memory interface) driving FPGA logic and all TXPLLs
- MMCM to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate FPGA logic
- Clocking pair of the interface must be in the same SLR of memory interface for the stacked silicon interconnect (SSI) technology devices

Requirements

GCIO

- Must use a differential I/O standard
- Must be in the same I/O column as the memory interface
- Must be in the same SLR of memory interface for the SSI technology devices
- The I/O standard and termination scheme are system dependent. For more information, consult the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 7].

MMCM

- MMCM is used to generate the FPGA logic system clock (1/4 of the memory clock)
- Must be located in the center bank of memory interface
- Must use internal feedback
- Input clock frequency divided by input divider must be ≥ 70 MHz (CLKINx / D ≥ 70 MHz)
- Must use integer multiply and output divide values

Input Clock Requirement

- The clock generator driving the GCIO should have jitter < 3 ps RMS.
- The input clock should always be clean and stable. The IP functionality is not guaranteed if this input system clock has a glitch, discontinuous, etc.
- No spread spectrum clock is allowed.

BUFGs and Clock Roots

- One BUFG is used to generate the system clock to FPGA logic and another BUFG is used to divide the system clock by two.
- BUFGs and clock roots must be located in center most bank of the memory interface.
    - For two bank systems, the bank with the higher number of bytes selected is chosen as the center bank. If the same number of bytes is selected in two banks, then the top bank is chosen as the center bank.
    - For four bank systems, either of the center banks can be chosen. RLDRAM 3 refers to the second bank from the top-most selected bank as the center bank.
    - Both the BUFGs must be in the same bank.

TXPLL

- CLKOUTPHY from TXPLL drives XIPHY within its bank
- TXPLL must be set to use a CLKFBOUT phase shift of 90°
- TXPLL must be held in reset until the MMCM lock output goes High
- Must use internal feedback

Figure 32-1 shows an example of the clocking structure for a three bank memory interface. The GCIO drives the MMCM located at the center bank of the memory interface. MMCM drives both the BUFGs located in the same bank. The BUFG (which is used to generate system clock to FPGA logic) output drives the TXPLLs used in each bank of the interface.

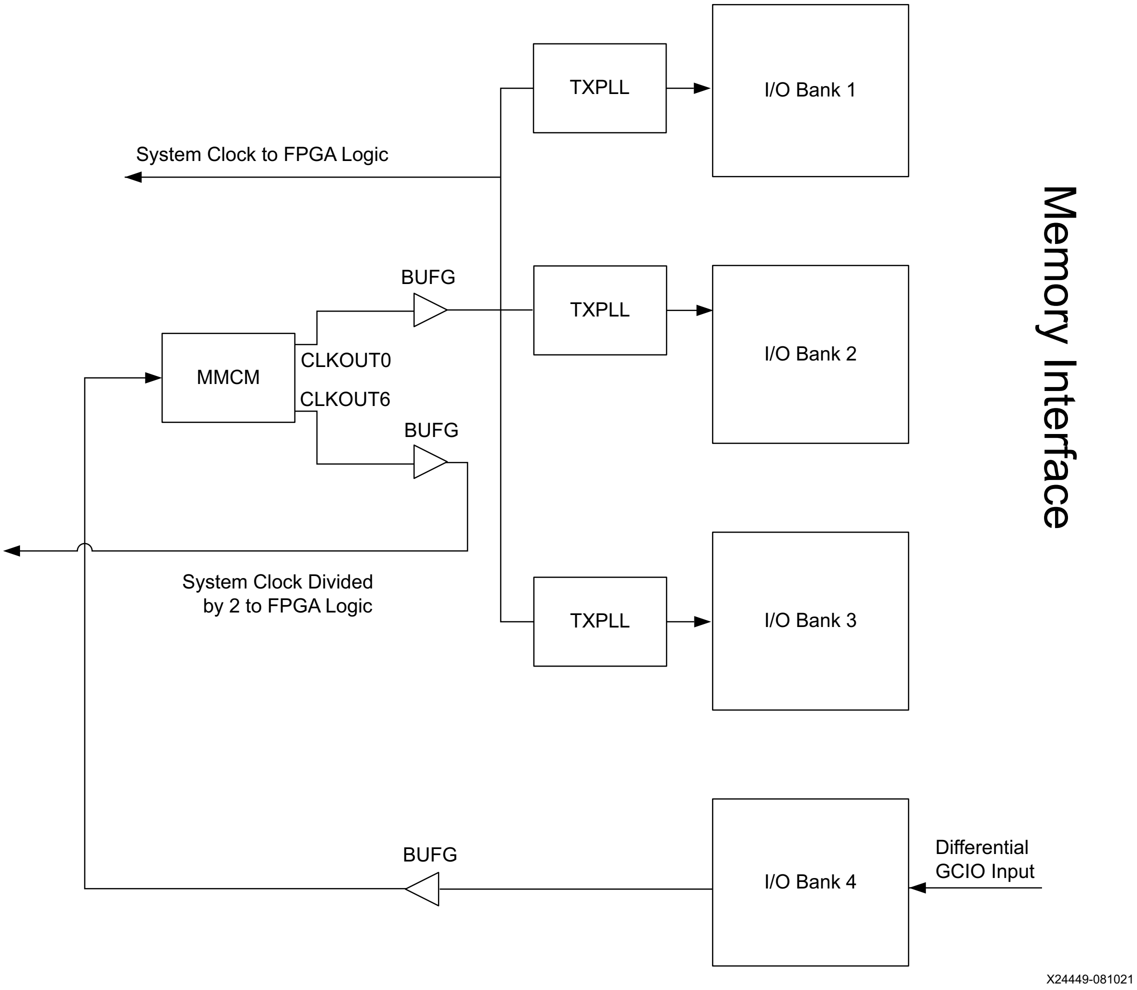


Figure 32-1: **Clocking Structure for Three Bank Memory Interface**

The MMCM is placed in the center bank of the memory interface.

- For two bank systems, MMCM is placed in a bank with the most number of bytes selected. If they both have the same number of bytes selected in two banks, then MMCM is placed in the top bank.
- For four bank systems, MMCM is placed in a second bank from the top.

For designs generated with System Clock configuration of **No Buffer**, MMCM must not be driven by another MMCM/PLL. Cascading clocking structures MMCM → BUFG → MMCM and PLL → BUFG → MMCM are not allowed.

If the MMCM is driven by the GCIO pin of the other bank, then the CLOCK_DEDICATED_ROUTE constraint with value "BACKBONE" must be set on the net that is driving MMCM or on the MMCM input. Setting up the CLOCK_DEDICATED_ROUTE constraint on the net is preferred. But when the same net is driving two MMCMs, the CLOCK_DEDICATED_ROUTE constraint must be managed by considering which MMCM needs the BACKBONE route.

In such cases, the CLOCK_DEDICATED_ROUTE constraint can be set on the MMCM input. To use the "BACKBONE" route, any clock buffer that exists in the same CMT tile as the GCIO must exist between the GCIO and MMCM input. The clock buffers that exists in the I/O CMT are BUFG, BUFGCE, BUFGCTRL, and BUFGCE_DIV. So RLDRAM 3 instantiates BUFG between the GCIO and MMCM when the GCIO pins and MMCM are not in the same bank (see Figure 32-1).

If the GCIO pin and MMCM are allocated in different banks, RLDRAM 3 generates CLOCK_DEDICATED_ROUTE constraints with value as "BACKBONE." If the GCIO pin and MMCM are allocated in the same bank, there is no need to set any constraints on the MMCM input.

Similarly when designs are generated with System Clock Configuration as a **No Buffer** option, you must take care of the "BACKBONE" constraint and the BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM if GCIO pin and MMCM are allocated in different banks. RLDRAM 3 does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations. For more information on clocking, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

XDC syntax for CLOCK_DEDICATED_ROUTE constraint is given here:

```
set_property CLOCK_DEDICATED_ROUTE BACKBONE [get_pins -hier -filter {NAME =~ */
u_rld3_infrastructure/gen_mmcme*.u_mmcme_adv_inst/CLKIN1}]
```

For more information on the CLOCK_DEDICATED_ROUTE constraints, see the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 9].

Note: If two different GCIO pins are used for two RLDRAM 3 IP cores in the same bank, center bank of the memory interface is different for each IP. RLDRAM 3 generates MMCM LOC and CLOCK_DEDICATED_ROUTE constraints accordingly.

Sharing of Input Clock Source (sys_clk_p)

If the same GCIO pin must be used for two IP cores, generate the two IP cores with the same frequency value selected for option **Reference Input Clock Period (ps)** and **System Clock Configuration** option as **No Buffer**. Perform the following changes in the wrapper file in which both IPs are instantiated:

1. RLDRAM 3 generates a single-ended input for system clock pins, such as `sys_clk_i`. Connect the differential buffer output to the single-ended system clock inputs (`sys_clk_i`) of both the IP cores.
2. System clock pins must be allocated within the same I/O column of the memory interface pins allocated. Add the pin LOC constraints for system clock pins and clock constraints in your top-level XDC.
3. You must add a "BACKBONE" constraint on the net that is driving the MMCM or on the MMCM input if GCIO pin and MMCM are not allocated in the same bank. Apart from this, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM to use the "BACKBONE" route.

Note:

- The UltraScale architecture includes an independent XIPHY power supply and TXPLL for each XIPHY. This results in clean, low jitter clocks for the memory system.
- Skew spanning across multiple BUFGs is not a concern because single point of contact exists between BUFG → TXPLL and the same BUFG → System Clock Logic.
- System input clock cannot span I/O columns because the longer the clock lines span, the more jitter is picked up.

TXPLL Usage

There are two TXPLLs per bank. If a bank is shared by two memory interfaces, both TXPLLs in that bank are used. One PLL per bank is used if a bank is used by a single memory interface. You can use a second PLL for other usage. To use a second PLL, you can perform the following steps:

1. Generate the design for the **System Clock Configuration** option as **No Buffer**.
2. RLDRAM 3 generates a single-ended input for system clock pins, such as `sys_clk_i`. Connect the differential buffer output to the single-ended system clock inputs (`sys_clk_i`) and also to the input of PLL (PLL instance that you have in your design).
3. You can use the PLL output clocks.

Additional Clocks

You can produce up to four additional clocks which are created from the same MMCM that generates `ui_clk`. Additional clocks can be selected from the **Clock Options** section in the **Advanced Options** tab. The GUI lists the possible clock frequencies from MMCM and the frequencies for additional clocks vary based on selected memory frequency (**Memory Device Interface Speed (ps)** value in the **Basic** tab), selected FPGA, and FPGA speed grade.

Resets

An asynchronous reset (`sys_rst`) input is provided. This is an active-High reset and the `sys_rst` must assert for a minimum pulse width of 5 ns. The `sys_rst` can be an internal or external pin.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must have a common reset input.*

For more information on reset, see the Reset Sequence in Chapter 31, Core Architecture.

PCB Guidelines for RLDRAM 3

Strict adherence to all documented RLDRAM 3 PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

Pin and Bank Rules

RLDRAM 3 Pin Rules

The rules are for single-rank memory interfaces.

- Address/control means `cs_n`, `ref_n`, `we_n`, `ba`, `ck`, `reset_n`, and `a`.
- All groups such as, Data, Address/Control, and System clock interfaces must be selected in a single column.
- Pins in a byte lane are numbered N0 to N12.
- Byte lanes in a bank are designed by T0, T1, T2, or T3. Nibbles within a byte lane are distinguished by a “U” or “L” designator added to the byte lane designator (T0, T1, T2, or T3). Thus they are T0L, T0U, T1L, T1U, T2L, T2U, T3L, and T3U.

Note: There are two PLLs per bank and a controller uses one PLL in every bank that is being used by the interface.

1. RLDRAM 3 interface can only be assigned to HP banks of the FPGA device.
2. Read Clock (`qk/qk_n`), Write Clock (`dk/dk_n`), `dq`, `qvld`, and `dm`.
   a. Read Clock pairs (`qkx_p/n`) must be placed on N0 and N1 pins. `dq` associated with a `qk/qk_n` pair must be in same byte lane on pins N2 to N11.
   b. For the data mask off configurations, ensure that `dm` pin on the RLDRAM 3 device is grounded. When data mask is enabled, one `dm` pin is associated with nine bits in x18 devices or with 18 bits in x36 devices. It must be placed in its associated `dq` byte lanes as listed:
      - For x18 part, `dm[0]` must be allocated in `dq[8:0]` allocated byte group and `dm[1]` must be allocated in `dq[17:9]`.
      - For x36 part, `dm[0]` must be allocated in `dq[8:0]` or `dq[26:18]` allocated byte lane. Similarly `dm[1]` must be allocated in `dq[17:9]` or `dq[35:27]` allocated byte group. `dq` and `dm` must be placed on one of the pins from N2 to N11 in the byte lane.
   c. `dk/dk_n` must be allocated to any P-N pair in the same byte lane as `ck/ck_n` in the address/control bank.

      Note: Pin 12 is not part of a pin pair and must not be used for differential clocks.
   d. `qvld` (x18 device) or `qvld0` (x36 device) must be placed on one of the pins from N2 to N12 in the `qk0` or `qk1` data byte lane. `qvld1` (x36 device) must be placed on one of the pins from N2 to N12 in of the `qk2` or `qk3` data byte lane.
3. Byte lanes are configured as either data or address/control.
   a. Pin N12 can be used for address/control in a data byte lane.
   b. No data signals (`qvalid, dq, dm`) can be placed in an address/control byte lane.
4. Address/control can be on any of the 13 pins in the address/control byte lanes. Address/control must be contained within the same bank. For three bank RLDRAM 3 interfaces, address/control must be in the centermost bank.
5. One `vrp` pin per bank is used and a DCI is required for the interfaces. A `vrp` pin is required in I/O banks containing inputs as well as output only banks. It is required in output only banks because address/control signals use SSTL12_DCI to enable usage of controlled output impedance. DCI cascade is allowed. When DCI cascade is selected, `vrp` pin can be used as a normal I/O. All rules for the DCI in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7] must be followed.
6. `ck` must be on the PN pair in the Address/Control byte lane.
7. `reset_n` can be on any pin as long as FPGA logic timing is met and I/O standard can be accommodated for the chosen bank (SSTL12).
8. Banks can be shared between two controllers.

 a. Each byte lane is dedicated to a specific controller (except for `reset_n`).
 b. Byte lanes from one controller cannot be placed inside the other. For example, with controllers A and B, "AABB" is allowed, while "ABAB" is not.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must share a common reset input.*

---

9. All I/O banks used by the memory interface must be in the same column.
10. All I/O banks used by the memory interface must be in the same SLR of the column for the SSI technology devices.
11. Maximum height of interface is three contiguous banks for 72-bit wide interface.
12. Bank skipping is not allowed.
13. The input clock for the MMCM in the interface must come from the a GCIO pair in the I/O column used for the memory interface. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements and MMCM requirements should be considered). For more information, see Clocking, page 501.
14. There are dedicated V_{REF} pins (not included in the rules above). If an external V_{REF} is not used, the V_{REF} pins must be pulled to ground by a resistor value specified in the *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7]. These pins must be connected appropriately for the standard in use.
15. The interface must be contained within the same I/O bank type (High Range or High Performance). Mixing bank types is not permitted with the exceptions of the `reset_n` in step 6 and the input clock mentioned in step 11.
16. RLDRAM 3 pins not mentioned in the cited pin rules (JTAG, MF, etc.) or ones that you choose not to use in your design must be connected as per Micron® RLDRAM 3 data sheet specification.
17. The system reset pin (`sys_rst_n`) must not be allocated to Pins N0 and N6 if the byte is used for the memory I/Os.

Pin Swapping

- Pins can swap freely within each byte group (data and address/control) (for more information, see the RLDRAM 3 Pin Rules, page 506).
- Byte groups (data and address/control) can swap easily with each other.
- Pins in the address/control byte groups can swap freely within and between their byte groups.
- No other pin swapping is permitted.

RLDRAM 3 Pinout Examples

IMPORTANT: *Due to the calibration stage, there is no need for* `set_input_delay/` `set_output_delay` *on the RLDRAM 3. Ignore the unconstrained inputs and outputs for RLDRAM 3 and the signals which are calibrated.*

Table 32-1 shows an example of an 18-bit RLDRAM 3 interface contained within one bank. This example is for a component interface using one x18 RLDRAM3 component with Address Multiplexing.

Table 32-1: **18-Bit RLDRAM 3 Interface Contained in One Bank**

Bank	Signal Name	Byte Group	I/O Type	Special Designation
1	qvld0	T3U_12	–	–
1	dq8	T3U_11	N	–
1	dq7	T3U_10	P	–
1	dq6	T3U_9	N	–
1	dq5	T3U_8	P	–
1	dq4	T3U_7	N	DBC-N
1	dq3	T3U_6	P	DBC-P
1	dq2	T3L_5	N	–
1	dq1	T3L_4	P	–
1	dq0	T3L_3	N	–
1	dm0	T3L_2	P	–
1	qk0_n	T3L_1	N	DBC-N
1	qk0_p	T3L_0	P	DBC-P
1	reset_n	T2U_12	–	–
1	we#	T2U_11	N	–
1	a18	T2U_10	P	–
1	a17	T2U_9	N	–
1	a14	T2U_8	P	–
1	a13	T2U_7	N	QBC-N
1	a10	T2U_6	P	QBC-P
1	a9	T2L_5	N	–
1	a8	T2L_4	P	–
1	a5	T2L_3	N	–
1	a4	T2L_2	P	–
1	a3	T2L_1	N	QBC-N

Table 32-1: **18-Bit RLDRAM 3 Interface Contained in One Bank *(Cont'd)***

Bank	Signal Name	Byte Group	I/O Type	Special Designation
1	a0	T2L_0	P	QBC-P
1	–	T1U_12	–	–
1	ba3	T1U_11	N	–
1	ba2	T1U_10	P	–
1	ba1	T1U_9	N	–
1	ba0	T1U_8	P	–
1	dk1_n	T1U_7	N	QBC-N
1	dk1_p	T1U_6	P	QBC-P
1	dk0_n	T1L_5	N	–
1	dk0_p	T1L_4	P	–
1	ck_n	T1L_3	N	–
1	ck_p	T1L_2	P	–
1	ref_n	T1L_1	N	QBC-N
1	cs_n	T1L_0	P	QBC-P
1	vrp	T0U_12	–	–
1	dq17	T0U_11	N	–
1	dq16	T0U_10	P	–
1	dq15	T0U_9	N	–
1	dq14	T0U_8	P	–
1	dq13	T0U_7	N	DBC-N
1	dq12	T0U_6	P	DBC-P
1	dq11	T0L_5	N	–
1	dq10	T0L_4	P	–
1	dq9	T0L_3	N	–
1	dm1	T0L_2	P	–
1	qk1_n	T0L_1	N	DBC-N
1	qk1_p	T0L_0	P	DBC-P

Protocol Description

This core has the following interfaces:

- Memory Interface
- User Interface
- Physical Interface

Memory Interface

The RLDRAM 3 core is customizable to support several configurations. The specific configuration is defined by Verilog parameters in the top-level of the core.

User Interface

The user interface connects to an FPGA user design to the RLDRAM 3 core to simplify interactions between the user design and the external memory device.

Command Request Signals

The user interface provides a set of signals used to issue a read or write command to the memory device. These signals are summarized in Table 32-2.

Table 32-2: **User Interface Request Signals**

Signal	I/O	Description
user_cmd_en	I	Command Enable. This signal issues a read or write request and indicates that the corresponding command signals are valid.
sys_clk_p/n	I	Primary clock to the IP.
sys_rst	I	Primary Active-High reset to the IP.

Table 32-2: **User Interface Request Signals *(Cont'd)***

Signal	I/O	Description
user_cmd[2 × CMD_PER_CLK – 1:0]	I	Command. This signal issues a read, write, or NOP request. When user_cmd_en is asserted: 2'b00 = Write Command 2'b01 = Read Command 2'b10 = NOP 2'b11 = NOP The NOP command is useful when more than one command per clock cycle must be provided to the Memory Controller yet not all command slots are required in a given clock cycle. The Memory Controller acts on the other commands provided and ignore the NOP command. NOP is not supported when CMD_PER_CLK == 1. CMD_PER_CLK is a top-level parameter used to determine how many memory commands are provided to the controller per FPGA logic clock cycle, it depends on nCK_PER_CLK and the burst length (see Figure 32-2)
user_addr[CMD_PER_CLK × ADDR_WIDTH – 1:0]	I	Command Address. This is the address to use for a command request. It is valid when user_cmd_en is asserted.
user_ba[CMD_PER_CLK × BANK_WIDTH – 1:0]	I	Command Bank Address. This is the address to use for a write request. It is valid when user_cmd_en is asserted.
user_wr_en	I	Write Data Enable. This signal issues the write data and data mask. It indicates that the corresponding user_wr_* signals are valid.
user_wr_data[2 × nCK_PER_CLK × DATA_WIDTH – 1:0]	I	Write Data. This is the data to use for a write request and is composed of the rise and fall data concatenated together. It is valid when user_wr_en is asserted.
user_wr_dm[2 × nCK_PER_CLK × DM_WIDTH – 1:0]	I	Write Data Mask. When active-High, the write data for a given selected device is masked and not written to the memory. It is valid when user_wr_en is asserted.
user_afifo_empty	O	Address FIFO empty. If asserted, the command buffer is empty.
user_wdfifo_empty	O	Write Data FIFO empty. If asserted, the write data buffer is empty.
user_afifo_full	O	Address FIFO full. If asserted, the command buffer is full, and any writes to the FIFO are ignored until deasserted.
user_wdfifo_full	O	Write Data FIFO full. If asserted, the write data buffer is full, and any writes to the FIFO are ignored until deasserted.

Table 32-2: **User Interface Request Signals *(Cont'd)***

Signal	I/O	Description
user_afifo_aempty	O	Address FIFO almost empty. If asserted, the command buffer is almost empty.
user_afifo_afull	O	Address FIFO almost full. If asserted, the command buffer is almost full.
user_wdfifo_aempty	O	Write Data FIFO almost empty. If asserted, the write data buffer is almost empty.
user_wdfifo_afull	O	Write Data FIFO almost full. If asserted, the Write Data buffer is almost full.
user_rd_valid[CMD_PER_CLK – 1:0]	O	Read Valid. This signal indicates that data read back from memory is available on user_rd_data and should be sampled.
user_rd_data[2 × nCK_PER_CLK × DATA_WIDTH – 1:0]	O	Read Data. This is the data read back from the read command.
init_calib_complete	O	Calibration Done. This signal indicates back to the user design that read calibration is complete and requests can now take place.
cx_rld3_ui_clk	O	This User Interface clock should be one quarter of the RLDRAM3 clock.
cx_rld3_ui_clk_sync_rst	O	This is the active-High user interface reset.
cx_calib_error	O	When asserted indicates error during calibration.
dbg_clk	O	Debug Clock. Do not connect any signals to dbg_clk and keep the port open during instantiation.

Interfacing with the Core through the User Interface

The width of certain user interface signals is dependent on the system clock frequency and the burst length. This allows the client to send multiple commands per FPGA logic clock cycle as might be required for certain configurations.

Note: Both write and read commands in the same `user_cmd` cycle is not allowed.

Figure 32-2 shows the `user_cmd` signal and how it is made up of multiple commands depending on the configuration.

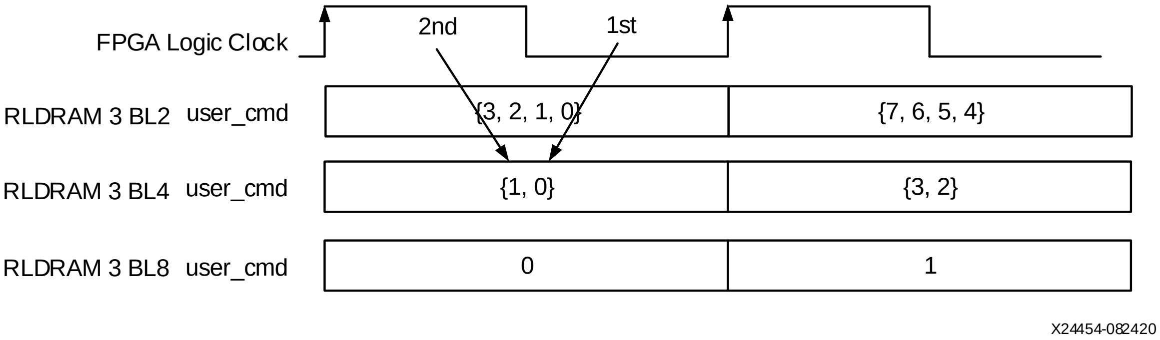


Figure 32-2: **Multiple Commands for user_cmd Signal**

As shown in Figure 32-2, four command slots are present in a single user interface clock cycle for BL2. Similarly, two command slots are present in a single user interface clock cycle for BL4. These command slots are serviced sequentially and the return data for read commands are presented at the user interface in the same sequence. Note that the read data might not be available in the same slot as that of its read command. The slot of a read data is determined by the timing requirements of the controller and its command slot. One such example is mentioned in the following BL2 design configuration.

Assume that the following set of commands is presented at the user interface for a given user interface cycle.

Table 32-3: **Command Set in User Interface Cycle**

Slots	Commands
0	RD0
1	NOP
2	RD1
3	NOP

It is not guaranteed that the read data appears in {DATA0, NOP, DATA1, NOP} order. It might also appear in {NOP, DATA0, NOP, DATA1} or {NOP, NOP, DATA0, DATA1} etc. orders. In any case, the sequence of the commands are maintained.

User Address Bit Allocation Based on RLDRAM 3 Configuration

Based on the RLDRAM 3 device selection, address width at the user interface is set in the multiple of 20 bits in case of 576 Mb device and 21 bits in case of 1.125 Gb device. Depending on the RLDRAM 3 device configuration, the actual address width can be less than the maximum address bits of 20 or 21 stated earlier. The width of the address bus does not include bank address bits. Table 32-4 summarizes the address width for various RLDRAM 3 configurations.

The address bits at the user interface are concatenated based on the burst length as shown in Table 32-4. Pad the unused address bits with zero. An example for x36 burst length 4 576 Mb device configuration is shown here:

{00, (18-bit address), 00, (18-bit address)}

Table 32-4: **User Address Width for 576 Mb and 1.125 Gb**

Burst Length	RLDRAM 3 Device Data Width	Address Width at RLDRAM 3 Interface		Address Width at User Interface
		Non-Multiplexed Mode	Multiplexed Mode	
576 Mb				
2	18	20	Not supported by RLDRAM 3	{20, 20, 20, 20}
2	36	19	Not supported by RLDRAM 3	({0, 19}, {0, 19}, {0, 19}, {0, 19})
4	18	19	11	({0, 19}, {0, 19})
4	36	18	11	({00, 18},{00, 18})
8	18	18	11	({00, 18})
8	36	Not supported by RLDRAM 3	Not supported by RLDRAM 3	N/A
1.125 Gb				
2	18	21	Not supported by RLDRAM 3	{21, 21, 21, 21}
2	36	20	Not supported by RLDRAM 3	({0, 20}, {0, 20}, {0, 20}, {0, 20})
4	18	20	11	({0, 20}, {0, 20})
4	36	19	11	({00, 19},{00, 19})
8	18	19	11	({00, 19})
8	36	Not supported by RLDRAM 3	Not supported by RLDRAM 3	N/A

Notes:

1. Two device configurations (2x18, 2x36) follow the same address mapping as one device configuration mentioned.

The user interface protocol for the RLDRAM 3 four-word burst architecture is shown in Figure 32-3.

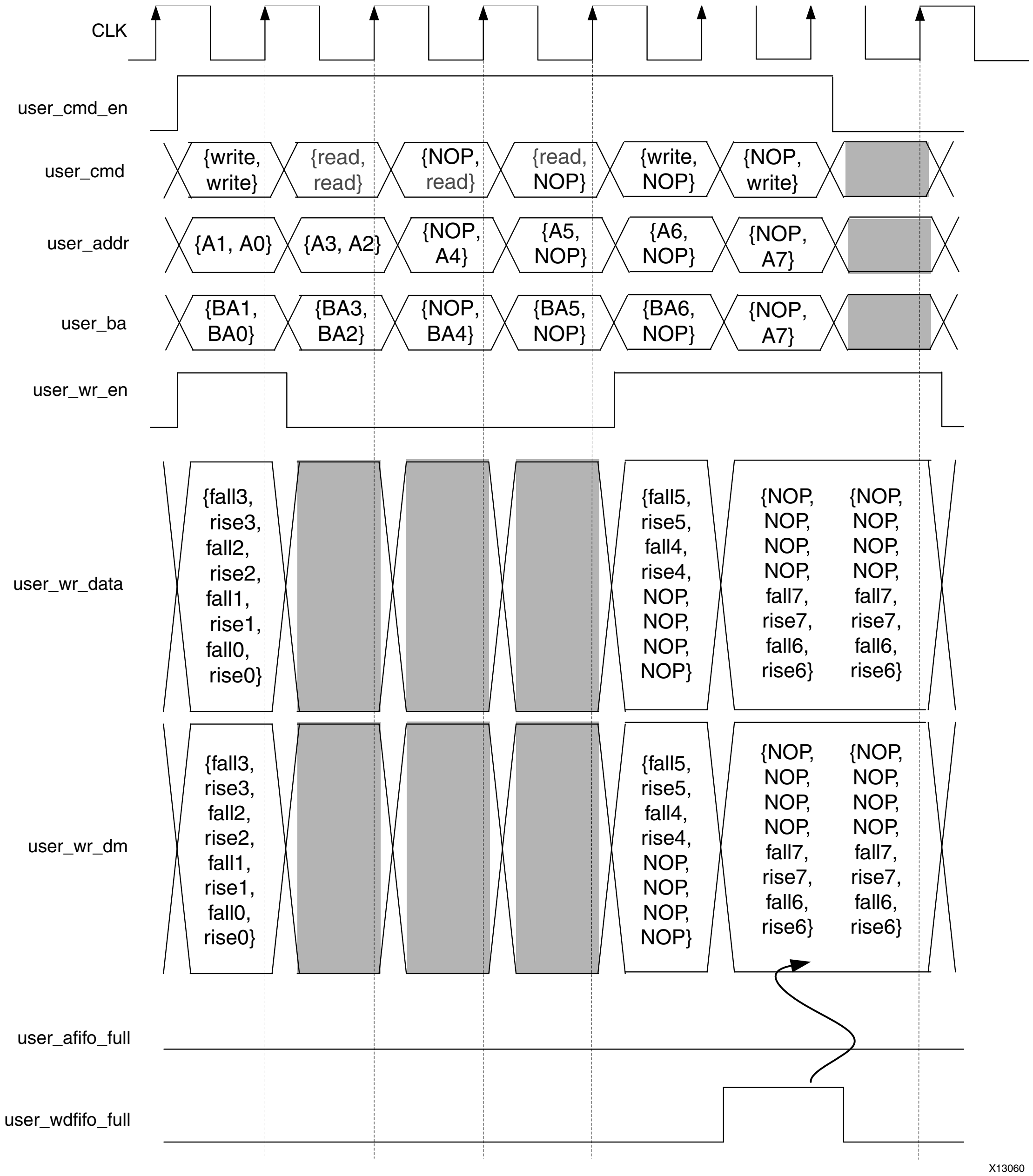


Figure 32-3: **RLDRAM 3 User Interface Protocol (Four-Word Burst Architecture)**

Before any requests can be accepted, the `ui_clk_sync_rst` signal must be deasserted Low. After the `ui_clk_sync_rst` signal is deasserted, the user interface FIFOs can accept commands and data for storage. The `init_calib_complete` signal is asserted after the memory initialization procedure and PHY calibration are complete, and the core can begin to service client requests.

A command request is issued by asserting **`user_cmd_en`** as a single cycle pulse. At this time, the **`user_cmd`**, **`user_addr`**, and **`user_ba`** signals must be valid. To issue a read request, **`user_cmd`** is set to **`2`b01`**, while for a write request, **`user_cmd`** is set to **`2'b00`**. For a write request, the data is to be issued in the same cycle as the command by asserting the **`user_wr_en`** signal High and presenting valid data on **`user_wr_data`** and **`user_wr_dm`**. The user interface protocol for the RLDRAM 3 eight-word burst architecture is shown in Figure 32-4.

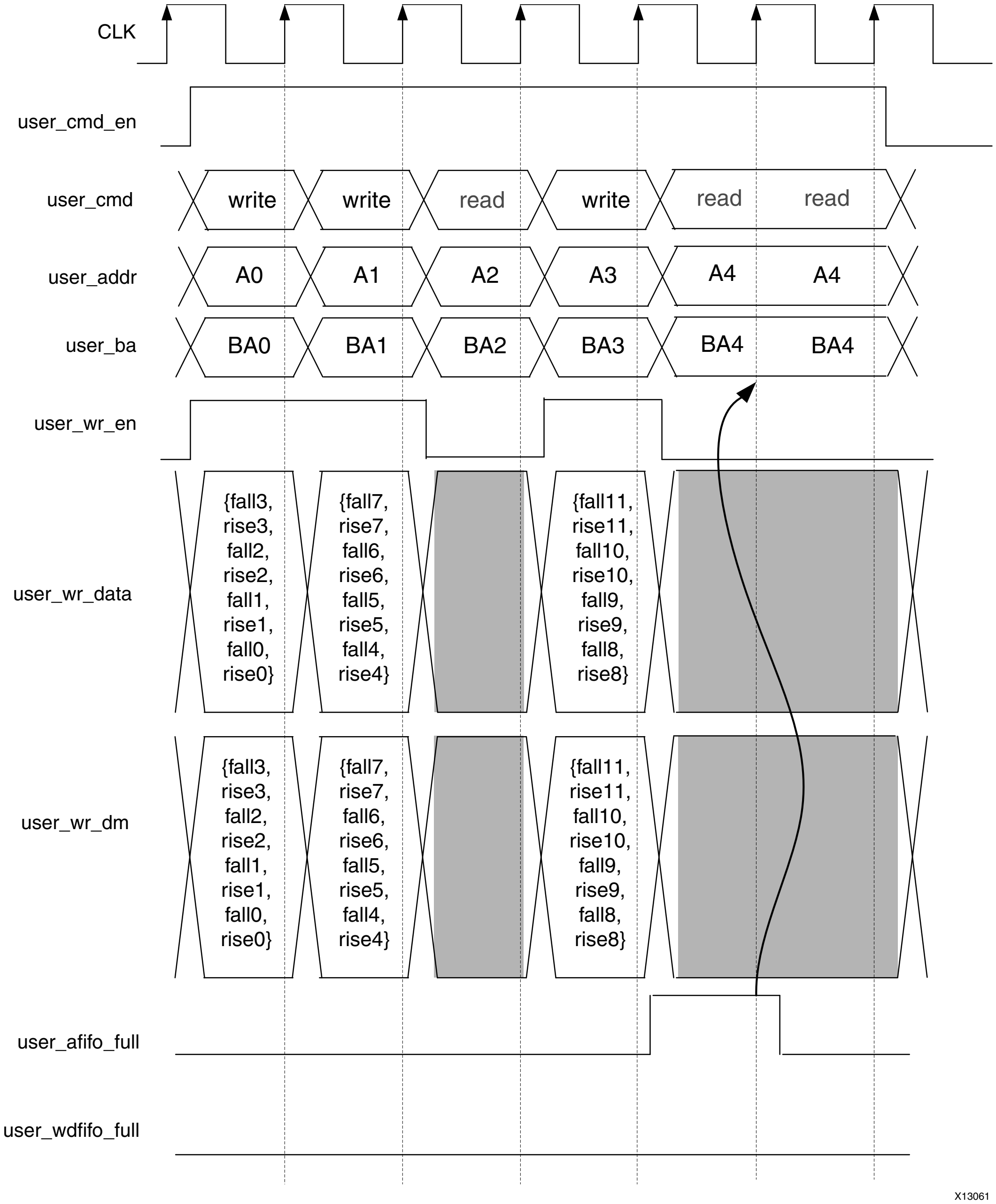


Figure 32-4: **RLDRAM 3 User Interface Protocol (Eight-Word Burst Architecture)**

When a read command is issued some time later (based on the configuration and latency of the system), the **`user_rd_valid[0]`** signal is asserted, indicating that **`user_rd_data`** is now valid, while **`user_rd_valid[1]`** is asserted indicating that **`user_rd_data`** is valid, as shown in Figure 32-5. The read data should be sampled on the same cycle that **`user_rd_valid[0]`** and **`user_rd_valid[1]`** are asserted because the core does not buffer returning data. This functionality can be added in, if desired.

The Memory Controller only puts commands on certain slots to the PHY such that the **`user_rd_valid`** signals are all asserted together and return the full width of data, but the extra **`user_rd_valid`** signals are provided in case of controller modifications.

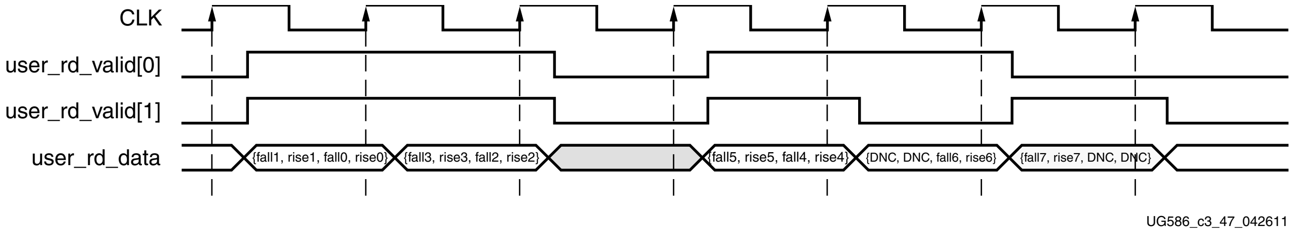


Figure 32-5: **User Interface Protocol Read Data**

Physical Interface

The physical interface is the connection from the FPGA core to an external RLDRAM 3 device. The I/O signals for this interface are defined in Table 32-5. These signals can be directly connected to the corresponding signals on the RLDRAM 3 device.

Table 32-5: **Physical Interface Signals**

Signal	I/O	Description
rld_ck_p	O	System Clock CK. This is the address/command clock to the memory device.
rld_ck_n	O	System Clock CK#. This is the inverted system clock to the memory device.
rld_dk_p	O	Write Clock DK. This is the write clock to the memory device.
rld_dk_n	O	Write Clock DK#. This is the inverted write clock to the memory device.
rld_a	O	Address. This is the address supplied for memory operations.
rld_ba	O	Bank Address. This is the bank address supplied for memory operations.
rld_cs_n	O	Chip Select CS#. This is the active-Low chip select control signal for the memory.
rld_we_n	O	Write Enable WE#. This is the active-Low write enable control signal for the memory.
rld_ref_n	O	Refresh REF#. This is the active-Low refresh control signal for the memory.
rld_dm	O	Data Mask DM. This is the active-High mask signal, driven by the FPGA to mask data that a user does not want written to the memory during a write command.
rld_dq	I/O	Data DQ. This is a bidirectional data port, driven by the FPGA for writes and by the memory for reads.
rld_qk_p	I	Read Clock QK. This is the read clock returned from the memory edge aligned with read data on rld_dq. This clock (in conjunction with QK#) is used by the PHY to sample the read data on rld_dq.

Table 32-5: **Physical Interface Signals *(Cont'd)***

Signal	I/O	Description
rld_qk_n	I	Read Clock QK#. This is the inverted read clock returned from the memory. This clock (in conjunction with QK) is used by the PHY to sample the read data on rld_dq.
rld_reset_n	O	RLDRAM 3 reset pin. This is the active-Low reset to the RLDRAM 3 device.
rld_qvld	I	This active-High data valid port indicates that the valid input data is available on the subsequent rising clock edge.

M and D Support for Reference Input Clock Speed

Memory IPs provide two possibilities to select the Reference Input Clock Speed. Value allowed for Reference Input Clock Speed (ps) is always ≥ Memory Device Interface Speed (ps).

- Memory IP lists the possible Reference Input Clock Speed values based on the targeted memory frequency (based on selected Memory Device Interface Speed).
- Otherwise, select **M and D Options** and target for desired Reference Input Clock Speed which is calculated based on selected CLKFBOUT_MULT (M), DIVCLK_DIVIDE (D), and CLKOUT0_DIVIDE (D0) values in the Advanced Clocking Tab.

The required Reference Input Clock Speed is calculated from the M, D, and D0 values entered in the GUI using the following formulas:

- MMCM_CLKOUT (MHz) = tCK / Phy_Clock_Ratio

  Where tCK is the Memory Device Interface Speed selected in the Basic tab.

- CLKIN (MHz) = (MMCM_CLKOUT (MHz) × D × D0) / M

  CLKIN (MHz) is the calculated Reference Input Clock Speed.

- VCO (MHz) = (CLKIN (MHz)) / D

  VCO (MHz) is the calculated VCO frequency.

- PFD (MHz) = CLKIN (MHz) / D

  PFD (MHz) is the calculated PFD frequency.

Calculated Reference Input Clock Speed from M, D, and D0 values are validated as per clocking guidelines. For more information on clocking rules, see Clocking.

Apart from the memory specific clocking rules, validation of the possible MMCM input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values are completed for M, D, and D0 in the GUI.

For UltraScale devices, see *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2] and *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For UltraScale+ devices, see *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 4], *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 5], and *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 6] for MMCM Input frequency range, MMCM VCO frequency range, and MMCM PFD frequency range values.

For possible M, D, and D0 values and detailed information on clocking and the MMCM, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 8].

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]

Customizing and Generating the Core

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

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the **`validate_bd_design`** command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

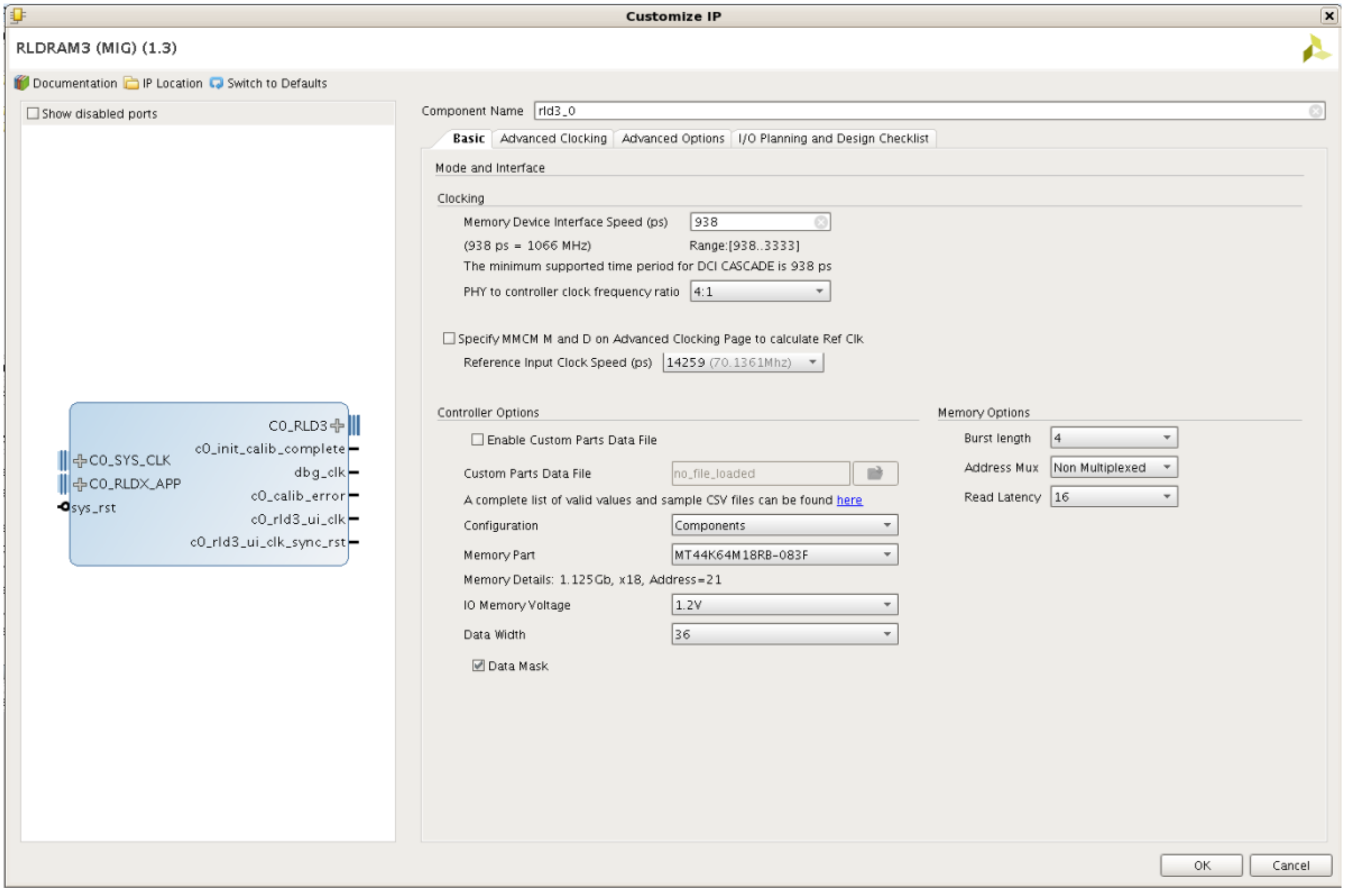
1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For more information about generating the core in Vivado, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

Basic Tab

Figure 33-1 shows the **Basic** tab when you start up the RLDRAM 3 SDRAM.



Component Name rld3_0

Mode and Interface

Clocking

Memory Device Interface Speed (ps) 938

(938 ps = 1066 MHz) Range:[938..3333]

The minimum supported time period for DCI CASCADE is 938 ps

PHY to controller clock frequency ratio 4:1

☐ Specify MMCM M and D on Advanced Clocking Page to calculate Ref Clk

Reference Input Clock Speed (ps) 14259 (70.1361Mhz)

Controller Options

☐ Enable Custom Parts Data File

Custom Parts Data File no_file_loaded

A complete list of valid values and sample CSV files can be found here

Configuration Components

Memory Part MT44K64M18RB-083F

Memory Details: 1.125Gb, x18, Address=21

IO Memory Voltage 1.2V

Data Width 36

☑ Data Mask

Memory Options

Burst length 4

Address Mux Non Multiplexed

Read Latency 16

Figure 33-1: **Vivado Customize IP Dialog Box – Basic**

IMPORTANT: *All parameters shown in the controller options dialog box are limited selection options in this release.*

For the Vivado IDE, all controllers (DDR3, DDR4, LPDDR3, QDR II+, QDR-IV, and RLDRAM 3) can be created and available for instantiation.

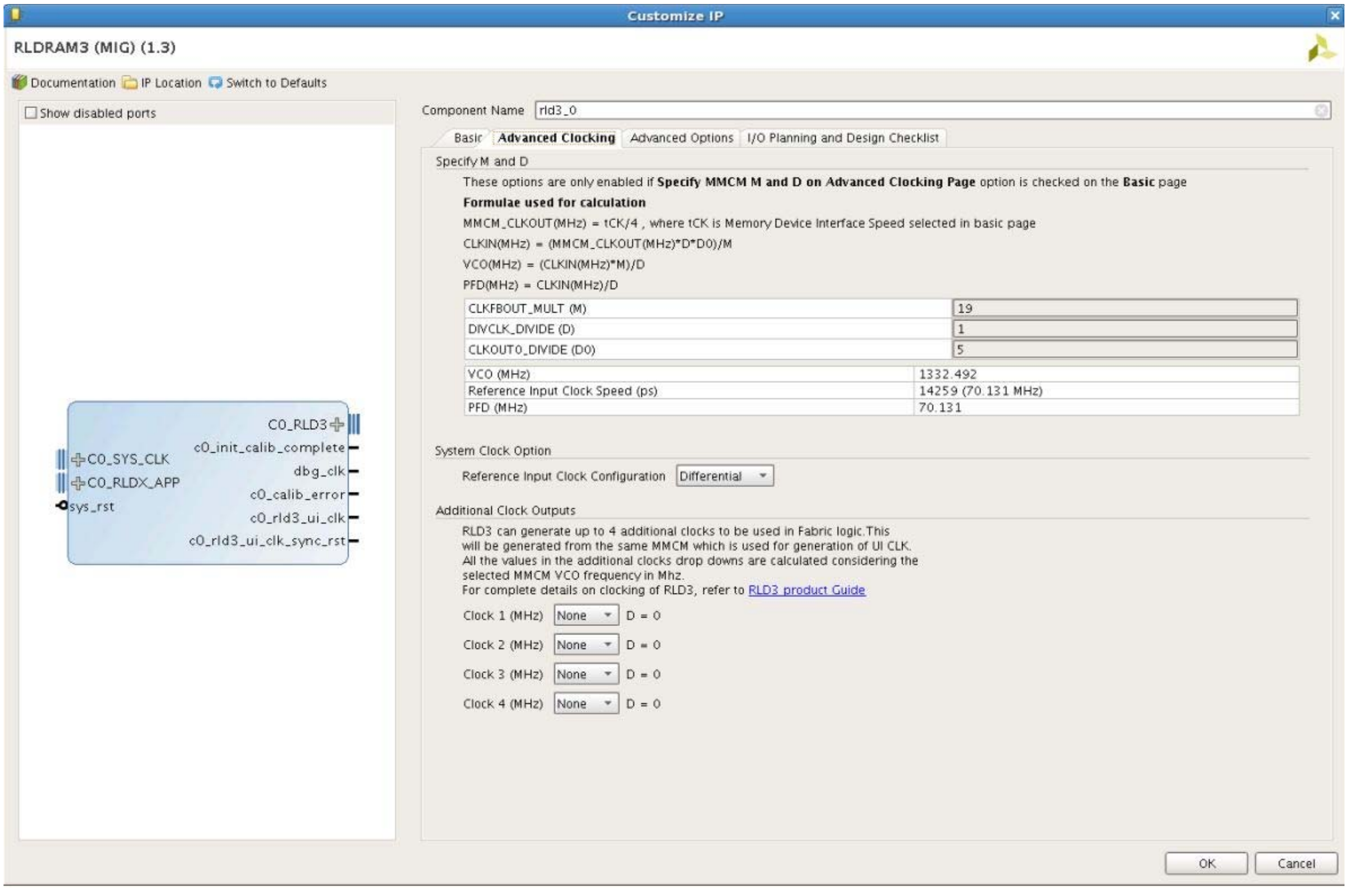
1. Select the settings in the **Clocking**, **Controller Options**, and **Memory Options**.

In **Clocking**, the **Memory Device Interface Speed** sets the speed of the interface. The speed entered drives the available **Reference Input Clock Speeds**. For more information on the clocking structure, see the Clocking, page 501.

2. To use memory parts which are not available by default through the RLDRAM 3 SDRAM Vivado IDE, you can create a custom parts CSV file, as specified in the AR: 63462. This CSV file has to be provided after enabling the **Custom Parts Data File** option. After selecting this option. you are able to see the custom memory parts along with the default memory parts. Note that, simulations are not supported for the custom part. Custom part simulations require manually adding the memory model to the simulation and might require modifying the test bench instantiation.

Advanced Clocking Tab

Figure 33-2 shows the next tab called **Advanced Clocking**. This displays the settings for **Specify M and D** value, **System Clock Options**, and **Additional Clock Outputs** for the specific controller.



Component Name: rld3_0

Specify M and D

These options are only enabled if **Specify MMCM M and D on Advanced Clocking Page** option is checked on the **Basic** page

Formulae used for calculation

MMCM_CLKOUT(MHz) = tCK/4 , where tCK is Memory Device Interface Speed selected in basic page

CLKIN(MHz) = (MMCM_CLKOUT(MHz)*D*D0)/M

VCO(MHz) = (CLKIN(MHz)*M)/D

PFD(MHz) = CLKIN(MHz)/D

CLKFBOUT_MULT (M)	19
DIVCLK_DIVIDE (D)	1
CLKOUT0_DIVIDE (D0)	5

VCO (MHz)	1332.492
Reference Input Clock Speed (ps)	14259 (70.131 MHz)
PFD (MHz)	70.131

System Clock Option

Reference Input Clock Configuration: Differential

Additional Clock Outputs

RLD3 can generate up to 4 additional clocks to be used in Fabric logic.This will be generated from the same MMCM which is used for generation of UI CLK. All the values in the additional clocks drop downs are calculated considering the selected MMCM VCO frequency in Mhz. For complete details on clocking of RLD3, refer to RLD3 product Guide

Clock 1 (MHz): None D = 0

Clock 2 (MHz): None D = 0

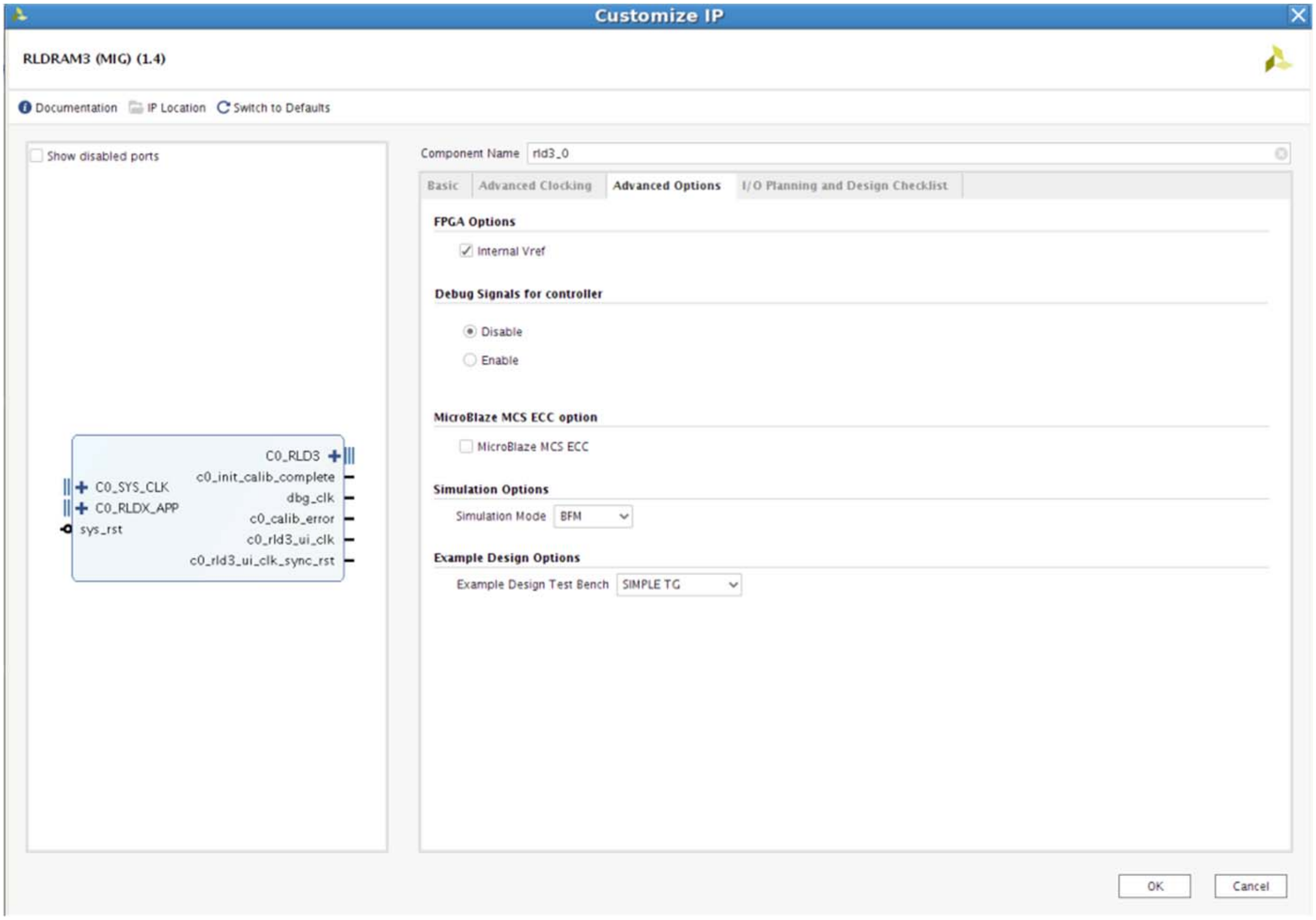
Clock 3 (MHz): None D = 0

Clock 4 (MHz): None D = 0

Figure 33-2: **Vivado Customize IP Dialog Box – Advanced Clocking**

Advanced Options Tab

Figure 33-3 shows the next tab called **Advanced Options**. This displays the advanced memory options for the specific controller.



☐ Show disabled ports

Component Name: rld3_0

FPGA Options

☑ Internal Vref

Debug Signals for controller

● Disable

○ Enable

MicroBlaze MCS ECC option

☐ MicroBlaze MCS ECC

Simulation Options

Simulation Mode: BFM

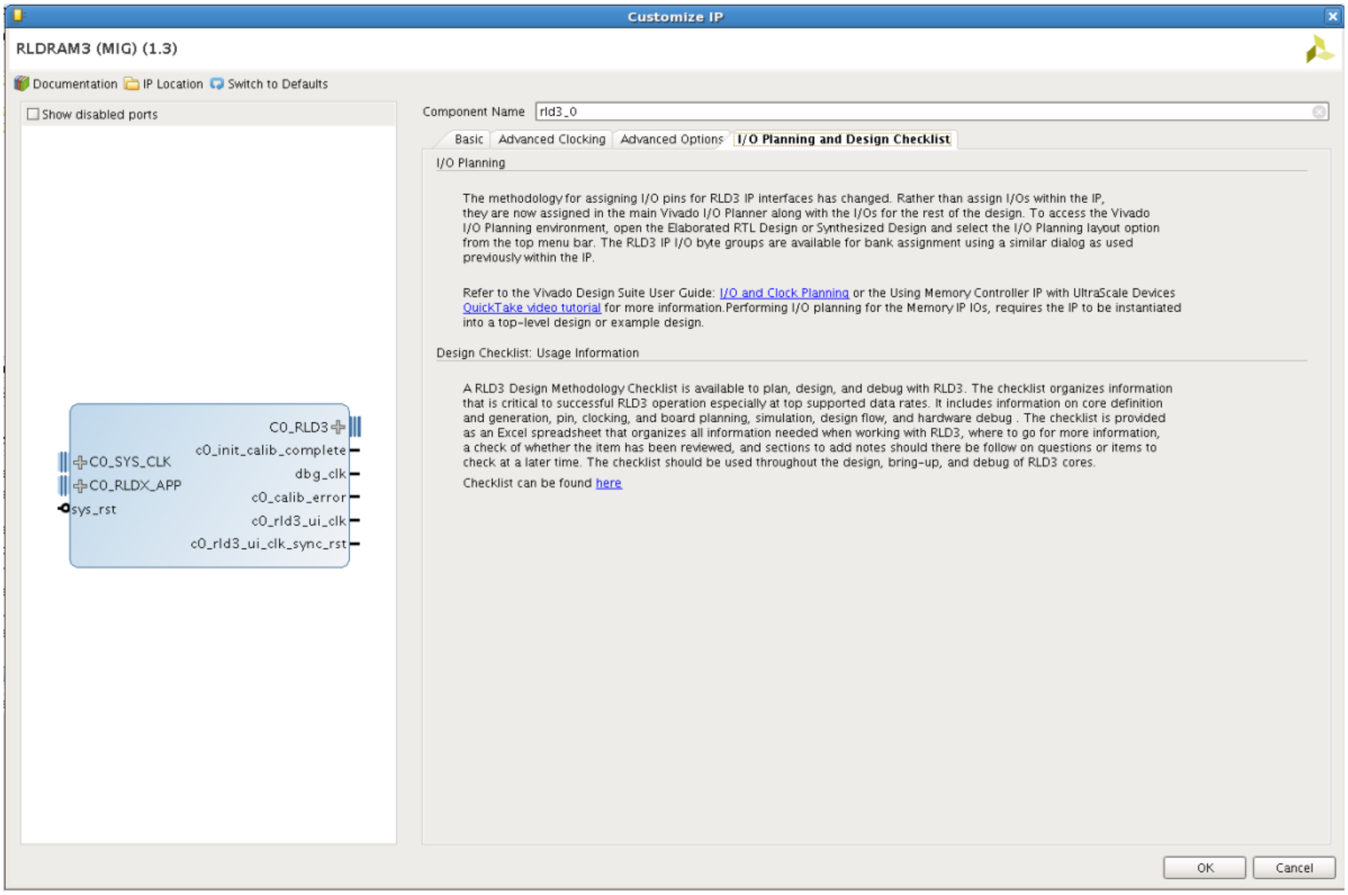
Example Design Options

Example Design Test Bench: SIMPLE TG

Figure 33-3: **Vivado Customize IP Dialog Box – Advanced Options**

RLDRAM 3 SDRAM I/O Planning and Design Checklist Tab

Figure 33-4 shows the **RLDRAM 3 SDRAM I/O Planning and Design Checklist** usage information.



Component Name: rld3_0

I/O Planning

The methodology for assigning I/O pins for RLD3 IP interfaces has changed. Rather than assign I/Os within the IP, they are now assigned in the main Vivado I/O Planner along with the I/Os for the rest of the design. To access the Vivado I/O Planning environment, open the Elaborated RTL Design or Synthesized Design and select the I/O Planning layout option from the top menu bar. The RLD3 IP I/O byte groups are available for bank assignment using a similar dialog as used previously within the IP.

Refer to the Vivado Design Suite User Guide: I/O and Clock Planning or the Using Memory Controller IP with UltraScale Devices QuickTake video tutorial for more information.Performing I/O planning for the Memory IP IOs, requires the IP to be instantiated into a top-level design or example design.

Design Checklist: Usage Information

A RLD3 Design Methodology Checklist is available to plan, design, and debug with RLD3. The checklist organizes information that is critical to successful RLD3 operation especially at top supported data rates. It includes information on core definition and generation, pin, clocking, and board planning, simulation, design flow, and hardware debug . The checklist is provided as an Excel spreadsheet that organizes all information needed when working with RLD3, where to go for more information, a check of whether the item has been reviewed, and sections to add notes should there be follow on questions or items to check at a later time. The checklist should be used throughout the design, bring-up, and debug of RLD3 cores.

Checklist can be found here

Figure 33-4: **Vivado Customize IP Dialog Box – I/O Planning and Design Checklist**

User Parameters

Table 33-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 33-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value
System Clock Configuration	System_Clock	Differential
Internal V_{REF}	Internal_Vref	TRUE
DCI Cascade	DCI_Cascade	FALSE
Debug Signal for Controller	Debug_Signal	Disable
Clock 1 (MHz)	ADDN_UI_CLKOUT1_FREQ_HZ	None
Clock 2 (MHz)	ADDN_UI_CLKOUT2_FREQ_HZ	None

Table 33-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value
Clock 3 (MHz)	ADDN_UI_CLKOUT3_FREQ_HZ	None
Clock 4 (MHz)	ADDN_UI_CLKOUT4_FREQ_HZ	None
Enable System Ports	Enable_SysPorts	TRUE
Default Bank Selections	Default_Bank_Selections	FALSE
Reference Clock	Reference_Clock	FALSE
Enable System Ports	Enable_SysPorts	TRUE
Clock Period (ps)	C0.RLD3_TimePeriod	1,071
Input Clock Period (ps)	C0.RLD3_InputClockPeriod	13,947
General Interconnect to Memory Clock Ratio	C0.RLD3_PhyClockRatio	4:1
Configuration	C0.RLD3_MemoryType	Components
Memory Part	C0.RLD3_MemoryPart	MT44K16M36RB-093
Data Width	C0.RLD3_DataWidth	36
Data Mask	C0.RLD3_DataMask	TRUE
Burst Length	C0.RLD3_BurstLength	8
Memory Voltage	C0.RLD3_MemoryVoltage	1.2

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Setting TWTR Check Parameter OFF for RLDRAM 3 Designs

This TWTR_CHECK_OFF switch provides the ability to turn OFF TWTR timing check inside the RLDRAM 3 controller. The default value of TWTR_CHECK parameter for RLDRAM 3 is set to ON. In many cases, it has been observed that some user traffic patterns never execute the TWTR timing. If the TWTR_CHECK_OFF switch is set to OFF, then the whole logic is bypassed. This can potentially help improve timing as well as improved bus efficiency. This can be changed through the Tcl command using the user parameter TWTR_CHECK_OFF for any RLDRAM 3 designs. Table shows details of the TWTR_CHECK_OFF user parameter.

Note: Do not turn this timing check off unless the access pattern will never cause a TWTR failure.

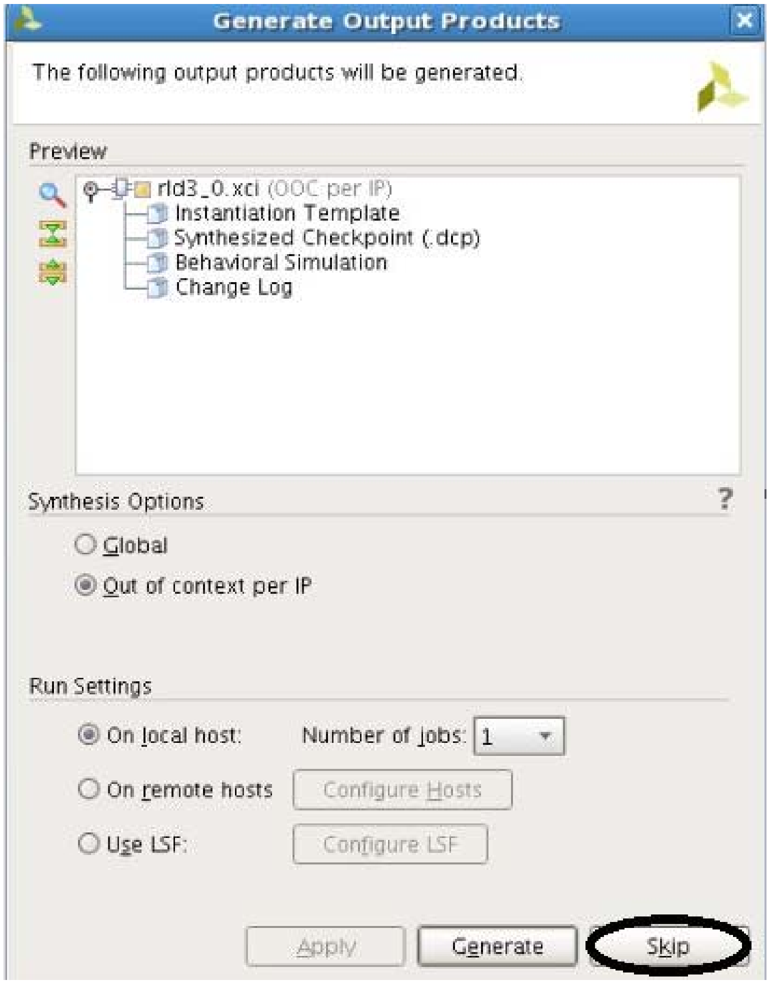
Table 33-2: **TWTR_CHECK_OFF User Parameter**

User Parameter	Value Format	Default Value	Possible Values
TWTR_CHECK_OFF	String	false	False – TWTR_CHECK parameter set to ON True – TWTR_CHECK parameter set to OFF

Follow these steps to change the TWTR check parameter value.

1. Generate RLDRAM 3 IP.

2. In the **Generate Output Products** option, do not select **Generate** instead select **Skip** (Figure 33-5).



The following output products will be generated.

- rld3_0.xci (OOC per IP)
  - Instantiation Template
  - Synthesized Checkpoint (.dcp)
  - Behavioral Simulation
  - Change Log

○ Global
● Out of context per IP

● On local host: Number of jobs: 1
○ On remote hosts
○ Use LSF:

Figure 33-5: **Generate Output Products Window – Skip**

3. Set the TWTR_CHECK_OFF value by running the following command on the Tcl Console:

```
set_property -dict [list config.TWTR_CHECK_OFF <value_to_be_set>] [get_ips
<ip_name>]
```

For example:

```
set_property -dict [list config.TWTR_CHECK_OFF {true}] [get_ips rld3_0]
```

4. Generate output files by selecting **Generate Output Products** after right-clicking IP (Figure 33-6).

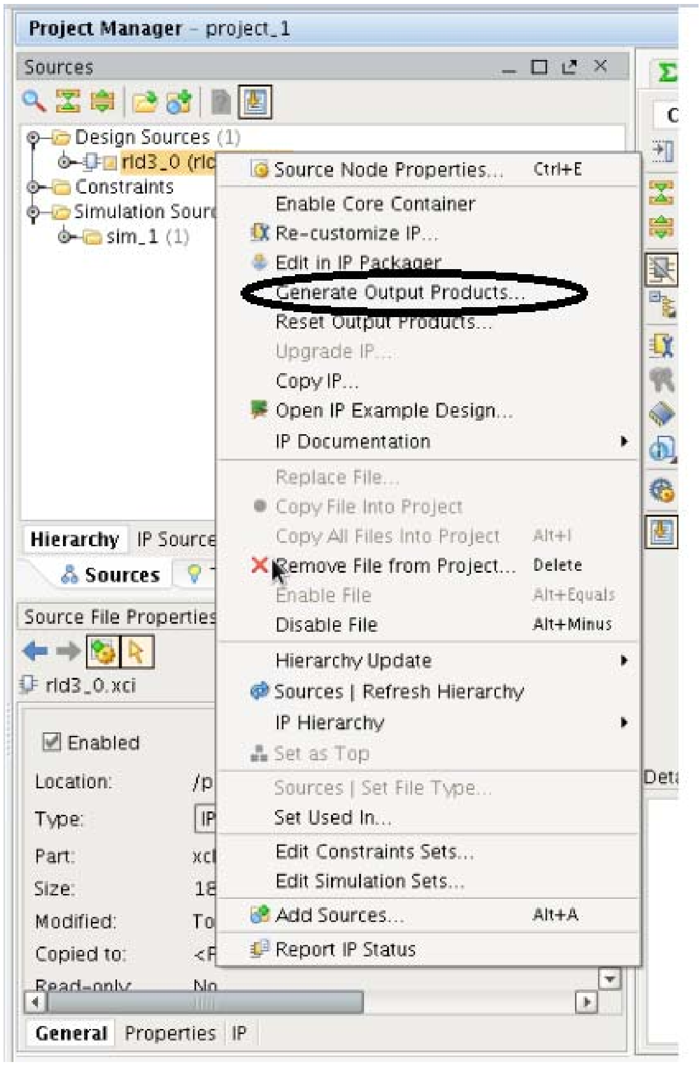


Figure 33-6: **Generate Output Products – Output Files**

The generated output files have the TWTR_CHECK parameter value set as per the selected value.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

I/O Planning

For details on I/O planning, see I/O Planning, page 235.

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite, if applicable.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For information on clocking, see Clocking, page 501.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The RLDRAM 3 tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

IMPORTANT: *The* `set_input_delay` *and* `set_output_delay` *constraints are not needed on the external memory interface pins in this design due to the calibration process that automatically runs at start-up. Warnings seen during implementation for the pins can be ignored.*

Simulation

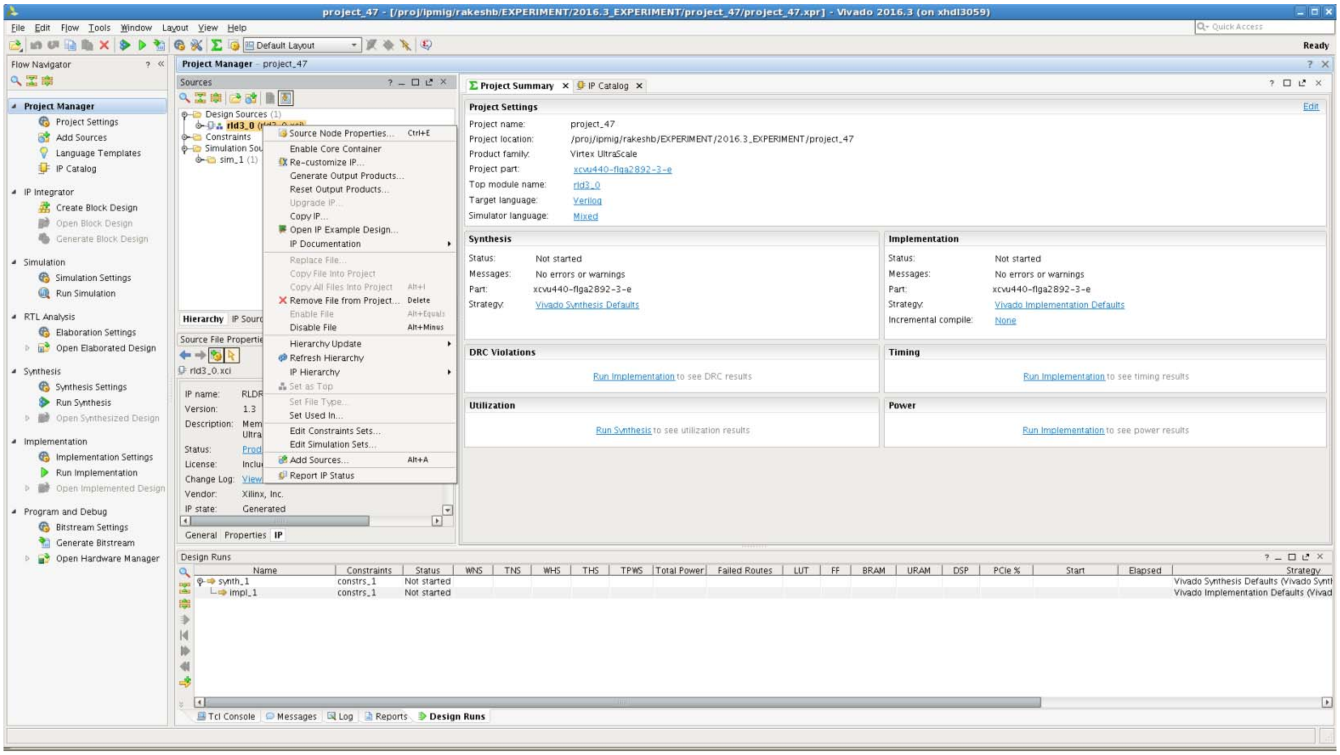
For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite. Vivado supports Open IP Example Design flow. To create the example design using this flow, right-click the IP in the **Source Window**, as shown in Figure 34-1 and select **Open IP Example Design**.



IP name: RLDR
Version: 1.3
Vendor: Xilinx, Inc.
IP state: Generated

Project Settings

Project name: project_47
Project location: /proj/ipmig/rakeshb/EXPERIMENT/2016.3_EXPERIMENT/project_47
Product family: Virtex UltraScale
Project part: xcvu440-flga2892-3-e
Top module name: rld3_0
Target language: Verilog
Simulator language: Mixed

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xcvu440-flga2892-3-e
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xcvu440-flga2892-3-e
Strategy: Vivado Implementation Defaults
Incremental compile: None

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	PCIe %	Start	Elapsed	Strategy
synth_1	constrs_1	Not started																Vivado Synthesis Defaults (Vivado Synt
impl_1	constrs_1	Not started																Vivado Implementation Defaults (Vivad

Figure 34-1: **Open IP Example Design**

This option creates a new Vivado project. Upon selecting the menu, a dialog box to enter the directory information for the new design project opens.

Select a directory, or use the defaults, and click **OK**. This launches a new Vivado with all of the example design files and a copy of the IP.

Simulating the Example Design (Designs with Standard User Interface)

The example design provides a synthesizable test bench to generate a fixed simple data pattern to the Memory Controller. This test bench consists of an IP wrapper and an **`example_tb`** that generates 100 writes and 100 reads.

The example design can be simulated using one of the methods in the following sections.

Project-Based Simulation

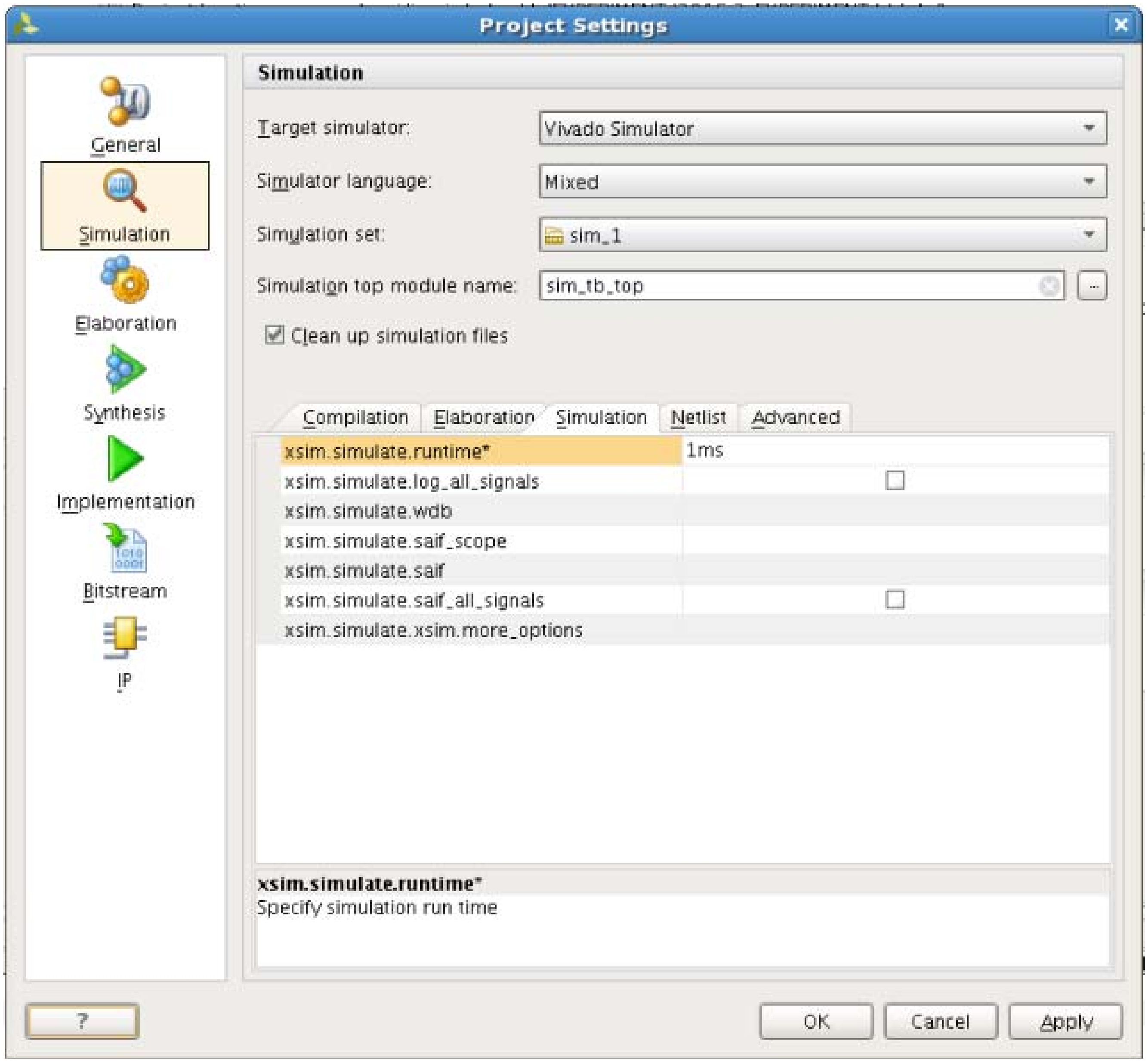
This method can be used to simulate the example design using the Vivado Integrated Design Environment (IDE). Memory IP delivers memory models for RLDRAM 3.

The Vivado simulator, Questa Advanced Simulator, IES, and VCS tools are used for RLDRAM 3. IP verification at each software release. The Vivado simulation tool is used for RLDRAM 3. IP verification from 2015.1 Vivado software release. The following subsections describe steps to run a project-based simulation using each supported simulator tool.

Project-Based Simulation Flow Using Vivado Simulator

1. In the **Open IP Example Design** Vivado project, under **Flow Navigator**, select **Simulation Settings**.
2. Select **Target simulator** as **Vivado Simulator**.

   Under the **Simulation** tab, set the **`xsim.simulate.runtime`** to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 34-2. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Set the **Simulation Language** to **Mixed**.
4. Apply the settings and select **OK**.



Simulation

Target simulator: Vivado Simulator

Simulator language: Mixed

Simulation set: sim_1

Simulation top module name: sim_tb_top

☑ Clean up simulation files

xsim.simulate.runtime*	1ms
xsim.simulate.log_all_signals	☐
xsim.simulate.wdb	
xsim.simulate.saif_scope	
xsim.simulate.saif	
xsim.simulate.saif_all_signals	☐
xsim.simulate.xsim.more_options	

xsim.simulate.runtime*
Specify simulation run time

Figure 34-2: **Simulation with Vivado Simulator**

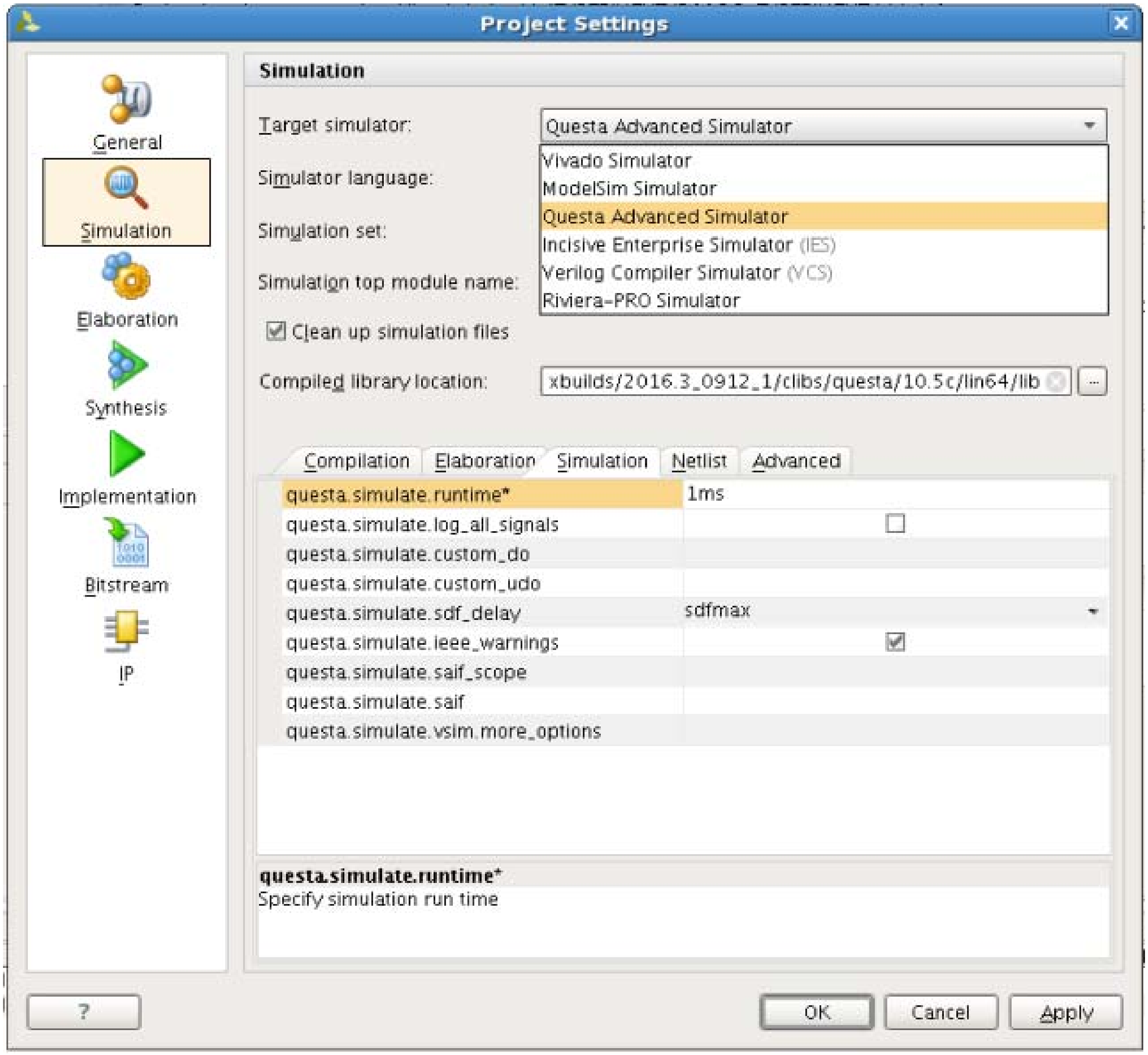
5. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 34-3.

Figure 34-3: **Run Behavioral Simulation**

6. Vivado invokes Vivado simulator and simulations are run in the Vivado simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using Questa Advanced Simulator

1. Open a RLDRAM 3 example Vivado project (**Open IP Example Design**...), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as **Questa Advanced Simulator**.
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `modelsim.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 34-4. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



Simulation

Target simulator: Questa Advanced Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: xbuilds/2016.3_0912_1/clibs/questa/10.5c/lin64/lib

questa.simulate.runtime*	1ms
questa.simulate.log_all_signals	☐
questa.simulate.custom_do	
questa.simulate.custom_udo	
questa.simulate.sdf_delay	sdfmax
questa.simulate.ieee_warnings	☑
questa.simulate.saif_scope	
questa.simulate.saif	
questa.simulate.vsim.more_options	

questa.simulate.runtime*
Specify simulation run time

Figure 34-4: **Simulation with Questa Advanced Simulator**

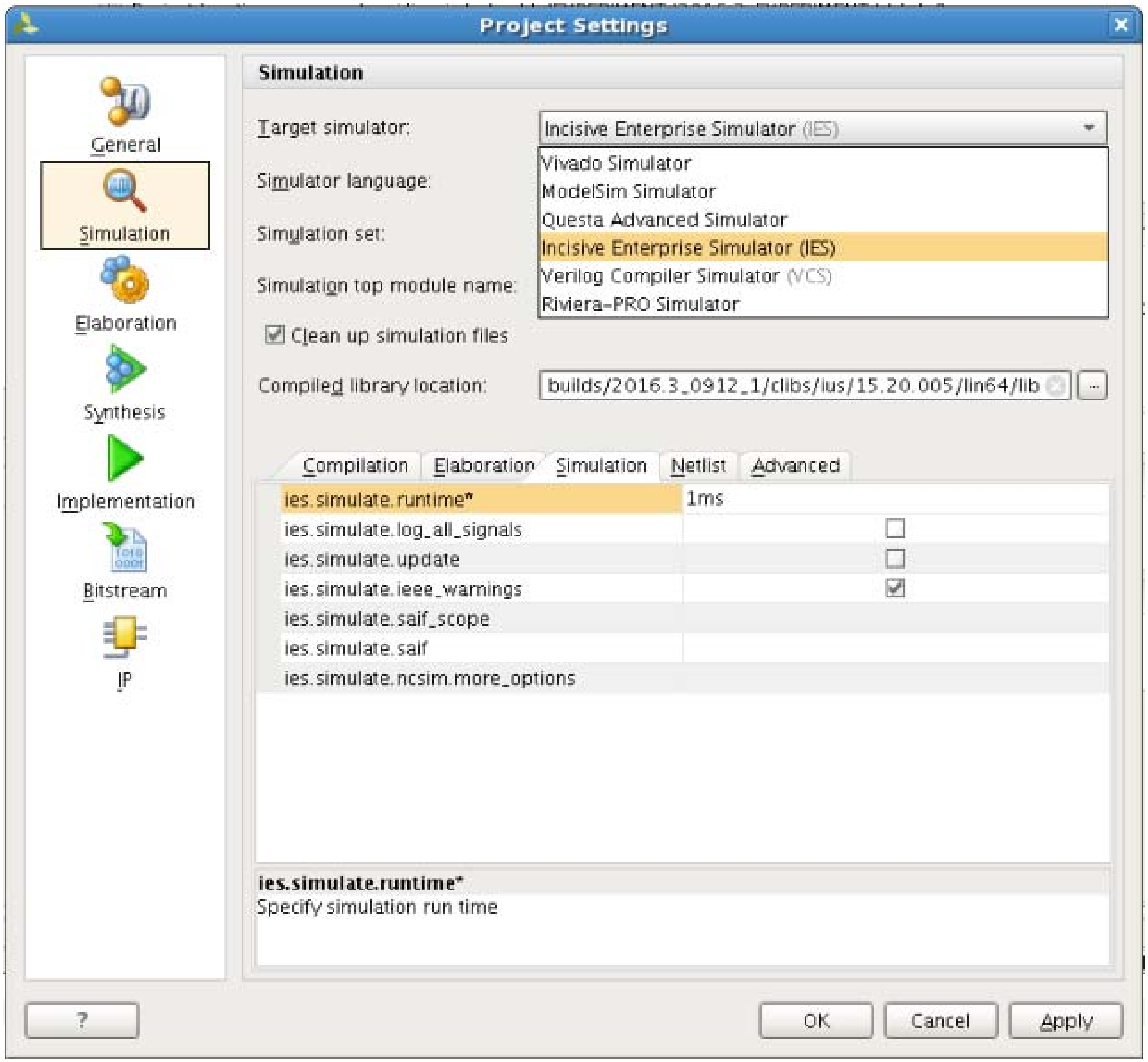
4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 34-5.

Figure 34-5: **Run Behavioral Simulation**

5. Vivado invokes Questa Advanced Simulator and simulations are run in the Questa Advanced Simulator tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using IES

1. Open a RLDRAM 3 example Vivado project (**Open IP Example Design**...), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as Incisive Enterprise Simulator (IES).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `ies.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 34-6. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



Simulation

Target simulator: Incisive Enterprise Simulator (IES)

- Vivado Simulator
- ModelSim Simulator
- Questa Advanced Simulator
- Incisive Enterprise Simulator (IES)
- Verilog Compiler Simulator (VCS)
- Riviera-PRO Simulator

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: builds/2016.3_0912_1/clibs/ius/15.20.005/lin64/lib

ies.simulate.runtime*	1ms
ies.simulate.log_all_signals	☐
ies.simulate.update	☐
ies.simulate.ieee_warnings	☑
ies.simulate.saif_scope	
ies.simulate.saif	
ies.simulate.ncsim.more_options	

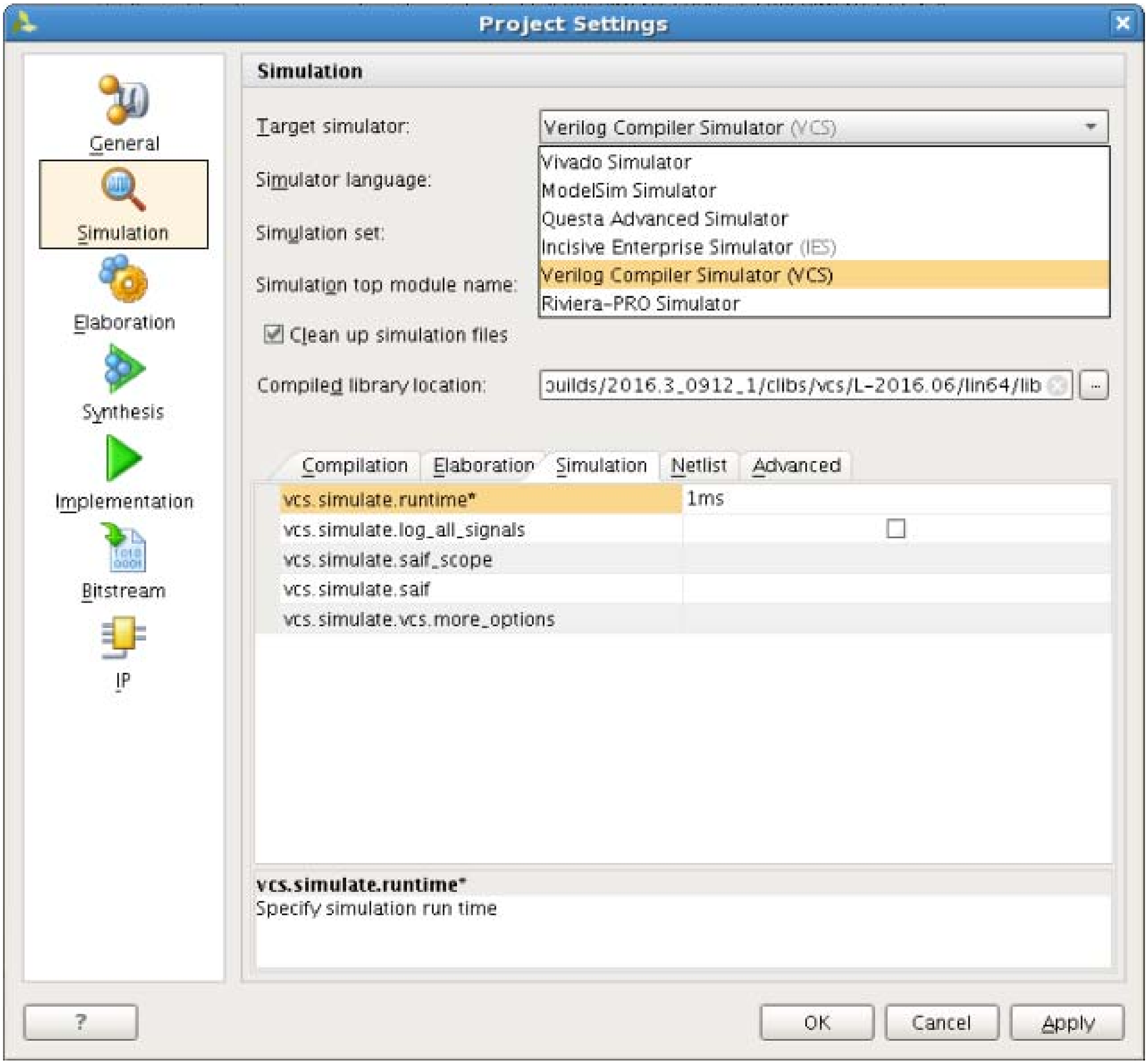
ies.simulate.runtime*
Specify simulation run time

Figure 34-6: **Simulation with IES Simulator**

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 34-5.
5. Vivado invokes IES and simulations are run in the IES tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Project-Based Simulation Flow Using VCS

1. Open a RLDRAM 3 example Vivado project (**Open IP Example Design...**), then under **Flow Navigator**, select **Simulation Settings.**
2. Select **Target simulator** as Verilog Compiler Simulator (VCS).
    a. Browse to the compiled libraries location and set the path on **Compiled libraries location** option.
    b. Under the **Simulation** tab, set the `vcs.simulate.runtime` to 1 ms (there are simulation RTL directives which stop the simulation after certain period of time, which is less than 1 ms) as shown in Figure 34-7. The **Generate Scripts Only** option generates simulation scripts only. To run behavioral simulation, **Generate Scripts Only** option must be de-selected.
3. Apply the settings and select **OK**.



Target simulator: Verilog Compiler Simulator (VCS)

Simulator language:

Simulation set:

Simulation top module name:

☑ Clean up simulation files

Compiled library location: builds/2016.3_0912_1/clibs/vcs/L-2016.06/lin64/lib

vcs.simulate.runtime*	1ms
vcs.simulate.log_all_signals	☐
vcs.simulate.saif_scope	
vcs.simulate.saif	
vcs.simulate.vcs.more_options	

vcs.simulate.runtime*
Specify simulation run time

Figure 34-7: **Simulation with VCS Simulator**

4. In the **Flow Navigator** window, select **Run Simulation** and select **Run Behavioral Simulation** option as shown in Figure 34-5.

5. Vivado invokes VCS and simulations are run in the VCS tool. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

Simulation Speed

RLDRAM 3 provides a Vivado IDE option to reduce the simulation speed by selecting behavioral XIPHY model instead of UNISIM XIPHY model. Behavioral XIPHY model simulation is a default option for RLDRAM 3 designs. To select the simulation mode, click the **Advanced Options** tab and find the **Simulation Options** as shown in Figure 33-3.

The SIM_MODE parameter in the RTL is given a different value based on the Vivado IDE selection.

- **SIM_MODE = BFM** – If fast mode is selected in the Vivado IDE, the RTL parameter reflects this value for the SIM_MODE parameter. This is the default option.
- **SIM_MODE = FULL** – If UNISIM mode is selected in the Vivado IDE, XIPHY UNISIMs are selected and the parameter value in the RTL is FULL.

Using Xilinx IP with Third-Party Synthesis Tools

For more information on how to use Xilinx IP with third-party synthesis tools, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation

If the GCIO pin and MMCM are not allocated in the same bank, the CLOCK_DEDICATED_ROUTE constraint must be set to BACKBONE. To use the BACKBONE route, BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV must be instantiated between GCIO and MMCM input. RLDRAM 3 manages these constraints for designs generated with the **Reference Input Clock** option selected as **Differential** (at **Advanced > FPGA Options > Reference Input**). Also, RLDRAM 3 handles the IP and example design flows for all scenarios.

If the design is generated with the **Reference Input Clock** option selected as **No Buffer** (at **Advanced > FPGA Options > Reference Input**), the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation based on GCIO and MMCM allocation needs to be handled manually for the IP flow. RLDRAM 3 does not generate clock constraints in the XDC file for **No Buffer** configurations and you must take care of the clock constraints for **No Buffer** configurations for the IP flow.

For an example design flow with **No Buffer** configurations, RLDRAM 3 generates the example design with differential buffer instantiation for system clock pins. RLDRAM 3 generates clock constraints in the **`example_design.xdc`**. It also generates a CLOCK_DEDICATED_ROUTE constraint as the “BACKBONE” and instantiates BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV between GCIO and MMCM input if the GCIO and MMCM are not in same bank to provide a complete solution. This is done for the example design flow as a reference when it is generated for the first time.

If in the example design, the I/O pins of the system clock pins are changed to some other pins with the I/O pin planner, the CLOCK_DEDICATED_ROUTE constraints and BUFG/BUFGCE/BUFGCTRL/BUFGCE_DIV instantiation need to be managed manually. A DRC error is reported for the same.

Chapter 35

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

The Memory Controller is generated along with a simple test bench to verify the basic read and write operations. The stimulus contains 100 consecutive writes followed by 100 consecutive reads for data integrity check.

SECTION VII: TRAFFIC GENERATOR

Chapter 36

Traffic Generator

Overview

This section describes the setup and behavior of the Traffic Generator. In the UltraScale™ architecture, Traffic Generator is instantiated in the example design (**`example_top.sv`**) to drive the memory design through the application interface (Figure 36-1).

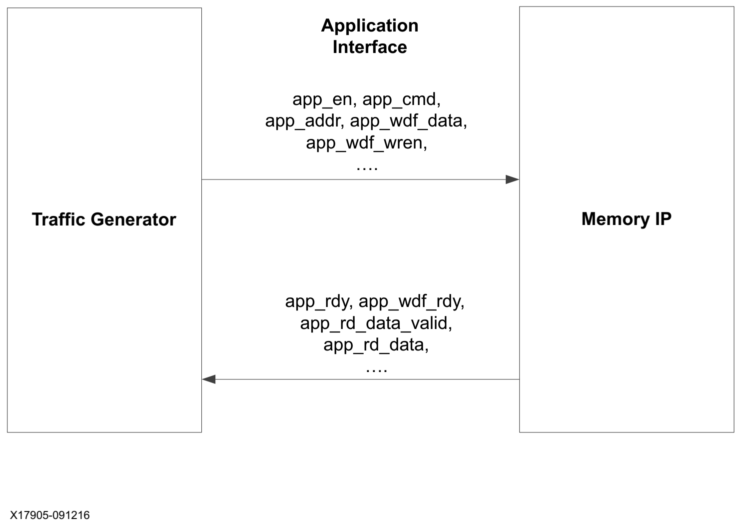


Figure 36-1: **Traffic Generator and Application Interface**

Two Traffic Generators are available to drive the memory design and they include:

- Simple Traffic Generator
- Advanced Traffic Generator

By default, Vivado® connects the memory design to the Simple Traffic Generator. You can choose to use the Advanced Traffic Generator by defining a switch HW_TG_EN in the **`example_top.sv`**. The Simple Traffic Generator is referred to as STG and the Advanced Traffic Generator is referred to as ATG for the remainder of this section.

Simple Traffic Generator

Memory IP generates the STG modules as `example_tb` for native interface and `example_tb_phy` for PHY only interface. The STG native interface generates 100 writes and 100 reads. The STG PHY only interface generates 10 writes and 10 reads. Both address and data increase linearly. Data check is performed during reads. Data error is reported using the `compare_error` signal.

Advanced Traffic Generator

The ATG is only supported for the user interface. When HW_TG_EN is defined, ATG is set to the default setting. The ATG could be programmed differently to test memory interface with different traffic patterns. In the example design created by Vivado, the ATG is set to default setting which is described in the next section. The default setting is recommended for most to get started. For further information on ATG programming, see the Traffic Generator Description section.

After memory initialization and calibration are done, the ATG starts sending write commands and read commands. If the memory read data does not match with the expected read data, the ATG flags compare errors through the status interface. For VIO or ILA debug, you have the option to connect status interface signals.

IMPORTANT: *For DDR3 and DDR4 interfaces, ATG is disabled with the AXI interface.*

Traffic Generator Default Behavior

In the default settings (parameter DEFAULT_MODE = 2016.3), the ATG performs memory writes followed by memory reads and data checks. Three types of patterns are generated sequentially:

1. PRBS23 data pattern
   a. PRBS23 data pattern is used per data bit. Each data bit has a different default starting seed value.
   b. Linear address pattern is used. Memory address space is walked through to cover full PRBS23 data pattern.
2. Hammer Zero pattern
   a. Hammer Zero pattern is used for all data bits.
   b. Linear address pattern is used. 1,024 Traffic Generator commands are issued.

3. PRBS address pattern
    a. PRBS23 data pattern is used per data bit. Each data bit has a different default starting seed value.
    b. PRBS address pattern is used. 1,024 Traffic Generator commands are issued.

The ATG repeats memory writes and reads on each of the two patterns infinitely. For simulations, ATG performs 1,000 PRBS23 pattern followed by 1,000 Hammer Zero pattern and 1,000 PRBS address pattern.

Traffic Generator Description

This section provides detailed information on using the ATG beyond the default settings.

Feature Support

In this section, the ATG basic feature support and mode of operation is described. The ATG allows you to program different traffic patterns, a read-write mode, and the duration of traffic burst based on their application.

Provide one traffic pattern for a simple traffic test in the direct instruction mode or program up to 32 traffic patterns into the traffic pattern table for a regression test in the traffic table mode.

Each traffic pattern can be programmed with the following options:

- **Address Mode** – Linear, PRBS, walking1/0.
- **Data Mode** – Linear, PRBS 8/10/23, walking1/0, and hammer1/0.
- **Read/Write Mode** – Write-read, write-only, read-only, write-once-read-forever.
    - **Read/Write Submode** – When read/write mode is set to write-read, you can choose to send write and read commands. The first choice sends all write commands follow by read commands.

      The second choice sends write and read command pseudo-randomly. This submode is valid for DDR3/DDR4 and RLDRAM 3 only.
- **Victim Mode** – No Victim, held1, held0, Non-inverted aggressor, inverted aggressor, delayed aggressor, delayed victim.
- **Victim Aggressor Delay** – Aggressor or victim delay when the Victim mode of "delayed aggressor" or "delayed victim" modes is used.
- **Victim Select** – Victim selected from the ATG VIO input or victim rotates per nibble/per byte/per interface width.
- **Number of Command Per Traffic Pattern**
- **Number of NOPs After Bursts**

- **Number of Bursts Before NOP**
- **Next Instruction Pointer**

Create one traffic pattern for simple traffic test using the direct instruction mode (**`vio_tg_direct_instr_en`**).

Also, create a sequence of traffic patterns by programming a "next instruction" (**`vio_tg_instr_nxt_instr`**) pointer pointing to one of the 32 traffic pattern entries for regression test in traffic table mode.

The example in Table 36-1 shows four traffic patterns programmed in the table mode.

The first pattern has PRBS data traffic written in Linear address space. The 1,000 write commands are issued followed by 1,000 read commands. Twenty cycles of NOPs are inserted between every 100 cycle of commands. After completion of instruction0, the next instruction points at instruction1.

Similarly, instruction1, instruction2, and instruction3 is executed and then looped back to instruction0.

Table 36-1: **Example of Instruction Program**

Instruction Number	Addr Mode	Data Mode	Read/Write Mode	Victim Mode	Number of Instruction Iteration	Insert M NOPs Between N-Burst (M)	Insert M NOPs Between N-Burst (N)	Next Instruction
0	Linear	PRBS	Write-Read	No Victim	1,000	20	100	1
1	Linear	PRBS	Write-Read	No Victim	1,000	0	500	2
2	Linear	Linear	Write-Only	No Victim	10,000	10	100	3
3	Linear	Walking1	Write-Read	No Victim	1,000	10	100	0
....								
31								

The ATG waits for calibration to complete (**`init_calib_complete`** and **`tg_start`** assertion). After the calibration complete and assertion of **`tg_start`**, the ATG starts sending the default traffic sequence according to traffic pattern table or direct instruction programmed. Memory Read/Write requests are then sent through the application interface, Memory Controller, and PHY. Either program the instruction table before asserting **`tg_start`** or pause the traffic generator (by asserting **`vio_tg_pause`**), reprogram the instruction table, and restart the test traffic for custom traffic pattern. For more information, see the Usage section.

The ATG performs error check when a traffic pattern is programmed to read/write modes that have write requests followed by read request (that is, Write-read-mode or Write-once-Read-forever-mode). The ATG first sends all write requests to the memory. After all write requests are sent, the ATG sends read requests to the same addresses as the write requests. Then the read data returning from memory is compared with the expected read data.

If there is no mismatch error and the ATG is not programmed into an infinite loop, **`vio_tg_status_done`** asserts to indicate run completion.

The ATG has watchdog logic. The watchdog logic checks if the ATG has any request sent to the application interface or the application interface has any read data return within N (parameter TG_WATCH_DOG_MAX_CNT) number of cycles. This provides information on whether memory traffic is running or stalled (because of reasons other than data mismatch).

Usage

In this section, basic usage and programming of the ATG is covered.

The ATG is programmed and controlled using the VIO interface. Table 36-2 shows instruction table programming options.

Table 36-2: **Traffic Generator Instruction Options**

Name	Bit Width	Description
Instruction Number	5	Instruction select. From 0 to 31.
Addr Mode	4	Address mode to be programmed. 0 = LINEAR; (with user-defined start address) 1 = PRBS; (PRBS supported range from 8 to 34 based on address width) 2= WALKING1 3 = WALKING0 4-15 = Reserved ***Note:*** QDR-IV SRAM only supports Linear address with start address equal to 0.
Data Mode	4	Data mode to be programmed. 0 = LINEAR 1 = PRBS (PRBS supported 8, 10, 23) 2 = WALKING1 3 = WALKING0 4 = HAMMER1 5 = HAMMER0 6 = Block RAM 7 = CAL_CPLX 8-15 = Reserved

Table 36-2: **Traffic Generator Instruction Options *(Cont'd)***

Name	Bit Width	Description
Read/Write Mode	4	0 = Read Only (No data check) 1 = Write Only (No data check) 2 = Write/Read (Read performs after Write and data value is checked against expected write data. For QDR II+ SRAM, one port is used for write and another port is used for read.) 3 = Write once and Read forever (Data check on Read data) 4-15 = Reserved

Read/Write Submode	2	Read/Write submode to be programmed.
		This is a submode option when vio_tg_instr_rw_mode is set to "WRITE_READ" mode.
		This mode is only valid for DDR3/DDR4 and RLDRAM 3. For QDR II+ and QDR-IV SRAM interfaces, this mode should be set to 0.
		WRITE_READ = 00; // Send all Write commands follow by Read commands defined in the instruction.
		WRITE_READ_SIMULTANEOUSLY = 01; // Send Write and Read commands pseudo-randomly. Note that Write is always ahead of Read.
		2 and 3 = Reserved

Victim Mode	3	Victim mode to be programmed.
		One victim bit could be programmed using global register vio_tg_victim_bit. The rest of the bits on signal bus are considered to be aggressors.
		The following program options define aggressor behavior:
		NO_VICTIM = 0;
		HELD1 = 1; // All aggressor signals held at 1
		HELD0 = 2; // All aggressor signals held at 0
		NONINV_AGGR = 3; // All aggressor signals are same as victim
		INV_AGGR = 4; // All aggressor signals are inversion of victim
		DELAYED_AGGR = 5; // All aggressor signals are delayed version of victim (num of cycle of delay is programmed at vio_tg_victim_aggr_delay)
		DELAYED_VICTIM = 6; // Victim signal is delayed version of all aggressors
		CAL_CPLX = 7; Complex Calibration pattern

Victim Aggressor Delay	5	Define aggressor/victim pattern to be N-delay cycle of victim/aggressor, where 0 ≤ N ≤ 24. It is used when victim mode "DELAY_AGGR" or "DELAY VICTIM" mode is used in traffic pattern.

Victim Select	3	Victim bit behavior programmed.
		VICTIM_EXTERNAL = 0; // Use Victim bit provided in vio_tg_glb_victim_bit
		VICTIM_ROTATE4 = 1; // Victim bit rotates from Bits[3:0] for every Nibble
		VICTIM_ROTATE8 = 2; // Victim bit rotates from Bits[7:0] for every Byte
		VICTIM_ROTATE_ALL = 3; // Victim bit rotates through all bits
		RESERVED = 4-7

Table 36-2: **Traffic Generator Instruction Options *(Cont'd)***

Name	Bit Width	Description
Number of instruction iteration	32	Number of Read and/or Write commands to be sent. N = APP_ADDR_WIDTH – 3 ***Note:*** Note: APP_ADDR_WIDTH is defined in example_top.sv. Linear Address Space Calculation: Max No. of iterations = $2^{(N)}$ PRBS Address Space Calculation Max No. of iterations = $(2^{(N)}) - 1$ Walking1/0 Address Space Calculation: Max No. of iterations = N
Insert M NOPs between N-burst (M)	10	M = Number of NOP cycles in between Read/Write commands at user interface at general interconnect clock, where M ≥ 1.
Insert M NOPs between N-burst (N)	32	N = Number of Read/Write commands before NOP cycle insertion at user interface at general interconnect clock, where N ≥ 1.
Next Instruction	6	Next instruction to run. To end traffic, next instruction should point at EXIT instruction. 6'b000000-6'b011111 – valid instruction 6'b1????? – EXIT instruction

Note: Application interface signals are not shown in section. See the corresponding memory section for the application interface data format.

Using VIO to Control ATG

VIO is instantiated for the DDR3/DDR4 example design to exercise the Traffic Generator modes when the design is generated with the ATG option.

The expected write data and the data that is read back are added to the ILA instance. Write and read data can be viewed in ILA for one byte only. Data of various bytes can be viewed by driving the appropriate value for **`vio_rbyte_sel`** which is driven through VIO. **`vio_rbyte_sel`** is a 4-bit signal and you need to pass the value through VIO for a required byte. Based on the value driven for **`vio_rbyte_sel`** through VIO, a corresponding DQ byte write and read data are listed in ILA.

The VIO to drive ATG modes is disabled in the default example design. To enable VIO which drives ATG modes for DDR3/DDR4 interfaces, define the macro as **`VIO_ATG_EN`** in the **`example_top`** module as follows:

```
'define VIO_ATG_EN
```

You have to manually instantiate VIO for other interfaces to exercise the Traffic Generator modes.

The ATG default control connectivity in the example design created by Vivado is listed in Table 36-3.

Note: Application interface signals are not shown in this table. See the corresponding memory section for application interface address/data width.

Table 36-3: **Default Traffic Generator Control Connection**

Signal	I/O	Width	Description	Default Value
clk	I	1	Traffic Generator Clock	Traffic Generator Clock
rst	I	1	Traffic Generator Reset	Traffic Generator Reset
init_calib_complete	I	1	Calibration Complete	Calibration Complete
General Control				
vio_tg_start	I	1	Enable traffic generator to proceed from "START" state to "LOAD" state after calibration completes. If you do not plan to program instruction table or PRBS data seed, tie this signal to 1'b1. If you plan to program instruction table or PRBS data seed, set this bit to 0 during reset. After reset deassertion and done with instruction/ seed programming, set this bit to 1 to start traffic generator.	Reserved signal. Tie to 1'b1.
vio_tg_rst	I	1	Reset traffic generator (synchronous reset, level sensitive). If there is outstanding traffic in memory pipeline, assert signal by some number of clock cycles until all outstanding transactions have completed.	Reserved signal. Tie to 0.
vio_tg_restart	I	1	Restart traffic generator after generator is done with traffic, paused or stopped with error (level sensitive). If there is outstanding traffic in memory pipeline, assert signal by some number of clock cycles until all outstanding transactions have completed.	Reserved signal. Tie to 0.
vio_tg_pause	I	1	Pause traffic generator (level sensitive).	Reserved signal. Tie to 0.
vio_tg_err_chk_en	I	1	If enabled, stop after first error detected. Read test is performed to determine whether "READ" or "WRITE" error occurred. If not enabled, continue traffic without stop.	Reserved signal. Tie to 0.

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
vio_tg_err_clear	I	1	Clear all error excluding sticky error bit (positive edge sensitive). Only use this signal when vio_tg_status_state is either TG_INSTR_ERRDONE or TG_INSTR_PAUSE. Error is cleared two cycles after vio_tg_err_clear is asserted.	Reserved signal. Tie to 0.
vio_tg_err_clear_all	I	1	Clear all error including sticky error bit (positive edge sensitive). Only use this signal when vio_tg_status_state is either TG_INSTR_ERRDONE or TG_INSTR_PAUSE. Error is cleared two cycles after vio_tg_err_clear_all is asserted.	Reserved signal. Tie to 0.
vio_tg_err_continue	I	1	Continue traffic after error(s) at TG_INSTR_ERRDONE state (positive edge sensitive).	Reserved signal. Tie to 0.
Instruction Table Programming				
vio_tg_direct_instr_en	I	1	0 = Traffic Table Mode – Traffic Generator uses traffic patterns programmed in 32-entry Traffic table which is found in ddr4_v2_2_tg_instr_bram.sv 1 = Direct Instruction Mode – Traffic Generator uses current traffic pattern presented at VIO interface	Reserved signal. Tie to 0.
vio_tg_instr_program_en	I	1	Enable instruction table programming (level sensitive).	Reserved signal. Tie to 0.
vio_tg_instr_num	I	5	Instruction number to be programmed.	Reserved signal. Tie to 0.
vio_tg_instr_addr_mode	I	4	Address mode to be programmed. 0 = LINEAR; (with user-defined start address) 1 = PRBS; (PRBS supported range from 8 to 34 based on address width) 2= WALKING1 3 = WALKING0 4-15 = Reserved ***Note:*** QDR-IV SRAM only supports Linear address with start address equal to 0.	Reserved signal. Tie to 0.

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
vio_tg_instr_data_mode	I	4	Data mode to be programmed. 0 = LINEAR 1 = PRBS (PRBS supported 8, 10, 23) 2 = WALKING1 3 = WALKING0 4 = HAMMER1 5 = HAMMER0 6 = Block RAM 7 = CAL_CPLX (Must be programmed along with victim mode CAL_CPLX) 8-15 = Reserved	Reserved signal. Tie to 0.
vio_tg_instr_rw_mode	I	4	0 = Read Only (No data check) 1 = Write Only (No data check) 2 = Write/Read (Read performs after Write and data value is checked against expected write data. For QDR II+ SRAM, one port is used for write and another port is used for read.) 3 = Write once and Read forever (Data check on Read data) 4-15 = Reserved	Reserved signal. Tie to 0.

vio_tg_instr_rw_submode	I	2	Read/Write submode to be programmed.
			This is a submode option when vio_tg_instr_rw_mode is set to "WRITE_READ" mode.
			This mode is only valid for DDR3/DDR4 and RLDRAM 3. For QDR II+ and QDR-IV SRAM interfaces, this mode should be set to 0.
			WRITE_READ = 0; // Send all Write commands follow by Read commands defined in the instruction.
			WRITE_READ_SIMULTANEOUSLY = 1; // Send Write and Read commands pseudo-randomly. Note that Write is always ahead of Read.
				Reserved signal. Tie to 0.

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value

vio_tg_instr_victim_mode	I	3	Victim mode to be programmed.
			One victim bit could be programmed using global register vio_tg_victim_bit. The rest of the bits on signal bus are considered to be aggressors.
			The following program options define aggressor behavior:
			NO_VICTIM = 0;
			HELD1 = 1; // All aggressor signals held at 1
			HELD0 = 2; // All aggressor signals held at 0
			NONINV_AGGR = 3; // All aggressor signals are same as victim
			INV_AGGR = 4; // All aggressor signals are inversion of victim
			DELAYED_AGGR = 5; // All aggressor signals are delayed version of victim (num of cycle of delay is programmed at vio_tg_victim_aggr_delay)
			DELAYED_VICTIM = 6; // Victim signal is delayed version of all aggressors
			CAL_CPLX = 7; Complex Calibration pattern (Must be programed along with Data Mode CAL_CPLX)
				Reserved signal. Tie to 0.

vio_tg_instr_victim_aggr_delay	I	5	Define aggressor/victim pattern to be N-delay cycle of victim/aggressor, where 0 ≤ N ≤ 24. It is used when victim mode "DELAY_AGGR" or "DELAY VICTIM" mode is used in traffic pattern.	Reserved signal. Tie to 0.

vio_tg_instr_victim_select	I	3	Victim bit behavior programmed.
			VICTIM_EXTERNAL = 0; // Use Victim bit provided in vio_tg_glb_victim_bit
			VICTIM_ROTATE4 = 1; // Victim bit rotates from Bits[3:0] for every Nibble
			VICTIM_ROTATE8 = 2; // Victim bit rotates from Bits[7:0] for every Byte
			VICTIM_ROTATE_ALL = 3; // Victim bit rotates through all bits
				Reserved signal. Tie to 0.

vio_tg_instr_num_of_iter	I	32	Number of Read/Write commands to issue (number of issue must be > 0 for each instruction programmed).	Reserved signal. Tie to 0.
vio_tg_instr_m_nops_btw_n_burst_m	I	10	M = Number of NOP cycles in between Read/Write commands at user interface at general interconnect clock N = Number of Read/Write commands before NOP cycle insertion at user interface at general interconnect clock	Reserved signal. Tie to 0.

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
vio_tg_instr_m_nops_btw_n_burst_n	I	32	M = Number of NOP cycles in between Read/Write commands at user interface at general interconnect clock N = Number of Read/Write commands before NOP cycle insertion at user interface at general interconnect clock	Reserved signal. Tie to 0.
vio_tg_instr_nxt_instr	I	6	Next instruction to run. To end traffic, next instruction should point at EXIT instruction. 6'b000000-6'b011111 – valid instruction 6'b1????? – EXIT instruction	Reserved signal. Tie to 0.
PRBS Data Seed Programming				
vio_tg_seed_program_en	I	1	0 = Traffic Table Mode – Traffic Generator uses traffic patterns programmed in 32-entry Traffic table 1 = Direct Instruction Mode – Traffic Generator uses current traffic pattern presented at VIO interface	Reserved signal. Tie to 0.
vio_tg_seed_num	I	8	Seed number to be programmed.	Reserved signal. Tie to 0.
vio_tg_seed_data	I	PRBS DATA WIDTH	PRBS seed to be programmed for a selected seed number (vio_tg_seed_num). PRBS_DATA_WIDTH is by default 23. PRBS_DATA_WIDTH can support 8, 10, and 23.	Reserved signal. Tie to 0.
Global Registers				
vio_tg_glb_victim_bit	I	8	Global register to define which bit in data bus is victim. It is used when victim mode is used in traffic pattern.	Reserved signal. Tie to 0.
vio_tg_glb_start_addr	I	APP_ADDR_WIDTH	Global register to define Start address seed for Linear Address Mode.	Reserved signal. Tie to 0.
vio_tg_glb_qdriv_rw_submode	I	2	Use for QDR-IV to control different traffic setup when Write-Read mode is selected. 2'b00 = Both Port A and Port B send Write traffic, follow by Read traffic 2'b01 = Port A sends Write traffic, while Port B sends Read traffic simultaneously 2'b10 = Port B sends Write traffic, while Port A sends Read traffic simultaneously 2'b11 = Both Port A and Port B send a mix of Write and Read traffic. Only Linear address mode is supported.	

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
Traffic Generator Internal Signals				
tg_qdriv_submode11_app_rd	I	1	TG internal signal. Connect the signal to default programming value. app_rd_data_valid derivative for QDR-IV submode11 use.	Reserved signal. Tie to 0.
Error Status Registers				
vio_tg_status_state	O	4	Traffic Generator state machine state.	
vio_tg_status_err_bit_valid	O	1	Error detected. It is used as trigger to detect read error.	
vio_tg_status_err_bit	O	APP_DATA_WIDTH	Error bit mismatch. Bitwise mismatch pattern. A 1 indicates error detected in that bit location.	
vio_tg_status_err_cnt	O	32	Saturated counter that counts the number of assertion of the signal vio_tg_status_err_bit_valid. The counter is reset by vio_tg_err_clear and vio_tg_err_clear_all.	
vio_tg_status_err_addr	O	APP_ADDR_WIDTH	Error Address Address location of failed read.	
vio_tg_status_exp_bit_valid	O	1	Expected read data valid.	
vio_tg_status_exp_bit	O	APP_DATA_WIDTH	Expected read data.	
vio_tg_status_read_bit_valid	O	1	Memory read data valid.	
vio_tg_status_read_bit	O	APP_DATA_WIDTH	Memory read data.	
vio_tg_status_first_err_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, vio_tg_status_first_err_bit_valid is set to 1 when first mismatch error is encountered. This register is not overwritten until vio_tg_err_clear, vio_tg_err_continue, and vio_tg_restart is triggered.	
vio_tg_status_first_err_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_err_bit_valid is set to 1, only the first error mismatch bit pattern is stored in this register.	
vio_tg_status_first_err_addr	O	APP_ADDR_WIDTH	If vio_tg_status_first_err_bit_valid is set to 1, only the first error address is stored in this register.	
vio_tg_status_first_exp_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, this represents expected read data valid when first mismatch error is encountered.	
vio_tg_status_first_exp_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_exp_bit_valid is set to 1, expected read data for the first error is stored in this register.	
vio_tg_status_first_read_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, this represents read data valid when first mismatch error is encountered.	

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
vio_tg_status_first_read_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_read_bit_valid is set to 1, read data from memory for the first error is stored in this register.	
vio_tg_status_err_bit_sticky_valid	O	1	Accumulated error mismatch valid over time. This register will be reset by vio_tg_err_clear, vio_tg_err_continue, vio_tg_restart.	
vio_tg_status_err_bit_sticky	O	APP_DATA_WIDTH	If vio_tg_status_err_bit_sticky_valid is set to 1, this represents accumulated error bit.	
vio_tg_status_err_cnt_sticky	O	32	Saturated counter that counts the number of assertion of the signal vio_tg_status_err_bit_sticky_valid. The counter is reset by vio_tg_err_clear_all.	
vio_tg_status_err_type_valid	O	1	If vio_tg_err_chk_en is set to 1, read test is performed after the first mismatch error. Read test returns error type of either "READ" or "WRITE" error. This register stores valid status of read test error type.	
vio_tg_status_err_type	O	1	If vio_tg_status_err_type_valid is set to 1, this represents error type result from read test. 0 = Write Error, 1 = Read Error	
vio_tg_status_done	O	1	All traffic programmed completed. ***Note:*** If infinite loop is programmed, vio_tg_status_done does not assert.	
vio_tg_status_wr_done	O	1	This signal pulses after a WRITE-READ mode instruction completes.	
vio_tg_status_watch_dog_hang	O	1	Watchdog hang. This register is set to 1 if there is no Read/Write command sent or no Read data return for a period of time (defined in tg_param.vh).	
compare_error	O	1	Accumulated error mismatch valid over time. This register is reset by vio_tg_err_clear, vio_tg_err_continue, and vio_tg_restart.	
tg_rd_err_addr_x1	O	50	Registered version of vio_tg_status_err_addr. Error Address location of failed read.	
tg_exp_data_valid_x1	O	1	Registered version of vio_tg_status_exp_bit_valid. Expected read data valid of failed read.	
tg_exp_data_x1	O	64	Registered version of vio_tg_status_exp_bit. Expected read data of selected data byte lane (vio_rbyte_sel) of failed read.	
tg_rd_data_valid_x1	O	1	Registered version of vio_tg_status_read_bit_valid. Memory read data valid.	

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
tg_rd_data_x1	O	64	Memory read data of selected data byte lane (vio_rbyte_sel) for the corresponding address.	
first_err_bit_valid_x1	O	1	Registered version of vio_tg_status_first_err_bit_valid. If vio_tg_err_chk_en is set to 1, first_err_bit_valid_x1 is set to 1 when first mismatch error is encountered.	
first_err_bit_x1	O	64	If vio_tg_status_first_err_bit_valid is set to 1, error mismatch bit pattern is stored in this register for the selected data byte lane (vio_rbyte_sel).	
acc_bit_err_valid_x1	O	1	Registered version of vio_tg_status_err_bit_sticky_valid. Accumulated error mismatch valid over time.	
acc_bit_err_x1	O	64	If vio_tg_status_err_bit_sticky_valid is set to 1, this represents accumulated error bits for the selected data byte lane using vio_rbyte_sel.	
err_type_valid_x1	O	1	Registered version of vio_tg_status_err_type_valid. This register stores valid status of read test error type.	
err_type_x1	O	1	Registered version of vio_tg_status_err_type_valid. If err_type_x1 is set to 1, this represents error type result from read test of the 0 = Write Error, 1 = Read Error.	
acc_bit_err_x1	O	APP_DATA_WIDTH	If vio_tg_status_err_bit_sticky_valid is set to 1, this represents accumulated error bit for the selected read byte lane (vio_rbyte_sel).	
acc_bit_err_valid_x1	O	1	Registered version of vio_tg_status_err_bit_sticky_valid. Accumulated error mismatch valid over time. This register resets by vio_tg_err_clear, vio_tg_err_continue, and vio_tg_restart.	
acc_byte_err_x1	O	10	If vio_tg_status_err_bit_sticky_valid is set to 1, each bit represents accumulated byte wise error.	
acc_dq_err_x2	O	8	This register indicated the accumulated data byte error.	
tg_rd_valid_cnt	O	32	Register which keeps a count of valid read data from Memory.	
tg_inst_cnt	O	32	Register which keeps a count of completed WR-RD mode instructions.	

Table 36-3: **Default Traffic Generator Control Connection *(Cont'd)***

Signal	I/O	Width	Description	Default Value
tg_cmp_err_x1_cnt	O	32	Delayed registered version of vio_tg_status_err_cnt. Saturated counter that counts the number of assertion of the signal vio_tg_status_err_bit_valid. The counter is reset by vio_tg_err_clear and vio_tg_err_clear_all.	
vio_rbyte_sel	O	4	Based on the value driven for vio_rbyte_sel through VIO, a corresponding DQ byte write and read data are listed in ILA.	

Traffic Error Detection

The ATG includes multiple data error reporting features. When using the Traffic Generator Default Behavior, check if there is a memory error in the Status register (**`vio_tg_status_err_sticky_valid`**) or if memory traffic stops (**`vio_tg_status_watch_dog_hang`**).

After the first memory error is seen, the ATG logs the error address (**`vio_tg_status_first_err_addr`**) and bit mismatch (**`vio_tg_status_first_err_bit`**).

Table 36-4 shows the common Traffic Generator Status register output which can be used for debug.

Table 36-4: **Common Traffic Generator Status Register for Debug**

Signal	I/O	Width	Description
vio_tg_status_err_bit_valid	O	1	Intermediate error detected. It is used as trigger to detect read error.
vio_tg_status_err_bit	O	APP_DATA_WIDTH	Intermediate error bit mismatch. Bitwise mismatch pattern.
vio_tg_status_err_addr	O	APP_ADDR_WIDTH	Intermediate error address. Address location of failed read.
vio_tg_status_first_err_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, first_err_bit_valid is set to 1 when first mismatch error is encountered. This register is not overwritten until vio_tg_err_clear, vio_tg_err_continue, and vio_tg_restart is triggered.
vio_tg_status_first_err_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_err_bit_valid is set to 1, error mismatch bit pattern is stored in this register.
vio_tg_status_first_err_addr	O	APP_ADDR_WIDTH	If vio_tg_status_first_err_bit_valid is set to 1, error address is stored in this register.
vio_tg_status_first_exp_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, this represents expected read data valid when first mismatch error is encountered.

Table 36-4: **Common Traffic Generator Status Register for Debug *(Cont'd)***

Signal	I/O	Width	Description
vio_tg_status_first_exp_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_exp_bit_valid is set to 1, expected read data is stored in this register.
vio_tg_status_first_read_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, this represents read data valid when first mismatch error is encountered.
vio_tg_status_first_read_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_read_bit_valid is set to 1, read data from memory is stored in this register.
vio_tg_status_err_bit_sticky_valid	O	1	Accumulated error mismatch valid over time. This register is reset by vio_tg_err_clear, vio_tg_err_continue, and vio_tg_restart.
vio_tg_status_err_bit_sticky	O	APP_DATA_WIDTH	If vio_tg_status_err_bit_sticky_valid is set to 1, this represents accumulated error bit.
vio_tg_status_done	O	1	All traffic programmed completes. ***Note:*** If infinite loop is programmed, vio_tg_status_done does not assert.
vio_tg_status_wr_done	O	1	This signal pulses after a Write-Read mode instruction completes.
vio_tg_status_watch_dog_hang	O	1	Watchdog hang. This register is set to 1 if there is no Read/Write command sent or no Read data return for a period of time (defined in tg_param.vh).

The VIO signal **`vio_tg_err_chk_en`** is used to enable error checking and can report read versus write data errors on **`vio_tg_status_err_type`** when **`vio_tg_status_err_type_valid`** is High. When using **`vio_tg_err_chk_en`**, the ATG can be programmed to have two different behaviors when traffic error is detected.

1. Stop traffic after first error is seen.

   The ATG stops traffic after first error. The ATG then performs a read-check to detect if the mismatch seen is a "WRITE" error or "READ" error. When **`vio_tg_status_state`** reaches ERRDone state, the read-check is completed. The **`vio_tg_restart`** can be pulsed to clear and restart ATG or the **`vio_tg_err_continue`** can be pulsed to continue traffic.

2. Continue traffic with error.

   The ATG continues sending traffic. The traffic can be restarted by asserting pause (**`vio_tg_pause`**), followed by pulse restart (**`vio_tg_restart`**), then deasserting pause.

In both cases, bitwise sticky bit mismatch is available in VIO for accumulated mismatch.

When a mismatch error is encountered, use the `vio_tg_status_err_bit_valid` to trigger the Vivado Logic Analyzer. All error status are presented in the `vio_tg_status_*` registers.

Depending on the goal, different `vio_tg_status_*` signals can be connected to ILA or VIO for observation. For example, if regression is run on a stable design, `compare_error` and `vio_tg_status_watch_dog_hang` can be used to detect error or hang conditions.

For a design debug, `vio_tg_status_err*` signals track errors seen on current read data return. `vio_tg_status_first*` signals store the first error seen. `vio_tg_status_err_bit_sticky*` signals accumulate all error bits seen.

Error bit buses could be very wide. It is recommended to add a MUX stage and a flop stage before connect the bus to ILA or VIO.

Error status can be cleared when the ATG is in either ERRDone or Pause states. Send a pulse to the `vio_tg_clear` to clear all error status except sticky bit. Send a pulse to the `vio_tg_clear_all` to clear all error status including sticky bit.

If `vio_tg_err_chk_en` is enabled, the ATG performs an error check to categorize whether a “READ” or “WRITE” error is encountered. The ATG categorizes error type using the following mechanism. When an error is first seen, the error is logged in the `vio_tg_status_first*` status registers. The error address would be read by the ATG for 1,024 times. If all the reads return data differently from the `vio_tg_status_first_exp_bit` register and all the reads return the same data, the error is categorized as “WRITE” error. Otherwise, the error is categorized as “READ” error.

For additional information on how to debug data errors using the ATG, see Debugging Data Errors in Chapter 38, Debugging.

How to Program Traffic Generator Instruction

After calibration is completed, the ATG starts sending current traffic pattern presented at the VIO interface if direct instruction mode is on; or default traffic sequence according to the traffic pattern table if the direct instruction mode is OFF.

If it is desired to run a custom traffic pattern, either program the instruction table before the ATG starts or pause the ATG. Program the instruction table and restart the test traffic through the VIO.

Steps to program the instruction table (wait for at least one general interconnect cycle between each step) are listed here.

Programming instruction table after reset:

1. Set the `vio_tg_start` to 0 to stop the ATG before reset deassertion.
2. Check if the `vio_tg_status_state` is TG_INSTR_START (hex0). Then go to step 4.

Programming instruction table after traffic started:

1. Set the **`vio_tg_start`** to 0 and set **`vio_tg_pause`** to 1 to pause the ATG.
2. Check and wait until the **`vio_tg_status_state`** is TG_INSTR_DONE (hexC), TG_INSTR_PAUSE (hex8), or TG_INSTR_ERRDONE (hex7).
3. Send a pulse to the **`vio_tg_restart`**. Then, go to step 4.

Common steps:

4. Set the **`vio_tg_instr_num_instr`** to the instruction number to be programmed.
5. Set all of the **`vio_tg_instr_*`** registers (instruction register) with desired traffic pattern.
6. Wait for four general interconnect cycles (optional for relaxing VIO write timing).
7. Set the **`vio_tg_instr_program_en`** to 1. This enables instruction table programming.
8. Wait for four general interconnect cycles (optional for relaxing VIO write timing).
9. Set the **`vio_tg_instr_program_en`** to 0. This disables instruction table programming.
10. Wait for four general interconnect cycles (optional for relaxing VIO write timing).
11. Repeat steps 3 to 9 if more than one instruction is programmed.
12. Optionally set the **`vio_tg_glb*`** registers (global register) if related features are programmed.
13. Optionally set the **`vio_tg_err_chk_en`** if you want the ATG to stop and perform read test in case of mismatch error.
14. Set the **`vio_tg_pause`** to 0 and set **`vio_tg_start`** to 1. This starts the ATG with new the programming.

In Figure 36-2, after **`c0_init_calib_complete`** signal is set, the ATG starts executing default instructions preloaded in the instruction table. Then, the **`vio_tg_pause`** is set to pause the ATG, and then pulse **`vio_tg_restart`**. Three ATG instructions are being re-programmed and the ATG is started again by deasserting **`vio_tg_pause`** and asserting **`tg_start`**.

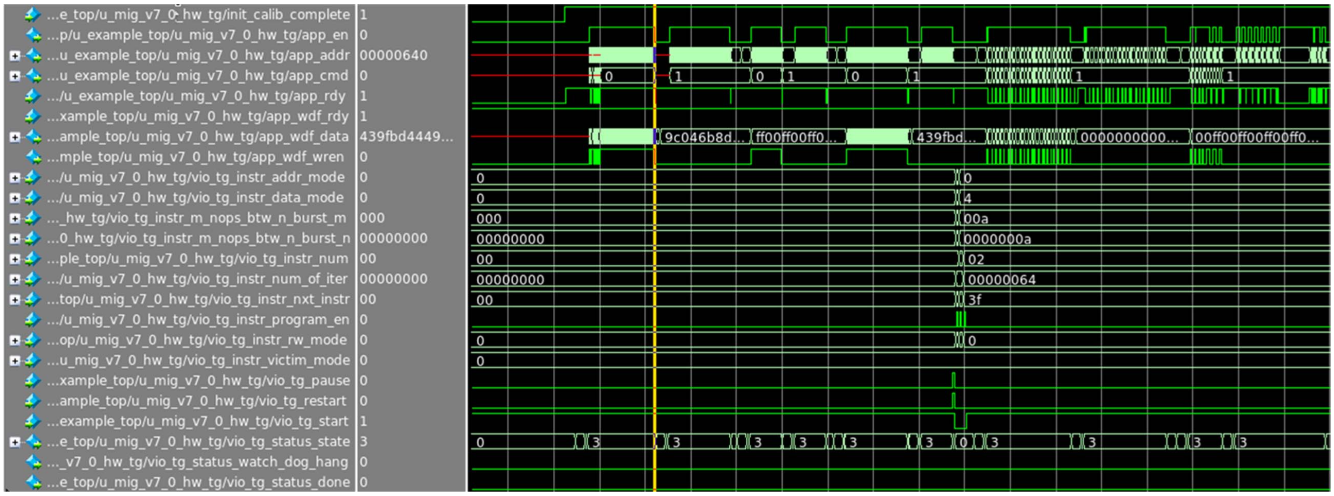


Figure 36-2: **Basic ATG Simulation**

Figure 36-3 zooms into the VIO instruction programming in Figure 36-2. After pausing the traffic pattern, **`vio_tg_restart`** is pulsed. Then **`vio_tg_instr_num`** and **`vio_tg_instr*`** are set, followed by **`vio_tg_program_en`** pulse (note that **`vio_tg_instr_num`** and **`vio_tg_instr`*** are stable for four general interconnect cycles before and after **`vio_tg_program_en`** pulse). After programming instructions are finished, the **`vio_tg_pause`** is deasserted and **`vio_tg_start`** is asserted.

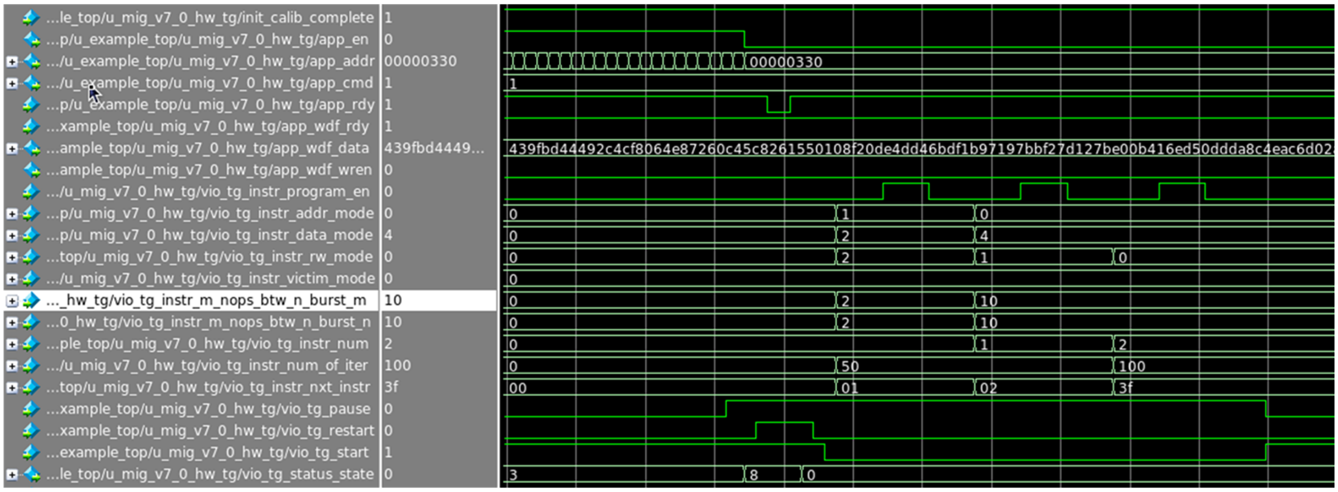


Figure 36-3: **ATG Re-Program Simulation**

Important Note:

1. For Write-read mode or Write-once-Read-forever modes, this ATG issues all write traffic, followed by all read traffic. During read data check, expected read traffic is generated on-the-fly and compared with read data.

If a memory address is written more than once with different data pattern, the ATG creates a false error check. Xilinx recommends for a given traffic pattern programmed, the number of command must be less than available address space programmed.

2. The ATG performs error check when read/write mode of a traffic pattern is programmed to be "Write-read mode" or "Write-once-Read-forever modes." For "Write-only" or "Read-only" modes error check is not performed.

To overwrite default ATG instruction table, update the **`mig_v1_2_tg_instr_bram.sv`**.

Traffic Generator Structure

In this section, the ATG logical structure and data flow is discussed.

The ATG data flow is summarized in Figure 36-4. The ATG is controlled and programmed through the VIO interface. Based on current instruction pointer value, an instruction is issued by the ATG state machine shown in Figure 36-5.

Based on the traffic pattern programmed in Read/Write mode, Read and/Write requests are sent to the application interface. Write patterns are generated by the Write Data Generation, Write Victim Pattern, and Write Address Generation engines (gray). Similarly, Read patterns are generated by Read Address Generation engine (dark gray).

When Write-Read-mode or Write-Once-Read-forever mode are programmed, Read data check is performed. Read data is compared against Expected Read pattern generated by the Expected Data Generation, Expected Victim Pattern, and Expected Address Generation engines (gray and white). Data compare is done in the Error Checker block. Error status is presented to the VIO interface.

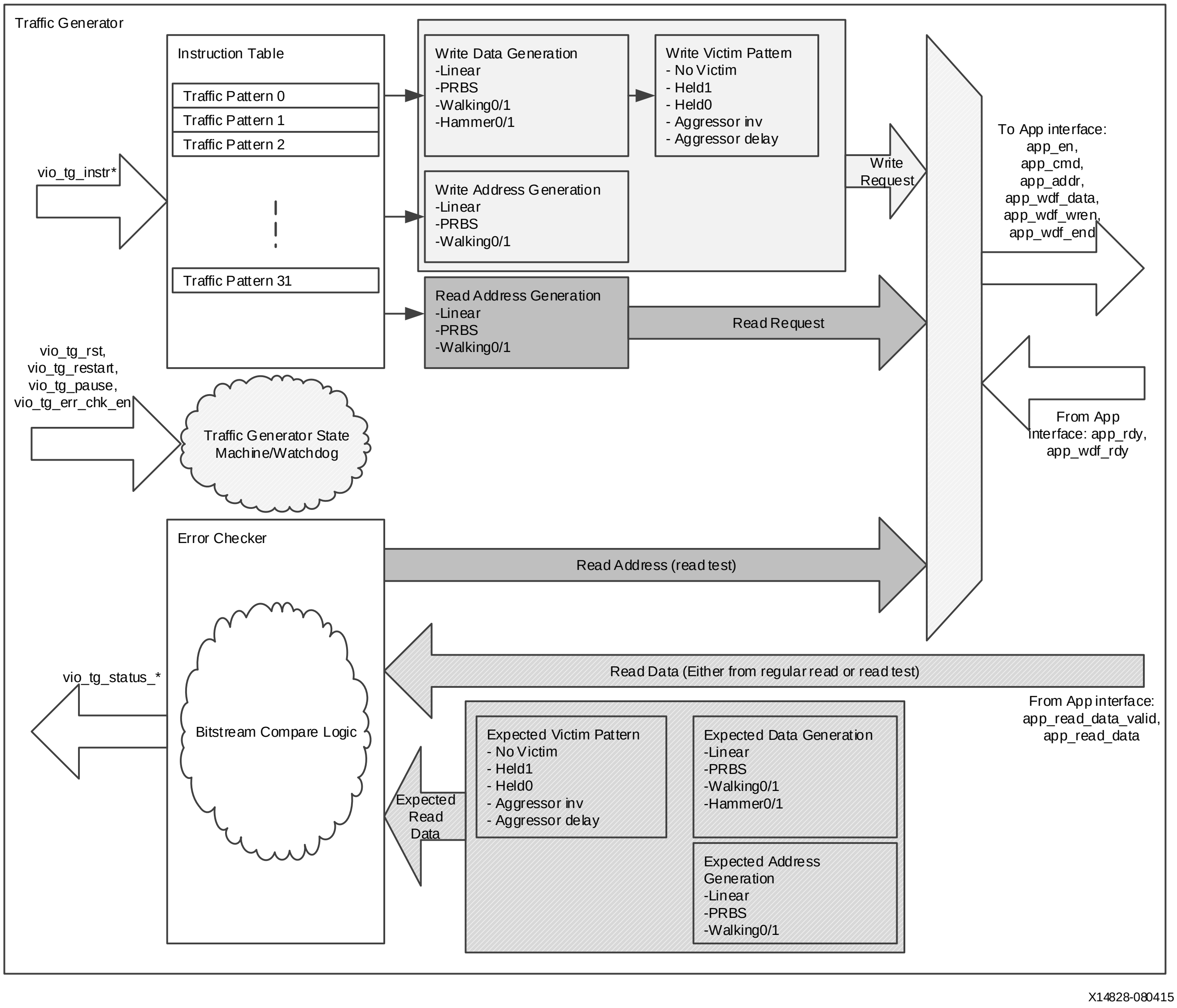


Figure 36-4: **Traffic Generator Data Flow**

Figure 36-5 and Table 36-5 show the ATG state machine and its states. The ATG resets at the "Start" state. After calibration completion (`init_calib_complete`) and the `tg_start` is asserted, the ATG state moves to instruction load called the "Load" state. The "Load" state performs next instruction load. When the instruction load is completed, the ATG state moves to Data initialization called the "Dinit" state. The "Dinit" state initializes all Data/ Address generation engines. After completion of data initialization, the ATG state moves to execution called the "exe" state. The "Exe" state issues Read and/or Write requests to the APP interface.

At the "Exe" state, you can pause the ATG and the ATG state moves to the "Pause" state. At the "Pause" state, the ATG can be restarted by issuing `tg_restart` through the VIO, or un-pause the ATG back to the "Exe" state.

At the "Exe" state, the ATG state goes through RWWait → RWload → Dinit states if Write-Read mode or Write-once-Read-forever modes are used. At the RWWait, the ATG waits for all Read requests to have data returned (for QDR II+ SRAM).

At the RWload state, the ATG transitions from Write mode to Read mode for DDR3/DDR4, RLDRAM II/RLDRAM 3, or from Write/Read mode to Read only mode for QDR II+ SRAM Write-once-Read-forever mode.

At the "Exe" state, the ATG state goes through LDWait → Load if the current instruction is completed. At the LDWait, the ATG waits for all Read requests to have data returned.

At the "Exe" state, the ATG state goes through DNWait → Done if the last instruction is completed. At the DNWait, the ATG waits for all Read requests to have data returned.

At the "Exe" state, the ATG state goes through ERRWait → ERRChk if an error is detected. At the ERRWait, the ATG waits for all Read requests to have data returned. The "ERRChk" state performs read test by issuing read requests to the application interface and determining whether "Read" or "Write" error occurred. After read test completion, the ATG state moves to "ERRDone."

At "Done," "Pause," and "ErrDone" states, the ATG can be restarted ATG by issuing `tg_restart`.

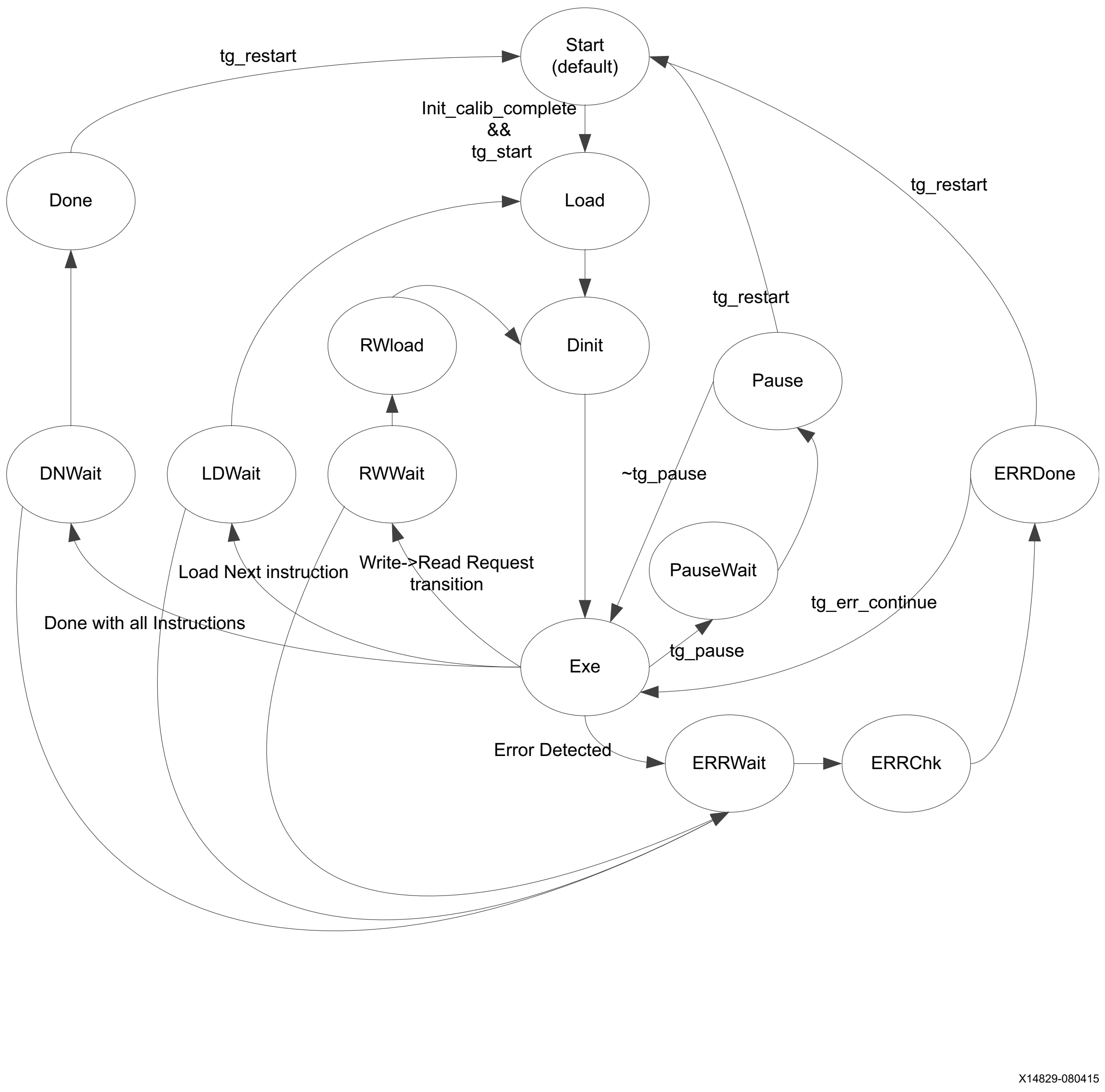


Figure 36-5: **Traffic Generator State Machine**

Table 36-5: **Traffic Generator State Machine States**

State	Enum	Description
Start (default)	0	Default state after reset. Proceed to "Load" state when init_calib_complete and vio_tg_start are TRUE.
Load	1	Load instruction into instruction pointer. Determine "Read" and/or "Write" requests to be made in "EXE" state based on read/write mode.
Dinit	2	Data initialization of all Data and Address Pattern generators.
Exe	3	Execute state. Sends "Read" and/or "Write" requests to APP interface until programmed request count is met.

Table 36-5: **Traffic Generator State Machine States *(Cont'd)***

State	Enum	Description
RWLoad	4	Update "Read" and/or "Write" requests to be made in "EXE" state based on read/write mode.
ERRWait	5	Waiting until all outstanding "Read" traffic has returned and checked.
ERRChk	6	Perform read test to determine if error type is "Read" or "Write" error.
ERRDone	7	Stopped after an error. You could continue or restart TG.
Pause	8	Pause traffic
PauseWait	13	Waiting until all outstanding "Read" traffic has returned and checked. Go to Pause state after all outstanding “Read” traffic are completed.
LDWait	9	Waiting until all outstanding "Read" traffic has returned and checked. Go to Load state after all outstanding “Read” traffic are completed.
RWWait	10	Waiting until all outstanding "Read" traffic has returned and checked. Go to RWLoad state after all outstanding “Read” traffic are completed.
DNWait	11	Waiting until all outstanding "Read" traffic has returned and checked. Go to Done state after all outstanding “Read” traffic are completed.
Done	12	All instruction completed. You can program or restart TG.

Traffic Generator Supported Interface and Configuration

The ATG supports DDR3/DDR4, RLDRAM II/RLDRAM 3, QDR II+ SRAM, and QDR-IV interfaces with various configurations. For each interface and configuration, the CMD_PER_CLK needs to be programmed with a different value.

Table 36-6: **CMD_PER_CLK Setting for 4:1 General Interconnect Cycle to Memory Clock Cycle**

Burst Length/ Mem Type	UltraScale		7 Series		
	DDR3/DDR4	RLDRAM 3	DDR3/DDR4	RLDRAM II	RLDRAM 3
8	1	1	1	1	1
4	–	2	–	2	2
2	–	4	–	–	4

Table 36-7: **CMD_PER_CLK Setting for 2:1 General Interconnect Cycle to Memory Clock Cycle**

Burst Length/ Mem Type	UltraScale	7 Series			
	QDR II+	DDR3/DDR4	RLDRAM II	RLDRAM 3	QDR II+
8	–	0.5	0.5	0.5	–
4	1	–	1	1	1
2	2	–	–	2	2

Note: For design with 2:1 general interconnect cycle to memory clock cycle ratio and burst length 8 (BL = 8), ATG error status interface `vio_tg_status_*` presents data in full burst (that is, double the APP_DATA_WIDTH).

How to Program Victim Mode/Victim Select/Victim Aggressor Delay

Basic cross-coupling patterns are supported in the victim mode. In a given Victim mode, the victim and aggressor behaviors are controlled by the Victim Select and the Victim Aggressor Delay.

First, program Victim mode to choose victim/aggressor relationship.

- **Held1** – All aggressors held at 1
- **Held0** – All aggressors held at 0
- **NONINV_AGGR** – All aggressors are same as victim pattern
- **INV_AGGR** – All aggressors are presented as inversion of victim pattern
- **DELAYED_AGGR** – All aggressors are presented as delayed version of victim pattern. Delay is programmable (`vio_tg_victim_aggr_delay`).
- **DELAYED_VICTIM** – Victim is presented as delayed version of aggressor pattern. Delay is programmable (`vio_tg_victim_aggr_delay`).
- **CAL_CPLX** – Both victim and aggressor are defined as calibration complex pattern. Both Data Mode and Victim Mode have to be programmed to CAL_CPLX.

After a Victim mode is selected, program the victim/aggressor select.

- Use the external VIO signal to choose victim bit (`vio_tg_glb_victim_bit`).
- Rotate victim per nibble (from Bits[3:0]) for every nibble.
- Rotate victim per byte (from Bits[7:0]) for every byte.
- Rotate victim in the whole memory interface.

If you selected Victim mode DELAYED_AGGR or DELAYED_VICTIM, the number of UI cycle shifted is programmed in `vio_tg_victim_aggr_delay` (where $0 \le N \le 24$).

Note: CAL_CPLX is a Xilinx internal mode that is used for the Calibration Complex Pattern.

How to Program PRBS Data Seed

One of the programmable traffic pattern data modes is PRBS data mode. In PRBS data mode, the PRBS Data Seed can be programmed per data bit using the VIO interface.

The following are steps to program PRBS Data Seed (wait for at least one general interconnect cycle between each step):

1. Set the `vio_tg_start` to 0 to stop traffic generator before reset deassertion.
2. Check the `vio_tg_status_state` to be TG_INSTR_START (hex0).

3. Set the **`vio_tg_seed_num`** and **`vio_tg_seed_data`** with the desired seed address number and seed.
4. Wait for four general interconnect cycles (optional for relaxing VIO write timing).
5. Set the **`vio_tg_seed_program_en`** to 1. This enables seed programming.
6. Wait for four general interconnect cycles (optional for relaxing VIO write timing).
7. Set the **`vio_tg_seed_program_en`** to 0. This disables seed programming.
8. Wait for four general interconnect cycles (optional for relaxing VIO write timing).
9. Repeat steps 3 to 8 if more than one seed (data bit) is programmed.
10. Set the **`vio_tg_start`** to 1. This starts traffic generator with new seed programming.

How to Program Linear Data Seed

One of the programmable traffic pattern data modes is Linear data mode. In Linear data mode, Linear data seed can be programmed by the parameter TG_PATTERN_MODE_LINEAR_DATA_SEED. The seed has a width of APP_DATA_WIDTH. For 4:1 general interconnect cycle to memory clock cycle ratio (nCK_PER_CLK), the seed format consists of eight data bursts of linear seed.

For 2:1 general interconnect cycle to memory clock cycle ratio, the seed format consists of four data bursts of linear seed. Each linear seed has a width of DQ_WIDTH.

For example, a 72-bit wide memory design with 4:1 general interconnect cycle to memory clock cycle ratio, linear seed starting with base of decimal 1,024 is presented by {**`72'd1031`**, **`72'd1030`**, **`72'd1029`**, **`72'd1028`**, **`72'd1027`**, **`72'd1026`**, **`72'd1025`**, and **`72'd1024`**}.

A second example, a 16-bit wide memory design with 2:1 general interconnect cycle to memory clock cycle ratio, linear seed starting with base of zero is presented by {**`16'd3`**, **`16'd2`**, **`16'd1`**, and **`16'd0`**}.

How to Program Linear Address Seed

One of the programmable traffic pattern address modes is Linear address mode. In Linear address mode, the Linear Address Seed can be programmed using the VIO input (**`vio_tg_glb_start_addr`**).

The seed has a width of APP_ADDR_WIDTH and it is formed by a concatenation of N number of consecutive linear address seeds, where the number N is listed in Table 36-8 and Table 36-9.

Table 36-8: **Linear Address Seed Look Up Table for 4:1 General Interconnect Cycle to Memory Clock Cycle**

Burst Length/ Mem Type	UltraScale		7 Series		
	DDR3/DDR4	RLDRAM 3	DDR3/DDR4	RLDRAM II	RLDRAM 3
8	1	1	1	1	1
4	–	1	–	1	1
2	–	1	–	–	1

Table 36-9: **Linear Address Seed Look Up Table for 2:1 General Interconnect Cycle to Memory Clock Cycle**

Burst Length/ Mem Type	UltraScale	7 Series			
	QDR II+	DDR3/DDR4	RLDRAM II	RLDRAM 3	QDR II+
8	–	1	1	1	–
4	1	–	1	1	1
2	2	–	–	1	2

Least significant bit(s) of Linear address seed is padded with zero. For DDR3/DDR4, the 3-bit of zero is padded because the burst length of eight is always used.

For RLDRAM 3, the 4-bit of zero is padded because the ATG cycles through 16 RLDRAM 3 banks automatically. For QDR II+ and QDR-IV SRAM interfaces, zero padding is not required.

Read/Write Submode

When Read/Write mode is programmed to Write/Read mode in an instruction, there are two options to perform the data write and read.

- ATG writes all data, then reads all data
- ATG switches between write and read pseudo-randomly. In this mode, data write is always ahead of data read.

IMPORTANT: *This mode is not supported in QDR II+ or QDR-IV SRAM interfaces.*

QDR II+ and QDR-IV SRAMs ATG Support

This section covers special supports for QDR II+ and QDR-IV SRAM interfaces.

For QDR II+ SRAM, the ATG supports separate Write and Read command signals in an application interface. When Write-Read mode is selected, the ATG issues Write and Read command simultaneously.

For QDR-IV SRAM, the memory controller supports two ports. In each port, there are four read/write channels. The QDR-IV ATG top-level module is **`mig_v1_2_hw_tg_qdriv`**. The **`mig_v1_2_hw_tg_qdriv`** instantiates two regular ATG (**`mig_v1_2_hw_tg`**) and has two ATG status register interfaces. Each of the status register interface maps into one of the two ports.

QDR-IV ATG supports four different modes of traffic setup. The traffic modes are programmed using **`vio_tg_glb_qdriv_rw_submode`**.

- Both PortA and PortB send Write traffic, follow by Read traffic.
    - PortA and PortB split address space for Write/Read equally.
- PortA sends Write traffic, while PortB sends Read traffic simultaneously. In this mode, **`vio_tg_start[1]`** should be set to 1 to disable ATG[1].
- PortB sends Write traffic, while PortA sends Read traffic simultaneously. In this mode, **`vio_tg_start[1]`** should be set to 1 to disable ATG[1].
- Both PortA and PortB send a mix of Write and Read traffic over each channel per port. Only Linear Address mode is supported.

    For a given address, **`address_bit[1:0]`** represent the QDR-IV channel. **`Address_bit[5:2]`** are used to form a write mask or read mask as shown in Table 10. A one in the Read/Write Mask denotes that one of the four channels has read or write being active. For a given time, Read and Write Mask should never collide. In addition, Write is always 16 cycles ahead of Read. After the first 16 writes, Read and Write are always interlocked and all four channels are occupied with Read and Write commands.

Table 36-10: **QDR-IV Read/Write Mask and Read/Write Channel Sharing Sequence**

Cycle	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Write Mask	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cycle	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Write Mask	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Mask	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cycle	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Write Mask	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 36-10: **QDR-IV Read/Write Mask and Read/Write Channel Sharing Sequence *(Cont'd)***

Cycle	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Read Mask	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cycle	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
Write Mask	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read Mask	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cycle	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
Write Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Read Mask	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

SECTION VIII: MULTIPLE IP CORES

Multiple IP Cores

This chapter describes the specifications and pin rules for generating multiple IP cores.

Creating a Design with Multiple IP Cores

The following steps must be followed to create a design with multiple IP cores:

1. Generate the target Memory IP. If the design includes multiple instances of the same Memory IP configuration, the IP only needs to be generated once. The same IP can be instantiated multiple times within the design.
    - If the IP shares the input `sys_clk`, select the **No Buffer** clocking option during IP generation with the same frequency value selected for option **Reference Input Clock Period (ps)**. Memory IP that share `sys_clk` must be allocated in the same I/O column. For more information on Sharing of Input Clock Source, see the Sharing of Input Clock Source for a link of each controller section.
2. Create a wrapper file to instantiate the target Memory IP cores.
3. Assign the pin locations for the Memory IP I/O signals. For more information on pin rules of the respective interface, see the Sharing of a Bank for a link of each controller section. Also, to learn more about the available Memory IP pin planning options, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 18].
4. Ensure the following specifications are followed.

Sharing of a Bank

Pin rules of each controller must be followed during IP generation. For more information on pin rules of each interface, see the respective IP sections:

- DDR3 Pin Rules in Chapter 4 and DDR4 Pin Rules in Chapter 4
- LPDDR3 Pin Rules in Chapter 11
- QDR II+ Pin Rules in Chapter 18
- QDR-IV Pin Rules in Chapter 25

- RLDRAM 3 Pin Rules in Chapter 32

The same bank can be shared across multiple IP cores, but Memory IP allows sharing of banks across multiple IP cores if the rules for combining I/O standards in the same bank are followed.

IMPORTANT: *If two controllers share a bank, they cannot be reset independently. The two controllers must have a common reset input.*

For more information on the rules for combining I/O standards in the same bank, see the section "Rules for Combining I/O Standards in the Same Bank," in *UltraScale™ Architecture SelectIO™ Resources User Guide* (UG571) [Ref 7]. The DCI I/O banking rules are also captured in UG571.

Sharing of Input Clock Source

One GCIO pin can be shared across multiple IP cores. There are certain rules that must be followed to share input clock source and you must perform a few manual changes in the wrapper files. For more information on Sharing of Input Clock Source, see the respective interfaces:

- Sharing of Input Clock Source (sys_clk_p) in Chapter 4 (DDR3/DDR4)
- Sharing of Input Clock Source (sys_clk_p) in Chapter 11 (LPDDR3)
- Sharing of Input Clock Source (sys_clk_p) in Chapter 18 (QDR II+ SRAM)
- Sharing of Input Clock Source (sys_clk_p) in Chapter 25 (QDR-IV SRAM)
- Sharing of Input Clock Source (sys_clk_p) in Chapter 32 (RLDRAM 3)

XSDB and dbg_clk Changes

The **`dbg_clk`** port is an output from the Memory IP and it automatically connects to the **`dbg_hub`** logic by Vivado® during implementation. If multiple IP cores are instantiated in the same project, Vivado automatically connects the first IP **`dbg_clk`** to **`dbg_bug`**.

In the wrapper file in which multiple Memory IP cores are instantiated, do not connect any signal to **`dbg_clk`** and keep the port open during instantiation. Vivado takes care of the **`dbg_clk`** connection to the **`dbg_hub`**.

MMCM Constraints

MMCM must be allocated in the center bank of the memory I/Os selected banks. Memory IP generates the LOC constraints for MMCM such that there is no conflict if the same bank is shared across multiple IP cores.

SECTION IX: DEBUGGING

Chapter 38

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the Memory IP, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

Documentation

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

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

Solution Centers

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

The Solution Center specific to the Memory IP core is located at Xilinx Memory IP Solution Center.

Answer Records

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

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

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

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

Master Answer Record for the Memory IP

AR: 58435

Technical Support

Xilinx provides technical support at the Xilinx support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

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

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

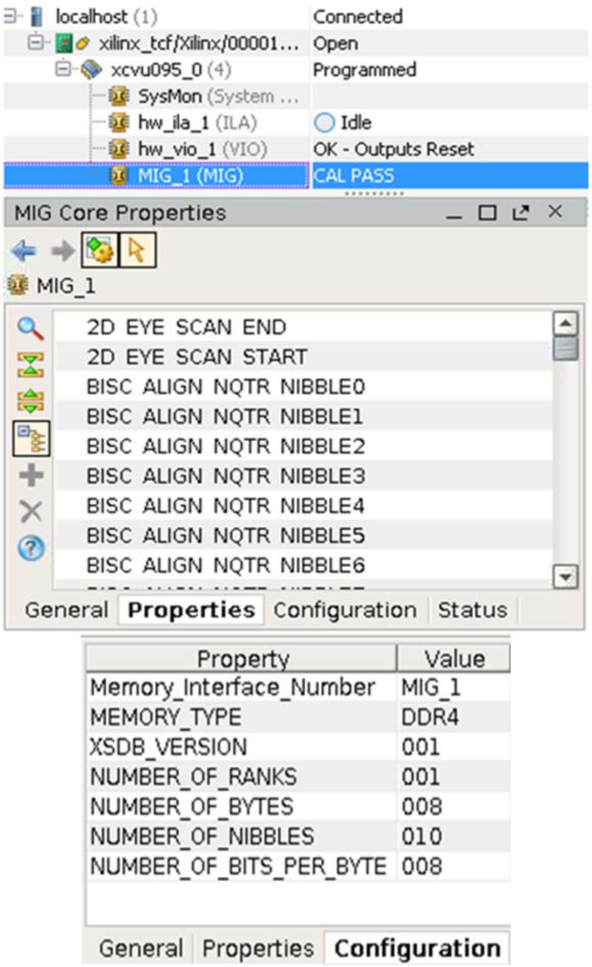
There are many tools available to address Memory IP design issues. It is important to know which tools are useful for debugging various situations.

XSDB Debug

Memory IP includes XSDB debug support. The Memory IP stores useful core configuration, calibration, and data window information within internal block RAM. The Memory IP debug XSDB interface can be used at any point to read out this information and get valuable statistics and feedback from the Memory IP. The information can be viewed through a Memory IP Debug GUI or through available Memory IP Debug Tcl commands.

Memory IP Debug GUI Usage

After configuring the device the Memory IP debug core and contents are visible in the Hardware Manager (Figure 38-1).



localhost (1)	Connected
xilinx_tcf/Xilinx/00001...	Open
xcvu095_0 (4)	Programmed
SysMon (System ...	
hw_ila_1 (ILA)	○ Idle
hw_vio_1 (VIO)	OK - Outputs Reset
MIG_1 (MIG)	CAL PASS

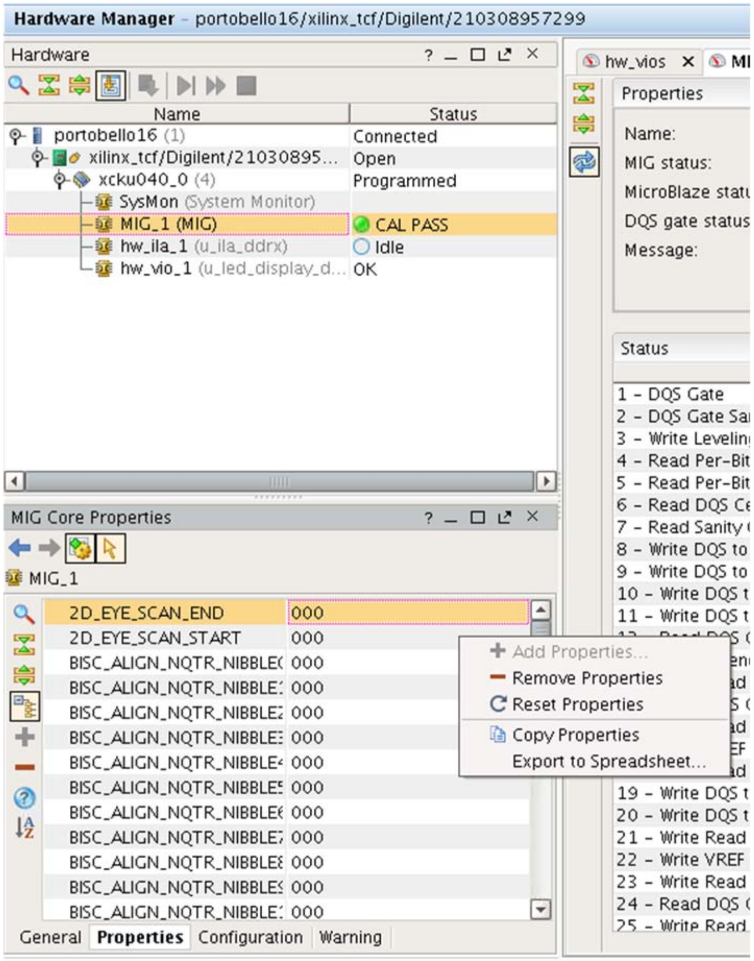
2D EYE SCAN END
2D EYE SCAN START
BISC ALIGN NQTR NIBBLE0
BISC ALIGN NQTR NIBBLE1
BISC ALIGN NQTR NIBBLE2
BISC ALIGN NQTR NIBBLE3
BISC ALIGN NQTR NIBBLE4
BISC ALIGN NQTR NIBBLE5
BISC ALIGN NQTR NIBBLE6

Property	Value
Memory_Interface_Number	MIG_1
MEMORY_TYPE	DDR4
XSDB_VERSION	001
NUMBER_OF_RANKS	001
NUMBER_OF_BYTES	008
NUMBER_OF_NIBBLES	010
NUMBER_OF_BITS_PER_BYTE	008

Figure 38-1: **Memory IP Debug Properties and Configuration Windows**

To export information about the properties to a spreadsheet, see Figure 38-2 which shows the Memory IP Core Properties window. Under the **Properties** tab, right-click anywhere in the field, and select the **Export to Spreadsheet** option in the context menu. Select the location and name of the file to save, use all the default options, and then select **OK** to save the file.

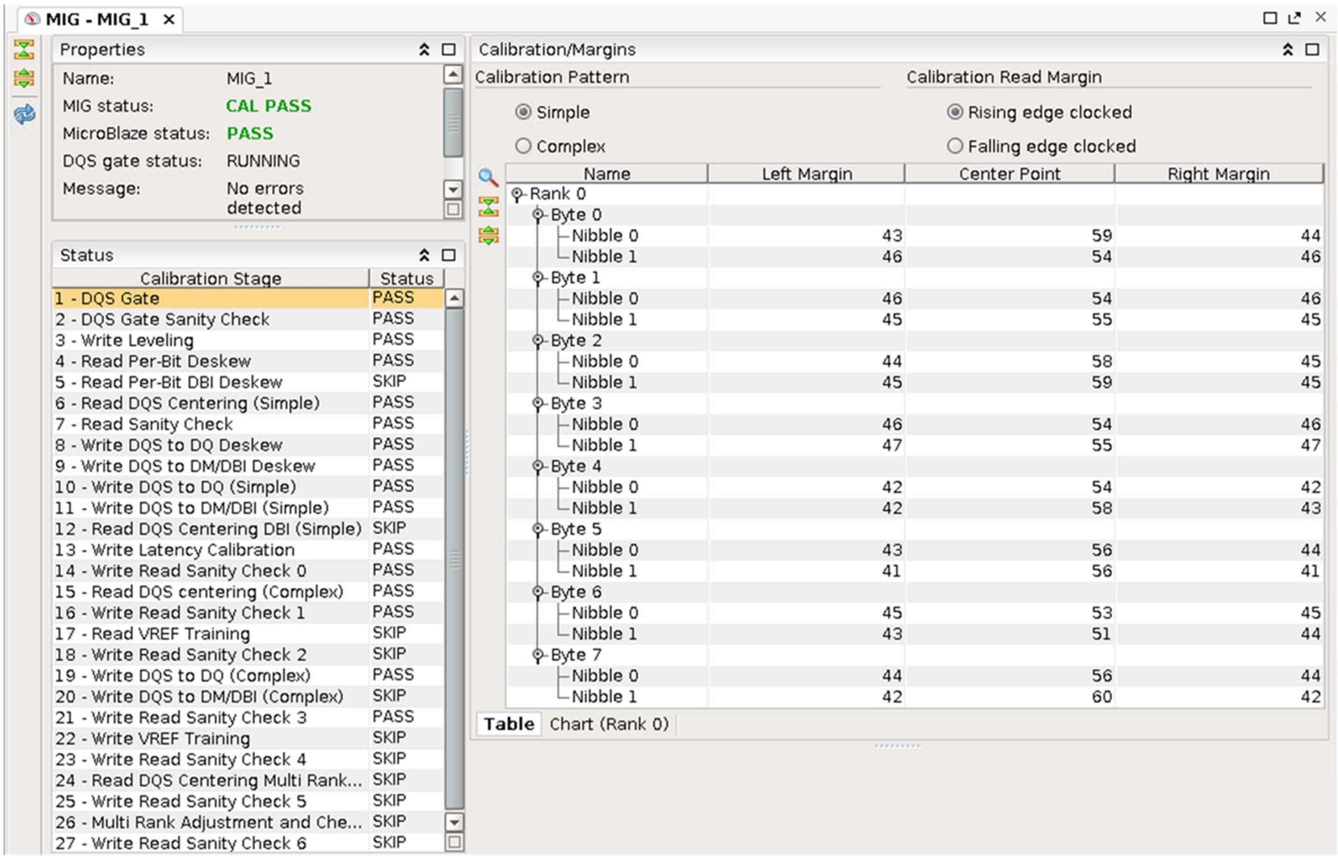
For more information on the **Properties** window menu commands, see the "Properties Window Popup Menu Commands" section in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 22].



Name	Status
portobello16 (1)	Connected
xilinx_tcf/Digilent/21030895...	Open
xcku040_0 (4)	Programmed
SysMon (System Monitor)	
MIG_1 (MIG)	● CAL PASS
hw_ila_1 (u_ila_ddrx)	○ Idle
hw_vio_1 (u_led_display_d...	OK

2D_EYE_SCAN_END	000
2D_EYE_SCAN_START	000
BISC_ALIGN_NQTR_NIBBLE(	000
BISC_ALIGN_NQTR_NIBBLE:	000
BISC_ALIGN_NQTR_NIBBLE2	000
BISC_ALIGN_NQTR_NIBBLE3	000
BISC_ALIGN_NQTR_NIBBLE4	000
BISC_ALIGN_NQTR_NIBBLE5	000
BISC_ALIGN_NQTR_NIBBLE6	000
BISC_ALIGN_NQTR_NIBBLE7	000
BISC_ALIGN_NQTR_NIBBLE8	000
BISC_ALIGN_NQTR_NIBBLE9	000
BISC_ALIGN_NQTR_NIBBLE:	000

Figure 38-2: **Memory IP Core Properties Export to Spreadsheet**



Properties

Name: MIG_1
MIG status: **CAL PASS**
MicroBlaze status: **PASS**
DQS gate status: RUNNING
Message: No errors detected

Status

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	PASS
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Che...	SKIP
27 - Write Read Sanity Check 6	SKIP

Calibration/Margins

Calibration Pattern
● Simple
○ Complex

Calibration Read Margin
● Rising edge clocked
○ Falling edge clocked

Name	Left Margin	Center Point	Right Margin
Rank 0			
Byte 0			
Nibble 0	43	59	44
Nibble 1	46	54	46
Byte 1			
Nibble 0	46	54	46
Nibble 1	45	55	45
Byte 2			
Nibble 0	44	58	45
Nibble 1	45	59	45
Byte 3			
Nibble 0	46	54	46
Nibble 1	47	55	47
Byte 4			
Nibble 0	42	54	42
Nibble 1	42	58	43
Byte 5			
Nibble 0	43	56	44
Nibble 1	41	56	41
Byte 6			
Nibble 0	45	53	45
Nibble 1	43	51	44
Byte 7			
Nibble 0	44	56	44
Nibble 1	42	60	42

Figure 38-3: **Example Display of Memory IP Debug Core**

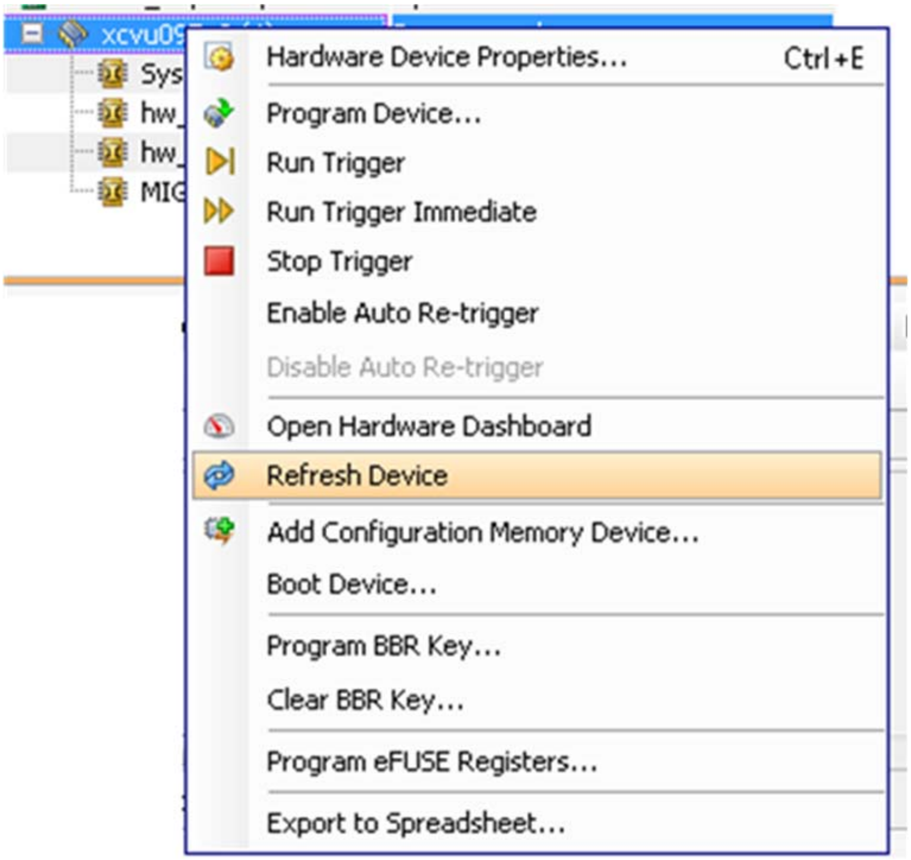


Figure 38-4: **Example of Refresh Device**

Memory IP Debug Tcl Usage

The following Tcl commands are available from the Vivado Tcl Console when connected to the hardware.

This outputs all XSDB Memory IP content that is displayed in the GUIs.

- **`get_hw_migs`** – Displays what Memory IP cores exist in the design
- **`refresh_hw_device`** – Refreshes the whole device including all cores
- **`refresh_hw_mig [lindex [get_hw_migs] 0]`** – Refreshes only the Memory IP core denoted by index (index begins with 0).
- **`report_property [lindex [get_hw_migs] 0]`** – Reports all of the parameters available for the Memory IP core. Where 0 is the index of the Memory IP core to be reported (index begins with 0).
- **`report_debug_core`** – Reports all debug core peripherals connected to the Debug Hub "**`dbg_hub`**." Associates the debug core "Index" with the "Instance Name." Useful when multiple instances of Memory IP are instantiated within the design to associate the debug core index with the each IP instantiation.

`report_debug_core` example:

```
Peripherals Connected to Debug Hub “dbg_hub” (2 Peripherals):
+-------+------------------------------+----------------------------------+
| Index | Type                         | Instance Name                    |
+-------+------------------------------+----------------------------------+
| 0     | vio_v3_0                     | gtwizard_ultrascale_0_vio_0_inst |
+-------+------------------------------+----------------------------------+
| 1     | labtools_xsdb_slave_lib_v2_1 | your_instance_name               |
+-------+------------------------------+----------------------------------+
| 2     | labtools_xsdb_slave_lib_v2_1 | your_instance_name               |
+-------+------------------------------+----------------------------------+
| 3     | labtools_xsdb_slave_lib_v2_1 | your_instance_name               |
+-------+------------------------------+----------------------------------+
| 4     | labtools_xsdb_slave_lib_v2_1 | your_instance_name               |
+-------+------------------------------+----------------------------------+
```

Example Design

Generation of a DDR3/DDR4 design through the Memory IP tool allows an example design to be generated using the Vivado **Generate IP Example Design** feature. The example design includes a synthesizable test bench with a traffic generator that is fully verified in simulation and hardware. This example design can be used to observe the behavior of the Memory IP design and can also aid in identifying board-related problems.

For complete details on the example design, see Chapter 6, Example Design. The following sections describe using the example design to perform hardware validation.

Debug Signals

The Memory IP UltraScale designs include an XSDB debug interface that can be used to very quickly identify calibration status and read and write window margin. This debug interface is always included in the generated Memory IP UltraScale designs.

Additional debug signals for use in the Vivado Design Suite debug feature can be enabled using the **Debug Signals** option on the **FPGA Options** Memory IP GUI screen. Enabling this feature allows example design signals to be monitored using the Vivado Design Suite debug feature. Selecting this option brings the debug signals to the top-level and creates a sample ILA core that debug signals can be port mapped into.

Furthermore, a VIO core can be added as needed. For details on enabling this debug feature, see Customizing and Generating the Core, page 217. The debug port is disabled for functional simulation and can only be enabled if the signals are actively driven by the user design.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 20].

Reference Boards

The KCU105 evaluation kit is a Xilinx development board that includes FPGA interfaces to a 64-bit (4 x16 components) DDR4 interface. This board can be used to test user designs and analyze board layout.

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado Design Suite debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Design Suite debug feature for debugging the specific problems.

Memory IP Usage

To focus the debug of calibration or data errors, use the provided Memory IP example design on the targeted board with the Debug Feature enabled through the Memory IP UltraScale GUI.

Note: Using the Memory IP example design and enabling the Debug Feature is not required to capture calibration and window results using XSDB, but it is useful to focus the debug on a known working solution.

However, the debug signals and example design are required to analyze the provided ILA and VIO debug signals within the Vivado Design Suite debug feature. The latest Memory IP release should be used to generate the example design.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

1. If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
2. If your outputs go to 0, check your licensing.
3. Ensure all guidelines referenced in Chapter 4, Designing with the Core and the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11] have been followed.
4. In Chapter 4, Designing with the Core, it includes information on clocking, pin/bank, and reset requirements. In the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11], it includes PCB guidelines such as trace matching, topology and routing, noise, termination, and I/O standard requirements. Adherence to these requirements, along with proper board design and signal integrity analysis is critical to the success of high-speed memory interfaces.
5. Measure all voltages on the board during idle and non-idle times to ensure the voltages are set appropriately and noise is within specifications.
    - Ensure the termination voltage regulator (V_{TT}) is powered on to $V_{CCO}/2$.
    - Ensure V_{REF} is measured when External V_{REF} is used and set to $V_{CCO}/2$.
6. When applicable, check `vrp` resistors.
7. Look at the clock inputs to ensure that they are clean.
8. Information on the clock input specifications can be found in the AC and DC Switching Characteristics data sheets (LVDS input requirements and PLL requirements should be considered).
9. Check the reset to ensure the polarity is correct and the signal is clean.
10. Check terminations. The *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11] should be used as a guideline.
11. Perform general signal integrity analysis.
    - Memory IP sets the most ideal ODT setting based on the memory parts and is described in the RTL as MR1. The RTL is `ddr3_0_ddr3.sv` for DDR3 and `ddr4_0_ddr4.sv` is for DDR4. IBIS simulations should be run to ensure terminations, the most ideal ODT, and output drive strength settings are appropriate.
    - For DDR3/DDR4, observe `dq/dqs` on a scope at the memory. View the alignment of the signals, V_{IL}/V_{IH}, and analyze the signal integrity during both writes and reads.
    - Observe the Address and Command signals on a scope at the memory. View the alignment, V_{IL}/V_{IH}, and analyze the signal integrity.

12. Verify the memory parts on the board(s) in test are the correct part(s) set through the Memory IP. The timing parameters and signals widths (that is, address, bank address) must match between the RTL and physical parts. Read/write failures can occur due to a mismatch.

13. If Data Mask (DM) is not being used for DDR3, ensure DM pin is tied low appropriately. For more information, see DDR3 Pin Rules in Chapter 4. Also, make sure that the GUI option for the DM selection is set correctly. If the DM is enabled in the IP but is not connected to the controller on the board, the calibration fails unpredictably.

14. For DDR3/DDR4, driving Chip Select (**`cs_n`**) from the FPGA is not required in single-rank designs. It can instead be tied low at the memory device according to the memory vendor's recommendations. Ensure the appropriate selection (**`cs_n`** enable or disable) is made when configuring the IP. Calibration sends commands differently based on whether **`cs_n`** is enabled or disabled. If the pin is tied low at the memory, ensure **`cs_n`** is disabled during IP configuration.

15. ODT is required for all DDR3/DDR4 interfaces and therefore must be driven from the FPGA. Memory IP sets the most ideal ODT setting based on extensive simulation. The most ideal ODT value is described in the RTL as MR1. The RTL file is **`ddr3_0_ddr3.sv`** for DDR3 and **`ddr4_0_ddr4.sv`** is for DDR4. External to the memory device, terminate ODT as specified in the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

16. Check for any floating pins.
    - The **`par`** input for command and address parity, **`alert_n`** input/output, and the TEN input for Connectivity Test Mode are not supported by the DDR4 UltraScale interface. Consult *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11] on how to connect these signals when not used.

      Note: The **`par`** is required for DDR3 RDIMM interfaces and is optional for DDR4 RDIMM/LRDIMM interfaces.

    - Floating **`reset_n/reset#`** or address pins can result in inconsistent failures across multiple resets and/or power supplies. If inconsistent calibration failures are seen, check the **`reset_n/reset#`** and address pins.

17. Measure the **`ck/ck_n`**, **`dqs/dqs_n`**, and system clocks for duty cycle distortion and general signal integrity.

18. If Internal V_{REF} is used (required for DDR4), ensure that the constraints are set appropriately in the XDC constraints file.

    An example of the Interval V_{REF} constraint is as follows:

    ```
    set_property INTERNAL_VREF 0.600 [get_iobanks 45]
    ```

19. Check the MMCM and PLL lock signals.

20. If no system clock is present after configuring the part, the following error is generated in Vivado Hardware Manager:

`mig_calibration_ddr3_0.csv does not exist`

21. Verify trace matching requirements are met as documented in the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].
22. Bring the **`init_calib_complete`** out to a pin and check with a scope or view whether calibration completed successfully in Hardware Manager in the Memory IP Debug GUI.
23. Verify the configuration of the Memory IP. The XSDB output can be used to verify the Memory IP settings. For example, the clock frequencies, version of Memory IP, Mode register settings, and the memory part configuration (see step 12) can be determined using Table 38-1.

Table 38-1: **Memory IP Configuration XSDB Parameters**

Variable Name	Description
CAL_MAP_VERSION	2015.1/2015.2 = 1 2015.3/2015.4 = 2 2016.1 = 3
CAL_STATUS_SIZE	7
CAL_VERSION_C_MB	C Code Version 2015.1 = 1 2015.2 = 2 2015.3 = 3 2015.4 = 4 2016.1 = 5
CAL_VERSION_RTL	RTL Code Version 2015.1 = 1 2015.2 = 2 2015.3 = 3 2015.4 = 4 2016.1 = 5
CONFIG_INFORMATION_0	Reserved
CONFIG_INFORMATION_0	Reserved
CONFIG_INFORMATION_1	Reserved
CONFIG_INFORMATION_2	Reserved
CONFIG_INFORMATION_3	Reserved
CONFIG_INFORMATION_4	Reserved
CONFIG_INFORMATION_5	Reserved
CONFIG_INFORMATION_6	Reserved
CONFIG_INFORMATION_7	Reserved
CONFIG_INFORMATION_8	Reserved
CONFIG_INFORMATION_9	Reserved

Table 38-1: **Memory IP Configuration XSDB Parameters *(Cont'd)***

Variable Name	Description
CONFIG_INFORMATION_10	Reserved
CONFIG_INFORMATION_11	Reserved
CONFIG_INFORMATION_12	Reserved
CONFIG_INFORMATION_13	Reserved
CONFIG_INFORMATION_14	Reserved
CONFIG_INFORMATION_15	Reserved
CONFIG_INFORMATION_16	Reserved
CONFIG_INFORMATION_17	Reserved
CONFIG_INFORMATION_18	Reserved
CONFIG_INFORMATION_19	Reserved
CONFIG_INFORMATION_20	Reserved
CONFIG_INFORMATION_21	Reserved
CONFIG_INFORMATION_22	Reserved
CONFIG_INFORMATION_23	Reserved
CONFIG_INFORMATION_24	Reserved
CONFIG_INFORMATION_25	Reserved
CONFIG_INFORMATION_26	Reserved
CONFIG_INFORMATION_27	Reserved
CONFIG_INFORMATION_28	Reserved
CONFIG_INFORMATION_29	Reserved
CONFIG_INFORMATION_30	Reserved
CONFIG_INFORMATION_31	Reserved
CONFIG_INFORMATION_32	Reserved
MR0_0	MR0[8:0] Setting
MR0_1	MR0[15:9] Setting
MR1_0	MR1[8:0] Setting
MR1_1	MR1[15:9] Setting
MR2_0	MR2[8:0] Setting
MR2_1	MR2[15:9] Setting
MR3_0	MR3[8:0] Setting
MR3_1	MR3[15:9] Setting
MR4_0	MR4[8:0] Setting
MR4_1	MR4[15:9] Setting
MR5_0	MR5[8:0] Setting
MR5_1	MR5[15:9] Setting

Table 38-1: **Memory IP Configuration XSDB Parameters *(Cont'd)***

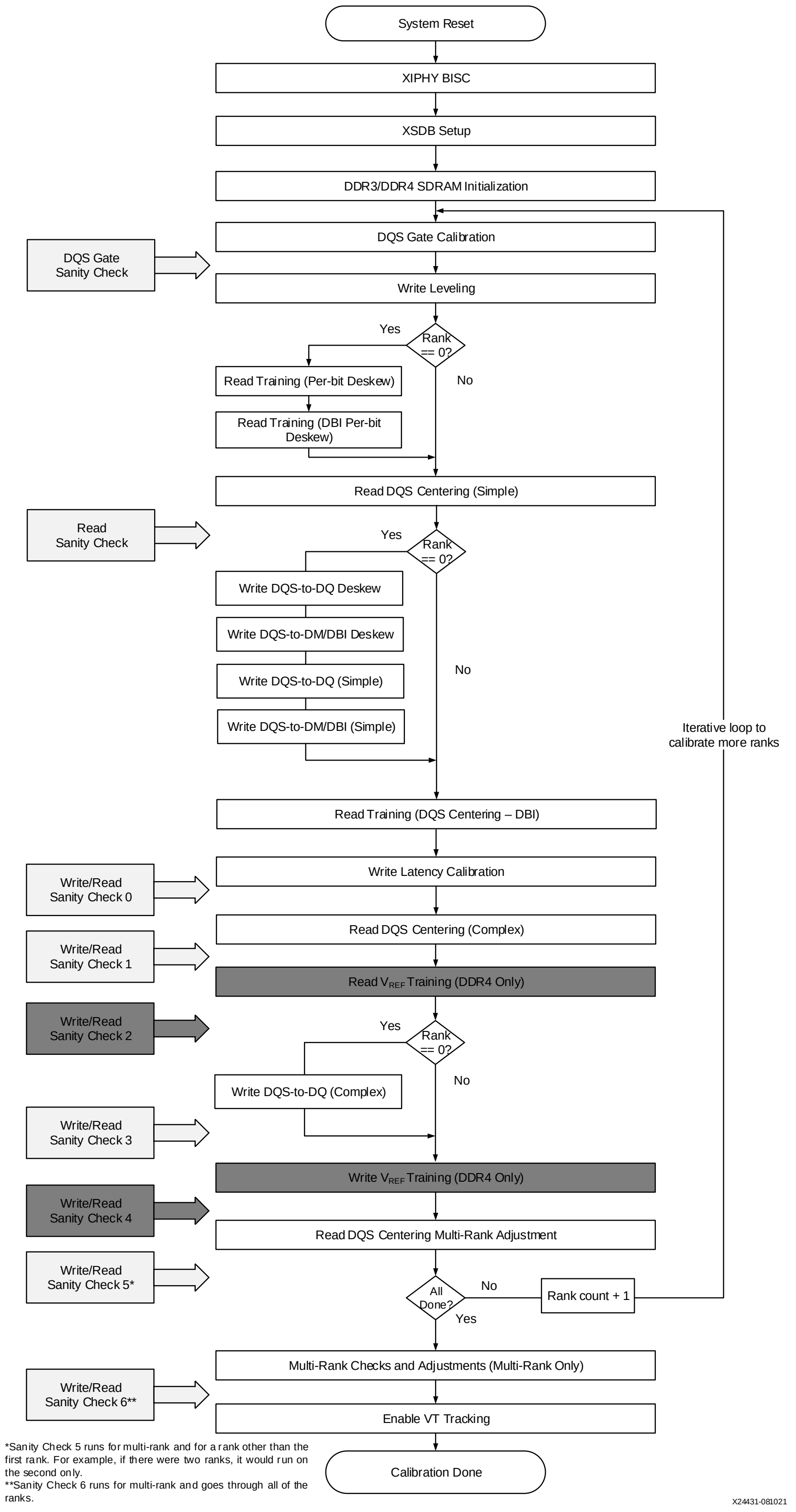
Variable Name	Description
MR6_0	MR6[8:0] Setting
MR6_1	MR6[15:9] Setting
Memory_Code_Name	Reserved
Memory_Frequency_0	Memory tCK [8:0]
Memory_Frequency_1	Memory tCK [16:9]
Memory_Module_Type	Module Type Component = 01 UDIMM = 02 SODIMM = 03 RDIMM = 04
Memory_Voltage	Memory Voltage 1.2V = 01 1.35V = 02 1.5V = 03
Mem_Type	Memory Type DDR3 = 01 DDR4 = 02 RLDRAM 3 = 03 QDR II+ SRAM = 04
PLL_M	CLKFBOUT_MULT_F value used in the core TXPLLs.
PLL_D	DIVCLK_DIVIDE value using in the core TXPLLs.
MMCM_M	CLKFBOUT_MULT_F value used in the core MMCM.
MMCM_D	DIVCLK_DIVIDE value using in the core MMCM.
Controller_Info	Reserved

24. Copy all of the data reported and submit it as part of a WebCase. For more information on opening a WebCase, see Technical Support, page 580.

Debugging DDR3/DDR4 Designs

Calibration Stages

Figure 38-5 shows the overall flow of memory initialization and the different stages of calibration. The dark gray color is not available for this release.



*Sanity Check 5 runs for multi-rank and for a rank other than the first rank. For example, if there were two ranks, it would run on the second only.

**Sanity Check 6 runs for multi-rank and goes through all of the ranks.

Figure 38-5: **PHY Overall Initialization and Calibration Sequence**

Memory Initialization

The PHY executes a JEDEC-compliant DDR3/DDR4 initialization sequence following the deassertion of system reset. Each DDR3/DDR4 SDRAM has a series of Mode registers accessed through Mode register set (MRS) commands. These Mode registers determine various SDRAM behaviors, such as burst length, read and write CAS latency, and additive latency. Memory IP designs never issue a calibration failure during Memory Initialization.

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

Debug Signals

There are two types of debug signals used in Memory IP UltraScale debug. The first set is a part of a debug interface that is always included in generated Memory IP UltraScale designs. These signals include calibration status and tap settings that can be read at any time throughout operation when the Hardware Manager is open using either Tcl commands or the Memory IP Debug GUI.

The second type of debug signals are fully integrated in the IP when the **Debug Signals** option in the Memory IP tool is enabled and when using the Memory IP Example Design. However, these signals are currently only brought up in the RTL and not connected to the debug VIO/ILA cores. Manual connection into either custom ILA/VIOs or the ILA generated when the **Debug Signals** option is enabled is currently required. These signals are documented in Table 38-2.

Table 38-2: **DDR3/DDR4 Debug Signals Used in Vivado Design Suite Debug Feature**

Signal	Signal Width	Signal Description
init_calib_complete	[0:0]	Signifies the status of calibration. 1'b0 = Calibration not complete 1'b1 = Calibration completed successfully
cal_pre_status	[8:0]	Signifies the status of the memory core before calibration has started. See Table 38-3 for decoding information.
cal_r*_status	[127:0]	Signifies the status of each stage of calibration. See Table 38-4 for decoding information. See the following relevant debug sections for usage information. ***Note:*** The * indicates the rank value. Each rank has a separate cal_r*_status bus.
cal_post_status	[8:0]	Signifies the status of the memory core after calibration has finished. See Table 38-5 for decoding information.

Table 38-2: **DDR3/DDR4 Debug Signals Used in Vivado Design Suite Debug Feature *(Cont'd)***

<table>
<tr><th>Signal</th><th>Signal Width</th><th>Signal Description</th></tr>
<tr><td>dbg_cal_seq</td><td>[2:0]</td><td>Calibration sequence indicator, when RTL is issuing commands to the DRAM.
[0] = 1'b0 -> Single Command Mode, one DRAM command only. 1'b1 -> Back-to-Back Command Mode. RTL is issuing back-to-back commands.
[1] = Write Leveling Mode.
[2] = Extended write mode enabled, where extra data and DQS pulses are sent to the DRAM before and after the regular write burst.</td></tr>
<tr><td>dbg_cal_seq_cnt</td><td>[31:0]</td><td>Calibration command sequence count used when RTL is issuing commands to the DRAM. Indicates how many DRAM commands are requested (counts down to 0 when all commands are sent out).</td></tr>
<tr><td>dbg_cal_seq_rd_cnt</td><td>[7:0]</td><td>Calibration read data burst count (counts down to 0 when all expected bursts return), used when RTL is issuing read commands to the DRAM.</td></tr>
<tr><td>dbg_rd_valid</td><td>[0:0]</td><td>Read Data Valid</td></tr>
<tr><td>dbg_cmp_byte</td><td>[5:0]</td><td>Calibration byte selection (used to determine which byte is currently selected and displayed in dbg_rd_data).
<table>
<tr><th>dbg_cmp_byte</th><th>DQS Byte</th></tr>
<tr><td>000000</td><td>0</td></tr>
<tr><td>000001</td><td>1</td></tr>
<tr><td>000010</td><td>2</td></tr>
<tr><td>000011</td><td>3</td></tr>
<tr><td>000100</td><td>4</td></tr>
<tr><td>000101</td><td>5</td></tr>
<tr><td>000110</td><td>6</td></tr>
<tr><td>000111</td><td>7</td></tr>
<tr><td>001000</td><td>8</td></tr>
<tr><td>001001</td><td>9</td></tr>
<tr><td>001010</td><td>10</td></tr>
<tr><td>001011</td><td>11</td></tr>
<tr><td>001100</td><td>12</td></tr>
<tr><td>001101</td><td>13</td></tr>
<tr><td>001110</td><td>14</td></tr>
<tr><td>001111</td><td>15</td></tr>
<tr><td>010000</td><td>16</td></tr>
<tr><td>010001</td><td>17</td></tr>
</table>
</td></tr>
</table>

Table 38-2: **DDR3/DDR4 Debug Signals Used in Vivado Design Suite Debug Feature *(Cont'd)***

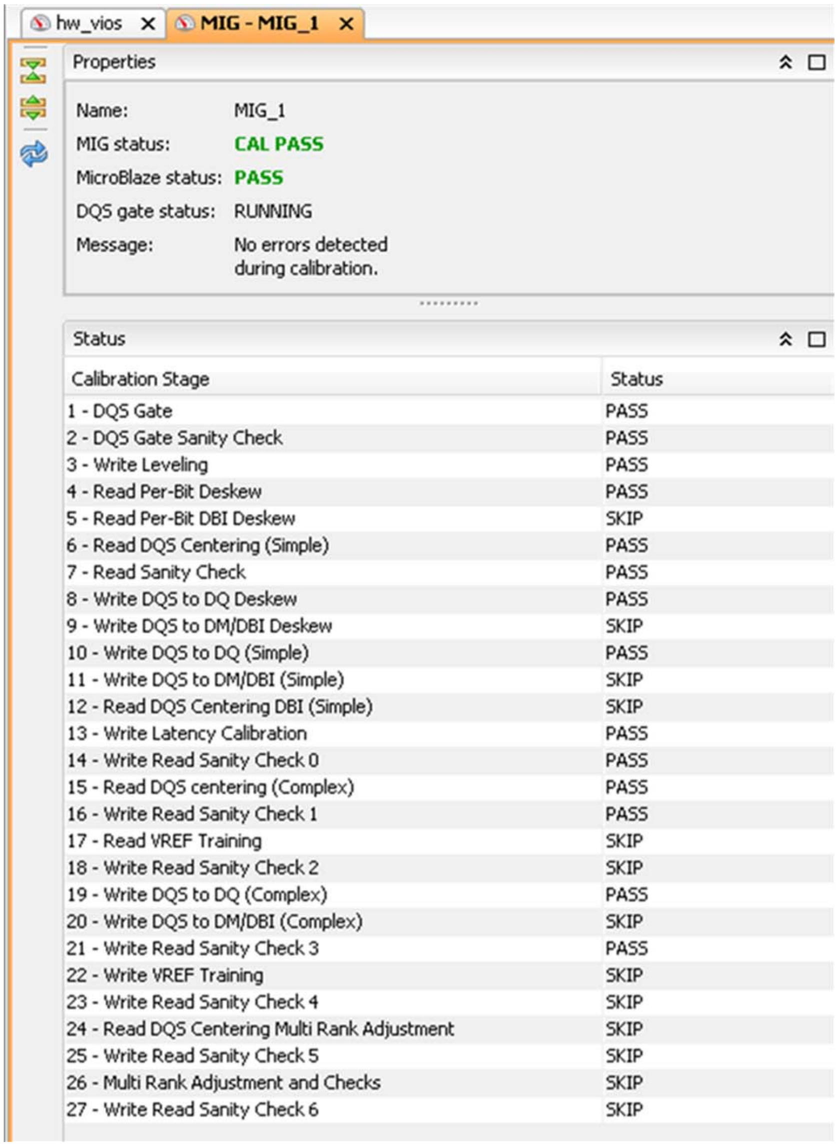
Signal	Signal Width	Signal Description
dbg_rd_data	[63:0]	Read Data from Input FIFOs
dbg_rd_data_cmp	[63:0]	Comparison of dbg_rd_data and dbg_expected_data
dbg_expected_data	[63:0]	Displays the expected data during calibration stages that use general interconnect-based data pattern comparison such as Read per-bit deskew or read DQS centering (complex).
dbg_cplx_config	[15:0]	Complex Calibration Configuration [0] = Start [1] = 1'b0 selects the read pattern. 1'b1 selects the write pattern. [3:2] = Rank selection [8:4] = Byte selection [15:9] = Number of loops through data pattern
dbg_cplx_status	[1:0]	Complex Calibration Status [0] = Busy [1] = Done
dbg_cplx_err_log	[63:0]	Complex calibration bitwise comparison result for all bits in the selected byte. Comparison is stored for each bit (1'b1 indicates compare mismatch): {fall3, rise3, fall2, rise2, fall1, rise1, fall0, rise0} [7:0] = Bit[0] of the byte [15:8] = Bit[1] of the byte [23:16] = Bit[2] of the byte [31:24] = Bit[3] of the byte [39:32] = Bit[4] of the byte [47:40] = Bit[5] of the byte [55:48] = Bit[6] of the byte [63:56] = Bit[7] of the byte
dbg_io_address	[27:0]	MicroBlaze I/O Address Bus
dbg_pllGate	[0:0]	PLL Lock Indicator
dbg_phy2clb_fixdly_rdy_low	[BYTES × 1 – 1:0]	XIPHY fixed delay ready signal (lower nibble)
dbg_phy2clb_fixdly_rdy_upp	[BYTES × 1 – 1:0]	XIPHY fixed delay ready signal (upper nibble)
dbg_phy2clb_phy_rdy_low	[BYTES × 1 – 1:0]	XIPHY PHY ready signal (lower nibble)
dbg_phy2clb_phy_rdy_upp	[BYTES × 1 – 1:0]	XIPHY PHY ready signal (upper nibble)
Traffic_error	[BYTES × 8× 8 – 1:0]	Reserved
Traffic_clr_error	[0:0]	Reserved
Win_start	[3:0]	Reserved

Determine the Failing Calibration Stage

XSDB can be used to very quickly determine which stage of calibration is failing, which byte/nibble/bit is causing the failure, and how the algorithm is failing.

Configure the device and, while the Hardware Manager is open, perform one of the following:

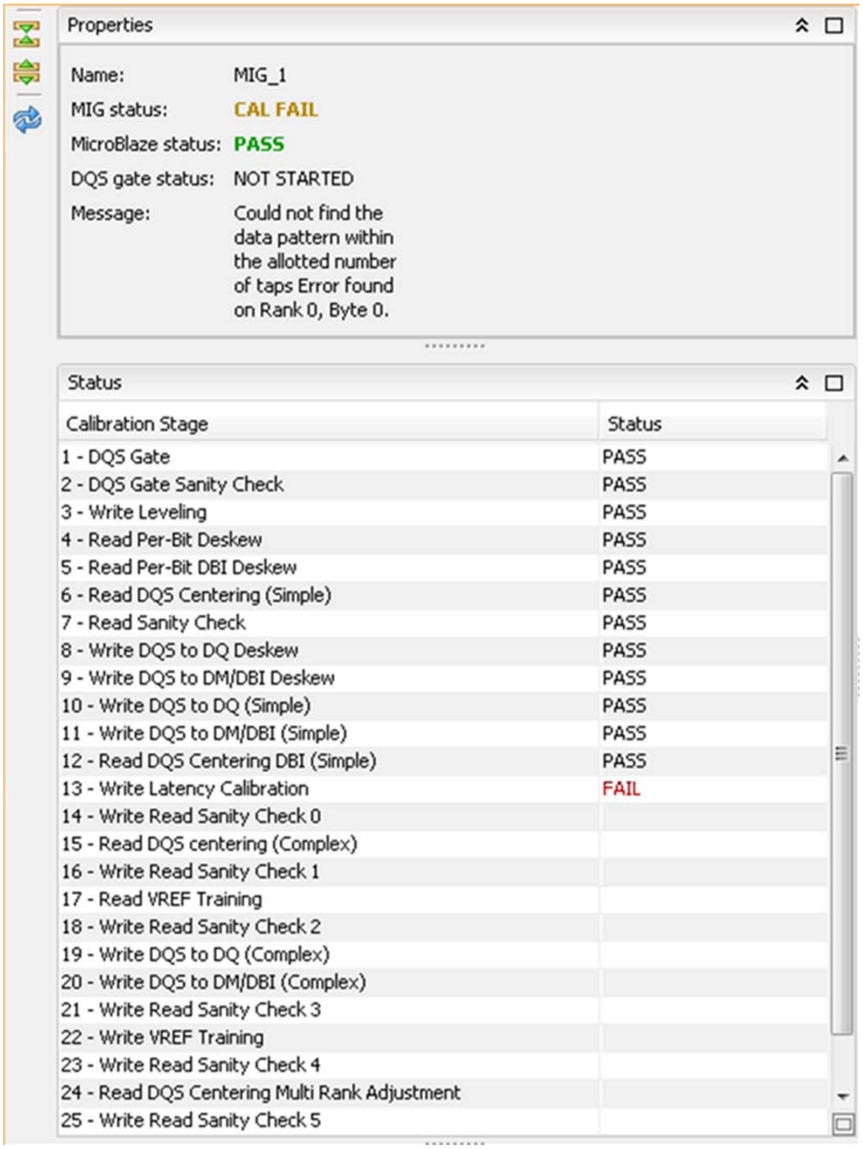
1. Use the available XSDB Memory IP GUI to identify which stages have completed, which, if any, has failed, and review the Memory IP properties window for a message on the failure. Here is a sample of the GUI for a passing and failing case:



Name: MIG_1
MIG status: **CAL PASS**
MicroBlaze status: **PASS**
DQS gate status: RUNNING
Message: No errors detected during calibration.

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	SKIP
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	SKIP
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank Adjustment	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Checks	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-6: **Memory IP XSDB Debug GUI Example – Calibration Pass**



Name: MIG_1
MIG status: CAL FAIL
MicroBlaze status: PASS
DQS gate status: NOT STARTED
Message: Could not find the data pattern within the allotted number of taps Error found on Rank 0, Byte 0.

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	PASS
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	PASS
12 - Read DQS Centering DBI (Simple)	PASS
13 - Write Latency Calibration	FAIL
14 - Write Read Sanity Check 0	
15 - Read DQS centering (Complex)	
16 - Write Read Sanity Check 1	
17 - Read VREF Training	
18 - Write Read Sanity Check 2	
19 - Write DQS to DQ (Complex)	
20 - Write DQS to DM/DBI (Complex)	
21 - Write Read Sanity Check 3	
22 - Write VREF Training	
23 - Write Read Sanity Check 4	
24 - Read DQS Centering Multi Rank Adjustment	
25 - Write Read Sanity Check 5	

Figure 38-7: **Memory IP XSDB Debug GUI Example – Calibration Failure**

2. Manually analyze the XSDB output by running the following commands in the Tcl prompt:

```
refresh_hw_device [lindex [get_hw_devices] 0]
report_property [lindex [get_hw_migs] 0]
```

Manually Analyzing the XSDB Output

The value of DDR_CAL_STATUS_RANK*_* can be used to determine which stages of calibration have passed on a per rank basis.

- RANK* within DDR_CAL_STATUS_RANK*_* denotes the physical DRAM RANK being calibrated.

- The _* at the end of DDR_CAL_STATUS_RANK*_* can be decoded in the "XSDB Status Reg" column in Table 38-4.
- XSDB Bit represents the nine bits assigned to each XSDB Status register.
- **`cal_r*_status`** represents the full port value used in simulation or when brought to an ILA core.

Note: A "1" in each bit position signifies the corresponding stage of calibration completed.

Table 38-3: **DDR3/DDR4 Pre-Cal Status**

XSDB Status Name	Bit	Description	Pre-Calibration Step
DDR_PRE_CAL_STATUS	0	Done	MicroBlaze has started up
	1	Done	Reserved
	2	Done	Reserved
	3	Done	Reserved
	4	Done	XSDB Setup Complete
	5		Reserved
	6		Reserved
	7		Reserved
	8		Reserved

Table 38-4: **DDR3/DDR4 DDR_CAL_STATUS_RANK*_* Decoding**

XSDB Status Reg	XSDB Bit	Status Bus Bits (Sim)	Description	Calibration Step
0	0	0	Start	DQS Gate
	1	1	Done	
	2	2	Start	Check for DQS gate
	3	3	Done	
	4	4	Start	Write leveling
	5	5	Done	
	6	6	Start	Read Per-bit Deskew
	7	7	Done	
	8	8	Start	Reserved

Table 38-4: **DDR3/DDR4 DDR_CAL_STATUS_RANK*_* Decoding *(Cont'd)***

XSDB Status Reg	XSDB Bit	Status Bus Bits (Sim)	Description	Calibration Step
1	0	9	Done	
	1	10	Start	Read DQS Centering (Simple)
	2	11	Done	
	3	12	Start	Read Sanity Check
	4	13	Done	
	5	14	Start	Write DQS-to-DQ Deskew
	6	15	Done	
	7	16	Start	Write DQS-to-DM Deskew
	8	17	Done	
2	0	18	Start	Write DQS-to-DQ (Simple)
	1	19	Done	
	2	20	Start	Write DQS-to-DM (Simple)
	3	21	Done	
	4	22	Start	Reserved
	5	23	Done	
	6	24	Start	Write Latency Calibration
	7	25	Done	
	8	26	Start	Write/Read Sanity Check 0
3	0	27	Done	
	1	28	Start	Read DQS Centering (Complex)
	2	29	Done	
	3	30	Start	Write/Read Sanity Check 1
	4	31	Done	
	5	32	Start	Reserved
	6	33	Done	
	7	34	Start	Write/Read Sanity Check 2
	8	35	Done	

Table 38-4: **DDR3/DDR4 DDR_CAL_STATUS_RANK*_* Decoding *(Cont'd)***

XSDB Status Reg	XSDB Bit	Status Bus Bits (Sim)	Description	Calibration Step
4	0	36	Start	Write DQS-to-DQ (Complex)
	1	37	Done	
	2	38	Start	Reserved
	3	39	Done	
	4	40	Start	Write/Read Sanity Check 3
	5	41	Done	
	6	42	Start	Reserved
	7	43	Done	
	8	44	Start	Write/Read Sanity Check 4
5	0	45	Done	
	1	46	Start	Read level multi-rank adjustment
	2	47	Done	
	3	48	Start	Write/Read Sanity Check 5 (for more than 1 rank)
	4	49	Done	
	5	50	Start	Multi-rank adjustments & Checks
	6	51	Done	
	7	52	Start	Write/Read Sanity Check 6 (all ranks)
	8	53	Done	

Table 38-5: **DDR3/DDR4 Post-Calibration Status**

XSDB Status Name	Bit	Description	Post-Calibration Step
DDR_POST_CAL_STATUS	0	Running	DQS Gate Tracking
	1	Idle	
	2	Fail	
	3	Running	Read Margin Check (Reserved)
	4	Running	Write Margin Check (Reserved)
	5		Handshake Failure (Reserved)
	6		Margin Check Failure (Reserved)
	7		Reserved
	8		Reserved

When the rank and calibration stage causing the failure are known, the failing byte, nibble, and/or bit position and error status for the failure can be identified using the signals listed in Table 38-6.

Table 38-6: **DDR3/DDR4 DDR_CAL_ERROR_0/_1/_CODE Decoding**

Variable Name	Description
DDR_CAL_ERROR_0	Bit position failing
DDR_CAL_ERROR_1	Nibble or byte position failing
DDR_CAL_ERROR_CODE	Error code specific to the failing stage of calibration. See the failing stage section below for details.

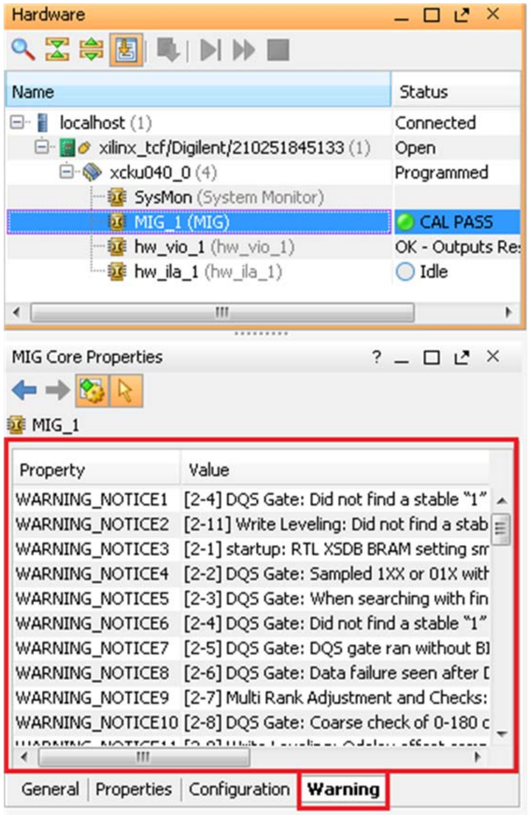
With these error codes, the failing stage of calibration, failing bit, nibble, and/or byte positions, and error code are known. The next step is to review the failing stage in the following section for specific debugging steps.

Understanding Calibration Warnings (Cal_warning)

A warning flag indicates something unexpected occurred but calibration can continue. Warnings can occur for multiple bits or bytes. Therefore, a limit on the number of warnings stored is not set. Warnings are outputs from the PHY, where the **`cal_warning`** signal is asserted for a single clock cycle to indicate a new warning.

In XSDB, the warnings are stored as part of the leftover address space in the block RAM used to store the XSDB data. The amount of space left over for warnings is dependent on the memory configuration (bus width, ranks, etc.).

The Vivado IDE displays warnings as highlighted in the example shown in Figure 38-8.

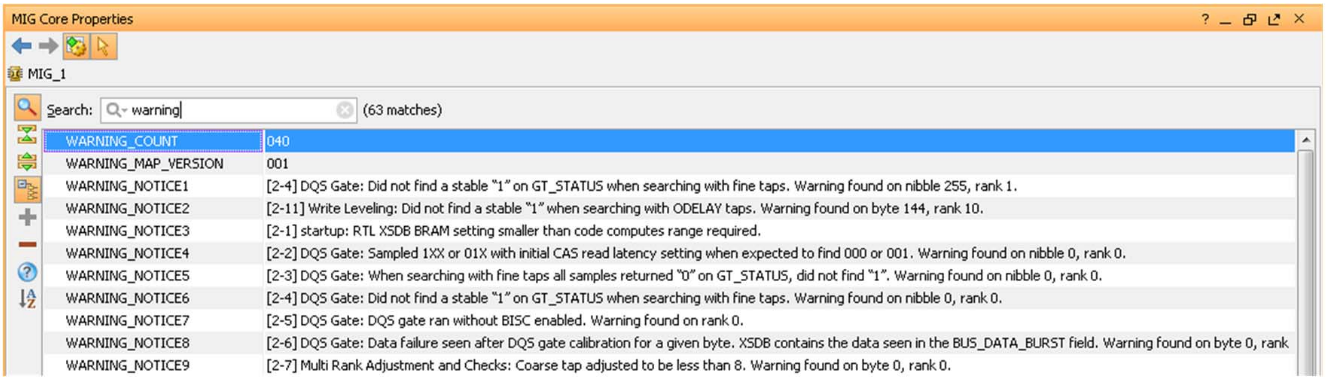


Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210251845133 (1)	Open
xcku040_0 (4)	Programmed
SysMon (System Monitor)	
MIG_1 (MIG)	CAL PASS
hw_vio_1 (hw_vio_1)	OK - Outputs Re
hw_ila_1 (hw_ila_1)	Idle

Property	Value
WARNING_NOTICE1	[2-4] DQS Gate: Did not find a stable "1"
WARNING_NOTICE2	[2-11] Write Leveling: Did not find a stab
WARNING_NOTICE3	[2-1] startup: RTL XSDB BRAM setting sm
WARNING_NOTICE4	[2-2] DQS Gate: Sampled 1XX or 01X with
WARNING_NOTICE5	[2-3] DQS Gate: When searching with fin
WARNING_NOTICE6	[2-4] DQS Gate: Did not find a stable "1"
WARNING_NOTICE7	[2-5] DQS Gate: DQS gate ran without BI
WARNING_NOTICE8	[2-6] DQS Gate: Data failure seen after D
WARNING_NOTICE9	[2-7] Multi Rank Adjustment and Checks:
WARNING_NOTICE10	[2-8] DQS Gate: Coarse check of 0-180 c

Figure 38-8: **Example Warnings Output in Warnings Tab**

The same warnings are displayed in the **Properties** window where the rest of the XSDB information is presented, as shown Figure 38-9. Apply a search filter of "warning" to find only the warning information.



Search: warning (63 matches)

WARNING_COUNT	040
WARNING_MAP_VERSION	001
WARNING_NOTICE1	[2-4] DQS Gate: Did not find a stable "1" on GT_STATUS when searching with fine taps. Warning found on nibble 255, rank 1.
WARNING_NOTICE2	[2-11] Write Leveling: Did not find a stable "1" when searching with ODELAY taps. Warning found on byte 144, rank 10.
WARNING_NOTICE3	[2-1] startup: RTL XSDB BRAM setting smaller than code computes range required.
WARNING_NOTICE4	[2-2] DQS Gate: Sampled 1XX or 01X with initial CAS read latency setting when expected to find 000 or 001. Warning found on nibble 0, rank 0.
WARNING_NOTICE5	[2-3] DQS Gate: When searching with fine taps all samples returned "0" on GT_STATUS, did not find "1". Warning found on nibble 0, rank 0.
WARNING_NOTICE6	[2-4] DQS Gate: Did not find a stable "1" on GT_STATUS when searching with fine taps. Warning found on nibble 0, rank 0.
WARNING_NOTICE7	[2-5] DQS Gate: DQS gate ran without BISC enabled. Warning found on rank 0.
WARNING_NOTICE8	[2-6] DQS Gate: Data failure seen after DQS gate calibration for a given byte. XSDB contains the data seen in the BUS_DATA_BURST field. Warning found on byte 0, rank
WARNING_NOTICE9	[2-7] Multi Rank Adjustment and Checks: Coarse tap adjusted to be less than 8. Warning found on byte 0, rank 0.

Figure 38-9: **Example Warnings Output in Properties Tab**

The following steps show how to manually read out the warnings.

1. Check the XSDB warnings fields to see if any warnings have occurred as listed in Table 38-7. If CAL_WARNINGS_END is non-zero then at least one warning has occurred.

Table 38-7: **DDR3/DDR4 DDR_CAL_ERROR_0/_1/_CODE Decoding**

Variable Name	Description
CAL_WARNINGS_START	Number of block RAM address locations used to store a single warning (set to 2).
CAL_WARNINGS_END	Total number of warnings stored in the block RAM.

2. Determine the end of the regular XSDB address range. END_ADDR0 and END_ADDR1 together form the end of the XSDB address range in the block RAM. The full address is made up by concatenating the two addresses together in binary (each made up of nine bits). For example, END_ADDR0 = `0x0AA` and END_ADDR1 = `0x004` means the end address is `0x8AA` (`` 18`b 00_0000_100 0_1010_1010 ``).
3. At the Hardware Manager Tcl Console, use the following command to read out a single warning:

```
read_hw -hw_core [ lindex [get_hw_cores] 0] 0 0x8AB 0x02
```

   This command reads out the XSDB block RAM location for the address provided up through the number of address locations requested. In the example above, the XSDB end address is `0x8AA`. Add 1 to this value to get to the warning storage area. The next field (`0x02` in the above example command) is the number of addresses to read from the starting location. Multiple addresses can be read out by changing `0x02` to whatever value is required.

4. The hex value read out is the raw data from the block RAM with four digits representing one register value. For example:

   A value of `00140000` is broken down into `0014` as the second register field and `0000` as the first register field where:

   - First field indicates bit/byte/nibble flag (depending on the warning)
   - Second field indicates the actual warning code, as shown in Table 38-8

Table 38-8 shows the description of the actual warning code.

Table 38-8: **DDR3/DDR4 Warning Code Decoding**

Stage of Calibration	Code (Decimal)	Unit 1[(1)]	Unit 2[(1)]	Description
Startup	1	N/A	N/A	RTL XSDB block RAM setting smaller than the code measures the range required.
DQS Gate	2	Nibble	Rank	(DDR4 only) Sampled 1XX or 01X with initial CAS read latency setting when expected to find 000 or 001.
DQS Gate	3	Nibble	Rank	When searching with fine taps, all samples returned 0 on GT_STATUS, did not find 1.

Table 38-8: **DDR3/DDR4 Warning Code Decoding *(Cont'd)***

Stage of Calibration	Code (Decimal)	Unit 1(1)	Unit 2(1)	Description
DQS Gate	4	Nibble	Rank	Did not find a stable 1 on GT_STATUS when searching with fine taps.
DQS Gate	5	N/A	Rank	(DDR3 only) DQS gate ran without BISC enabled.
DQS Gate	6	Byte	Rank	(DDR3 only) Data failure seen after DQS gate calibration for a given byte. XSDB contains the data seen in the BUS_DATA_BURST field.
DQS Gate	7	Byte	Rank	Multi-rank Only: Coarse taps and read latency adjusted to limit coarse taps < 8.
DQS Gate	8	Byte	Rank	After initial pattern found, the forward coarse check failed to find expected stable 1 region before searching with fine taps.
WRLVL	9	Byte	N/A	ODELAY offset computation from BISC results is 0.
WRLVL	10	Byte	N/A	Step size accelerate computation from BISC results is 0.
WRLVL	11	Byte	Rank	Did not find a stable 1 when searching with ODELAY taps.
WRLVL	12	Byte	Rank	Lowest ODELAY setting is maximum ODELAY taps allowed.
Read DQS Centering (Simple)	13	Nibble	Rank	Small window found (< 33% of the bit time) for a given rising edge nibble (P).
Read DQS Centering (Complex)	14	Nibble	Rank	Small window found (< 33% of the bit time) for a given rising edge nibble (P).
Read DQS Centering (Simple)	15	Nibble	Rank	Small window found (< 33% of the bit time) for a given falling edge nibble (N).
Read DQS Centering (Complex)	16	Nibble	Rank	Small window found (< 33% of the bit time) for a given falling edge nibble (N).
Read DQS Centering (Simple)	17	Nibble	Rank	Right edge tap setting recorded is smaller than left edge tap (P).
Read DQS Centering (Complex)	18	Nibble	Rank	Right edge tap setting recorded is smaller than left edge tap (P).
Read DQS Centering (Simple)	19	Nibble	Rank	Right edge tap setting recorded is smaller than left edge tap (N).
Read DQS Centering (Complex)	20	Nibble	Rank	Right edge tap setting recorded is smaller than left edge tap (N).
Read DQS Centering (Simple)	21	Nibble	Rank	Hit end of tap delay before finding true right edge (P).
Read DQS Centering (Complex)	22	Nibble	Rank	Hit end of tap delay before finding true right edge (P).
Read DQS Centering (Simple)	23	Nibble	Rank	Hit end of tap delay before finding true right edge (N).
Read DQS Centering (Complex)	24	Nibble	Rank	Hit end of tap delay before finding true right edge (N).
Multi-Rank Read Adjust	25	Nibble	Rank	Final XSDB PQTR value did not match what was left in the RIU.

Table 38-8: **DDR3/DDR4 Warning Code Decoding *(Cont'd)***

Stage of Calibration	Code (Decimal)	Unit 1[1]	Unit 2[1]	Description
Multi-Rank Read Adjust	26	Nibble	Rank	Final XSDB NQTR value did not match what was left in the RIU.
Write DQS-to-DQ (Simple)	27	Byte	N/A	Small window found for DQ (< 33% of the bit time).
Write DQS-to-DQ (Complex)	28	Byte	N/A	Small window found for DQ (< 33% of the bit time).
Write DQS-to-DQ (Simple)	29	Byte	N/A	Size of the DQ window found is < 4x the difference of the left and right edge.
Write DQS-to-DQ (Complex)	30	Byte	N/A	Size of the DQ window found is < 4x the difference of the left and right edge.
Write DQS-to-DQ (Simple)	31	Byte	N/A	When computing aggregate eye size between DQ and DM, the DQ eye was recorded as 0.
Write DQS-to-DQ (Simple)	32	Byte	N/A	Small window found for DM (< 33% of the bit time)
Write DQS-to-DQ (Simple)	33	Byte	N/A	DM calibrated wanted to underflow the DQS ODELAY
Write DQS-to-DQ (Simple)	34	Byte	N/A	DM calibrated wanted to overflow the DQS ODELAY
Write V_{REF}	35	Byte	Rank	V_{REF} value read back from the DRAM did not match expected value.
WRLVL	36	Byte	Rank	Could not preserve the full offset skew on the Write DQS-to-DQ/DM output for the given rank.

Notes:

1. Unit refers to value stored in the XSDB block RAM. Three locations are used in the block RAM for storage of a single warning, the first contains the code, Unit 1 is the next address, and Unit 2 is the following address.

Debugging DQS Gate Calibration Failures

The XIPHY is used to capture read data from the DRAM by using the DQS strobe to clock in read data and transfer the data to an internal FIFO using that strobe. The first step in capturing data is to evaluate where that strobe is so the XIPHY can open the gate and allow the DQS to clock the data into the rest of the PHY.

The XIPHY uses an internal clock to sample the DQS during a read burst and provides a single binary value back called GT_STATUS. This sample is used as part of a training algorithm to determine where the first rising edge of the DQS is in relation to the sampling clock.

Calibration logic issues individual read commands to the DRAM and asserts the `clb2phy_rd_en` signal to the XIPHY to open the gate which allows the sample of the DQS to occur. The `clb2phy_rd_en` signal has control over the timing of the gate opening on a DRAM-clock-cycle resolution (DQS_GATE_READ_LATENCY_RANK#_BYTE#). This signal is controlled on a per-byte basis in the PHY and is set in the **`ddr_mc_pi`** block for use by both calibration and the controller.

Calibration is responsible for determining the value used on a per-byte basis for use by the controller. The XIPHY provides for additional granularity in the time to open the gate through coarse and fine taps. Coarse taps offer 90° DRAM clock-cycle granularity (16 available) and each fine tap provides a 2.5 to 15 ps granularity for each tap (512 available). BISC provides the number of taps for 1/4 of a memory clock cycle by taking (BISC_PQTR_NIBBLE#-BISC_ALIGN_PQTR_NIBBLE#) or (BISC_NQTR_NIBBLE#-BISC_ALIGN_NQTR_NIBBLE#). These are used to estimate the per-tap resolution for a given nibble.

The search for the DQS begins with an estimate of when the DQS is expected back. The total latency for the read is a function of the delay through the PHY, PCB delay, and the configured latency of the DRAM (CAS latency, Additive latency, etc.). The search starts three DRAM clock cycles before the expected return of the DQS. The algorithm must start sampling before the first rising edge of the DQS, preferably in the preamble region. DDR3 and DDR4 have different preambles for the DQS as shown in Figure 38-10.

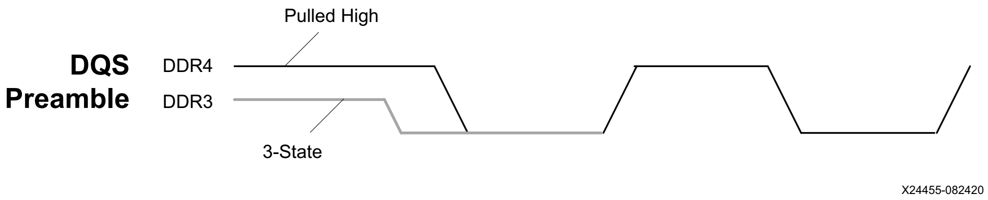


Figure 38-10: **DDR3/DDR4 DQS Preamble**

The specification for the DDR3 preamble is longer (3/4 of a DRAM clock cycle) and starts from the terminated 3-state while the DDR4 preamble is shorter (1/2 of a DRAM clock cycle) and starts from the rail terminated level. For DDR4, the preamble training mode is enabled during DQS gate calibration, so the DQS is driven low whenever the DQS is idle. This allows for the algorithm to look for the same sample pattern on the DQS for DDR3/DDR4 where the preamble is larger than half a clock cycle for both cases.

Given that DDR3 starts in the 3-state region before the burst, any accepted sample taken can either be a 0 or 1. To avoid this result, 20 samples (in hardware) are taken for each individual sample such that the probability of the 3-state region or noise in the sampling clock/strobe being mistaken for the actual DQS is low. This probability is given by the binomial probability shown in the binomial probability equation.

X = expected outcome

n= number of tries

P = probability of a single outcome

$$P(X = x) = \frac{n!}{x!(n-x)!}p^{x}(1-p)^{n-x}$$

When sampling in the 3-state region the result can be 0 or 1, so the probability of 20 samples all arriving at the same value is roughly 9.5 × 10^{-6}. Figure 38-11 shows an example of samples of a DQS burst with the expected sampling pattern to be found as the coarse taps are adjusted. The pattern is the expected level seen on the DQS over time as the sampling clock is adjusted in relation to the DQS.

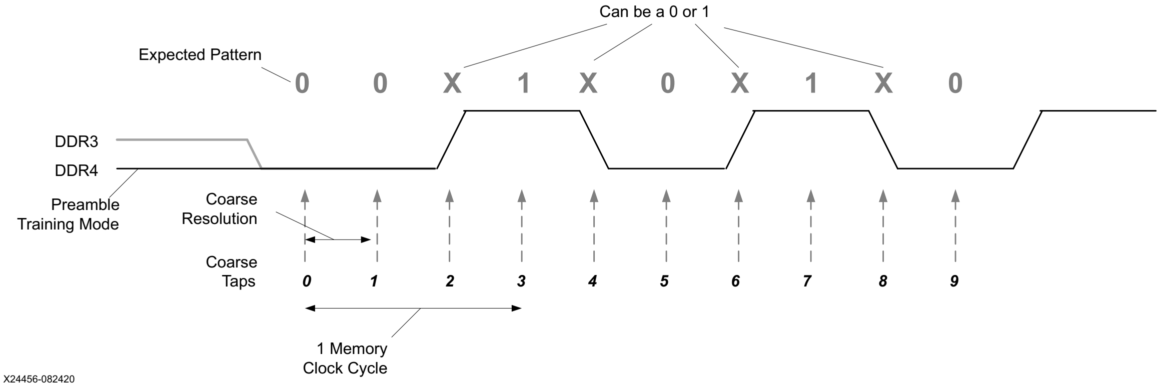


Figure 38-11: **Example DQS Gate Samples Using Coarse Taps**

Each individual element of the pattern is 20 read bursts from the DRAM and samples from the XIPHY. The gate in the XIPHY is opened and a new sample is taken to indicate the level seen on the DQS. If each of the samples matches with the first sample taken, the value is accepted. If all samples are not the same value that value is marked as "X" in the pattern. The "X" in the pattern shown is to allow for jitter and DCD between the clocks, and to deal with uncertainty when dealing with clocks with an unknown alignment. Depending on how the clocks line up they can resolve to all 0s, all 1s, or a mix of values, and yet the DQS pattern can still be found properly.

The coarse taps in the XIPHY are incremented and the value recorded at each individual coarse tap location, looking for the full pattern "00X1X0X1X0." For the algorithm to incorrectly calculate the 3-state region as the actual DQS pattern, you would have to take 20 samples of all 0s at a given coarse tap, another 20 samples of all 0s at another, then 20 coarse taps of all 1s for the initial pattern ("00X1"). The probability of this occurring is 8.67 × 10^{-19}. This also only covers the initial scan and does not include the full pattern which scans over 10 coarse taps.

While the probability is fairly low, there is a chance of coupling or noise being mistaken as a DQS pattern. In this case, each sample is no longer random but a signal that can be fairly repeatable. To guard against mistaking the 3-state region in DDR3 systems with the actual DQS pulse, an extra step is taken to read data from the MPR register to validate the gate alignment. The read path is set up by BISC for capture of data, placing the capture clock roughly in the middle of the expected bit time back from the DRAM.

Because the algorithm is looking for a set pattern and does not know the exact alignment of the DQS with the clock used for sampling the data, there are four possible patterns, as shown in Figure 38-12.

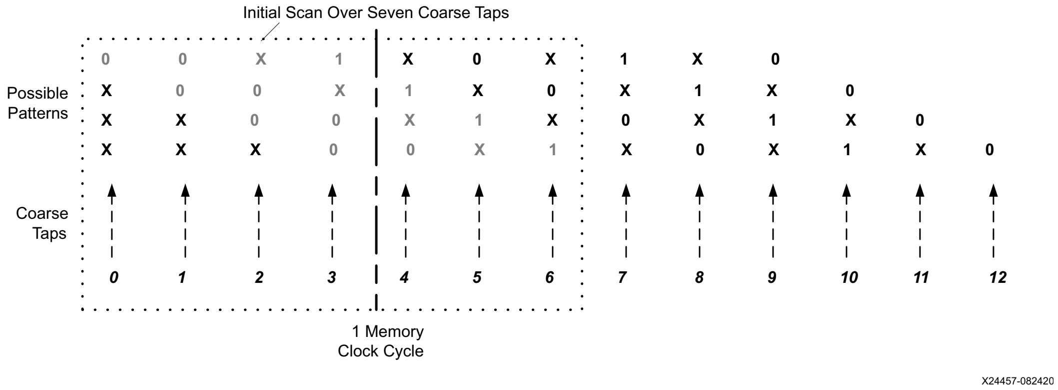


Figure 38-12: **DQS Gate Calibration Possible Patterns**

To speed up the pattern search, only the initial seven coarse taps are used to determine if the starting pattern is found. This eliminates the need to search additional coarse taps if the early samples do not match the expected result. If the result over the first coarse seven coarse taps is not one of the four shown in Figure 38-12, the following occurs:

- Coarse taps are reset to 0
- **`clb2phy_rd_en`** general interconnect control is adjusted to increase by one DRAM clock cycle
- Search starts again (this is the equivalent of starting at coarse tap four in Figure 38-12)

For DDR4, if the algorithm samples 1XX or 01X this means it started the sampling too late in relation to the DQS burst. The algorithm decreases the **`clb2phy_rd_en`** general interconnect control and try again. If the **`clb2phy_rd_en`** is at the low limit already it issues an error.

If all allowable values of **`clb2phy_rd_en`** for a given latency are checked and the expected pattern is still not found, the search begins again from the start but this time the sampling is offset by an estimated 45° using fine taps (half a coarse tap). This allows the sampling to occur at a different phase than the initial relationship. Each time through if the pattern is not found, the offset is reduced by half until all offset values have been exhausted.

Figure 38-13 shows an extreme case of DCD on the DQS that would result in the pattern not being found until an offset being applied using fine taps.

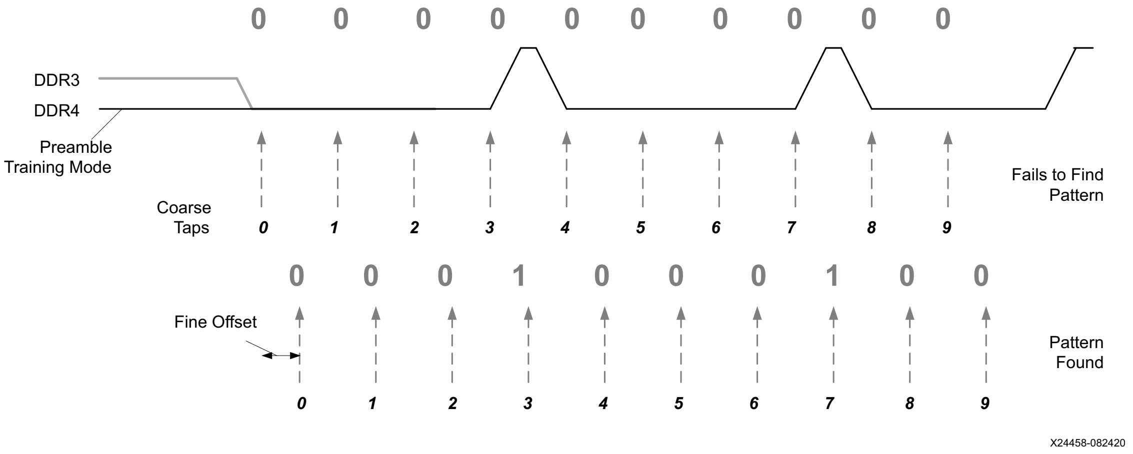


Figure 38-13: **DQS Gate Calibration Fine Offset Example**

After the pattern has been found, the final coarse tap (DQS_GATE_COARSE_RANK#_BYTE#) is set based on the alignment of the pattern previously checked (shown in Figure 38-12). The coarse tap is set to be the last 0 seen before the 1 (3 is used to indicate an unstable region, where multiple samples return 0 and 1) was found in the pattern shown in Figure 38-14. During this step, the final value of the coarse tap is set between 3 to 6. If the coarse value of 7 to 9 is chosen, the coarse taps are decremented by 4 and the general interconnect read latency is incremented by 1, so the value falls in the 3 to 5 range instead.

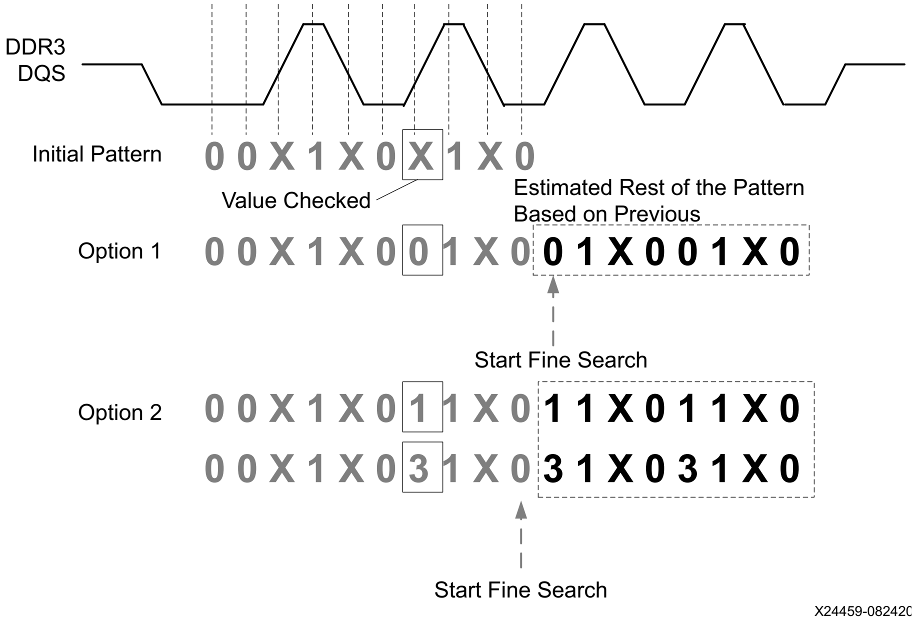


Figure 38-14: **DQS Gate Coarse Setting Before Fine Search**

From this point the `clb2phy_rd_en` (DQS_GATE_READ_LATENCY_RANK#_BYTE#) is increased by 1 to position the gate in the final location before the start of the fine sweep. This is done to ensure the proper timing of the gate in relation to the full DQS burst during normal operation. Because this is sampling the strobe with another signal it can have jitter in relation to one another.

For example, when they are lined up taking multiple samples it might give you a different result each time as a new sample is taken. The fine search begins in an area where all samples returned a 0 so it is relatively stable, as shown in Figure 38-15. The fine taps are incremented until a non-zero value is returned (which indicates the left edge of the unstable region) and that value recorded as shown in Figure 38-17 (DQS_GATE_FINE_LEFT_RANK#_BYTE#).

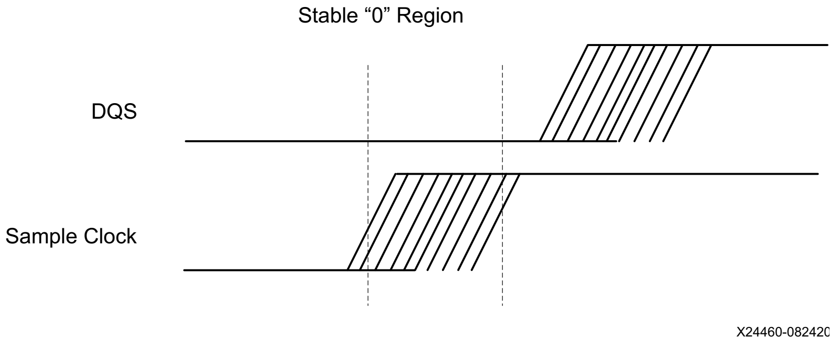


Figure 38-15: **DQS Gate Fine Adjustment, Sample a 0**

The fine taps are then incremented until all samples taken return a 1, as shown in Figure 38-16. This is recorded as the right edge of the uncertain region as shown in Figure 38-17 (DQS_GATE_FINE_RIGHT_RANK#_BYTE#).

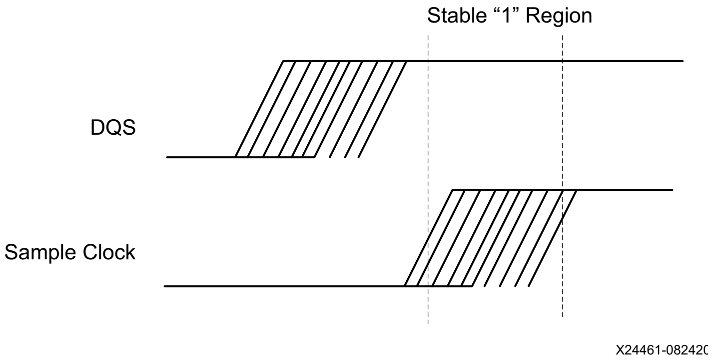


Figure 38-16: **DQS Gate Fine Adjustment, Sample a 1**

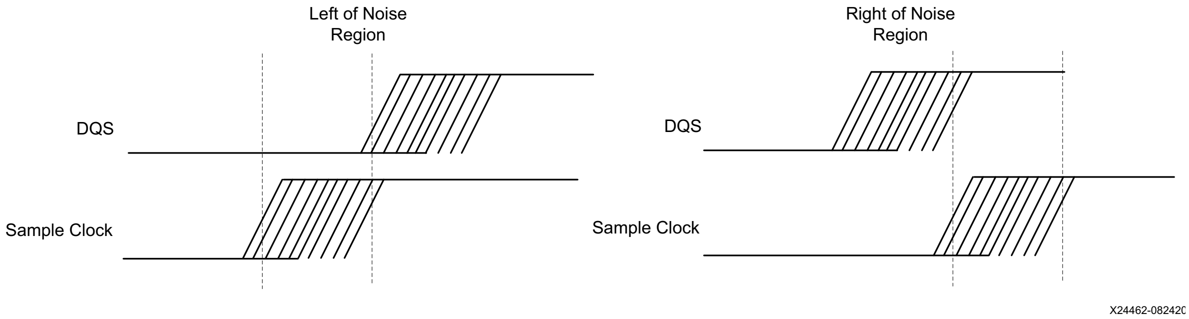


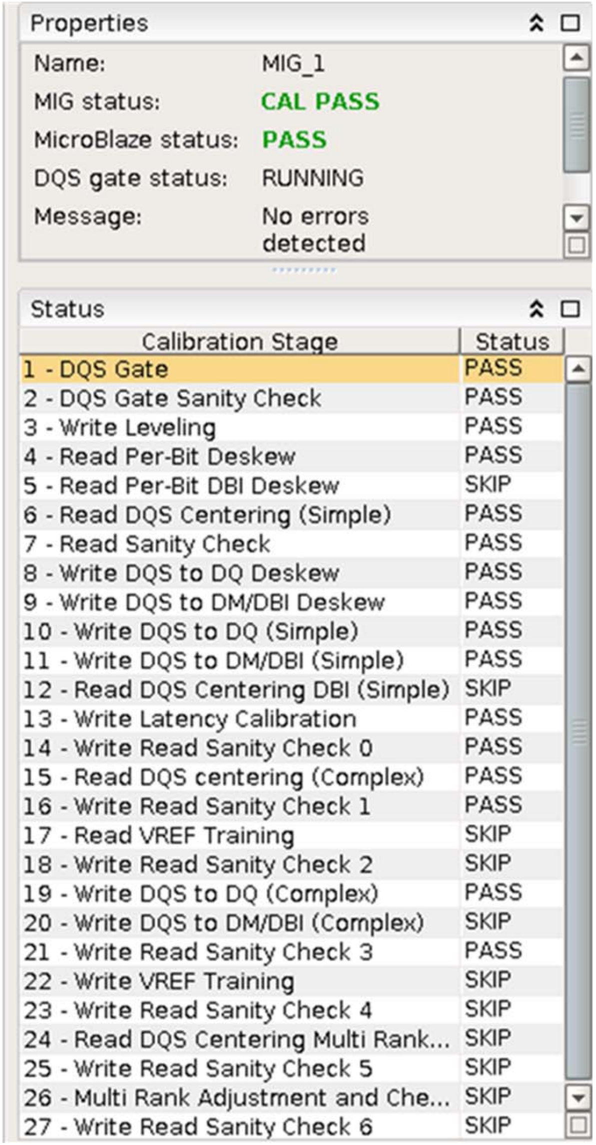
Figure 38-17: **DQS Gate Fine Adjustment, Uncertain Region**

The final fine tap is computed as the midpoint of the uncertain region, (right – left)/2 + left (DQS_GATE_FINE_CENTER_RANK#_BYTE#). This ensures optimal placement of the gate in relation to the DQS. For simulation, speeding up a faster search is implemented for the fine tap adjustment. This is performed by using a binary search to jump the fine taps by larger values to quickly find the 0 to 1 transition.

For multi-rank systems, separate control exists in the XIPHY for each rank and every rank can be trained separately for coarse and fine taps. After calibration is complete, adjustments are made so that for each byte, the **`clb2phy_rd_en`** (DQS_GATE_READ_LATENCY_RANK#_BYTE#) value for a given byte matches across all ranks. The coarse taps are incremented/decremented accordingly to adjust the timing of the gate signal to match the timing found in calibration. If a common **`clb2phy_rd_en`** setting cannot be found for a given byte across all ranks, an error is asserted.

Debug

To determine the status of DQS Gate Calibration, click the **DQS_GATE** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in the **Memory IP Properties** identifies how the stage failed, or notes if it passed successfully.



Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	PASS
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Che...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-18: **Memory IP XSDB Debug GUI Example – DQS Gate**

The status of DQS Gate can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-9. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-9: **DDR_CAL_ERROR Decode for DQS Preamble Detection Calibration**

DQS Gate Code	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	Byte	Logical Nibble	Based on the calculated latency from the MR register, back off and start sampling. If the sample occurs too late in the DQS burst and it cannot decrease the latency, then issue an error.	Check the PCB routing guidelines against the routing on the PCB being tested. Measure the Chip Select and the returning DQS and check if the time of the returning DQS matches the expected CAS latency. Check the levels on the DQS signal itself.
0x2	Byte	Logical Nibble	Expected Pattern not found on GT_STATUS.	Check the DQS_GATE_PATTERN_* stored in XSDB. This stores what the DQS pattern found around the expected CAS latency. More generic version of error 0x4/0x5 where not all samples found matched. Probe the DQS when a read command occurs and look at the signal levels of the P/N pair. Check the VRP resistor value.
0x3	Byte	Logical Nibble	CAS latency is too low. Calibration starts at a CAS latency (CL) minus 3; For allowable CAS latencies, see Table 4-75, page 173.	Check CAS latency parameter in the XSDB MR fields against what is allowed in Table 4-75, page 173.
0x4	Byte	Logical Nibble	Pattern not found on GT_STATUS, all samples were 0. Expecting to sample the preamble.	Check power and pinout on the PCB/ Design. This is the error found when the DRAM does not respond to the Read command. Probe if the read DQS is generated when a read command is sent out.
0x5	Byte	Logical Nibble	Pattern not found on GT_STATUS, all samples were 1. Expecting to sample the preamble.	Check power and pinout on the PCB/ Design. This is the error found when the DRAM does not respond to the Read command. Probe if the read DQS is generated when a read command is sent out.

Table 38-9: **DDR_CAL_ERROR Decode for DQS Preamble Detection Calibration *(Cont'd)***

DQS Gate Code	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x6	Byte	Logical Nibble	Could not find the 0->1 transition with fine taps in at least ½ tck (estimated) of fine taps.	Check the BISC values in XSDB (for the nibbles associated with the DQS) to determine the 90° offset value in taps. Check if any warnings are generated, look if any are 0x13 or 0x014. For DDR3, BISC must be run and a data check is used to confirm the DQS gate settings, but if the data is wrong the algorithm keeps searching and could end up in this failure. Check data connections, vrp settings, V_{REF} resistor in the PCB (or if internal V_{REF} set properly for all bytes).
0x7	Byte	Logical Nibble	Underflow of coarse taps when trying to limit maximum coarse tap setting.	Check calibrated coarse tap (DQS_GATE_COARSE_RANK*_BYTE*) setting for failing DQS to be sure the value is in the range of 1–6.
0x8	Byte	Logical Nibble	Violation of maximum read latency limit.	Check DQS and CK trace lengths. Ensure the maximum trace length is not violated. For debug purposes, try a lower frequency where more search range is available and check if the stage is successful.

Table 38-10 shows the signals and values adjusted or used during the DQS Preamble Detection stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the Memory IP core properties in the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-10: **Additional XSDB Signals of Interest during DQS Preamble Detection**

Signal	Usage	Signal Description
DQS_GATE_COARSE_RANK*_BYTE*	One value per rank and DQS group	Final RL_DLY_COARSE tap value.
DQS_GATE_FINE_CENTER_RANK*_BYTE*	One value per rank and DQS group	Final RL_DLY_FINE tap value. This is adjusted during alignment of sample clock to DQS.
DQS_GATE_FINE_LEFT_RANK*_BYTE*	One value per rank and DQS group	RL_DLY_FINE tap value when left edge was detected.
DQS_GATE_FINE_RIGHT_RANK*_BYTE*	One value per rank and DQS group	RL_DLY_FINE tap value when right edge was detected.

Table 38-10: **Additional XSDB Signals of Interest during DQS Preamble Detection *(Cont'd)***

Signal	Usage	Signal Description
DQS_GATE_PATTERN_0/1/2_RANK*_BYTE*	One value per rank and DQS group	The DQS pattern detected during DQS preamble detection. When a DQS Preamble Detection error occurs where the pattern is not found (DDR_CAL_ERROR code 0x0, 0x2, 0x4, or 0x5), the pattern seen during CL+1 is saved here. The full pattern could be up to 13 bits. The first nine bits are stored on _0. Overflow bits are stored on _1. Currently, _2 is reserved. For example, 9'b0_1100_1100 9'b1_1001_1000 9'b1_0011_0000 9'b0_0110_0000 Examples shown here are not comprehensive, as the expected pattern looks like: 10'b0**X**1**X**0**X**1**X**00 Where **X** above can be a 0 or 1. The LSB within this signals is the pattern detected when Coarse = 0, the next bit is the pattern detected when Coarse = 1, etc. Additionally, there can be up to three padded zeros before start of the pattern. In some cases, extra information of interest is stored in the overflow register. The full pattern stored can be: 13'b0_0110_1100_0000 So the pattern is broken up and stored in two locations: 9'b0_0110_0000 <- PATTERN_0 9'b0_0001_0011 <- PATTERN_1
DQS_GATE_READ_LATENCY_RANK*_BYTE*	One value per rank and DQS group	Read Latency value last used during DQS Preamble Detection. The Read Latency field is limited to CAS latency -3 to CAS latency + 7. If the DQS is toggling yet was not found check the latency of the DQS signal coming back in relation to the chip select.
BISC_ALIGN_PQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_ALIGN_NQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.

Table 38-10: **Additional XSDB Signals of Interest during DQS Preamble Detection *(Cont'd)***

Signal	Usage	Signal Description
BISC_PQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_PQTR – BISC_ALIGN_PQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_PQTR – BISC_ALIGN_PQTR). Useful for error code 0x6.
BISC_NQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_NQTR – BISC_ALIGN_NQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_NQTR – BISC_ALIGN_NQTR). Useful for error code 0x6.

This is a sample of the results for the DQS Preamble Detection XSDB debug signals:

```
DQS_GATE_COARSE_RANK0_BYTE0                string true true 007
DQS_GATE_COARSE_RANK0_BYTE1                string true true 006
DQS_GATE_COARSE_RANK0_BYTE2                string true true 007
DQS_GATE_COARSE_RANK0_BYTE3                string true true 007
DQS_GATE_COARSE_RANK0_BYTE4                string true true 008
DQS_GATE_COARSE_RANK0_BYTE5                string true true 008
DQS_GATE_COARSE_RANK0_BYTE6                string true true 008
DQS_GATE_COARSE_RANK0_BYTE7                string true true 008
DQS_GATE_COARSE_RANK0_BYTE8                string true true 008
DQS_GATE_FINE_CENTER_RANK0_BYTE0           string true true 005
DQS_GATE_FINE_CENTER_RANK0_BYTE1           string true true 02b
DQS_GATE_FINE_CENTER_RANK0_BYTE2           string true true 024
DQS_GATE_FINE_CENTER_RANK0_BYTE3           string true true 019
DQS_GATE_FINE_CENTER_RANK0_BYTE4           string true true 022
DQS_GATE_FINE_CENTER_RANK0_BYTE5           string true true 021
DQS_GATE_FINE_CENTER_RANK0_BYTE6           string true true 011
DQS_GATE_FINE_CENTER_RANK0_BYTE7           string true true 008
DQS_GATE_FINE_CENTER_RANK0_BYTE8           string true true 000
DQS_GATE_FINE_LEFT_RANK0_BYTE0             string true true 002
DQS_GATE_FINE_LEFT_RANK0_BYTE1             string true true 028
DQS_GATE_FINE_LEFT_RANK0_BYTE2             string true true 021
DQS_GATE_FINE_LEFT_RANK0_BYTE3             string true true 015
DQS_GATE_FINE_LEFT_RANK0_BYTE4             string true true 020
DQS_GATE_FINE_LEFT_RANK0_BYTE5             string true true 01f
DQS_GATE_FINE_LEFT_RANK0_BYTE6             string true true 00f
DQS_GATE_FINE_LEFT_RANK0_BYTE7             string true true 006
DQS_GATE_FINE_LEFT_RANK0_BYTE8             string true true 000
DQS_GATE_FINE_RIGHT_RANK0_BYTE0            string true true 008
DQS_GATE_FINE_RIGHT_RANK0_BYTE1            string true true 02f
DQS_GATE_FINE_RIGHT_RANK0_BYTE2            string true true 028
DQS_GATE_FINE_RIGHT_RANK0_BYTE3            string true true 01e
DQS_GATE_FINE_RIGHT_RANK0_BYTE4            string true true 025
DQS_GATE_FINE_RIGHT_RANK0_BYTE5            string true true 024
DQS_GATE_FINE_RIGHT_RANK0_BYTE6            string true true 014
DQS_GATE_FINE_RIGHT_RANK0_BYTE7            string true true 00b
DQS_GATE_FINE_RIGHT_RANK0_BYTE8            string true true 001
```

```
DQS_GATE_PATTERN_0_RANK0_BYTE0                 string true true 130
DQS_GATE_PATTERN_0_RANK0_BYTE1                 string true true 198
DQS_GATE_PATTERN_0_RANK0_BYTE2                 string true true 130
DQS_GATE_PATTERN_0_RANK0_BYTE3                 string true true 130
DQS_GATE_PATTERN_0_RANK0_BYTE4                 string true true 060
DQS_GATE_PATTERN_0_RANK0_BYTE5                 string true true 060
DQS_GATE_PATTERN_0_RANK0_BYTE6                 string true true 060
DQS_GATE_PATTERN_0_RANK0_BYTE7                 string true true 060
DQS_GATE_PATTERN_0_RANK0_BYTE8                 string true true 060
DQS_GATE_PATTERN_1_RANK0_BYTE0                 string true true 001
DQS_GATE_PATTERN_1_RANK0_BYTE1                 string true true 001
DQS_GATE_PATTERN_1_RANK0_BYTE2                 string true true 001
DQS_GATE_PATTERN_1_RANK0_BYTE3                 string true true 001
DQS_GATE_PATTERN_1_RANK0_BYTE4                 string true true 003
DQS_GATE_PATTERN_1_RANK0_BYTE5                 string true true 003
DQS_GATE_PATTERN_1_RANK0_BYTE6                 string true true 003
DQS_GATE_PATTERN_1_RANK0_BYTE7                 string true true 003
DQS_GATE_PATTERN_1_RANK0_BYTE8                 string true true 003
DQS_GATE_PATTERN_2_RANK0_BYTE0                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE1                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE2                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE3                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE4                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE5                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE6                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE7                 string true true 000
DQS_GATE_PATTERN_2_RANK0_BYTE8                 string true true 000
DQS_GATE_READ_LATENCY_RANK0_BYTE0              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE1              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE2              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE3              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE4              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE5              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE6              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE7              string true true 010
DQS_GATE_READ_LATENCY_RANK0_BYTE8              string true true 010
BISC_ALIGN_NQTR_NIBBLE0                        string true true 000
BISC_ALIGN_NQTR_NIBBLE1                        string true true 000
BISC_ALIGN_NQTR_NIBBLE2                        string true true 000
BISC_ALIGN_NQTR_NIBBLE3                        string true true 000
BISC_ALIGN_NQTR_NIBBLE4                        string true true 000
BISC_ALIGN_NQTR_NIBBLE5                        string true true 000
BISC_ALIGN_NQTR_NIBBLE6                        string true true 000
BISC_ALIGN_NQTR_NIBBLE7                        string true true 000
BISC_ALIGN_NQTR_NIBBLE8                        string true true 000
BISC_ALIGN_NQTR_NIBBLE9                        string true true 000
BISC_ALIGN_NQTR_NIBBLE10                       string true true 000
BISC_ALIGN_NQTR_NIBBLE11                       string true true 000
BISC_ALIGN_NQTR_NIBBLE12                       string true true 000
BISC_ALIGN_NQTR_NIBBLE13                       string true true 000
BISC_ALIGN_NQTR_NIBBLE14                       string true true 000
BISC_ALIGN_NQTR_NIBBLE15                       string true true 000
BISC_ALIGN_NQTR_NIBBLE16                       string true true 000
BISC_ALIGN_NQTR_NIBBLE17                       string true true 000
BISC_ALIGN_PQTR_NIBBLE0                        string true true 004
BISC_ALIGN_PQTR_NIBBLE1                        string true true 006
BISC_ALIGN_PQTR_NIBBLE2                        string true true 005
BISC_ALIGN_PQTR_NIBBLE3                        string true true 005
BISC_ALIGN_PQTR_NIBBLE4                        string true true 004
```

```
BISC_ALIGN_PQTR_NIBBLE5                       string true true 006
BISC_ALIGN_PQTR_NIBBLE6                       string true true 003
BISC_ALIGN_PQTR_NIBBLE7                       string true true 004
BISC_ALIGN_PQTR_NIBBLE8                       string true true 007
BISC_ALIGN_PQTR_NIBBLE9                       string true true 006
BISC_ALIGN_PQTR_NIBBLE10                      string true true 003
BISC_ALIGN_PQTR_NIBBLE11                      string true true 006
BISC_ALIGN_PQTR_NIBBLE12                      string true true 004
BISC_ALIGN_PQTR_NIBBLE13                      string true true 004
BISC_ALIGN_PQTR_NIBBLE14                      string true true 004
BISC_ALIGN_PQTR_NIBBLE15                      string true true 006
BISC_ALIGN_PQTR_NIBBLE16                      string true true 004
BISC_ALIGN_PQTR_NIBBLE17                      string true true 007
BISC_NQTR_NIBBLE0                             string true true 030
BISC_NQTR_NIBBLE1                             string true true 02f
BISC_NQTR_NIBBLE2                             string true true 031
BISC_NQTR_NIBBLE3                             string true true 031
BISC_NQTR_NIBBLE4                             string true true 02e
BISC_NQTR_NIBBLE5                             string true true 030
BISC_NQTR_NIBBLE6                             string true true 02f
BISC_NQTR_NIBBLE7                             string true true 031
BISC_NQTR_NIBBLE8                             string true true 030
BISC_NQTR_NIBBLE9                             string true true 031
BISC_NQTR_NIBBLE10                            string true true 02f
BISC_NQTR_NIBBLE11                            string true true 030
BISC_NQTR_NIBBLE12                            string true true 02f
BISC_NQTR_NIBBLE13                            string true true 032
BISC_NQTR_NIBBLE14                            string true true 031
BISC_NQTR_NIBBLE15                            string true true 031
BISC_NQTR_NIBBLE16                            string true true 031
BISC_NQTR_NIBBLE17                            string true true 031
BISC_PQTR_NIBBLE0                             string true true 030
BISC_PQTR_NIBBLE1                             string true true 032
BISC_PQTR_NIBBLE2                             string true true 031
BISC_PQTR_NIBBLE3                             string true true 032
BISC_PQTR_NIBBLE4                             string true true 030
BISC_PQTR_NIBBLE5                             string true true 030
BISC_PQTR_NIBBLE6                             string true true 02e
BISC_PQTR_NIBBLE7                             string true true 02f
BISC_PQTR_NIBBLE8                             string true true 033
BISC_PQTR_NIBBLE9                             string true true 033
BISC_PQTR_NIBBLE10                            string true true 030
BISC_PQTR_NIBBLE11                            string true true 034
BISC_PQTR_NIBBLE12                            string true true 030
BISC_PQTR_NIBBLE13                            string true true 030
BISC_PQTR_NIBBLE14                            string true true 030
BISC_PQTR_NIBBLE15                            string true true 031
BISC_PQTR_NIBBLE16                            string true true 031
BISC_PQTR_NIBBLE17                            string true true 033
```

Expected Results

Table 38-11 provides expected results for the coarse, fine, and read latency parameters during DQS Preamble Detection. These values can be compared to the results found in hardware testing.

Table 38-11: **Expected Results for DQS Preamble Detection Coarse/Fine Tap and RL**

Parameter	Description
DQS_GATE_COARSE_RANK*_BYTE*	Final RL_DLY_COARSE tap value. Expected values 3-6 only.
DQS_GATE_FINE_CENTER_RANK*_BYTE*	Final RL_DLY_FINE tap value. Expected value should be less than 90 degrees (use BISC values to estimate the 90° value) and between DQS_GATE_FINE_LEFT and DQS_GATE_FINE_RIGHT.
DQS_GATE_READ_LATENCY_RANK*_BYTE*	Read Latency value last used during DQS Preamble Detection. Expected value is dependent on the PCB trace length but should be in the range CL-2 to CL+4.

Hardware Measurements

This is the first stage of calibration. Therefore, any general setup issue can result in a failure during DQS Preamble Detection Calibration. The first items to verify are proper clocking and reset setup as well as usage of unmodified Memory IP RTL that is generated specifically for the SDRAM(s) in hardware. The General Checks, page 588 section should be verified when a failure occurs during DQS Preamble Detection.

After the General Checks, page 588 have been verified, hardware measurements on DQS, and specifically the DQS byte that fails during DQS Preamble Detection, should be captured and analyzed. DQS must be toggling during DQS Preamble Detection. If this stage fails, after failure, probe the failing DQS at the FPGA using a high quality scope and probes. When a failure occurs, the calibration goes into an error loop routine, continually issuing read commands to the DRAM to allow for probing of the PCB. While probing DQS, validate:

1. Continuous DQS pulses exist with gaps between each BL8 read.
2. The signal integrity of DQS:
   - Ensure V_{IL} and V_{IH} are met for the specific I/O Standard in use. For more information, see the *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2].
   - Look for 50% duty cycle periods.
   - Ensure that the signals have low jitter/noise that can result from any power supply or board noise.

If DQS pulses are not present and the General Checks, page 588 have been verified, probe the read commands at the SDAM and verify:

1. The appropriate read commands exist – CS# = 0, RAS# = 1, CAS# = 0, WE# = 1.
2. The signal integrity of each command signal is valid.
   - Ensure V_{IL} and V_{IH} are met. For more information, see the JESD79-3F, *DDR3 SDRAM Standard* and JESD79-4, *DDR4 SDRAM Standard*, JEDEC Solid State Technology Association [Ref 1].

3. CK to command timing.
4. RESET# voltage level.
5. Memory initialization routine.

Debugging Write Leveling Calibration Failures

The DDR3/DDR4 SDRAM memory modules use a fly-by topology on clocks, address, commands, and control signals to improve signal integrity. This topology causes a skew between DQS and CK at each memory device on the module. Write leveling is a feature in DDR3/DDR4 SDRAMs that allows the controller to adjust each write DQS phase independently with respect to the clock (CK) forwarded to the DDR3/DDR4 device to compensate for this skew and meet the tDQSS specification [Ref 1].

During write leveling, DQS is driven by the FPGA memory interface and DQ is driven by the DDR3/DDR4 SDRAM device to provide feedback. To start write leveling, an MRS command is sent to the DRAM to enable the feedback feature, while another MRS command is sent to disable write leveling at the end. Figure 38-19 shows the block diagram for the write leveling implementation.

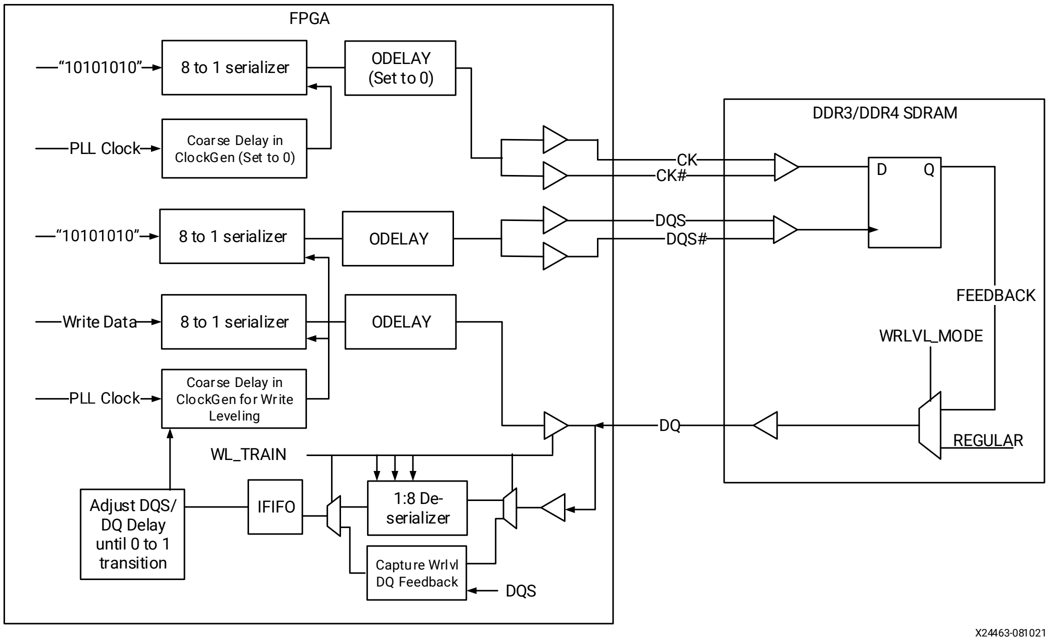


Figure 38-19: **Write Leveling Block Diagram**

The XIPHY is set up for write leveling by setting various attributes in the RIU. WL_TRAIN is set to decouple the DQS and DQ when driving out the DQS. This allows the XIPHY to capture the returning DQ from the DRAM. Because the DQ is returned without the returning DQS strobe for capture, the RX_GATE is set to 0 in the XIPHY to disable DQS gate operation. While the write leveling algorithm acts on a single DQS at a time, all the XIPHY bytes are set up for write leveling to ensure there is no contention on the bus for the DQ.

DQS is delayed with ODELAY and coarse delay (WL_DLY_CRSE[12:9] applies to all bits in a nibble) provided in the RIU WL_DLY_RNKx register. The WL_DLY_FINE[8:0] location in the RIU is used to store the ODELAY value for write leveling for a given nibble (used by the XIPHY when switching ranks).

A DQS train of pulses is output by the FPGA to the DRAM to detect the relationship of CK and DQS at the DDR3/DDR4 memory device. DQS is delayed using the ODELAY and coarse taps in unit tap increments until a 0 to 1 transition is detected on the feedback DQ input. A single typical burst length of eight pattern is first put out on the DQS (four clock pulses), followed by a gap, and then 100 bursts length of eight patterns are sent to the DRAM (Figure 38-20).

The first part is to ensure the DRAM updates the feedback sample on the DQ being sent back, while the second provides a clock that is used by the XIPHY to clock into the XIPHY the level seen on the DQ. Sampling the DQ while driving the DQS helps to avoid ringing on the DQS at the end of a burst that can be mistaken as a clock edge by the DRAM.

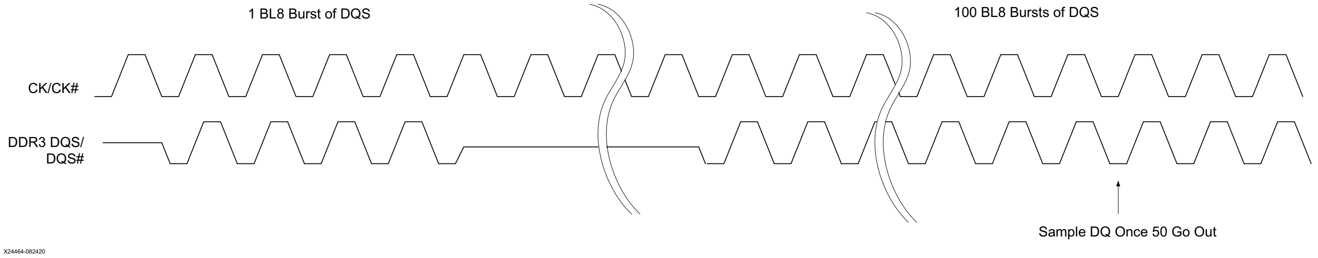


Figure 38-20: **Write Leveling DQS Bursts**

To avoid false edge detection around the CK negative edge due to jitter, the DQS delays the entire window to find the large stable 0 and 1 region (Stable 0 or 1 indicates all samples taken return the same value). Check that you are to the left of this stable 1 region as the right side of this region is the CK negative edge being captured with the DQS, as shown in Figure 38-21.

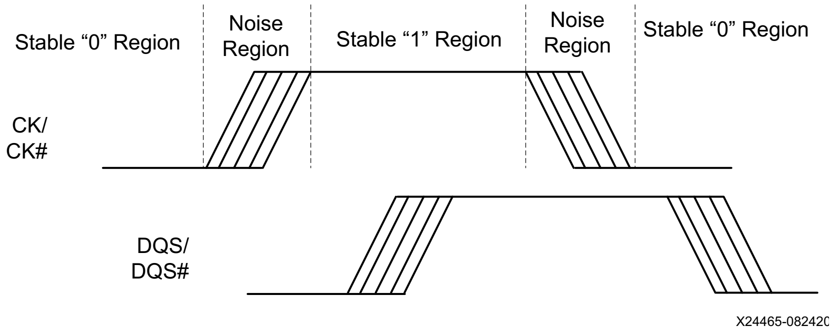


Figure 38-21: **Write Leveling Regions**

Write leveling is performed in the following two steps:

1. Find the transition from 0 to 1 using coarse taps and ODELAY taps (if needed).

   During the first step, look for a static 0 to be returned from all samples taken. This means 64 samples were taken and it is certain the data is a 0. Record the coarse tap setting and keep incrementing the coarse tap.

   - If the algorithm receives another stable 0 update the setting (WRLVL_COARSE_STABLE0_RANK_BYTE) and continue.
   - If the algorithm receives a non-zero result (noise) or a stable 1 reading (WRLVL_COARSE_STABLE0_RANK_BYTE), the search has gone too far and the delay is backed up to the last coarse setting that gave a stable 0. This reference allows you to know the algorithm placed the coarse taps to the left of the transition desired.
   - If the algorithm never sees a transition from a stable 0 to the noise or stable 1 using the coarse taps, the ODELAY of the DQS is set to an offset value (first set at 45°, WRLVL_ODELAY_INITIAL_OFFSET_BYTE) and the coarse taps are checked again from 0. Check for the stable 0 to stable 1 transition (the algorithm might need to perform this if the noise region is close to 90° or there is a large amount of DCD).
   - If the transition is still not found, the offset is halved and the algorithm tries again. The final offset value used is stored at WRLVL_ODELAY_LAST_OFFSET_RANK_BYTE. Because the algorithm is aligning the DQS with the nearest clock edge the coarse tap sweep is limited to five, which is 1.25 clock cycles. The final coarse setting is stored at WRLVL_COARSE_STABLE0_RANK_BYTE.

2. Find the center of the noise region around that transition from 0 to 1 using ODELAY taps.

   The second step is to sweep with ODELAY taps and find both edges of the noise region (WRLVL_ODELAY_STABLE0_RANK_BYTE, WRLVL_ODELAY_STABLE1_RANK_BYTE while WRLVL_ODELAY_CENTER_RANK_BYTE holds the final value). The number of ODELAY taps used is determined by the initial alignment of the DQS and CK and the size of this noise region as shown in Figure 38-22.

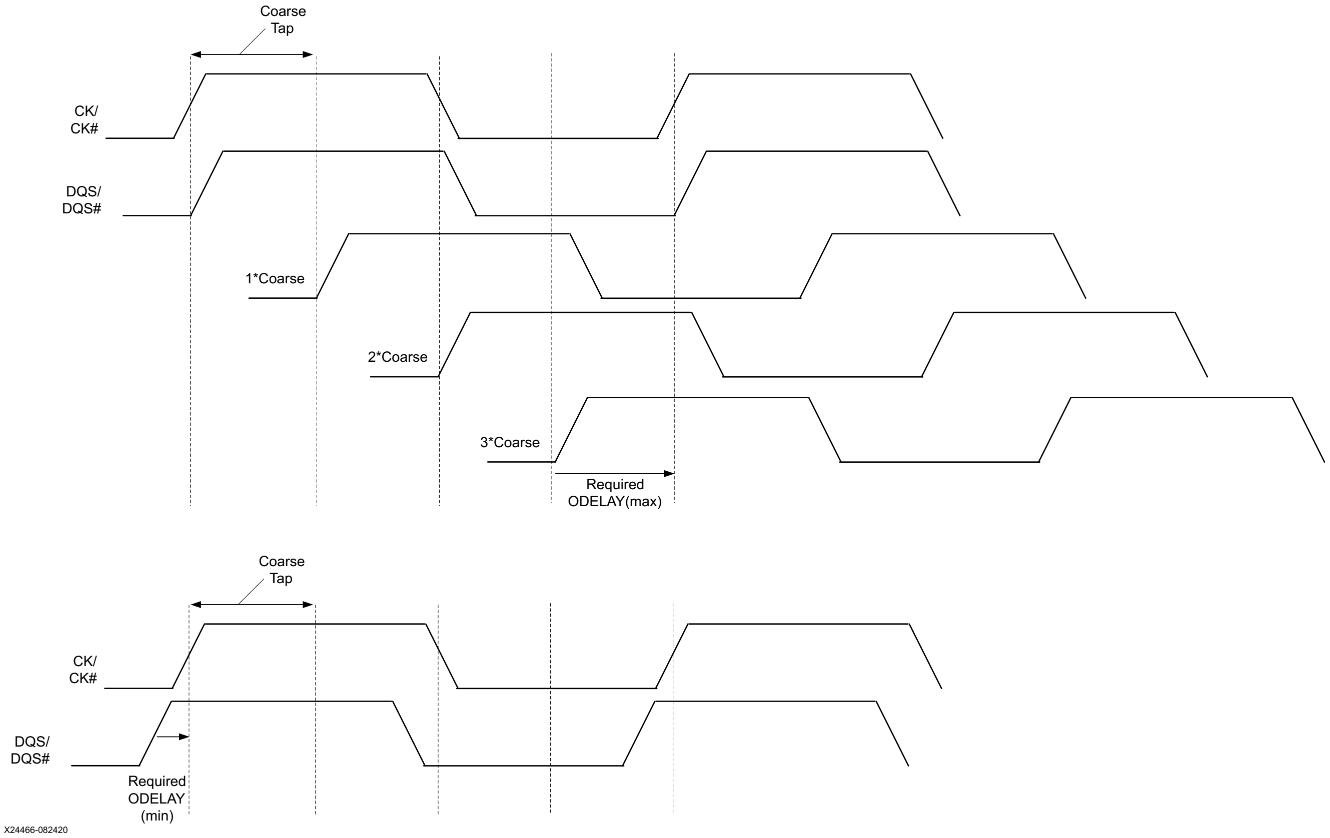


Figure 38-22: **Worst Case ODELAY Taps (Maximum and Minimum)**

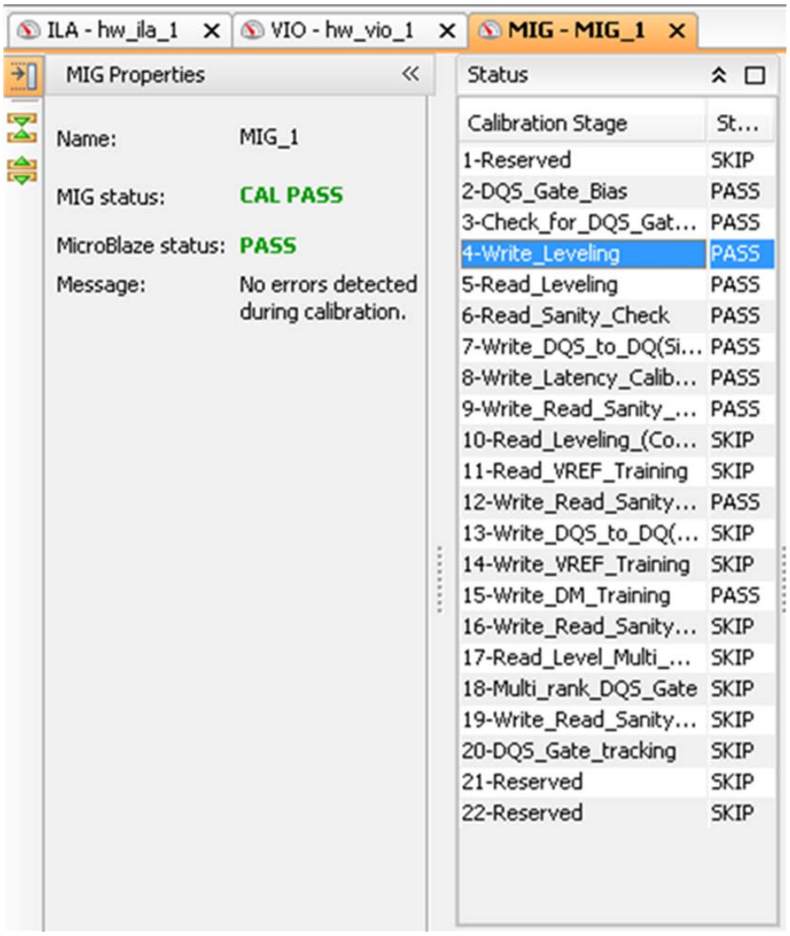
After the final ODELAY setting is found, the value of ODELAY is loaded in the RIU in the WL_DLY_RNKx[8:0] register. This value is also loaded in the ODELAY register for the DQ and the DM to match the DQS. If any deskew has been performed on the DQS/DQ/DM when reaching this point (multi-rank systems), the deskew information is preserved and the offset is applied.

The lowest ODELAY value is stored at WRLVL_ODELAY_LOWEST_COMMON_BYTE, which is used to preserve the WRLVL element with the deskew portion of ODELAY for a given byte. During normal operation in a multi-rank system, the XIPHY is responsible for loading the ODELAY with the value stored for a given rank.

After write leveling, the MPR command is sent to the DRAM to disable the write leveling feature, the WL_TRAIN is set back to the default OFF setting, and the DQS gate is turned back on to allow for capture of the DQ with the returning strobe DQS.

Debug

To determine the status of Write Leveling Calibration, click the **Write Leveling** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Name: MIG_1

MIG status: CAL PASS

MicroBlaze status: PASS

Message: No errors detected during calibration.

Calibration Stage	St...
1-Reserved	SKIP
2-DQS_Gate_Bias	PASS
3-Check_for_DQS_Gat...	PASS
4-Write_Leveling	PASS
5-Read_Leveling	PASS
6-Read_Sanity_Check	PASS
7-Write_DQS_to_DQ(Si...	PASS
8-Write_Latency_Calib...	PASS
9-Write_Read_Sanity_...	PASS
10-Read_Leveling_(Co...	SKIP
11-Read_VREF_Training	SKIP
12-Write_Read_Sanity...	PASS
13-Write_DQS_to_DQ(...	SKIP
14-Write_VREF_Training	SKIP
15-Write_DM_Training	PASS
16-Write_Read_Sanity...	SKIP
17-Read_Level_Multi_...	SKIP
18-Multi_rank_DQS_Gate	SKIP
19-Write_Read_Sanity...	SKIP
20-DQS_Gate_tracking	SKIP
21-Reserved	SKIP
22-Reserved	SKIP

Figure 38-23: **Memory IP XSDB Debug GUI Example – Write Leveling**

The status of Write Leveling can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-12. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-12: **DDR_CAL_ERROR Decode for Write Leveling Calibration**

Write Leveling Code	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	Byte	N/A	Cannot find stable 0 region	For failures on the second rank of a multi-rank DIMM, check if the DIMM uses mirroring and make sure the design generated matches what the DIMM expects. Check the pinout and connections of the address/control bus, specifically A7 which is used to power on the write leveling mode in the DRAM.
0x2	Byte	N/A	Cannot find stable 1 region	Check XSDB BUS_DATA_BURST fields to see what the data looked like. Check if a single BIT is stuck at a certain value. If possible, add an ILA to look at the dbg_rd_data to check multiple bursts of data.
0x3	Byte	N/A	Cannot find the left edge of noise region with fine taps	Check the BISC values in XSDB (for the nibbles associated with the DQS) to determine the 90° offset value in taps.
0x4	Byte	N/A	Could not find the 0->1 transition with ODELAY taps in at least 1 tck (estimated) of ODELAY taps	Check the BISC values in XSDB (for the nibbles associated with the DQS) to determine the 90° offset value in taps.

Table 38-13 describes the signals and values adjusted or used during the Write Leveling stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within Hardware Manager or by executing the Tcl commands noted in the XSDB Debug.

Table 38-13: **Signals of Interest for Write Leveling Calibration**

Signal	Usage	Signal Description
WRLVL_COARSE_STABLE0 _RANK*_BYTE*	One per rank per Byte	WRLVL course tap setting to find Stable 0.
WRLVL_COARSE_STABLE1 _RANK*_BYTE*	One per rank per Byte	WRLVL coarse tap setting to find Stable 1 or noise.
WRLVL_ODELAY_INITIAL_OFFSET_BYTE*	One per Byte	ODELAY Offset used during Write Leveling. Used to estimate number of ODELAY taps to equal one coarse tap, for offsetting alignment during algorithm.
WRLVL_ODELAY_STABLE0_RANK*_BYTE*	One per rank per Byte	Left side of noise region when edge aligned (or last stable 0 received) before getting noisy data or stable 1.
WRLVL_ODLEAY_STABLE1_ RANK*_BYTE*	One per rank per Byte	Right side of noise region when edge aligned (or first stable 1 received) after getting noisy data or stable 0.

Table 38-13: **Signals of Interest for Write Leveling Calibration *(Cont'd)***

Signal	Usage	Signal Description
WRLVL_ODELAY_CENTER_ RANK*_BYTE*	One per rank per Byte	Midpoint between WRLVL_ODELAY_STABLE0 and WRLVL_ODELAY_STABLE1. Final ODELAY setting for the byte after WRLVL.
WRLVL_ ODELAY_LAST_OFFSET_RANK*_BYTE*	One per rank per Byte	Final Offset setting used in the algorithm (may be smaller than WRLVL_ODELAY_INITIAL_OFFSET_BYTE*)
WRLVL_ODELAY_LOWEST_COMMON_Byte*	One per Byte	Final ODELAY setting programmed into the RIU.
BUS_DATA_BURST (Available in 2014.2 and later)		General purpose area for storing read bursts of data. This register is intended to store up to four bursts of data for a x8 byte. During Write Leveling, the bus is being used to store the DQ data that may be useful when an error occurs (such as a stuck-at-bit) without having to check general interconnect data. During the first part of the algorithm data is sampled coming back at multiple coarse taps, and the data is stored in these locations. Given the number of samples taken and the limitation of space to store all samples, what is stored is the value found on the bus across multiple samples, as well as the last value seen for a given setting. The data is returned per bit and stored in a 32-bit register such that single bit data is in the format of {f3, r3, f2, r2, f1, r1, f0, r0} (8-bits for a single bit of a burst). A single general interconnect 32-bit register holds data for bits {3, 2, 1, 0} while another holds data for bits {7, 6, 5, 4}. For a x8 device, all bits are read in and "OR'd" together to create a "sample." This sample is used to determine stable 0 or stable 1. When dealing with multiple samples, if any sample does not match with the first sample, the data is marked as unstable internally (0x01010101). The register is split up such that: Bus_Data_Burst_0_Bit0, Bus_Data_Burst_0_Bit1, Bus_Data_Burst_0_Bit2, Bus_Data_Burst_0_Bit3 will hold the aggregate value found across all samples for a given tap setting. This might be for coarse = 0.

Table 38-13: **Signals of Interest for Write Leveling Calibration *(Cont'd)***

Signal	Usage	Signal Description
BUS_DATA_BURST (Available in 2014.2 and later) Continued		Then the following: Bus_Data_Burst_0_Bit4, Bus_Data_Burst_0_Bit5, Bus_Data_Burst_0_Bit6, Bus_Data_Burst_0_Bit7 would hold the last single sample when taking multiple samples. For example, if it is set up to take five samples, this would hold the fifth sample, while the previous bit locations would hold the aggregate of all samples which might be UNSTABLE (0x01010101). Unstable can easily happen if the edges are close to being aligned already. Given that there are only four burst locations yet the algorithm could try up to six coarse taps, there are not enough locations to store all data (4 & 5 would overwrite locations 0 & 1). Some of the data will be overwritten in that case. This is mostly to aid in what is actually seen on the DQ bus as the coarse taps are adjusted. It provides a window into the data as the DQS is adjusted in relation to the CK for a full clock cycle. If the coarse adjustment is found in the first step, a single location is used in case of a failure in the fine search. When no stable 0 is found during the fine adjustment, the value received is stored at: Bus_Data_Burst_0_Bit0, Bus_Data_Burst_0_Bit1, Bus_Data_Burst_0_Bit2, Bus_Data_Burst_0_Bit3 Much in the same way as before, 0 to 3 stores the aggregate, while 4 to 7 stores the final reading of a set of samples.
BISC_ALIGN_PQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_ALIGN_NQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.

Table 38-13: **Signals of Interest for Write Leveling Calibration *(Cont'd)***

Signal	Usage	Signal Description
BISC_PQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_PQTR – BISC_ALIGN_PQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_PQTR – BISC_ALIGN_PQTR). Useful for error code 0x6.
BISC_NQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_NQTR – BISC_ALIGN_NQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_NQTR – BISC_ALIGN_NQTR). Useful for error code 0x6.

This is a sample of results for the Write Leveling XSDB debug signals:

```
WRLVL_COARSE_STABLE0_RANK0_BYTE0               string true true 003
WRLVL_COARSE_STABLE0_RANK0_BYTE1               string true true 000
WRLVL_COARSE_STABLE0_RANK0_BYTE2               string true true 000
WRLVL_COARSE_STABLE0_RANK0_BYTE3               string true true 000
WRLVL_COARSE_STABLE0_RANK0_BYTE4               string true true 002
WRLVL_COARSE_STABLE0_RANK0_BYTE5               string true true 001
WRLVL_COARSE_STABLE0_RANK0_BYTE6               string true true 001
WRLVL_COARSE_STABLE0_RANK0_BYTE7               string true true 001
WRLVL_COARSE_STABLE0_RANK0_BYTE8               string true true 001
WRLVL_COARSE_STABLE1_RANK0_BYTE0               string true true 004
WRLVL_COARSE_STABLE1_RANK0_BYTE1               string true true 001
WRLVL_COARSE_STABLE1_RANK0_BYTE2               string true true 001
WRLVL_COARSE_STABLE1_RANK0_BYTE3               string true true 001
WRLVL_COARSE_STABLE1_RANK0_BYTE4               string true true 003
WRLVL_COARSE_STABLE1_RANK0_BYTE5               string true true 002
WRLVL_COARSE_STABLE1_RANK0_BYTE6               string true true 002
WRLVL_COARSE_STABLE1_RANK0_BYTE7               string true true 002
WRLVL_COARSE_STABLE1_RANK0_BYTE8               string true true 002
WRLVL_ODELAY_CENTER_RANK0_BYTE0                string true true 02b
WRLVL_ODELAY_CENTER_RANK0_BYTE1                string true true 010
WRLVL_ODELAY_CENTER_RANK0_BYTE2                string true true 020
WRLVL_ODELAY_CENTER_RANK0_BYTE3                string true true 02b
WRLVL_ODELAY_CENTER_RANK0_BYTE4                string true true 008
WRLVL_ODELAY_CENTER_RANK0_BYTE5                string true true 02c
WRLVL_ODELAY_CENTER_RANK0_BYTE6                string true true 01b
WRLVL_ODELAY_CENTER_RANK0_BYTE7                string true true 02b
WRLVL_ODELAY_CENTER_RANK0_BYTE8                string true true 016
WRLVL_ODELAY_INITIAL_OFFSET_BYTE0              string true true 016
WRLVL_ODELAY_INITIAL_OFFSET_BYTE1              string true true 017
WRLVL_ODELAY_INITIAL_OFFSET_BYTE2              string true true 016
WRLVL_ODELAY_INITIAL_OFFSET_BYTE3              string true true 016
WRLVL_ODELAY_INITIAL_OFFSET_BYTE4              string true true 017
WRLVL_ODELAY_INITIAL_OFFSET_BYTE5              string true true 017
WRLVL_ODELAY_INITIAL_OFFSET_BYTE6              string true true 017
WRLVL_ODELAY_INITIAL_OFFSET_BYTE7              string true true 017
WRLVL_ODELAY_INITIAL_OFFSET_BYTE8              string true true 017
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE0           string true true 016
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE1           string true true 017
```

```
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE2           string true true 016
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE3           string true true 016
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE4           string true true 017
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE5           string true true 017
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE6           string true true 017
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE7           string true true 017
WRLVL_ODELAY_LAST_OFFSET_RANK0_BYTE8           string true true 017
WRLVL_ODELAY_LOWEST_COMMON_BYTE0               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE1               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE2               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE3               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE4               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE5               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE6               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE7               string true true 000
WRLVL_ODELAY_LOWEST_COMMON_BYTE8               string true true 000
WRLVL_ODELAY_STABLE0_RANK0_BYTE0               string true true 028
WRLVL_ODELAY_STABLE0_RANK0_BYTE1               string true true 00d
WRLVL_ODELAY_STABLE0_RANK0_BYTE2               string true true 01d
WRLVL_ODELAY_STABLE0_RANK0_BYTE3               string true true 027
WRLVL_ODELAY_STABLE0_RANK0_BYTE4               string true true 004
WRLVL_ODELAY_STABLE0_RANK0_BYTE5               string true true 027
WRLVL_ODELAY_STABLE0_RANK0_BYTE6               string true true 017
WRLVL_ODELAY_STABLE0_RANK0_BYTE7               string true true 027
WRLVL_ODELAY_STABLE0_RANK0_BYTE8               string true true 014
WRLVL_ODELAY_STABLE1_RANK0_BYTE0               string true true 02e
WRLVL_ODELAY_STABLE1_RANK0_BYTE1               string true true 014
WRLVL_ODELAY_STABLE1_RANK0_BYTE2               string true true 023
WRLVL_ODELAY_STABLE1_RANK0_BYTE3               string true true 02f
WRLVL_ODELAY_STABLE1_RANK0_BYTE4               string true true 00c
WRLVL_ODELAY_STABLE1_RANK0_BYTE5               string true true 031
WRLVL_ODELAY_STABLE1_RANK0_BYTE6               string true true 020
WRLVL_ODELAY_STABLE1_RANK0_BYTE7               string true true 030
WRLVL_ODELAY_STABLE1_RANK0_BYTE8               string true true 018
BISC_ALIGN_NQTR_NIBBLE0                        string true true 000
BISC_ALIGN_NQTR_NIBBLE1                        string true true 000
BISC_ALIGN_NQTR_NIBBLE2                        string true true 000
BISC_ALIGN_NQTR_NIBBLE3                        string true true 000
BISC_ALIGN_NQTR_NIBBLE4                        string true true 000
BISC_ALIGN_NQTR_NIBBLE5                        string true true 000
BISC_ALIGN_NQTR_NIBBLE6                        string true true 000
BISC_ALIGN_NQTR_NIBBLE7                        string true true 000
BISC_ALIGN_NQTR_NIBBLE8                        string true true 000
BISC_ALIGN_NQTR_NIBBLE9                        string true true 000
BISC_ALIGN_NQTR_NIBBLE10                       string true true 000
BISC_ALIGN_NQTR_NIBBLE11                       string true true 000
BISC_ALIGN_NQTR_NIBBLE12                       string true true 000
BISC_ALIGN_NQTR_NIBBLE13                       string true true 000
BISC_ALIGN_NQTR_NIBBLE14                       string true true 000
BISC_ALIGN_NQTR_NIBBLE15                       string true true 000
BISC_ALIGN_NQTR_NIBBLE16                       string true true 000
BISC_ALIGN_NQTR_NIBBLE17                       string true true 000
BISC_ALIGN_PQTR_NIBBLE0                        string true true 004
BISC_ALIGN_PQTR_NIBBLE1                        string true true 006
BISC_ALIGN_PQTR_NIBBLE2                        string true true 005
BISC_ALIGN_PQTR_NIBBLE3                        string true true 005
BISC_ALIGN_PQTR_NIBBLE4                        string true true 004
BISC_ALIGN_PQTR_NIBBLE5                        string true true 006
BISC_ALIGN_PQTR_NIBBLE6                        string true true 003
```

```
BISC_ALIGN_PQTR_NIBBLE7                       string true true 004
BISC_ALIGN_PQTR_NIBBLE8                       string true true 007
BISC_ALIGN_PQTR_NIBBLE9                       string true true 006
BISC_ALIGN_PQTR_NIBBLE10                      string true true 003
BISC_ALIGN_PQTR_NIBBLE11                      string true true 006
BISC_ALIGN_PQTR_NIBBLE12                      string true true 004
BISC_ALIGN_PQTR_NIBBLE13                      string true true 004
BISC_ALIGN_PQTR_NIBBLE14                      string true true 004
BISC_ALIGN_PQTR_NIBBLE15                      string true true 006
BISC_ALIGN_PQTR_NIBBLE16                      string true true 004
BISC_ALIGN_PQTR_NIBBLE17                      string true true 007
BISC_NQTR_NIBBLE0                             string true true 030
BISC_NQTR_NIBBLE1                             string true true 02f
BISC_NQTR_NIBBLE2                             string true true 031
BISC_NQTR_NIBBLE3                             string true true 031
BISC_NQTR_NIBBLE4                             string true true 02e
BISC_NQTR_NIBBLE5                             string true true 030
BISC_NQTR_NIBBLE6                             string true true 02f
BISC_NQTR_NIBBLE7                             string true true 031
BISC_NQTR_NIBBLE8                             string true true 030
BISC_NQTR_NIBBLE9                             string true true 031
BISC_NQTR_NIBBLE10                            string true true 02f
BISC_NQTR_NIBBLE11                            string true true 030
BISC_NQTR_NIBBLE12                            string true true 02f
BISC_NQTR_NIBBLE13                            string true true 032
BISC_NQTR_NIBBLE14                            string true true 031
BISC_NQTR_NIBBLE15                            string true true 031
BISC_NQTR_NIBBLE16                            string true true 031
BISC_NQTR_NIBBLE17                            string true true 031
BISC_PQTR_NIBBLE0                             string true true 030
BISC_PQTR_NIBBLE1                             string true true 032
BISC_PQTR_NIBBLE2                             string true true 031
BISC_PQTR_NIBBLE3                             string true true 032
BISC_PQTR_NIBBLE4                             string true true 030
BISC_PQTR_NIBBLE5                             string true true 030
BISC_PQTR_NIBBLE6                             string true true 02e
BISC_PQTR_NIBBLE7                             string true true 02f
BISC_PQTR_NIBBLE8                             string true true 033
BISC_PQTR_NIBBLE9                             string true true 033
BISC_PQTR_NIBBLE10                            string true true 030
BISC_PQTR_NIBBLE11                            string true true 034
BISC_PQTR_NIBBLE12                            string true true 030
BISC_PQTR_NIBBLE13                            string true true 030
BISC_PQTR_NIBBLE14                            string true true 030
BISC_PQTR_NIBBLE15                            string true true 031
BISC_PQTR_NIBBLE16                            string true true 031
BISC_PQTR_NIBBLE17                            string true true 033
```

Expected Results

The tap variance across DQS byte groups vary greatly due to the difference in trace lengths with fly-by-routing. When an error occurs, an error loop is started that generates DQS strobes to the DRAM while still in WRLVL mode. This error loop runs continuously until a reset or power cycle to aid in debug. Table 38-14 provides expected results for the coarse and fine parameters during Write Leveling.

Table 38-14: **Expected Write Leveling Results**

Parameter	Description
WRLVL_COARSE_STABLE0_RANK*_BYTE*	WRLVL Coarse tap setting after calibration. Expected values 0 to 4.
WRLVL_ODELAY_STABLE1_RANK*_BYTE*	WRLVL ODELAY tap setting to find Stable 1 or noise. Expected values 0 to 90° setting of ODELAY taps (depending on the tap resolution).
WRLVL_ODELAY_CENTER_ RANK*_BYTE*	Midpoint between WRLVL_ODELAY_STABLE0 and WRLVL_ODELAY_STABLE1. Expected value should be less than 90° (use BISC values to estimate the 90° value) and between WRLVL_FINE_LEFT and WRLVL_FINE_RIGHT.

Hardware Measurements

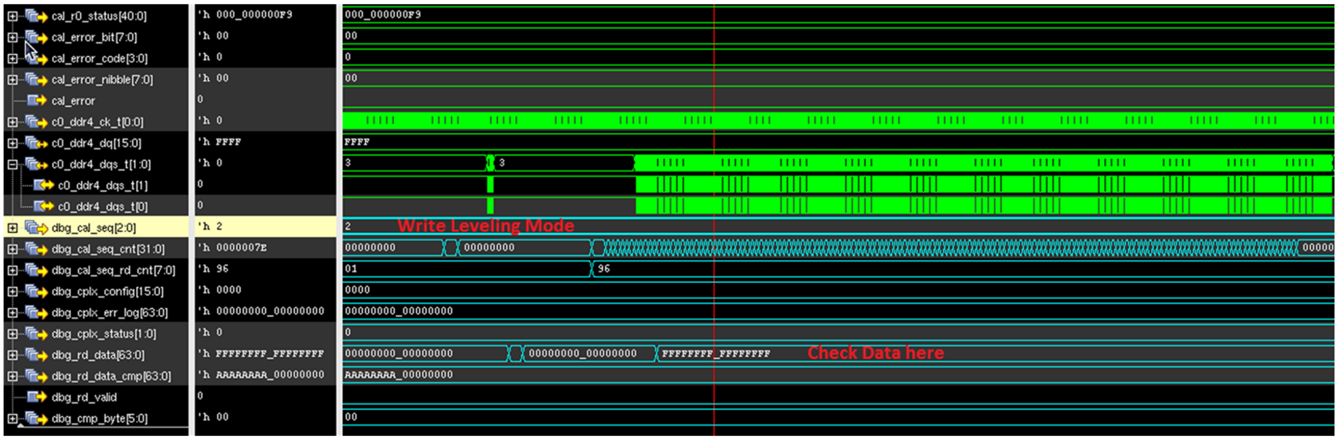
The following measurements can be made during the error loop or when triggering on the status bit that indicates the start of WRLVL (**`dbg_cal_seq[1]`** = 1'b1).

- Verify DQS and CK are toggling on the board. The FPGA sends DQS and CK during Write Leveling. If they are not toggling, something is wrong with the setup and the General Checks, page 588 section should be thoroughly reviewed.
- Verify fly-by-routing is implemented correctly on the board.
- Verify CK to DQS trace matching. The required matching is documented with the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11]. Failure to adhere to this spec can result in Write Leveling failures.
- Trigger on the start of Write Leveling by bringing **`dbg_cal_seq[1]`** to an I/O and using the rising edge (**`1`b1`**) as the scope trigger.
  Monitor the following:
    - MRS command at the memory to enable Write Leveling Mode. The Mode registers must be properly set up to enable Write Leveling. Specifically, address bit A7 must be correct. If the part chosen in the Memory IP is not accurate or there is an issue with the connection of the address bits on the board, this could be an issue. If the Mode registers are not set up to enable Write Leveling, the 0-to-1 transition is not seen.

      Note: For dual-rank design when address mirroring is used, address bit A7 is not the same between the two ranks.
    - Verify the ODT pin is connected and being asserted properly during the DQS toggling.
    - Check the signal levels of all the DQ bits being returned. Any stuck-at-bits (Low/High) or floating bits that are not being driven to a given rail can cause issues.
    - Verify the DQS to CK relationship changes as the algorithm makes adjustments to the DQS. Check the DQ value being returned as this relationship changes.

  - For DDR3 check the V_{REF} voltage, while for DDR4 check the V_{REF} settings are correct in the design.

- Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the trigger to `dbg_cal_seq` = 0R0 (R signifies rising edge). The following simulation example shows how the debug signals should behave during successful Write Leveling.



cal_r0_status[40:0]	'h 000_000000F9
cal_error_bit[7:0]	'h 00
cal_error_code[3:0]	'h 0
cal_error_nibble[7:0]	'h 00
cal_error	0
c0_ddr4_ck_t[0:0]	'h 0
c0_ddr4_dq[15:0]	'h FFFF
c0_ddr4_dqs_t[1:0]	'h 0
c0_ddr4_dqs_t[1]	0
c0_ddr4_dqs_t[0]	0
dbg_cal_seq[2:0]	'h 2
dbg_cal_seq_cnt[31:0]	'h 0000007E
dbg_cal_seq_rd_cnt[7:0]	'h 96
dbg_cplx_config[15:0]	'h 0000
dbg_cplx_err_log[63:0]	'h 00000000_00000000
dbg_cplx_status[1:0]	'h 0
dbg_rd_data[63:0]	'h FFFFFFFF_FFFFFFFF
dbg_rd_data_cmp[63:0]	'h AAAAAAAA_00000000
dbg_rd_valid	0
dbg_cmp_byte[5:0]	'h 00

Figure 38-24: **RTL Debug Signals during Write Leveling**

Read Leveling Calibration Overview

After the gate has been trained and Write Leveling has completed, the next step is to ensure reliable capture of the read data with the DQS. This stage of Read Leveling is divided into two phases, Per-Bit Deskew and Read DQS Centering. Read DQS Centering utilizes the DDR3 and DDR4 Multi Purpose Register (MPR). The MPR contains a pattern that can be used to train the read DQS and DQ for read capture. While DDR4 allows for several patterns, DDR3 only has a single repeating pattern available.

To perform per-bit deskew, a non-repeating pattern is useful to deal with or diagnose cases of extreme skew between different bits in a byte. Because this is limited by the DDR3 MPR pattern, a long pattern is first written to the DRAM and then read back to perform per-bit deskew (only done on the first rank of a multi-rank system). When per-bit deskew is complete, the simple repeating pattern available through both DDR3 and DDR4 MPR is used to center the DQS in the DQ read eye.

The XIPHY provides separate delay elements (2.5 to 15 ps per tap, 512 total) for the DQS to clock the rising and falling edge DQ data (PQTR for rising edge, NQTR for falling edge) on a per-nibble basis (four DQ bits per PQTR/NQTR). This allows the algorithm to center the rising and falling edge DQS strobe independently to ensure more margin when dealing with DCD. The data captured in the PQTR clock domain is transferred to the NQTR clock domain before being sent to the read FIFO and to the general interconnect clock domain.

Due to this transfer of clock domains, the PQTR and NQTR clocks must be roughly 180° out of phase. This relationship between the PQTR/NQTR clock paths is set up as part of the BISC start-up routine, and thus calibration needs to maintain this relationship as part of the training (BISC_ALIGN_PQTR, BISC_ALIGN_NQTR, BISC_PQTR, BISC_NQTR).

Debugging Read Per-Bit Deskew Failures

First, write `0x00` to address `0x000`. Because the write latency calibration has not yet been performed, the address DQ is held for eight clock cycles before and after the expected write latency is expected. The DQS toggles extra time before/after is shown in Figure 38-25. This ensures the data is written to the DRAM if the burst does not occur at the correct time the DRAM expects it.

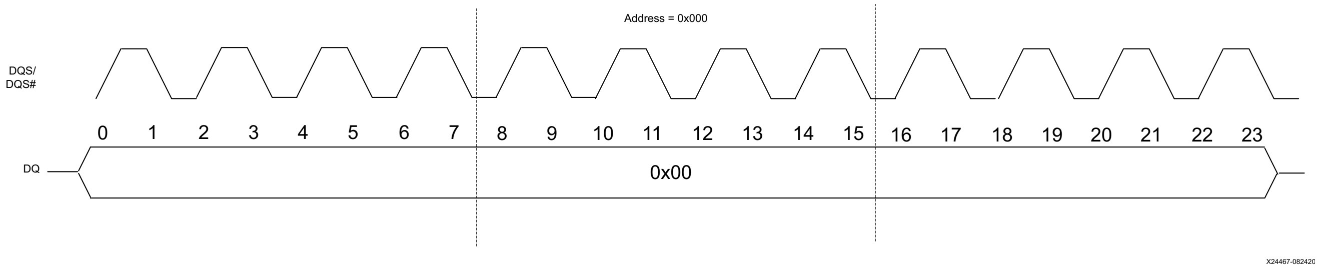


Figure 38-25: **Per-Bit Deskew – Write 0x00 to Address 0x000**

Next, write `0xFF` to a different address to allow for back-to-back reads (Figure 38-26). For DDR3 address `0x008` is used, while for DDR4 address `0x000` and bank group `0x1` is used. At higher frequencies, DDR4 requires a change in the bank group to allow for back-to-back bursts of eight.

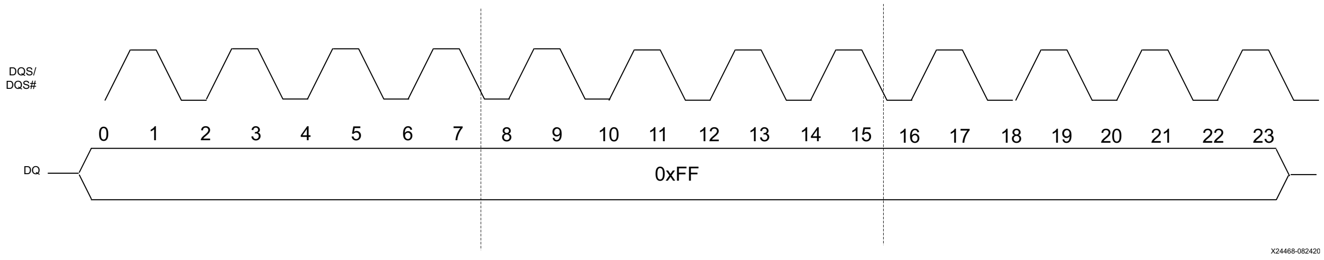


Figure 38-26: **Per-Bit Deskew – Write 0xFF to Other Address**

After the data is written, back-to-back reads are issued to the DRAM to perform per-bit deskew (Figure 38-27).

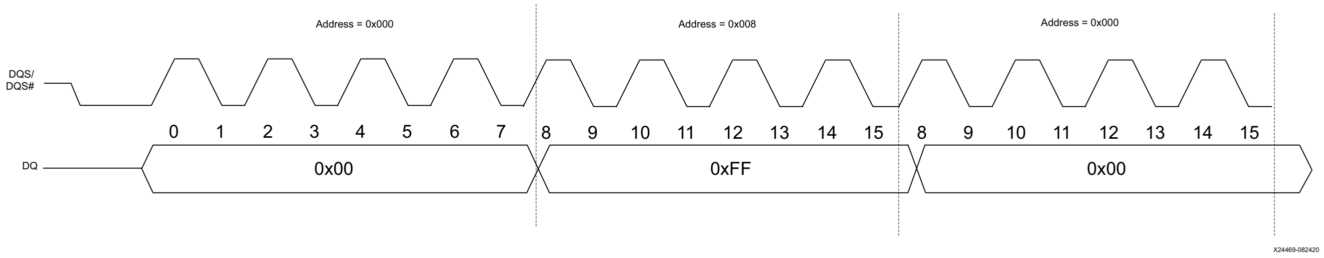


Figure 38-27: **Per-Bit Deskew – Back-to-Back Reads (No Gaps)**

Using this pattern each bit in a byte is left edge aligned with the DQS strobe (PQTR/NQTR). More than a bit time of skew can be seen and corrected as well.

RECOMMENDED: *In general, a bit time of skew between bits is not ideal. Ensure the DDR3/DDR4 trace matching guidelines within DQS byte are met. See PCB Guidelines for DDR3, page 87 and PCB Guidelines for DDR4, page 87.*

At the start of deskew, the PQTR/NQTR are decreased down together until one of them hits 0 (to preserve the initial relationship setup by BISC). Next, the data for a given bit is checked for the matching pattern. Only the rising edge data is checked for correctness. The falling edge comparison is thrown away to allow for extra delay on the PQTR/NQTR relative to the DQ.

While in the ideal case, the PQTR/NQTR are edge aligned with the DQ when the delays are set to 0. Due to extra delay in the PQTR/NQTR path, the NQTR might be pushed into the next burst transaction at higher frequencies and so it is excluded from the comparison (Figure 38-28 through Figure 38-29). More of the rising edge data of a given burst would need to be discarded to deal with more than a bit time of skew. If the last part of the burst was not excluded, the failure would cause the PQTR/NQTR to be pushed instead of the DQ IDELAY.

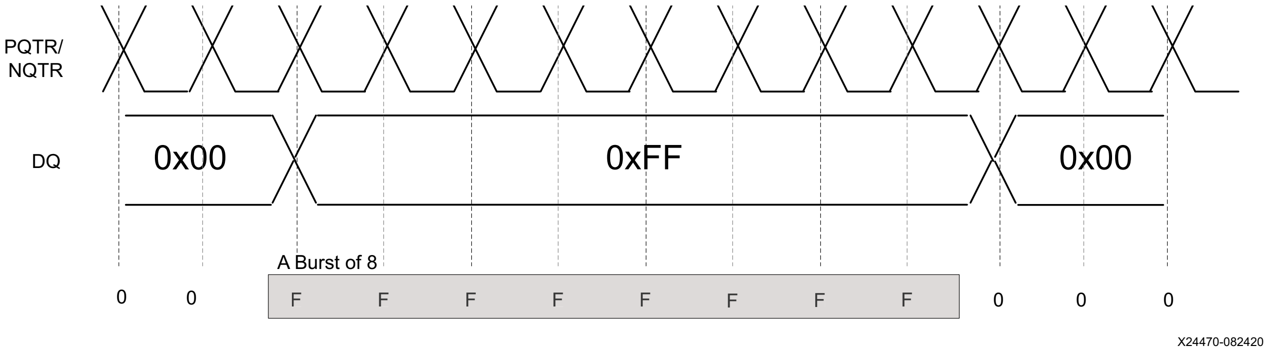


Figure 38-28: **Per-Bit Deskew – Delays Set to 0 (Ideal)**

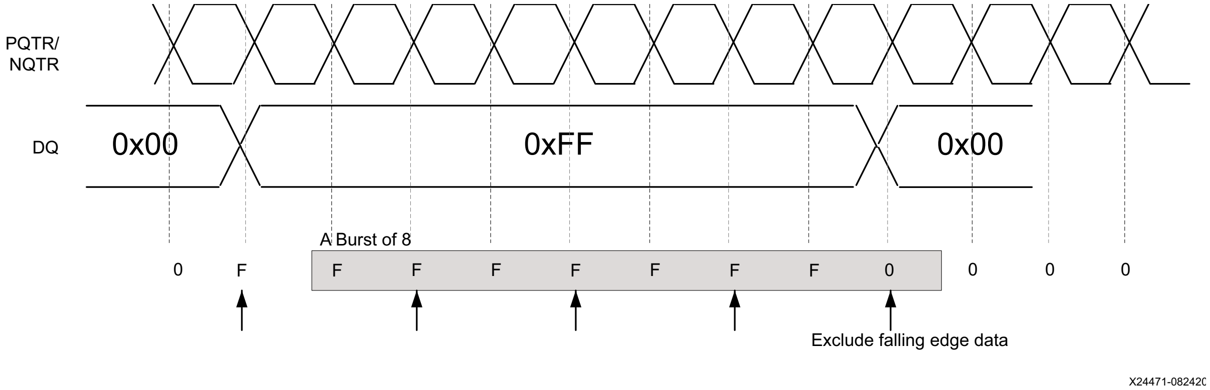


Figure 38-29: **Per-Bit Deskew – Delays Set to 0**

If the pattern is found, the given IDELAY on that bit is incremented by 1, then checked again. If the pattern is not seen, the PQTR/NQTR are incremented by 1 and the data checked again. The algorithm checks for the passing and failing region for a given bit, adjusting either the PQTR/NQTR delays or the IDELAY for that bit.

To guard against noise in the uncertain region, the passing region is defined by a minimum window size (10), hence the passing region is not declared as found unless the PQTR/NQTR are incremented and a contiguous region of passing data is found for a given bit. All of the bits are cycled through to push the PQTR/NQTR out to align with the latest bit in a given nibble. Figure 38-30 through Figure 38-33 show an example of the PQTR/NQTR and various bits being aligned during the deskew stage.

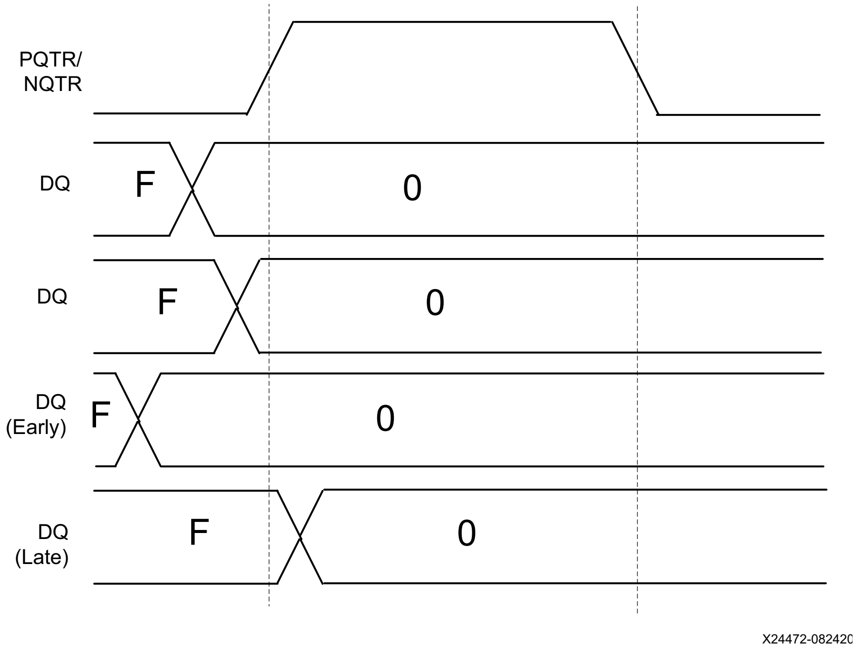


Figure 38-30: **Per-Bit Deskew – Initial Relationship Example**

The algorithm takes the result of each bit at a time and decides based on the results of that bit only. The common PQTR/NQTR are delayed as needed to align with each bit, but is not decremented. This ensures it gets pushed out to the latest bit.

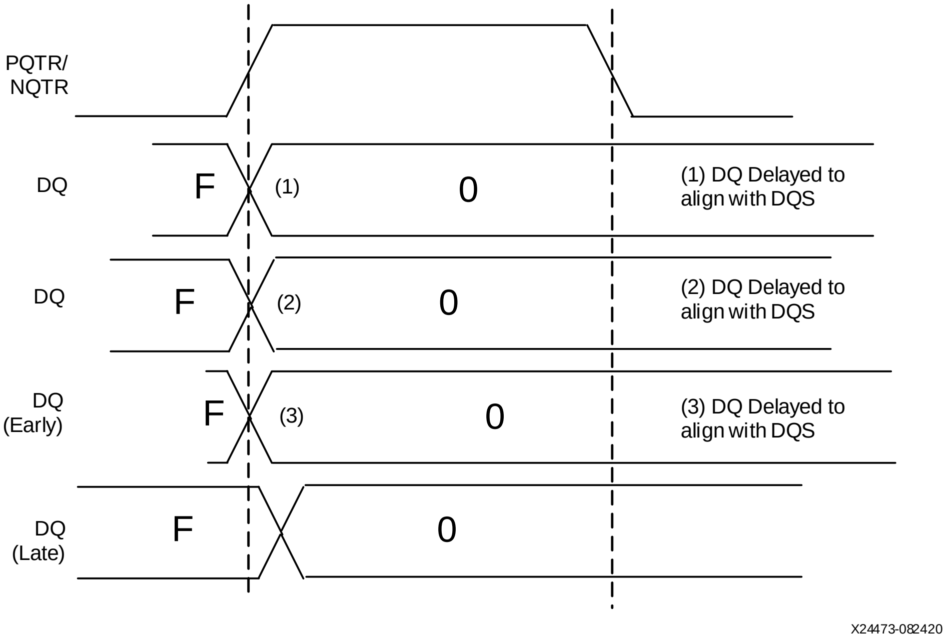


Figure 38-31: **Per-Bit Deskew – Early Bits Pushed Out**

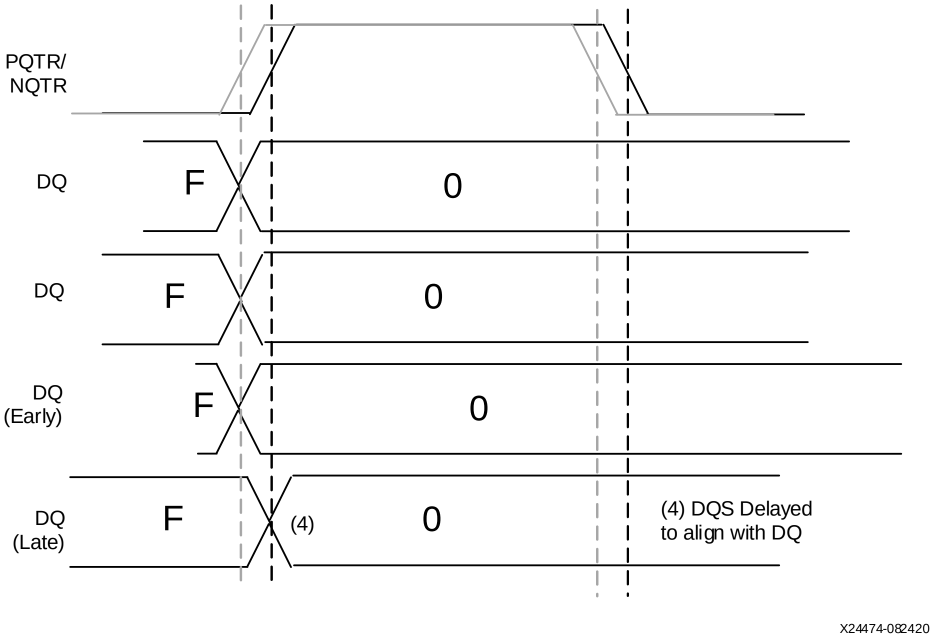
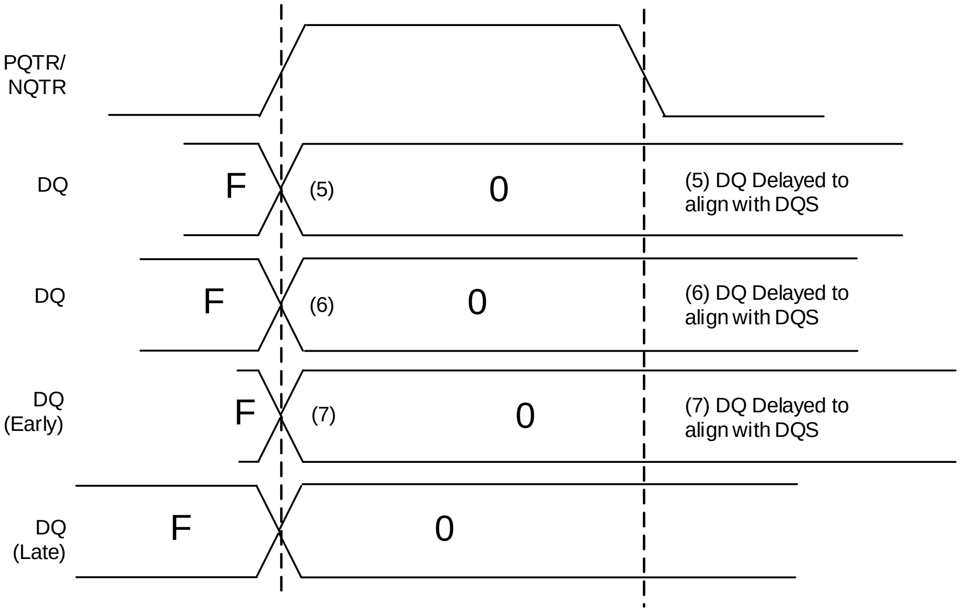


Figure 38-32: **Per-Bit Deskew – PQTR/NQTR Delayed to Align with Late Bit**

When completed, the PQTR/NQTR are pushed out to align with the latest DQ bit (RDLVL_DESKEW_PQTR_nibble, RDLVL_DESKEW_NQTR_nibble), but DQ bits calibrated first might have been early as shown in the example. Accordingly, all bits are checked once again and aligned as needed (Figure 38-33).



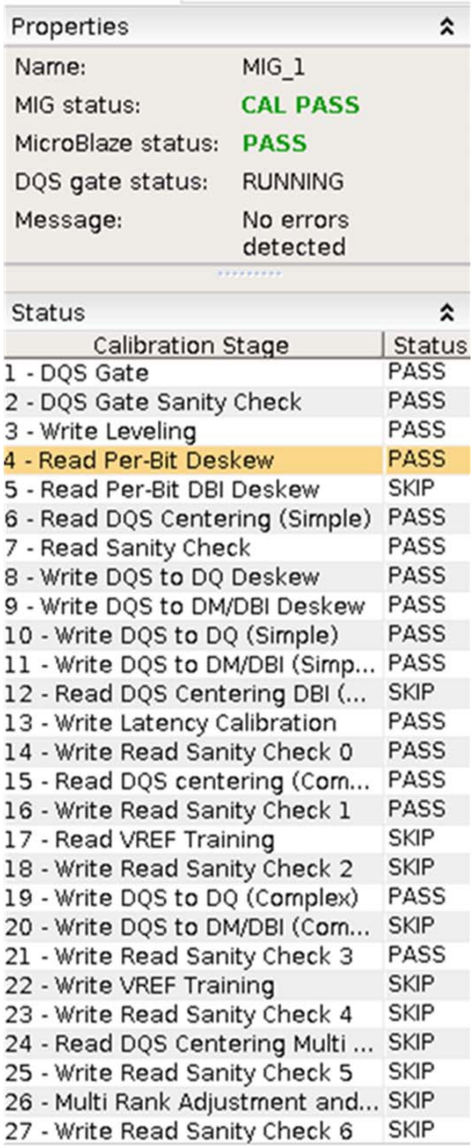
X24475-082420

Figure 38-33: **Per-Bit Deskew – Push Early Bits as Needed to Align**

The final DQ IDELAY value from deskew is stored at RDLVL_DESKEW_IDELAY_Byte_Bit.

Debug

To determine the status of Read Per-Bit Deskew Calibration, click the **Read Per-Bit Deskew** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Properties

Name: MIG_1

MIG status: CAL PASS

MicroBlaze status: PASS

DQS gate status: RUNNING

Message: No errors detected

Status

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simp...	PASS
12 - Read DQS Centering DBI (...	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Com...	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Com...	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi ...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-34: **Memory IP XSDB Debug GUI Example – Read Per-Bit Deskew**

The status of Read Per-Bit Deskew can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-15. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-15: **DDR_CAL_ERROR Decode for Read Deskew Calibration**

Per-Bit Deskew DDR_CAL_ ERROR_CODE	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	Nibble	Bit	No valid data found for a given bit in the nibble (deskew pattern)	Check the BUS_DATA_BURST fields in XSDB. Check the dbg_rd_data, dbg_rd_data_cmp, and dbg_expected_data signals in the ILA. Check the pinout and look for any STUCK-AT-BITs, check vrp resistor, V_{REF} resistor. Check BISC_PQTR, BISC_NQTR for starting offset between rising/falling clocks. Probe the board and check for the returning pattern to determine if the initial write to the DRAM happened properly, or if it is a read failure. Check ODT if it is a write issue.
0xF	Nibble	Bit	Timeout error waiting for read data to return	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-16 describes the signals and values adjusted or used during the Read Per-Bit Deskew stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-16: **Signals of Interest for Read Deskew Calibration**

Signal	Usage	Signal Description
RDLVL_DESKEW_PQTR_NIBBLE*	One per nibble	Read leveling PQTR when left edge of read data valid window is detected during per bit read DQ deskew.
RDLVL_DESKEW_NQTR_NIBBLE*	One per nibble	Read leveling NQTR when left edge of read data valid window is detected during per bit read DQ deskew.

Table 38-16: **Signals of Interest for Read Deskew Calibration**

Signal	Usage	Signal Description
RDLVL_DESKEW_IDELAY_BYTE_BIT*	One per Bit	Read leveling IDELAY delay value found during per bit read DQ deskew.
BUS_DATA_BURST (2014.3+)		When a failure occurs during deskew, some data is saved to indicate what the data looks like for a byte across some tap settings for a given byte the failure occurred for (DQ IDELAY is left wherever the algorithm left it). Deskew (Figure 38-35): BUS_DATA_BURST_0 holds first part of two burst data (should be all 0) when PQTR/NQTR set to 0 taps. BUS_DATA_BURST_1 holds second part of two burst data (should be all 1). when PQTR/NQTR set to 0 taps. BUS_DATA_BURST_2 holds first part of two burst data (should be all 0) when PQTR/NQTR set to 90°. BUS_DATA_BURST_3 holds second part of two burst data (should be all 1) when PQTR/NQTR set to 90°.

Figure 38-35 shows an example of the behavior described in the BUS_DATA_BURST description in Table 38-16.

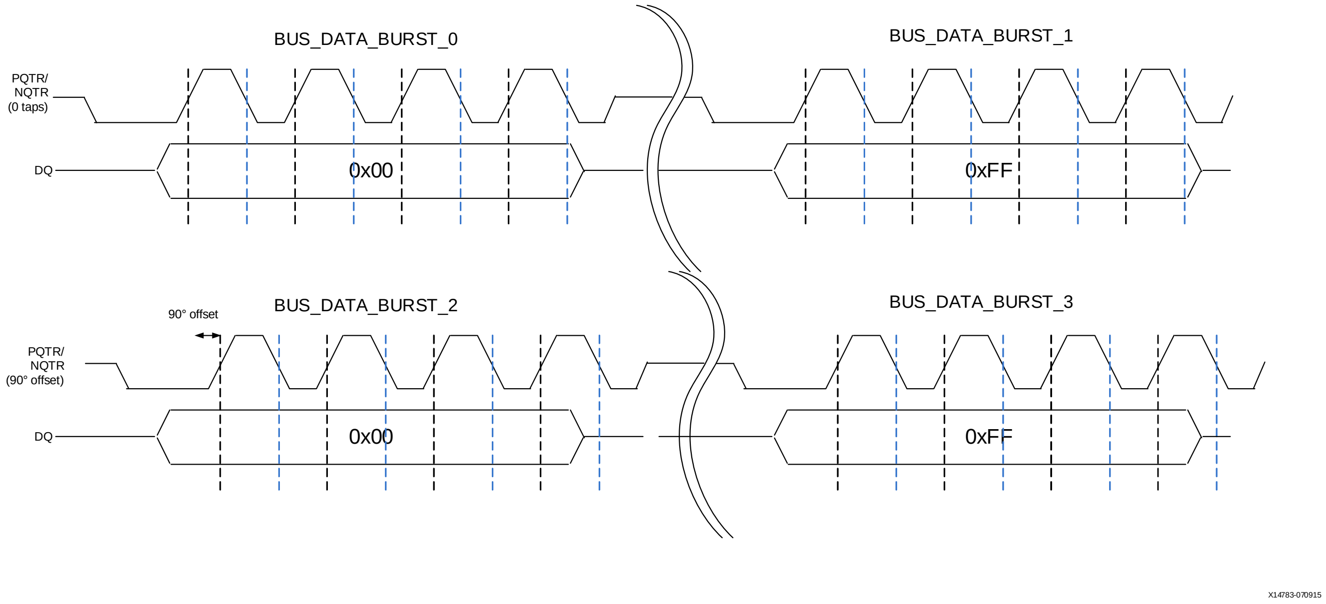


Figure 38-35: **Deskew Error (XSDB BUS_DATA_BURST)**

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope™. Figure 38-36 is an example of how the data is converted.

Note: For this stage of calibration which is using a data pattern of all 0s or all 1s, the conversion is not visible.

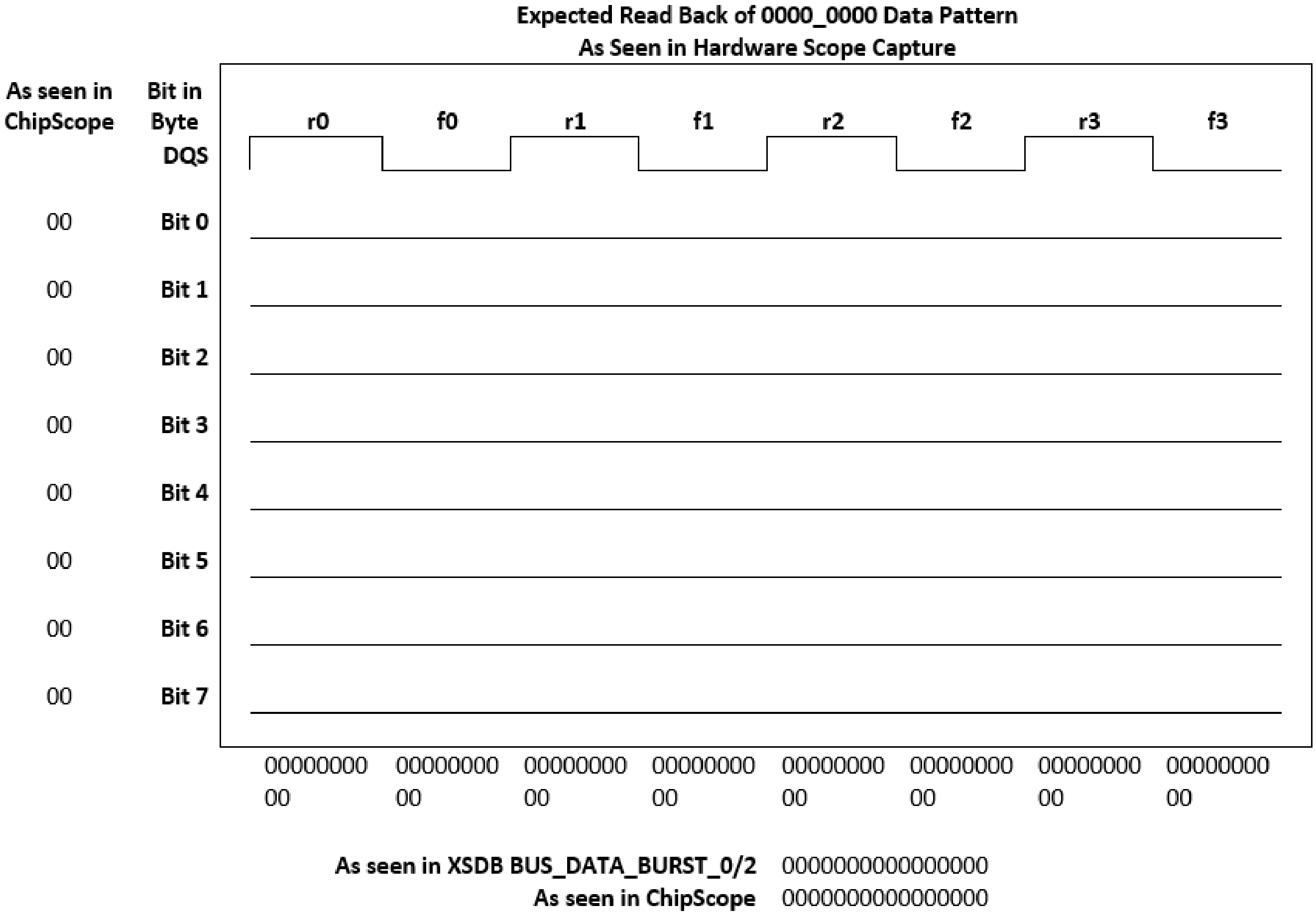


Figure 38-36: **Expected Read Back 0000_0000 Data Pattern**

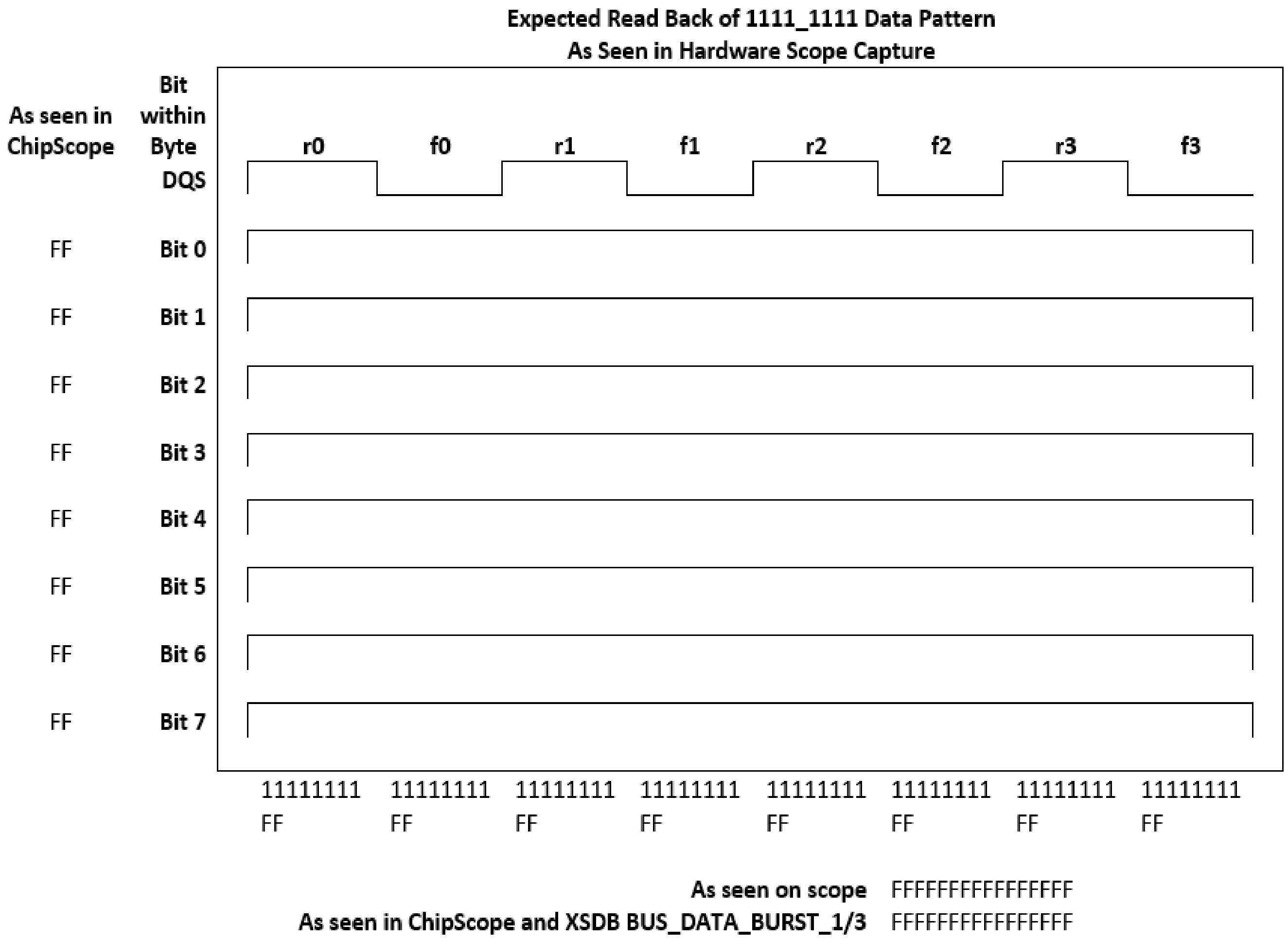


Figure 38-37: **Expected Read Back 1111_1111 Data Pattern**

This is a sample of results for the Read Per-Bit Deskew XSDB debug signals:

```
RDLVL_DESKEW_IDELAY_BYTE0_BIT0                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE0_BIT1                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE0_BIT2                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE0_BIT3                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE0_BIT4                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE0_BIT5                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE0_BIT6                 string true true 033
RDLVL_DESKEW_IDELAY_BYTE0_BIT7                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE1_BIT0                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE1_BIT1                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE1_BIT2                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE1_BIT3                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE1_BIT4                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE1_BIT5                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE1_BIT6                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE1_BIT7                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE2_BIT0                 string true true 033
RDLVL_DESKEW_IDELAY_BYTE2_BIT1                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE2_BIT2                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE2_BIT3                 string true true 028
```

```
RDLVL_DESKEW_IDELAY_BYTE2_BIT4                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE2_BIT5                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE2_BIT6                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE2_BIT7                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT0                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE3_BIT1                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE3_BIT2                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT3                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT4                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT5                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE3_BIT6                 string true true 028
RDLVL_DESKEW_IDELAY_BYTE3_BIT7                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE4_BIT0                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE4_BIT1                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE4_BIT2                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE4_BIT3                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE4_BIT4                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE4_BIT5                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE4_BIT6                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE4_BIT7                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE5_BIT0                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE5_BIT1                 string true true 02a
RDLVL_DESKEW_IDELAY_BYTE5_BIT2                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE5_BIT3                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE5_BIT4                 string true true 028
RDLVL_DESKEW_IDELAY_BYTE5_BIT5                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE5_BIT6                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE5_BIT7                 string true true 026
RDLVL_DESKEW_IDELAY_BYTE6_BIT0                 string true true 028
RDLVL_DESKEW_IDELAY_BYTE6_BIT1                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE6_BIT2                 string true true 025
RDLVL_DESKEW_IDELAY_BYTE6_BIT3                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE6_BIT4                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE6_BIT5                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE6_BIT6                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE6_BIT7                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE7_BIT0                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE7_BIT1                 string true true 02a
RDLVL_DESKEW_IDELAY_BYTE7_BIT2                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE7_BIT3                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE7_BIT4                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE7_BIT5                 string true true 02a
RDLVL_DESKEW_IDELAY_BYTE7_BIT6                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE7_BIT7                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE8_BIT0                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE8_BIT1                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE8_BIT2                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE8_BIT3                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE8_BIT4                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE8_BIT5                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE8_BIT6                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE8_BIT7                 string true true 02f
RDLVL_DESKEW_NQTR_NIBBLE0                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE1                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE2                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE3                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE4                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE5                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE6                      string true true 001
```

```
RDLVL_DESKEW_NQTR_NIBBLE7                      string true true 002
RDLVL_DESKEW_NQTR_NIBBLE8                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE9                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE10                     string true true 000
RDLVL_DESKEW_NQTR_NIBBLE11                     string true true 000
RDLVL_DESKEW_NQTR_NIBBLE12                     string true true 000
RDLVL_DESKEW_NQTR_NIBBLE13                     string true true 002
RDLVL_DESKEW_NQTR_NIBBLE14                     string true true 001
RDLVL_DESKEW_NQTR_NIBBLE15                     string true true 000
RDLVL_DESKEW_NQTR_NIBBLE16                     string true true 000
RDLVL_DESKEW_NQTR_NIBBLE17                     string true true 000
RDLVL_DESKEW_PQTR_NIBBLE0                      string true true 000
RDLVL_DESKEW_PQTR_NIBBLE1                      string true true 003
RDLVL_DESKEW_PQTR_NIBBLE2                      string true true 000
RDLVL_DESKEW_PQTR_NIBBLE3                      string true true 001
RDLVL_DESKEW_PQTR_NIBBLE4                      string true true 002
RDLVL_DESKEW_PQTR_NIBBLE5                      string true true 000
RDLVL_DESKEW_PQTR_NIBBLE6                      string true true 000
RDLVL_DESKEW_PQTR_NIBBLE7                      string true true 000
RDLVL_DESKEW_PQTR_NIBBLE8                      string true true 003
RDLVL_DESKEW_PQTR_NIBBLE9                      string true true 002
RDLVL_DESKEW_PQTR_NIBBLE10                     string true true 001
RDLVL_DESKEW_PQTR_NIBBLE11                     string true true 004
RDLVL_DESKEW_PQTR_NIBBLE12                     string true true 001
RDLVL_DESKEW_PQTR_NIBBLE13                     string true true 000
RDLVL_DESKEW_PQTR_NIBBLE14                     string true true 000
RDLVL_DESKEW_PQTR_NIBBLE15                     string true true 000
RDLVL_DESKEW_PQTR_NIBBLE16                     string true true 000
RDLVL_DESKEW_PQTR_NIBBLE17                     string true true 002
```

Expected Results

- Look at the individual IDELAY taps for each bit. The IDELAY taps should only vary by 0 to 20 taps, and is dependent on PCB trace delays. For Deskew, the IDELAY taps are typically in the 50 to 70 tap range, while PQTR and NQTR are usually in the 0 to 5 tap range.
- Determine if any bytes completed successfully. The per-bit algorithm sequentially steps through each DQS byte.

Hardware Measurements

1. Probe the write commands and read commands at the memory:
    - Write = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 1; `act_n` = 1 (DDR4 only)
    - Read = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 0; `act_n` = 1 (DDR4 only)
2. Probe a data pin to check for data being returned from the DRAM.
3. Probe the writes checking the signal level of the write DQS and the write DQ.
4. Probe the V_{REF} level at the DRAM (for DDR3).
5. Probe the DM pin which should be deasserted during the write burst (or tied off on the board with an appropriate value resistor).

6. Probe the read burst after the write and check if the expected data pattern is being returned.
7. Check for floating address pins if the expected data is not returned.
8. Check for any stuck-at level issues on DQ pins whose signal level does not change. If at all possible probe at the receiver to check termination and signal integrity.
9. Check the DBG port signals and the full read data and comparison result to check the data in general interconnect. The calibration algorithm has RTL logic issue the commands and check the data.
   Check if the `dbg_rd_valid` aligns with the data pattern or is off (which can indicate an issue with DQS gate calibration). Set up a trigger when the error gets asserted to capture signals in the hardware debugger for analysis.
10. Re-check results from DQS gate or other previous calibration stages. Compare passing byte lanes against failing byte lanes for previous stages of calibration. If a failure occurs during simple pattern calibration, check the values found during deskew for example.
11. All of the data comparison for read deskew occurs in the general interconnect, so it can be useful to pull in the debug data in the hardware debugger and take a look at what the data looks like coming back as taps are adjusted, see Figure 38-38. The screen captures are from simulation, with a small burst of five reads. Look at `dbg_rd_data`, `dbg_rd_data_cmp`, and `dbg_rd_valid`.
12. Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the Read Deskew trigger to `cal_r*_status[6]` = R (rising edge). To view each byte, add an additional trigger on `dbg_cmp_byte` and set to the byte of interest. The following simulation example shows how the debug signals should behave during successful Read Deskew.

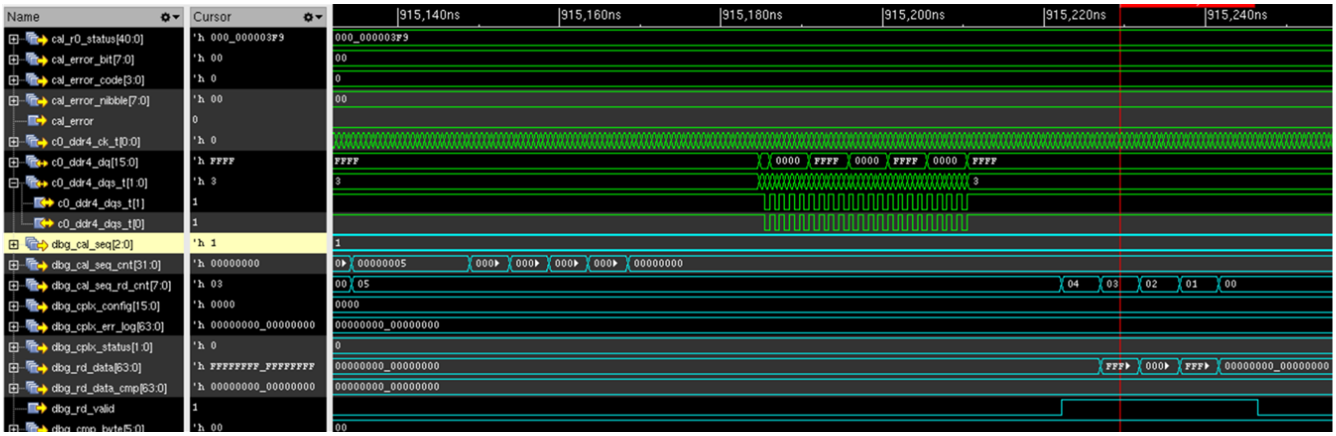


Figure 38-38: **RTL Debug Signals during Read Deskew (No Error)**

13. After failure during this stage of calibration, the design goes into a continuous loop of read commands to allow board probing.

Debugging Read Per-Bit DBI Deskew Failures

If the read DBI option is selected for DDR4, the DBI pin needs to be calibrated along with the DQ bits being captured.

The regular deskew algorithm performs a per-bit deskew on every DQ bit in a nibble against the PQTR/NQTR, pushing early DQ bits to line up with late bits. Because the DBI pin is an input to one of the nibbles, it could have an effect on the PQTR/NQTR settings or even the other DQ pins if the DQ pins need to be pushed to align with the DBI pin. A similar mechanism as the DQ per-bit deskew is ran but the DBI pin is deskewed instead in relation to the PQTR/NQTR.

1. Turn on DBI on the read path (MRS setting in the DRAM and a fabric switch that inverts the read data when value read from the DBI pin is asserted).
2. If the nibble does not contain the DBI pin, skip the nibble and go to the next nibble.
3. Start from the previous PQTR/NQTR settings found during DQ deskew (edge alignment for bits in the nibble).
4. Issue back-to-back reads to address `0x000`/Bank Group 0 and `0x000`/Bank Group 1. This is repeated until per-bit DBI deskew is complete as shown in Figure 38-39.

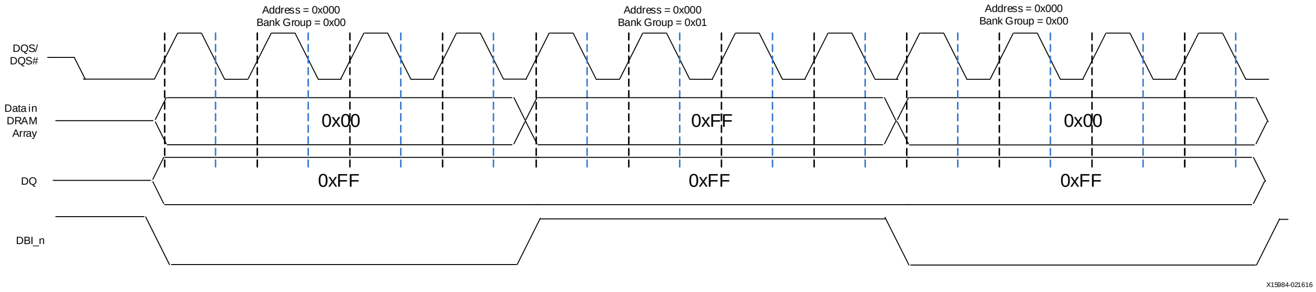


Figure 38-39: **DBI Deskew Read Pattern**

5. Delay the DBI pin with IDELAY to edge align with the PQTR/NQTR clock. If the PQTR/NQTR delay needs to be adjusted, the other DQ bits in the nibble are adjusted accordingly. This occurs if the DBI pin arrives later than all other bits in the nibble.
6. Loop through all nibbles in the interface for the rank.
7. Turn off DBI on the read path (MRS setting in the DRAM and fabric switch).

Debug

To determine the status of Read Per-Bit DBI Deskew Calibration, click the **Read Per-Bit DBI Deskew Calibration** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.

The status of Read Per-Bit DBI Deskew can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-17.

Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-17: **DDR_CAL_ERROR Decode for Read Per-Bit DBI Deskew**

Per-Bit DBI Deskew DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Nibble	N/A	No valid data found for a given bit in the nibble when running the deskew pattern	Check the BUS_DATA_BURST fields in XSDB. Check the dbg_rd_data, dbg_rd_data_cmp, and dbg_expected_data signals in the ILA. Check the pinout for the DBI pin. Probe the board and check for the returning pattern to determine if the initial write to the DRAM happened properly, or if it is a read failure. Probe the DBI pin during the read.
0xF	Nibble	N/A	Timeout error waiting for all read data bursts to return	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-18 shows the signals and values adjusted or used during the Read Per-Bit Deskew stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-18: **Signals of Interest for Read Per-Bit DBI Deskew Calibration**

Signal	Usage	Signal Description
RDLVL_DESKEW_DBI_PQTR	One per nibble	Read leveling PQTR when left edge of read data valid window is detected during per-bit read DBI deskew.
RDLVL_DESKEW_DBI_NQTR	One per nibble	Read leveling NQTR when left edge of read data valid window is detected during per-bit read DBI deskew.
RDLVL_DESKEW_DBI_IDELAY_BYTE	One per Byte	Read leveling IDELAY delay value found during per-bit read DBI deskew.
RDLVL_DESKEW_PQTR_NIBBLE	One per nibble	Read leveling PQTR when left edge of read data valid window is detected during per-bit read DQ deskew.
RDLVL_DESKEW_NQTR_NIBBLE	One per nibble	Read leveling NQTR when left edge of read data valid window is detected during per-bit read DQ deskew.

Table 38-18: **Signals of Interest for Read Per-Bit DBI Deskew Calibration *(Cont'd)***

Signal	Usage	Signal Description
RDLVL_DESKEW_IDELAY_BYTE_BIT*	One per Bit	Read leveling IDELAY delay value found during per-bit read DQ deskew.
BUS_DATA_BURST		When a failure occurs during deskew, some data is saved to indicate what the data looks like for a byte across some tap settings for a given byte the failure occurred for (DQ IDELAY is left wherever the algorithm left it). Deskew (Figure 38-35): BUS_DATA_BURST_0 holds first part of two burst data (should be all 0) when PQTR/NQTR set to t taps. BUS_DATA_BURST_1 holds second part of two burst data (should be all 1). When PQTR/NQTR set to 0 taps. BUS_DATA_BURST_2 holds first part of two burst data (should be all 0) when PQTR/NQTR set to 90°. BUS_DATA_BURST_3 holds second part of two burst data (should be all 1) when PQTR/NQTR set to 90°.

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope™. Figure 38-40 is an example of how the data is converted.

Note: For this stage of calibration which is using a data pattern of all 0s or all 1s, the conversion is not visible.

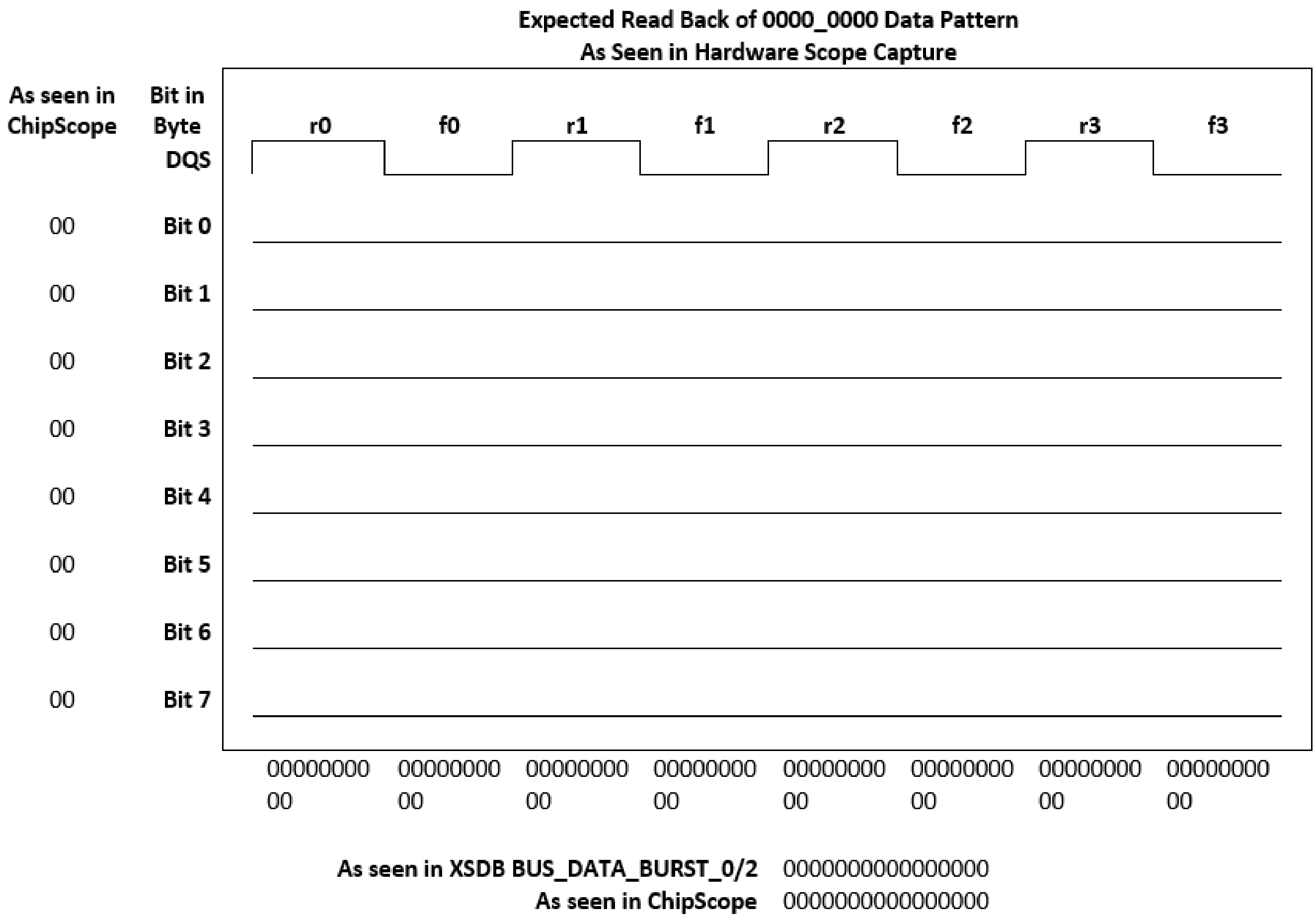


Figure 38-40: **Expected Read Back 0000_0000 Data Pattern**

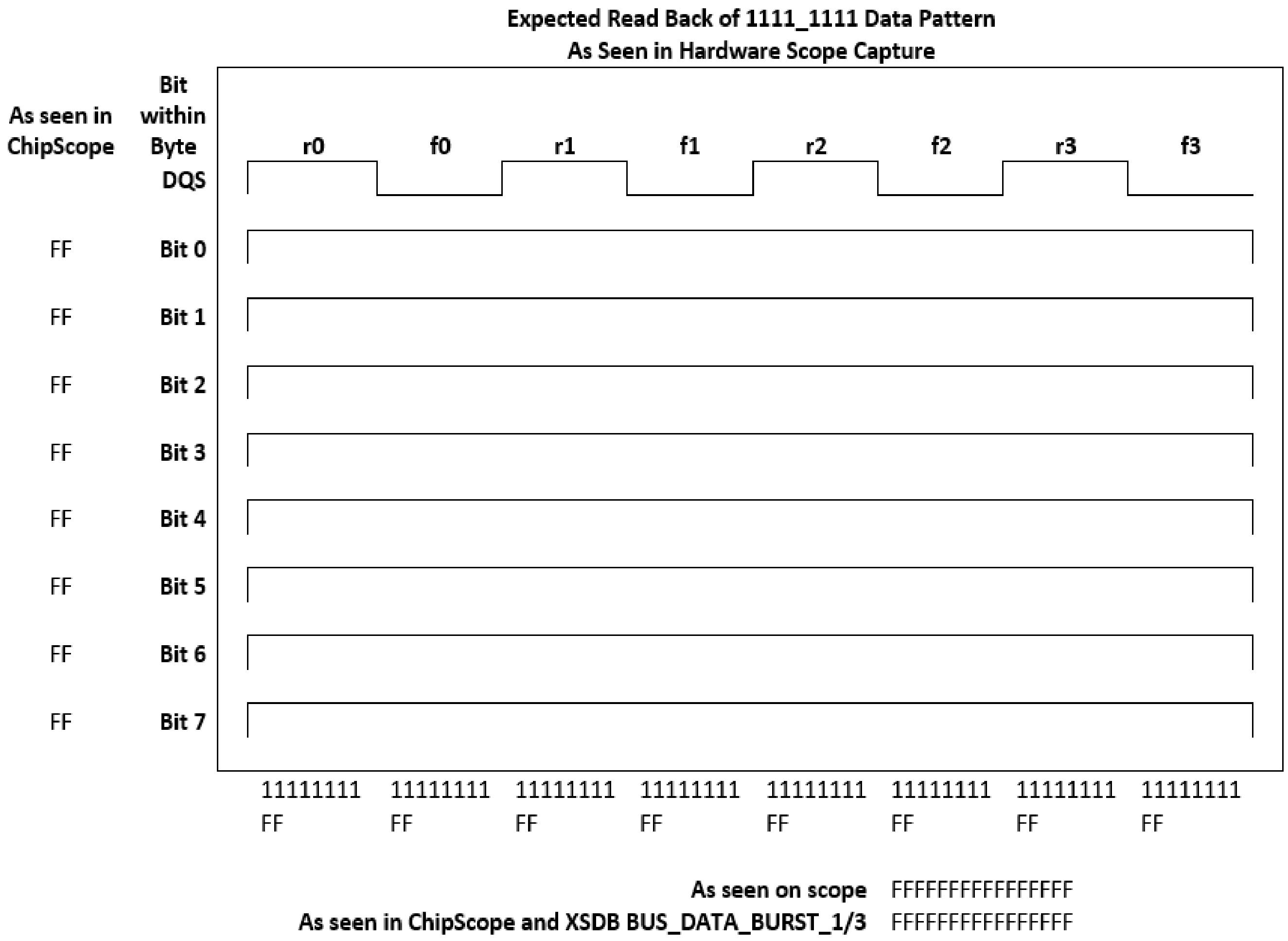


Figure 38-41: **Expected Read Back 1111_1111 Data Pattern**

This is a sample of results for the Read Per-Bit DBI Deskew XSDB debug signals:

```
RDLVL_DESKEW_IDELAY_BYTE0_BIT0                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE0_BIT1                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE0_BIT2                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE0_BIT3                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE0_BIT4                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE0_BIT5                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE0_BIT6                 string true true 033
RDLVL_DESKEW_IDELAY_BYTE0_BIT7                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE1_BIT0                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE1_BIT1                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE1_BIT2                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE1_BIT3                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE1_BIT4                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE1_BIT5                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE1_BIT6                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE1_BIT7                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE2_BIT0                 string true true 033
RDLVL_DESKEW_IDELAY_BYTE2_BIT1                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE2_BIT2                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE2_BIT3                 string true true 028
```

```
RDLVL_DESKEW_IDELAY_BYTE2_BIT4                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE2_BIT5                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE2_BIT6                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE2_BIT7                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT0                 string true true 02f
RDLVL_DESKEW_IDELAY_BYTE3_BIT1                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE3_BIT2                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT3                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT4                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE3_BIT5                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE3_BIT6                 string true true 028
RDLVL_DESKEW_IDELAY_BYTE3_BIT7                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE4_BIT0                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE4_BIT1                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE4_BIT2                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE4_BIT3                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE4_BIT4                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE4_BIT5                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE4_BIT6                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE4_BIT7                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE5_BIT0                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE5_BIT1                 string true true 02a
RDLVL_DESKEW_IDELAY_BYTE5_BIT2                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE5_BIT3                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE5_BIT4                 string true true 028
RDLVL_DESKEW_IDELAY_BYTE5_BIT5                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE5_BIT6                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE5_BIT7                 string true true 026
RDLVL_DESKEW_IDELAY_BYTE6_BIT0                 string true true 028
RDLVL_DESKEW_IDELAY_BYTE6_BIT1                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE6_BIT2                 string true true 025
RDLVL_DESKEW_IDELAY_BYTE6_BIT3                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE6_BIT4                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE6_BIT5                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE6_BIT6                 string true true 032
RDLVL_DESKEW_IDELAY_BYTE6_BIT7                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE7_BIT0                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE7_BIT1                 string true true 02a
RDLVL_DESKEW_IDELAY_BYTE7_BIT2                 string true true 030
RDLVL_DESKEW_IDELAY_BYTE7_BIT3                 string true true 02d
RDLVL_DESKEW_IDELAY_BYTE7_BIT4                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE7_BIT5                 string true true 02a
RDLVL_DESKEW_IDELAY_BYTE7_BIT6                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE7_BIT7                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE8_BIT0                 string true true 029
RDLVL_DESKEW_IDELAY_BYTE8_BIT1                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE8_BIT2                 string true true 02b
RDLVL_DESKEW_IDELAY_BYTE8_BIT3                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE8_BIT4                 string true true 02e
RDLVL_DESKEW_IDELAY_BYTE8_BIT5                 string true true 02c
RDLVL_DESKEW_IDELAY_BYTE8_BIT6                 string true true 031
RDLVL_DESKEW_IDELAY_BYTE8_BIT7                 string true true 02f
RDLVL_DESKEW_NQTR_NIBBLE0                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE1                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE2                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE3                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE4                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE5                      string true true 000
RDLVL_DESKEW_NQTR_NIBBLE6                      string true true 001
```

```
RDLVL_DESKEW_NQTR_NIBBLE7                    string true true 002
RDLVL_DESKEW_NQTR_NIBBLE8                    string true true 000
RDLVL_DESKEW_NQTR_NIBBLE9                    string true true 000
RDLVL_DESKEW_NQTR_NIBBLE10                   string true true 000
RDLVL_DESKEW_NQTR_NIBBLE11                   string true true 000
RDLVL_DESKEW_NQTR_NIBBLE12                   string true true 000
RDLVL_DESKEW_NQTR_NIBBLE13                   string true true 002
RDLVL_DESKEW_NQTR_NIBBLE14                   string true true 001
RDLVL_DESKEW_NQTR_NIBBLE15                   string true true 000
RDLVL_DESKEW_NQTR_NIBBLE16                   string true true 000
RDLVL_DESKEW_NQTR_NIBBLE17                   string true true 000
RDLVL_DESKEW_PQTR_NIBBLE0                    string true true 000
RDLVL_DESKEW_PQTR_NIBBLE1                    string true true 003
RDLVL_DESKEW_PQTR_NIBBLE2                    string true true 000
RDLVL_DESKEW_PQTR_NIBBLE3                    string true true 001
RDLVL_DESKEW_PQTR_NIBBLE4                    string true true 002
RDLVL_DESKEW_PQTR_NIBBLE5                    string true true 000
RDLVL_DESKEW_PQTR_NIBBLE6                    string true true 000
RDLVL_DESKEW_PQTR_NIBBLE7                    string true true 000
RDLVL_DESKEW_PQTR_NIBBLE8                    string true true 003
RDLVL_DESKEW_PQTR_NIBBLE9                    string true true 002
RDLVL_DESKEW_PQTR_NIBBLE10                   string true true 001
RDLVL_DESKEW_PQTR_NIBBLE11                   string true true 004
RDLVL_DESKEW_PQTR_NIBBLE12                   string true true 001
RDLVL_DESKEW_PQTR_NIBBLE13                   string true true 000
RDLVL_DESKEW_PQTR_NIBBLE14                   string true true 000
RDLVL_DESKEW_PQTR_NIBBLE15                   string true true 000
RDLVL_DESKEW_PQTR_NIBBLE16                   string true true 000
RDLVL_DESKEW_PQTR_NIBBLE17                   string true true 002
```

Expected Results

- Look at the individual IDELAY taps for each bit. The IDELAY taps should only vary by 0 to 20 taps, and is dependent on PCB trace delays. For Deskew, the IDELAY taps are typically in the 50 to 70 tap range, while PQTR and NQTR are usually in the 0 to 5 tap range.
- Determine if any bytes completed successfully. The per-bit algorithm sequentially steps through each DQS byte.

Hardware Measurements

1. Probe the write commands and read commands at the memory:
   - Write = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 1; `act_n` = 1
   - Read = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 0; `act_n` = 1
2. Probe the data and DBI pins to check for data being returned from the DRAM.
3. Probe the writes checking the signal level of the write DQS and the write DQ.
4. Probe the DBI pin which should be deasserted during the write burst. The DBI pin should not be asserted because DBI write should be off.
5. Probe the read burst after the write and check if the expected data pattern is being returned.

6. Check for floating address pins if the expected data is not returned.
7. Check for any stuck-at level issues on DQ/DBI pins whose signal level does not change. If at all possible probe at the receiver to check termination and signal integrity.
8. Check the DBG port signals and the full read data and comparison result to check the data in general interconnect. The calibration algorithm has RTL logic issue the commands and check the data.
   Check if the **`dbg_rd_valid`** aligns with the data pattern or is off (which can indicate an issue with DQS gate calibration). Set up a trigger when the error gets asserted to capture signals in the hardware debugger for analysis.
9. Re-check results from DQS gate or other previous calibration stages. Compare passing byte lanes against failing byte lanes for previous stages of calibration. If a failure occurs during simple pattern calibration, check the values found during deskew for example.
10. All of the data comparison for read deskew occurs in the general interconnect, so it can be useful to pull in the debug data in the hardware debugger and take a look at what the data looks like coming back as taps are adjusted, see Figure 38-42. The screen captures are from simulation, with a small burst of five reads. Look at **`dbg_rd_data`**, **`dbg_rd_data_cmp`**, and **`dbg_rd_valid`**.
11. Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the Read DBI Deskew trigger to **`cal_r*_status[8]`** = R (rising edge). To view each byte, add an additional trigger on **`dbg_cmp_byte`** and set to the byte of interest. The following simulation example shows how the debug signals should behave during successful Read DBI Deskew.

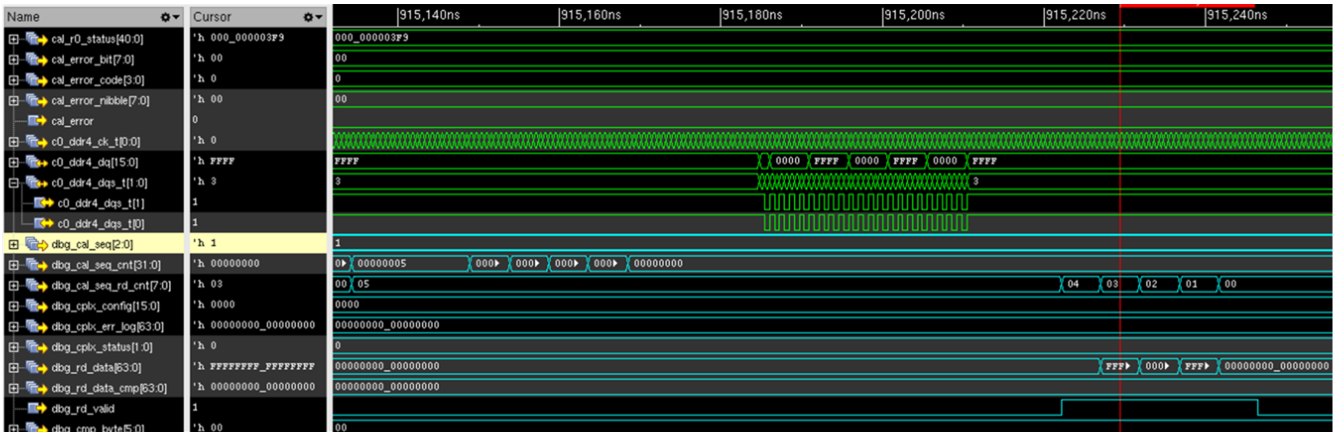


Figure 38-42: **RTL Debug Signals during Read DBI Deskew (No Error)**

12. After failure during this stage of calibration, the design goes into a continuous loop of read commands to allow board probing.

Debugging Read DQS Centering (Simple/MPR) Failures

When the data is deskewed, the PQTR/NQTR delays need to be adjusted to center in the aggregate data valid window for a given nibble. The DRAM MPR register is used to provide the data pattern for centering. Therefore, the pattern changes each bit time and does not rely on being written into the DRAM first, eliminating some uncertainty. The simple clock pattern is used to allow for the same pattern checking for DDR3 and DDR4. Gaps in the reads to the DRAM are used to stress the initial centering to incorporate the effects of ISI on the first DQS pulse as shown in Figure 38-43.

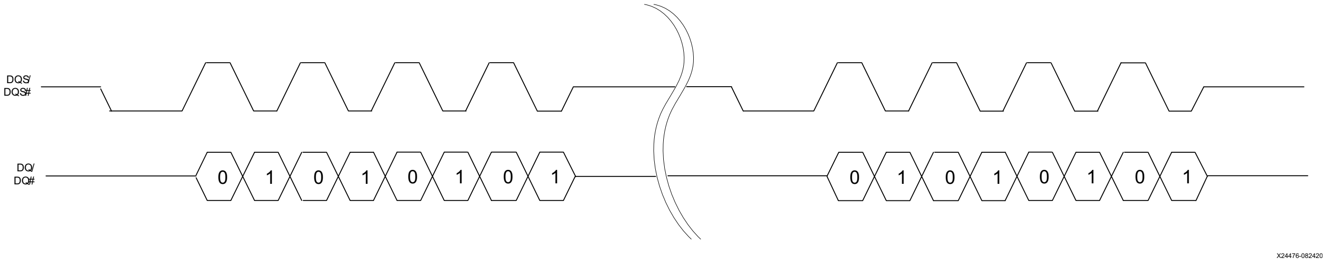


Figure 38-43: **Gap between MPR Reads**

To properly account for jitter on the data and clock returned from the DRAM, multiple data samples are taken at a given tap value. 64 read bursts are used in hardware while five are used in simulation. More samples mean finding the best alignment in the data valid window.

Given that the PHY has two capture strobes PQTR/NQTR that need to be centered independently yet moved together, calibration needs to take special care to ensure the clocks stay in a certain phase relationship with one another.

The data and PQTR/NQTR delays start with the value found during deskew. Data is first delayed with IDELAY such that both the PQTR and NQTR clocks start out just to the left of the data valid window for all bits in a given nibble so the entire read window can be scanned with each clock (Figure 38-44, RDLVL_IDELAY_VALUE_Rank_Byte_Bit). Scanning the window with the same delay element and computing the center with that delay element helps to minimize uncertainty in tap resolution that might arise from using different delay lines to find the edges of the read window.

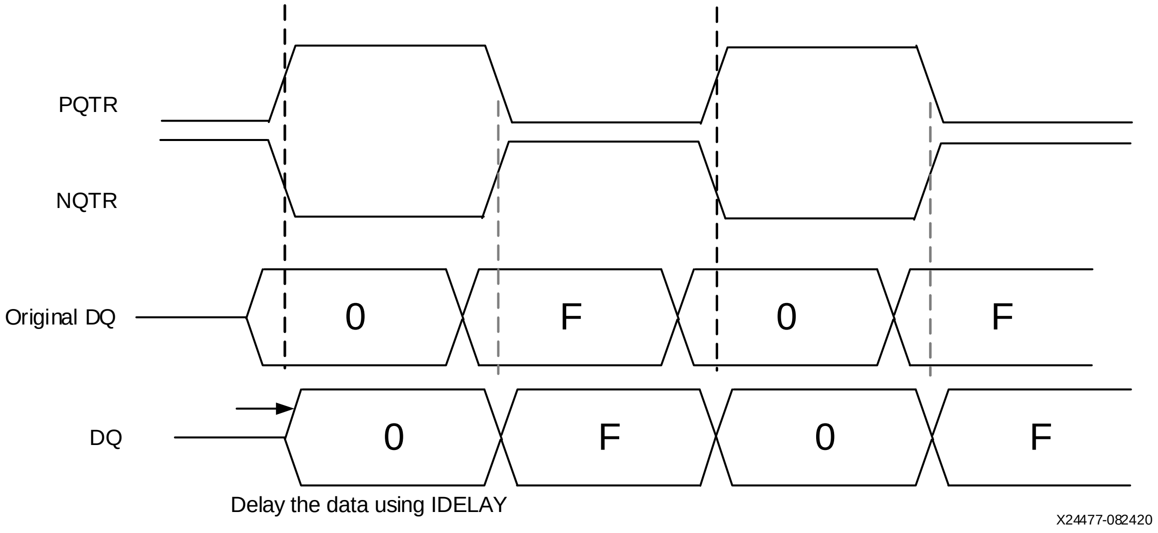


Figure 38-44: **Delay DQ Thus PQTR and NQTR in Failing Region**

At the start of training, the PQTR/NQTR and data are roughly edge aligned, but because the pattern is different from the deskew step the edge might have changed a bit. Also, during deskew the aggregate edge for both PQTR/NQTR is found while you want to find a separate edge for each clock.

After making sure both PQTR/NQTR start outside the data valid region, the clocks are incremented to look for the passing region (Figure 38-45). Rising edge data is checked for PQTR while falling edge data is checked for NQTR, with a separate check being kept to indicate where the passing region/falling region is for each clock.

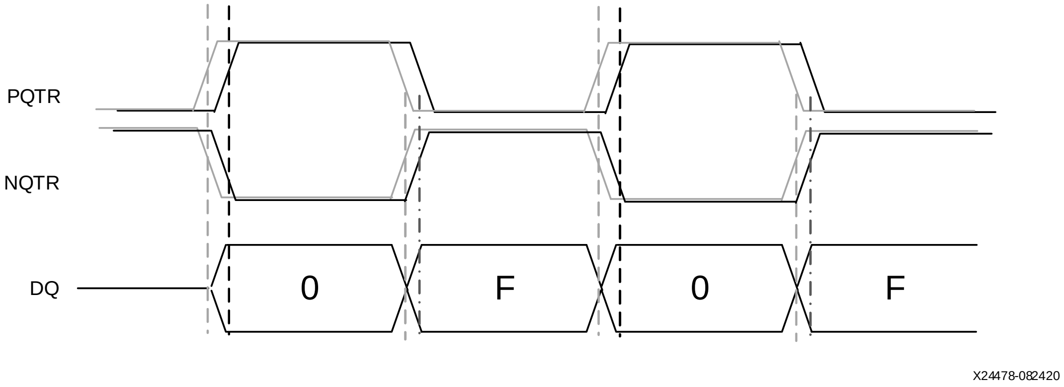


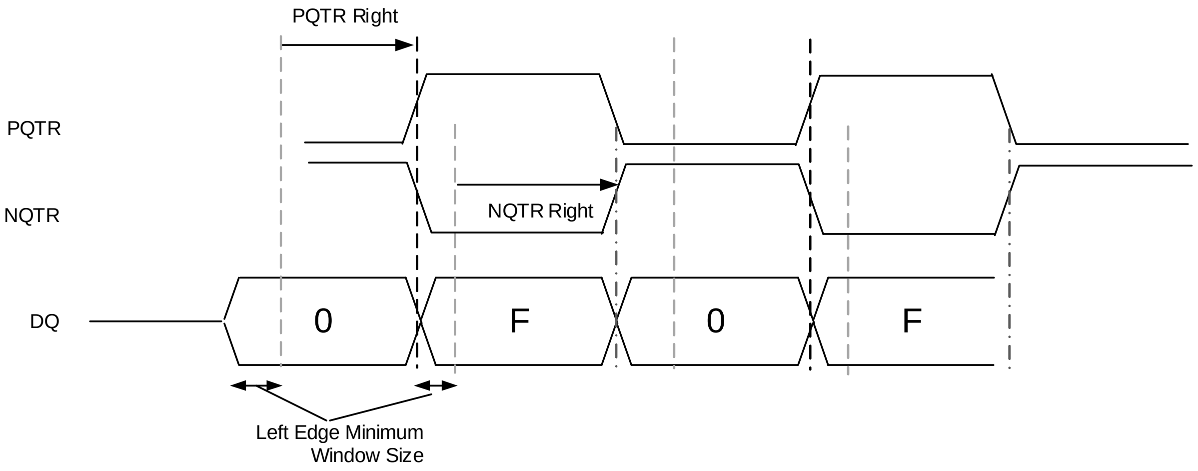
Figure 38-45: **PQTR and NQTR Delayed to Find Passing Region (Left Edge)**

When searching for the edge, a minimum window size of 10 is used to guarantee the noise region has been cleared and the true edge is found. The PQTR/NQTR delays are increased past the initial passing point until the minimum window size is found before the left edge is declared as found. If the minimum window is not located across the entire tap range for either clock, an error is asserted.

After the left edge is found (RDLVL_PQTR_LEFT_Rank_Nibble, RDLVL_NQTR_LEFT_Rank_Nibble), the right edge of the data valid window can be searched starting from the left edge + minimum window size. A minimum window size is not used when searching for the right edge, as the starting point already guarantees a minimum window size has been met.

Again, the PQTR/NQTR delays are incremented together and checked for error independently to keep track of the right edge of the window. Because the data from the PQTR domain is transferred into the NQTR clock domain in the XIPHY, the edge for NQTR is checked first, keeping track of the results for PQTR along the way (Figure 38-46).

When the NQTR edge is located, a flag is checked to see if the PQTR edge is found as well. If the PQTR edge was not found, the PQTR delay continues to search for the edge, while the NQTR delay stays at its right edge (RDLVL_PQTR_RIGHT_Rank_Nibble, RDLVL_NQTR_RIGHT_Rank_Nibble). For simulation, the right edge detection is sped up by having the delays adjusted by larger than one tap at a time.



X24479-082420

Figure 38-46: **PQTR and NQTR Delayed to Find Failing Region (Right Edge)**

After both rising and falling edge windows are found, the final center point is calculated based on the left and right edges for each clock. The final delay for each clock (RDLVL_PQTR_CENTER_Rank_Nibble, RDLVL_NQTR_CENTER_Rank_Nibble) is computed by:

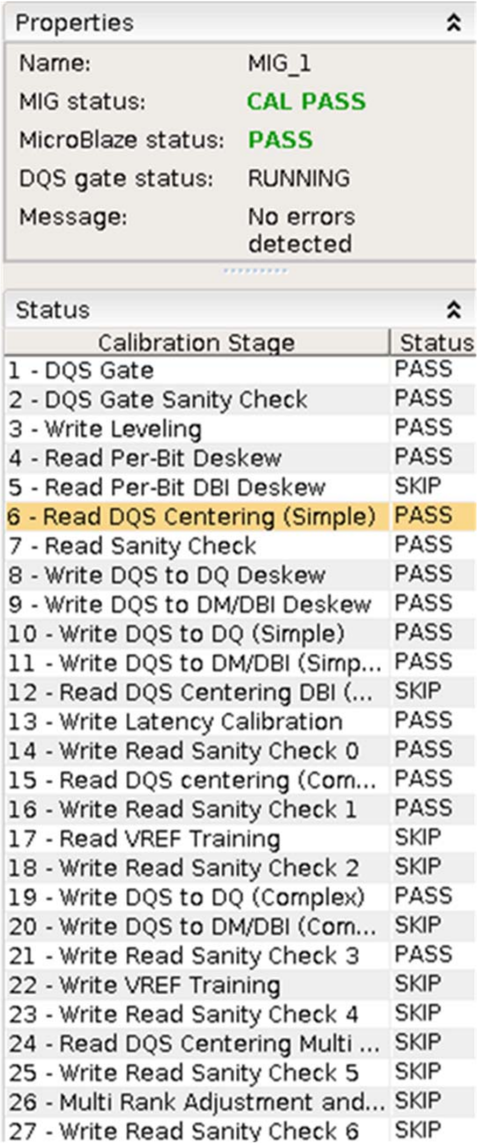
left + ((right – left)/2).

For multi-rank systems deskew only runs on the first rank, while read DQS centering using the PQTR/NQTR runs on all ranks. After calibration is complete for all ranks, for a given DQ bit the IDELAY is set to the center of the range of values seen for all ranks (RDLVL_IDELAY_FINAL_BYTE_BIT). The PQTR/NQTR final value is also computed based on the range of values seen between all of the ranks (RDLVL_PQTR_CENTER_FINAL_NIBBLE, RDLVL_NQTR_CENTER_FINAL_NIBBLE).

IMPORTANT: *For multi-rank systems, there must be overlap in the read window computation. Also, there is a limit in the allowed skew between ranks, see the PCB Guidelines for DDR3 in Chapter 4 and PCB Guidelines for DDR4 in Chapter 4.*

Debug

To determine the status of Read MPR DQS Centering Calibration, click the **Read DQS Centering (Simple)** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simp...	PASS
12 - Read DQS Centering DBI (...	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Com...	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Com...	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi ...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-47: **Memory IP XSDB Debug GUI Example – Read DQS Centering (Simple)**

The status of Read MPR DQS Centering can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-19. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-19: **DDR_CAL_ERROR Decode for Read Leveling Calibration**

Read DQS Centering DDR_CAL_ ERROR_ CODE	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	Nibble	Bit	No valid data found for a given bit in the nibble	Check the BUS_DATA_BURST fields in XSDB. Check the dbg_rd_data, dbg_rd_data_cmp, and dbg_expected_data signals in the ILA. Check the pinout and look for any STUCK-AT-BITs, check VRP resistor, VREF resistor. Check the RDLVL_DESKEW_* fields of XSDB to check if any delays are much larger/smaller than others.
0x2	Nibble	Bit	Could not find the left Edge (error condition) to determine window size	Check for a mapping issue. This usually implies a delay is not moving when it should. Check the connections going to the XIPHY and ensure the correct RIU is selected based on the byte being adjusted.
0xF	Nibble	Bit	Timeout error waiting for read data to return	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-20 shows the signals and values adjusted or used during the Read MPR DQS Centering stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-20: **Signals of Interest for Read Leveling Calibration**

Signal	Usage	Signal Description
RDLVL_PQTR_LEFT_RANK*_NIBBLE*	One per rank per nibble	Read leveling PQTR tap position when left edge of read data valid window is detected (simple pattern).
RDLVL_NQTR_LEFT_RANK*_NIBBLE*	One per rank per nibble	Read leveling NQTR tap position when left edge of read data valid window is detected (simple pattern).
RDLVL_PQTR_RIGHT_RANK*_NIBBLE*	One per rank per nibble	Read leveling PQTR tap position when right edge of read data valid window is detected (simple pattern).
RDLVL_NQTR_RIGHT_RANK*_NIBBLE*	One per rank per nibble	Read leveling NQTR tap position when right edge of read data valid window is detected (simple pattern).

Table 38-20: **Signals of Interest for Read Leveling Calibration *(Cont'd)***

Signal	Usage	Signal Description
RDLVL_PQTR_CENTER_RANK*_NIBBLE*	One per rank per nibble	Read leveling PQTR center tap position found at the end of read DQS centering (simple pattern).
RDLVL_NQTR_CENTER_RANK*_NIBBLE*	One per rank per nibble	Read leveling NQTR center tap position found at the end of read DQS centering (simple pattern).
RDLVL_IDELAY_VALUE_RANK*_BYTE*_BIT*	One per rank per Bit	Read leveling IDELAY delay value found during per bit read DQS centering (simple pattern).
RDLVL_IDELAY_DBI_RANK*_BYTE*	One per rank per Byte	Reserved
BISC_ALIGN_PQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_ALIGN_NQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_PQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_PQTR – BISC_ALIGN_PQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_PQTR – BISC_ALIGN_PQTR). Useful for error code 0x6.
BISC_NQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_NQTR – BISC_ALIGN_NQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_NQTR – BISC_ALIGN_NQTR). Useful for error code 0x6.
RDLVL_PQTR_FINAL_NIBBLE*	One per nibble	Final Read leveling PQTR tap position from the XIPHY.
RDLVL_NQTR_FINAL_NIBBLE*	One per nibble	Final Read leveling NQTR tap position from the XIPHY.
RDLVL_IDELAY_FINAL_BYTE*_BIT*	One per Bit	Final IDELAY tap position from the XIPHY.

Table 38-20: **Signals of Interest for Read Leveling Calibration *(Cont'd)***

Signal	Usage	Signal Description
RDLVL_IDELAY_DBI_FINAL_BYTE*	One per Byte	Reserved
BUS_DATA_BURST (2014.3+)		When a failure occurs during simple pattern read training, some data is saved to indicate what the data looks like for a byte across some tap settings for a given byte the failure occurred for (DQ IDELAY is left wherever the algorithm left it). Read DQS centering (Figure 38-48): BUS_DATA_BURST_0 holds a single burst of data when PQTR/NQTR set to 0 taps. BUS_DATA_BURST_1 holds a single burst of data when PQTR/NQTR set to 90°. BUS_DATA_BURST_2 holds a single burst of data when PQTR/NQTR set to 180°. BUS_DATA_BURST_3 holds a single burst of data when PQTR/NQTR set to 270°.

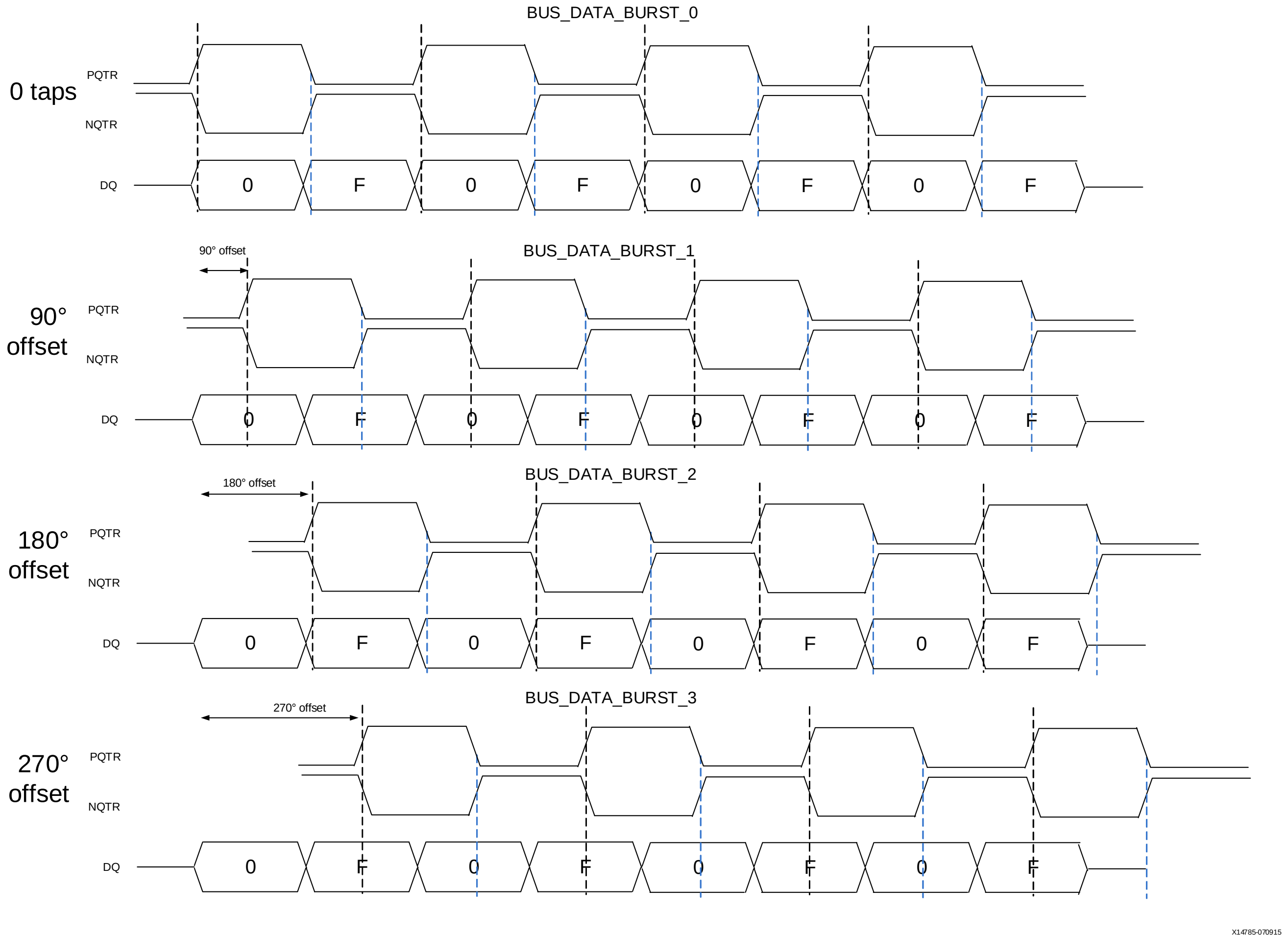


Figure 38-48: **Read DQS Centering Error (XSDB BUS_DATA_BURST)**

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figure 38-49 and Figure 38-50 are examples of how the data is converted.

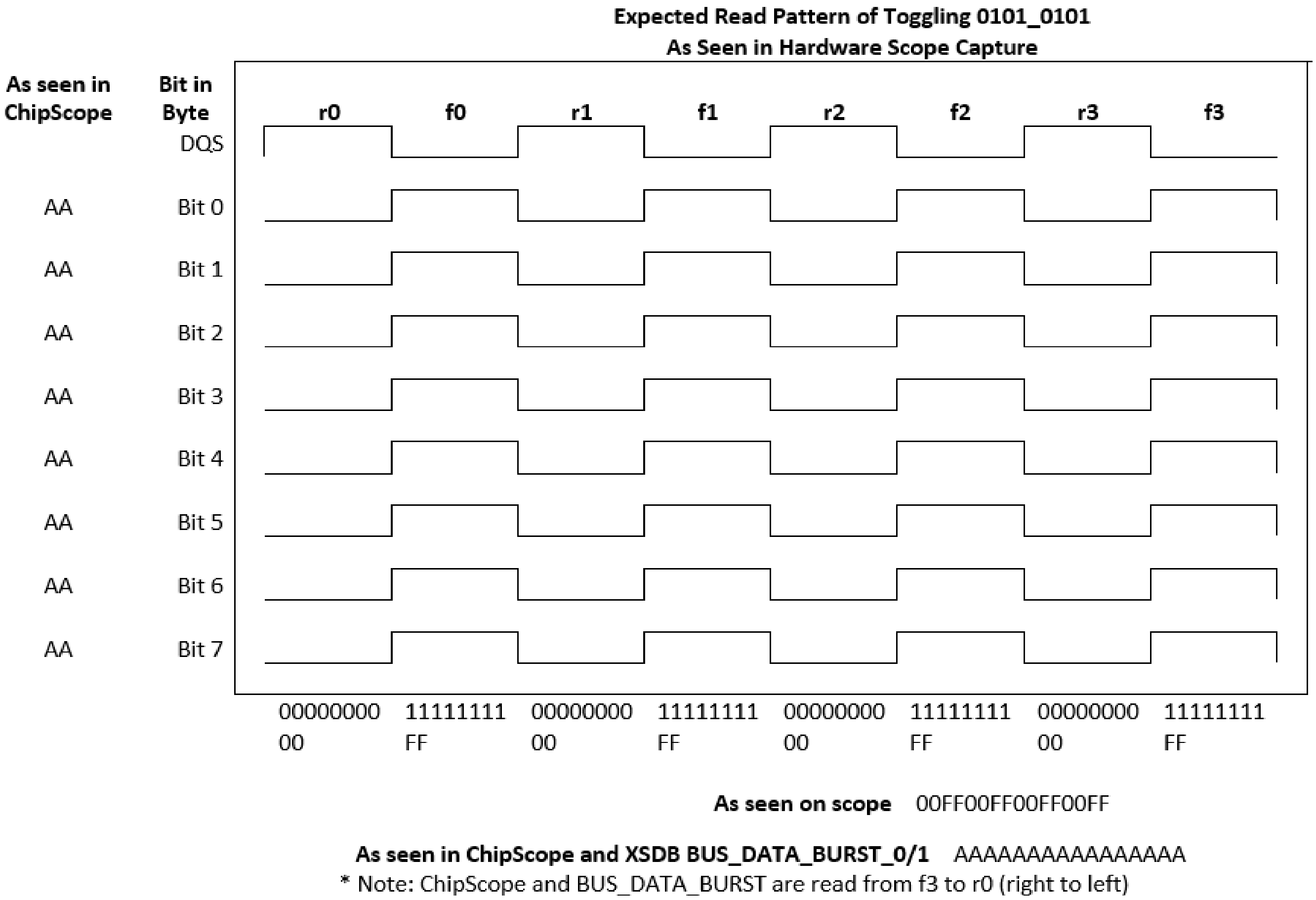


Figure 38-49: **Expected Read Pattern of Toggling 0101_0101**

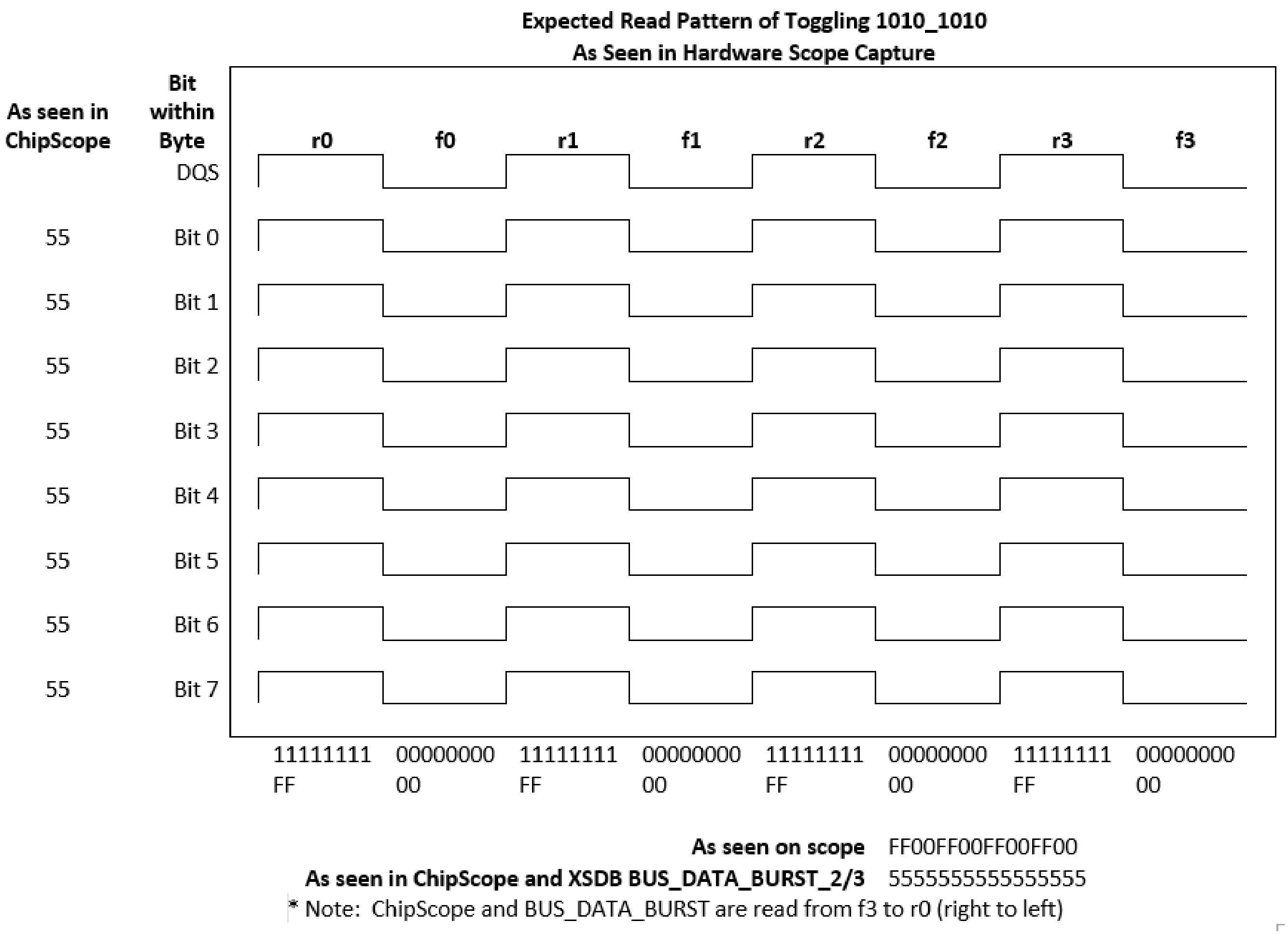


Figure 38-50: **Expected Read Pattern of Toggling 1010_1010**

This is a sample of results for Read MPR DQS Centering using the Memory IP Debug GUI within the Hardware Manager.

Note: Either the "Table" or "Chart" view can be used to look at the window.

Figure 38-51 and Figure 38-52 are screen captures from 2015.1 and might vary from the current version.

Figure 38-51: **Example Read Calibration Margin from Memory IP Debug GUI**

This is a sample of results for the Read Per-Bit Deskew XSDB debug signals:

```
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT0          string true true 042
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT1          string true true 042
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT2          string true true 042
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT3          string true true 045
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT4          string true true 03a
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT5          string true true 03e
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT6          string true true 040
RDLVL_IDELAY_VALUE_RANK0_BYTE0_BIT7          string true true 03d
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT0          string true true 038
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT1          string true true 03d
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT2          string true true 03e
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT3          string true true 039
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT4          string true true 03a
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT5          string true true 034
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT6          string true true 03c
RDLVL_IDELAY_VALUE_RANK0_BYTE1_BIT7          string true true 033
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT0          string true true 041
```

```
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT1              string true true 042
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT2              string true true 031
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT3              string true true 040
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT4              string true true 040
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT5              string true true 033
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT6              string true true 036
RDLVL_IDELAY_VALUE_RANK0_BYTE2_BIT7              string true true 031
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT0              string true true 038
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT1              string true true 038
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT2              string true true 035
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT3              string true true 035
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT4              string true true 036
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT5              string true true 03c
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT6              string true true 038
RDLVL_IDELAY_VALUE_RANK0_BYTE3_BIT7              string true true 037
RDLVL_NQTR_CENTER_RANK0_NIBBLE0                  string true true 03c
RDLVL_NQTR_CENTER_RANK0_NIBBLE1                  string true true 03a
RDLVL_NQTR_CENTER_RANK0_NIBBLE2                  string true true 03a
RDLVL_NQTR_CENTER_RANK0_NIBBLE3                  string true true 039
RDLVL_NQTR_CENTER_RANK0_NIBBLE4                  string true true 044
RDLVL_NQTR_CENTER_RANK0_NIBBLE5                  string true true 038
RDLVL_NQTR_CENTER_RANK0_NIBBLE6                  string true true 039
RDLVL_NQTR_CENTER_RANK0_NIBBLE7                  string true true 03b
RDLVL_NQTR_LEFT_RANK0_NIBBLE0                    string true true 009
RDLVL_NQTR_LEFT_RANK0_NIBBLE1                    string true true 006
RDLVL_NQTR_LEFT_RANK0_NIBBLE2                    string true true 00b
RDLVL_NQTR_LEFT_RANK0_NIBBLE3                    string true true 008
RDLVL_NQTR_LEFT_RANK0_NIBBLE4                    string true true 010
RDLVL_NQTR_LEFT_RANK0_NIBBLE5                    string true true 006
RDLVL_NQTR_LEFT_RANK0_NIBBLE6                    string true true 006
RDLVL_NQTR_LEFT_RANK0_NIBBLE7                    string true true 00a
RDLVL_NQTR_RIGHT_RANK0_NIBBLE0                   string true true 06f
RDLVL_NQTR_RIGHT_RANK0_NIBBLE1                   string true true 06e
RDLVL_NQTR_RIGHT_RANK0_NIBBLE2                   string true true 06a
RDLVL_NQTR_RIGHT_RANK0_NIBBLE3                   string true true 06a
RDLVL_NQTR_RIGHT_RANK0_NIBBLE4                   string true true 078
RDLVL_NQTR_RIGHT_RANK0_NIBBLE5                   string true true 06a
RDLVL_NQTR_RIGHT_RANK0_NIBBLE6                   string true true 06c
RDLVL_NQTR_RIGHT_RANK0_NIBBLE7                   string true true 06d
RDLVL_PQTR_CENTER_RANK0_NIBBLE0                  string true true 040
RDLVL_PQTR_CENTER_RANK0_NIBBLE1                  string true true 040
RDLVL_PQTR_CENTER_RANK0_NIBBLE2                  string true true 037
RDLVL_PQTR_CENTER_RANK0_NIBBLE3                  string true true 03a
RDLVL_PQTR_CENTER_RANK0_NIBBLE4                  string true true 043
RDLVL_PQTR_CENTER_RANK0_NIBBLE5                  string true true 037
RDLVL_PQTR_CENTER_RANK0_NIBBLE6                  string true true 03e
RDLVL_PQTR_CENTER_RANK0_NIBBLE7                  string true true 040
RDLVL_PQTR_LEFT_RANK0_NIBBLE0                    string true true 013
RDLVL_PQTR_LEFT_RANK0_NIBBLE1                    string true true 015
RDLVL_PQTR_LEFT_RANK0_NIBBLE2                    string true true 008
RDLVL_PQTR_LEFT_RANK0_NIBBLE3                    string true true 00b
RDLVL_PQTR_LEFT_RANK0_NIBBLE4                    string true true 018
RDLVL_PQTR_LEFT_RANK0_NIBBLE5                    string true true 008
RDLVL_PQTR_LEFT_RANK0_NIBBLE6                    string true true 00d
RDLVL_PQTR_LEFT_RANK0_NIBBLE7                    string true true 012
RDLVL_PQTR_RIGHT_RANK0_NIBBLE0                   string true true 06e
RDLVL_PQTR_RIGHT_RANK0_NIBBLE1                   string true true 06c
RDLVL_PQTR_RIGHT_RANK0_NIBBLE2                   string true true 066
RDLVL_PQTR_RIGHT_RANK0_NIBBLE3                   string true true 06a
```

```
RDLVL_PQTR_RIGHT_RANK0_NIBBLE4                    string true true 06f
RDLVL_PQTR_RIGHT_RANK0_NIBBLE5                    string true true 067
RDLVL_PQTR_RIGHT_RANK0_NIBBLE6                    string true true 06f
RDLVL_PQTR_RIGHT_RANK0_NIBBLE7                    string true true 06f
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT0                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT1                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT2                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT3                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT4                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT5                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT6                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE0_BIT7                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT0                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT1                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT2                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT3                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT4                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT5                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT6                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE1_BIT7                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT0                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT1                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT2                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT3                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT4                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT5                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT6                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE2_BIT7                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT0                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT1                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT2                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT3                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT4                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT5                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT6                string true true 000
MULTI_RANK_RDLVL_IDELAY_BYTE3_BIT7                string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE0                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE1                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE2                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE3                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE4                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE5                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE6                     string true true 000
MULTI_RANK_RDLVL_NQTR_NIBBLE7                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE0                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE1                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE2                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE3                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE4                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE5                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE6                     string true true 000
MULTI_RANK_RDLVL_PQTR_NIBBLE7                     string true true 000
BISC_ALIGN_NQTR_NIBBLE0                           string true true 000
BISC_ALIGN_NQTR_NIBBLE1                           string true true 000
BISC_ALIGN_NQTR_NIBBLE2                           string true true 000
BISC_ALIGN_NQTR_NIBBLE3                           string true true 000
BISC_ALIGN_NQTR_NIBBLE4                           string true true 000
BISC_ALIGN_NQTR_NIBBLE5                           string true true 000
BISC_ALIGN_NQTR_NIBBLE6                           string true true 000
```

```
BISC_ALIGN_NQTR_NIBBLE7                        string true true 000
BISC_ALIGN_PQTR_NIBBLE0                        string true true 007
BISC_ALIGN_PQTR_NIBBLE1                        string true true 004
BISC_ALIGN_PQTR_NIBBLE2                        string true true 006
BISC_ALIGN_PQTR_NIBBLE3                        string true true 005
BISC_ALIGN_PQTR_NIBBLE4                        string true true 005
BISC_ALIGN_PQTR_NIBBLE5                        string true true 004
BISC_ALIGN_PQTR_NIBBLE6                        string true true 004
BISC_ALIGN_PQTR_NIBBLE7                        string true true 004
BISC_NQTR_NIBBLE0                              string true true 036
BISC_NQTR_NIBBLE1                              string true true 033
BISC_NQTR_NIBBLE2                              string true true 037
BISC_NQTR_NIBBLE3                              string true true 035
BISC_NQTR_NIBBLE4                              string true true 037
BISC_NQTR_NIBBLE5                              string true true 036
BISC_NQTR_NIBBLE6                              string true true 036
BISC_NQTR_NIBBLE7                              string true true 036
BISC_PQTR_NIBBLE0                              string true true 038
BISC_PQTR_NIBBLE1                              string true true 036
BISC_PQTR_NIBBLE2                              string true true 038
BISC_PQTR_NIBBLE3                              string true true 035
BISC_PQTR_NIBBLE4                              string true true 037
BISC_PQTR_NIBBLE5                              string true true 037
BISC_PQTR_NIBBLE6                              string true true 035
BISC_PQTR_NIBBLE7                              string true true 036
```

Expected Results

- Look at the individual PQTR/NQTR tap settings for each nibble. The taps should only vary by 0 to 20 taps. Use the BISC values to compute the estimated bit time in taps.
    - For example, Byte 7 Nibble 0 in Figure 38-52 is shifted and smaller compared to the remaining nibbles. This type of result is not expected. For this specific example, the FPGA was not properly loaded into the socket.

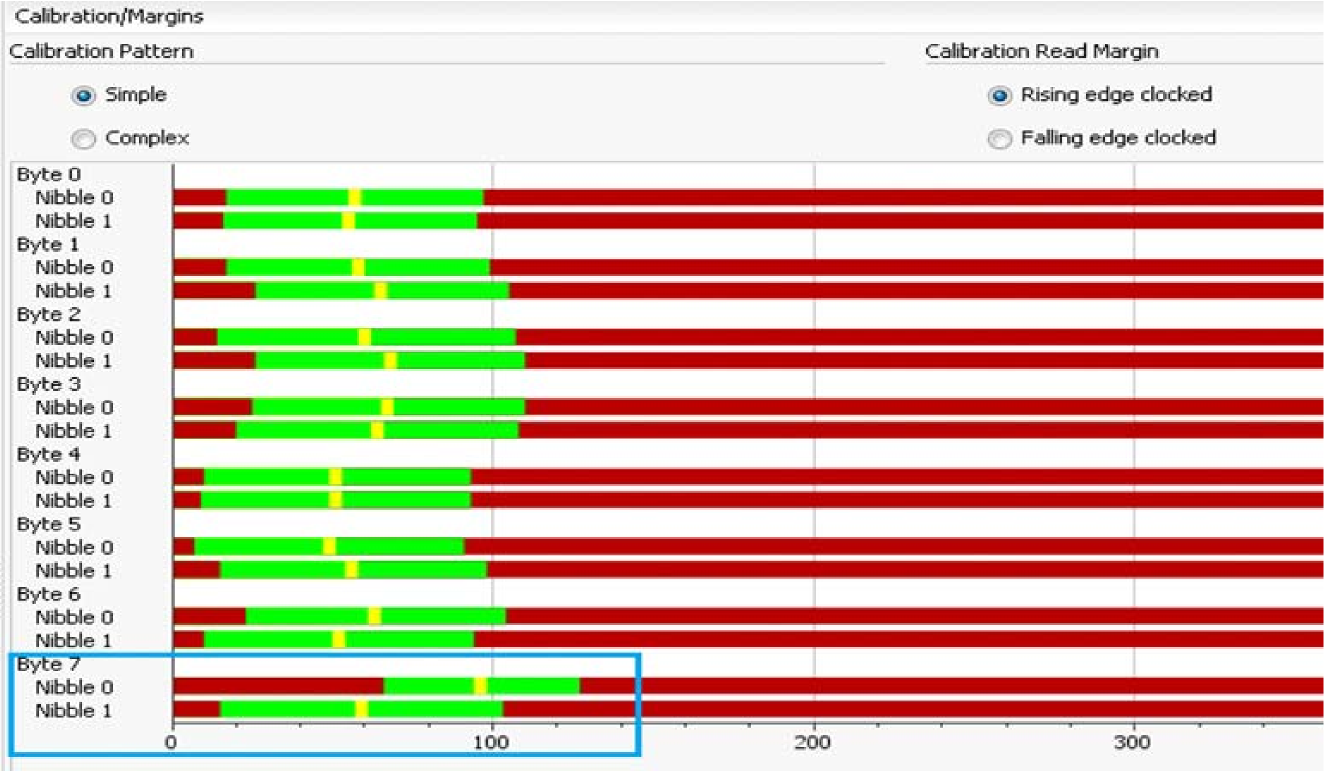


Figure 38-52: **Example of Suspicious Calibration Read Margin**

- Determine if any bytes completed successfully. The read DQS Centering algorithm sequentially steps through each DQS byte group detecting the capture edges.
- To analyze the window size in ps, see the Determining Window Size in ps, page 773. In some cases, simple pattern calibration might show a better than ideal rise or fall window. Because a simple pattern (clock pattern) is used, it is possible for the rising edge clock to always find the same value (for example, 1) and the falling edge to always find the opposite (for example, 0). This can occur due to a non-ideal starting V_{REF} value which causes duty cycle distortion making the rise or fall larger than the other. If the rise and fall window sizes are added together and compared against the expected clock cycle time, the result should be more reasonable.

  As a general rule of thumb, the window size for a healthy system should be ≥ 30% of the expected UI size.

Hardware Measurements

1. Using high quality probes and scope, probe the address/command to ensure the load register command to the DRAM that enables MPR was correct. To enable the MPR, a Mode register set (MRS) command is issued to the MR3 register with bit A2 = 1. To make this measurement, bring a scope trigger to an I/O based on the following conditions:
    - `cal_r*_status[9]` = R (rising edge) && `dbg_rd_valid` = 1'b0 && `cal_seq_cnt[2:0] = 3`b0`

  - To view each byte, add an additional trigger on **`dbg_cmp_byte`** and set to the byte of interest.

  Within this capture, A2 (must be 1) and **`we_n`** (must be 0).

2. Probe the read commands at the memory:
    - Read = **`cs_n`** = 1; **`ras_n`** = 0; **`cas_n`** = 1; **`we_n`** = 0; **`act_n`** = 1 (DDR4 only)
3. Probe a data pin to check for data being returned from the DRAM.
4. Probe the read burst and check if the expected data pattern is being returned.
5. Check for floating address pins if the expected data is not returned.
6. Check for any stuck-at level issues on DQ pins whose signal level does not change. If at all possible probe at the receiver to check termination and signal integrity.
7. Check the DBG port signals and the full read data and comparison result to check the data in general interconnect. The calibration algorithm has RTL logic issue the commands and check the data. Check if the **`dbg_rd_valid`** aligns with the data pattern or is OFF (which can indicate an issue with DQS gate calibration). Set up a trigger when the error gets asserted to capture signals in the hardware debugger for analysis.
8. Re-check results from DQS gate or other previous calibration stages. Compare passing byte lanes against failing byte lanes for previous stages of calibration. If a failure occurs during simple pattern calibration, check the values found during deskew for example.
9. All of the data comparison for read DQS centering occurs in the general interconnect, so it can be useful to pull in the debug data in the hardware debugger and take a look at what the data looks like coming back as taps are adjusted, see Figure 38-53 and Figure 38-54. Screenshots shown are from simulation, with a small burst of five reads. Look at **`dbg_rd_data`**, **`dbg_rd_data_cmp`**, and **`dbg_rd_valid`**.
10. Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the Read Centering trigger to (**`cal_r*_status[10]`** = R (rising edge) && **`dbg_rd_valid`** = 1’b0 && **`cal_seq_cnt[2:0] = 3`b0`**). To view each byte, add an additional trigger on **`dbg_cmp_byte`** and set to the byte of interest. The following simulation example shows how the debug signals should behave during successful Read DQS Centering.

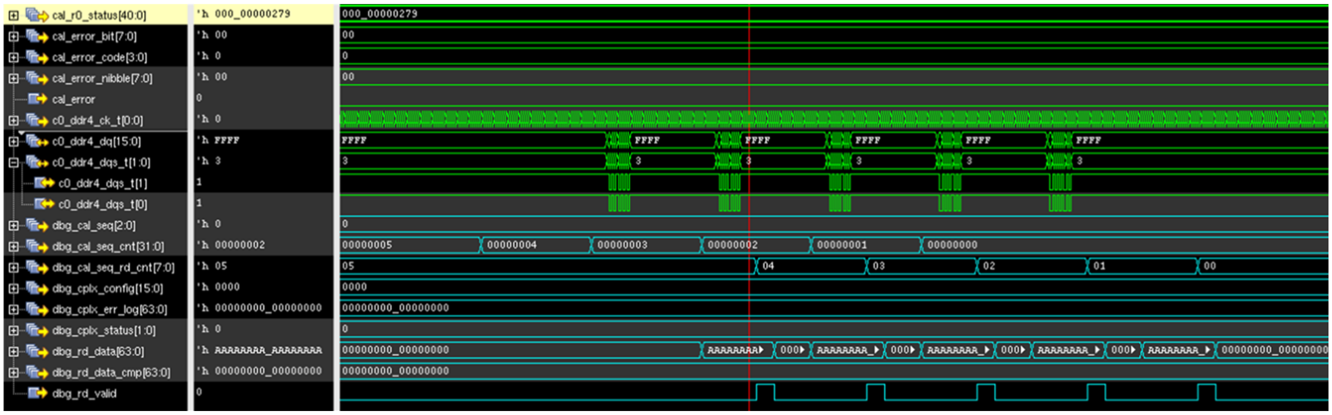


Figure 38-53: **RTL Debug Signals during Read DQS Centering (No Error)**

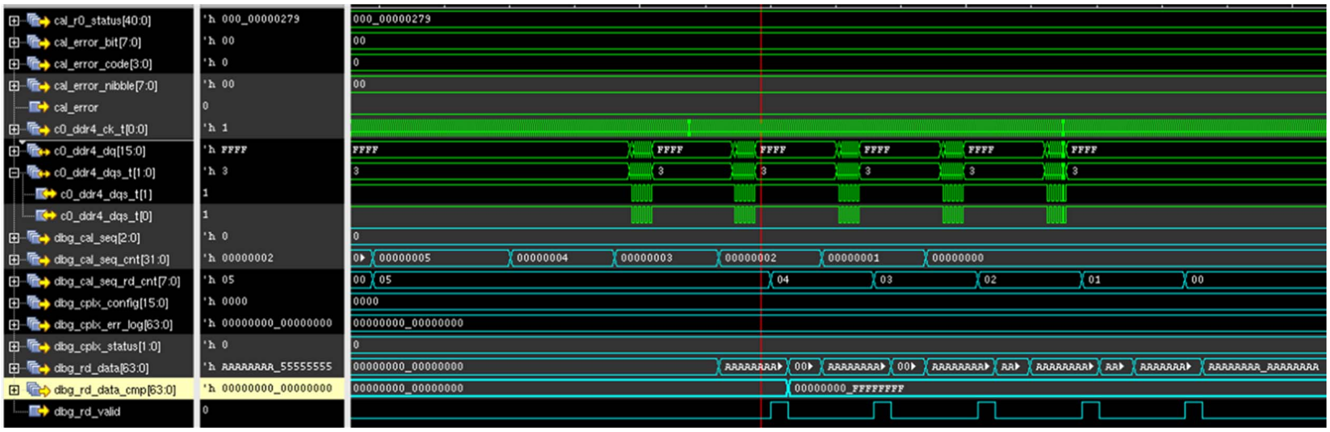


Figure 38-54: **RTL Debug Signals during Read DQS Centering (Error Case Shown)**

11. After failure during this stage of calibration, the design goes into a continuous loop of read commands to allow board probing.

Write Calibration Overview

Note: The calibration step is only enabled for the first rank in a multi-rank system.

The DRAM requires the write DQS to be center-aligned with the DQ to ensure maximum write margin. Initially the write DQS is set to be roughly 90° out of phase with the DQ using the XIPHY TX_DATA_PHASE set for the DQS. The TX_DATA_PHASE is an optional per-bit adjustment that uses a fast internal XIPHY clock to generate a 90° offset between bits. The DQS and DQ ODELAY are used to fine tune the 90° phase alignment to ensure maximum margin at the DRAM.

A simple clock pattern of `10101010` is used initially because the write latency has not yet been determined. Due to fly-by routing on the PCB/DIMM module, the command to data timing is unknown until the next stage of calibration. Just as in read per-bit deskew when issuing a write to the DRAM, the DQS and DQ toggles for eight clock cycles before and after the expected write latency. This is used to ensure the data is written into the DRAM even if the command-to-write data relationship is still unknown. Write DQS-to-DQ is completed in two stages, per-bit deskew and DQS centering.

Debugging Write Per-Bit Deskew Failures

Initially all DQ bits have the same ODELAY setting based on the write leveling results, but the ODELAY for each bit might need to be adjusted to account for skew between bits. Figure 38-55 shows an example of the initial timing relationship between a write DQS and DQ.

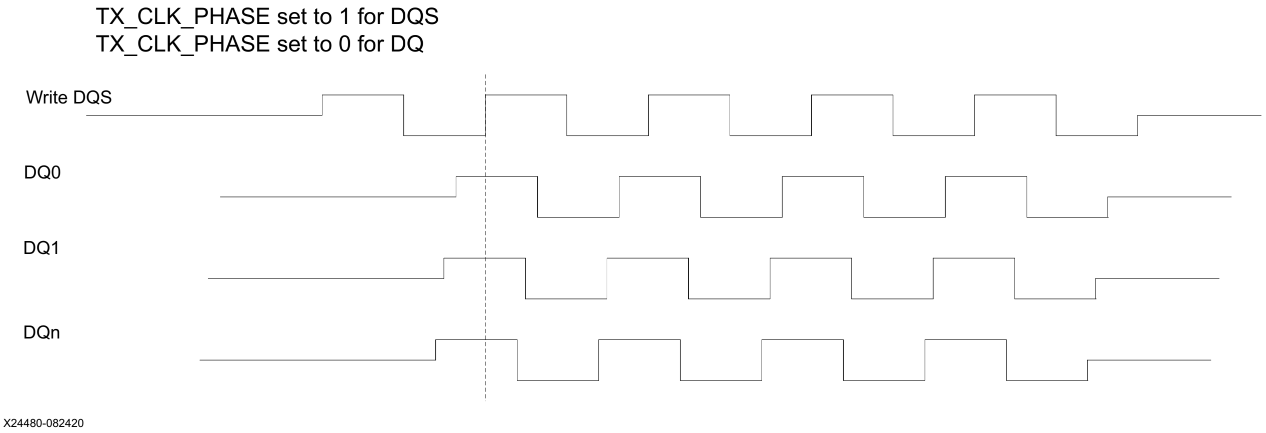


Figure 38-55: **Initial Write DQS and DQ with Skew between Bits**

1. Set TX_DATA_PHASE to 1 for DQ to add the 90° shift on the DQS relative to the DQ for a given byte (Figure 38-56). The data read back on some DQ bits are `10101010` while other DQ bits might be `01010101`.

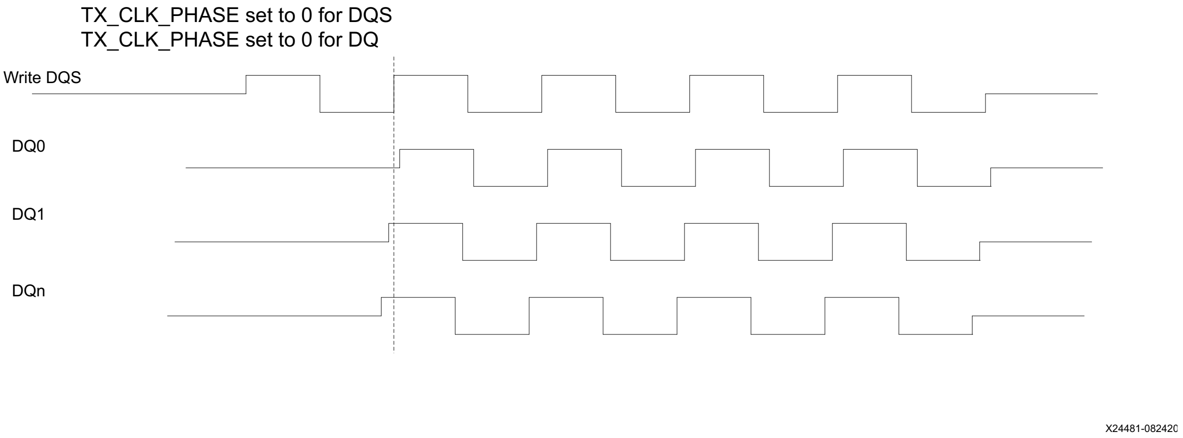


Figure 38-56: **Add 90° Shift on DQ**

2. If all the data for the byte does not match the expected data pattern, increment DQS ODELAY one tap at a time until the expected data pattern is found on all bits and save the delay as WRITE_DQS_TO_DQ_DESKEW_DELAY_Byte (Figure 38-57). As the DQS ODELAY is incremented, it moves away from the edge alignment with the CK. The deskew data is the inner edge of the data valid window for writes.

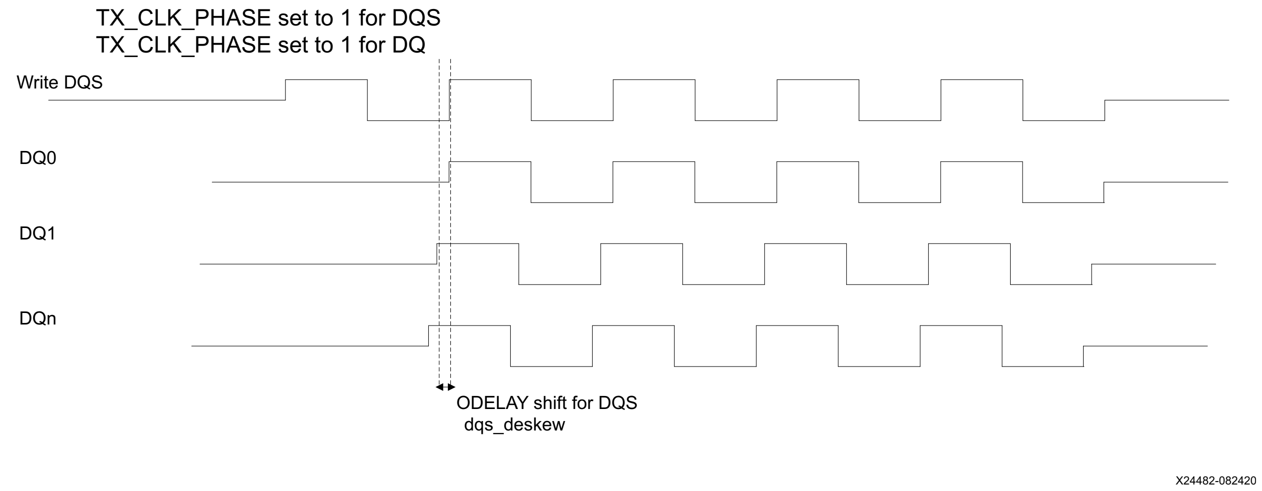


Figure 38-57: **Increment Write DQS ODELAY until All Bits Captured with Correct Pattern**

3. Increment each DQ ODELAY until each bit fails to return the expected data pattern (the data is edge aligned with the write DQS, Figure 38-58).

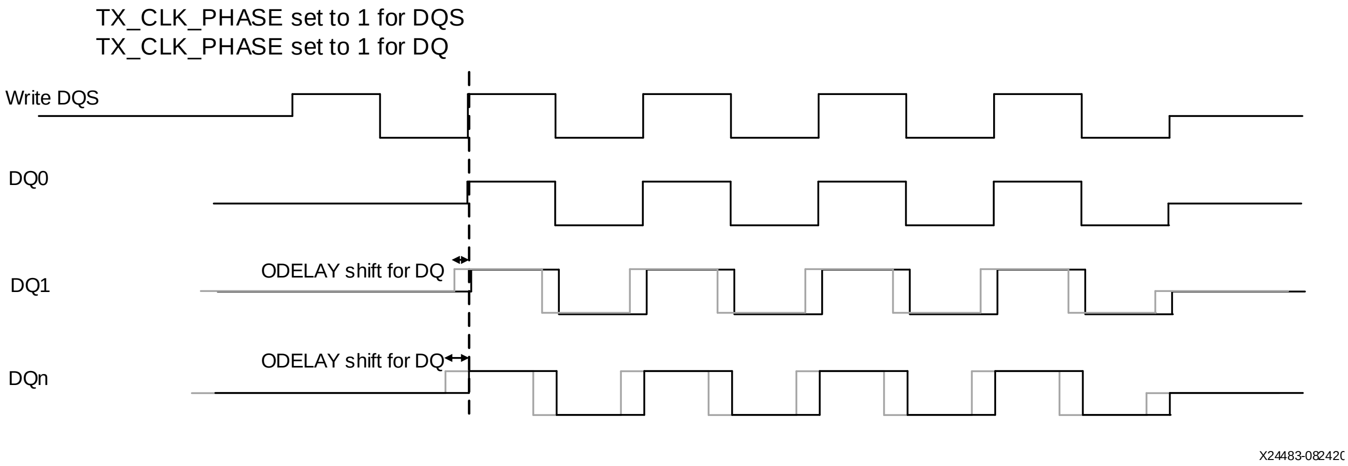


Figure 38-58: **Per-Bit Write Deskew**

4. Return the DQ to the original position at the 0° shift using the TX_DATA_PHASE. Set DQS ODELAY back to starting value (Figure 38-59).

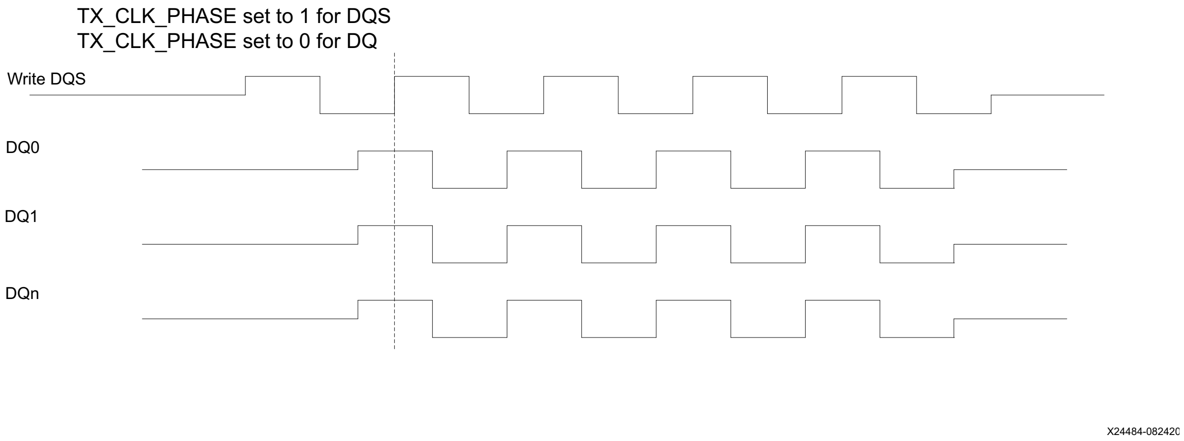
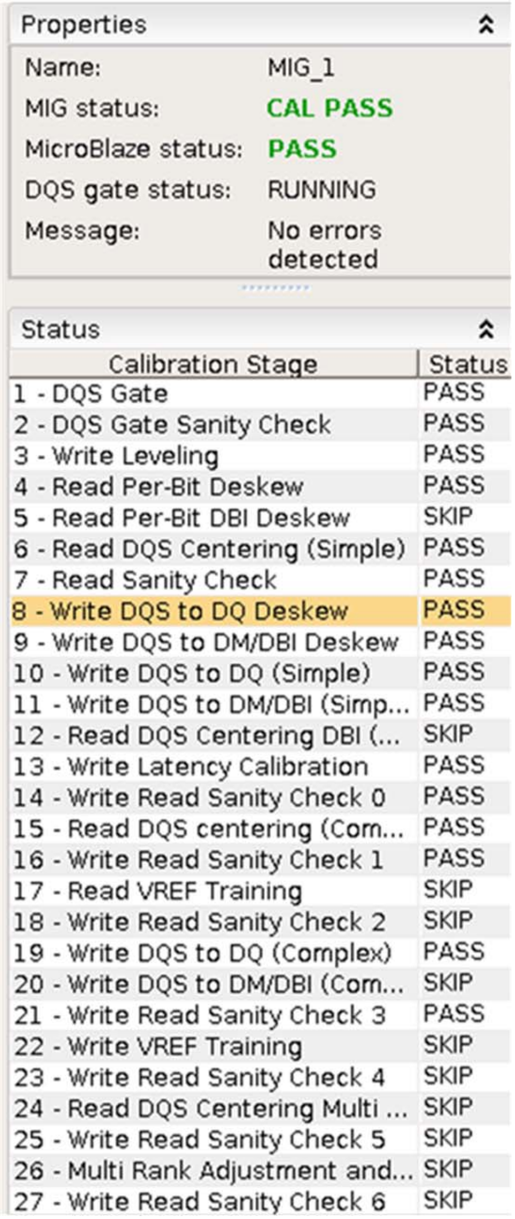


Figure 38-59: **DQ Returned to Approximate 90° Offset with DQS**

Debug

To determine the status of Write Per-Bit Deskew Calibration, click the **Write DQS to DQ Deskew** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Properties

Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Status

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simp...	PASS
12 - Read DQS Centering DBI (...	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Com...	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Com...	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi ...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-60: **Memory IP XSDB Debug GUI Example – Write DQS to DQ Deskew**

The status of Write Per-Bit Deskew can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to the Table 38-21. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-21: **DDR_CAL_ERROR Decode for Write DQS Centering Calibration**

Write DQS-to-DQ Deskew DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Byte	Bit	DQS Deskew Error. Ran out of taps, no valid data found.	Check BUS_DATA_BURST XSDB field to check what values were returned. Check the alignment of DQS to DQ during a write burst with a scope on the PCB. Check the DQS-to-CK alignment. Check the WRLVL fields in XSDB for a given byte.
0x2	Byte	Bit	DQ (or DM) Deskew Error. Failure point not found (bit only indicated when set to CAL_FULL).	Check for a mapping issue. This usually implies a delay is not moving when it should. Check the connections going to the XIPHY and ensure the correct RIU is selected based on the byte being adjusted.
0xF	Byte	N/A	Timeout error waiting for read data to return.	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-22 shows the signals and values adjusted or used during the Write Per-Bit Deskew stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-22: **Signals of Interest for Write Per-Bit Deskew**

Signal	Usage	Signal Description
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE*	One per Byte	ODELAY value required to place DQS into the byte write data valid window during write per-bit deskew.
WRITE_DQS_ODELAY_FINAL_BYTE*	One per Byte	Final DQS ODELAY value.
WRITE_DQ_ODELAY_FINAL_BYTE*_BIT*	One per Bit	Final DQ ODELAY value.
BUS_DATA_BURST (2014.3+)		During calibration for a byte an example data burst is saved for later analysis in case of failure. BUS_DATA_BURST_0 holds an initial read data burst pattern for a given byte with the starting alignment prior to write deskew (TX_DATA_PHASE set to 1 for DQS, 0 for DQ). The ODELAY values for DQS and DQ are the initial WRLVL values. After a byte calibrates, the example read data saved in the BUS_DATA_BURST registers is cleared. BUS_DATA_BURST_1, BUS_DATA_BURST_2, and BUS_DATA_BURST_3 are not used.

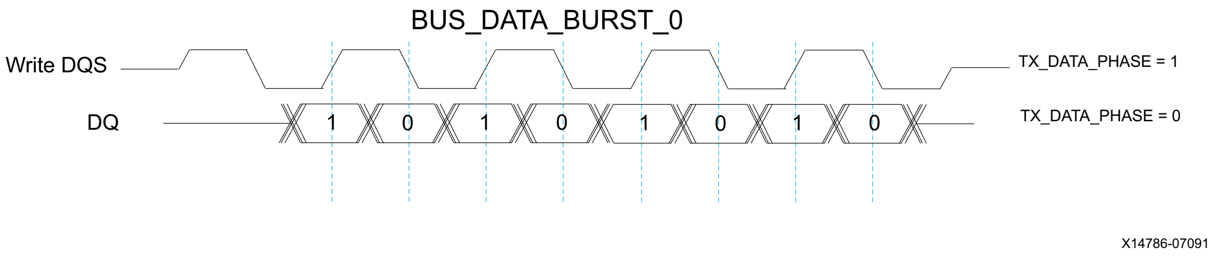


Figure 38-61: **Write DQS Centering (XSDB BUS_DATA_BURST_0)**

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figure 38-62 is an example of how the data is converted.

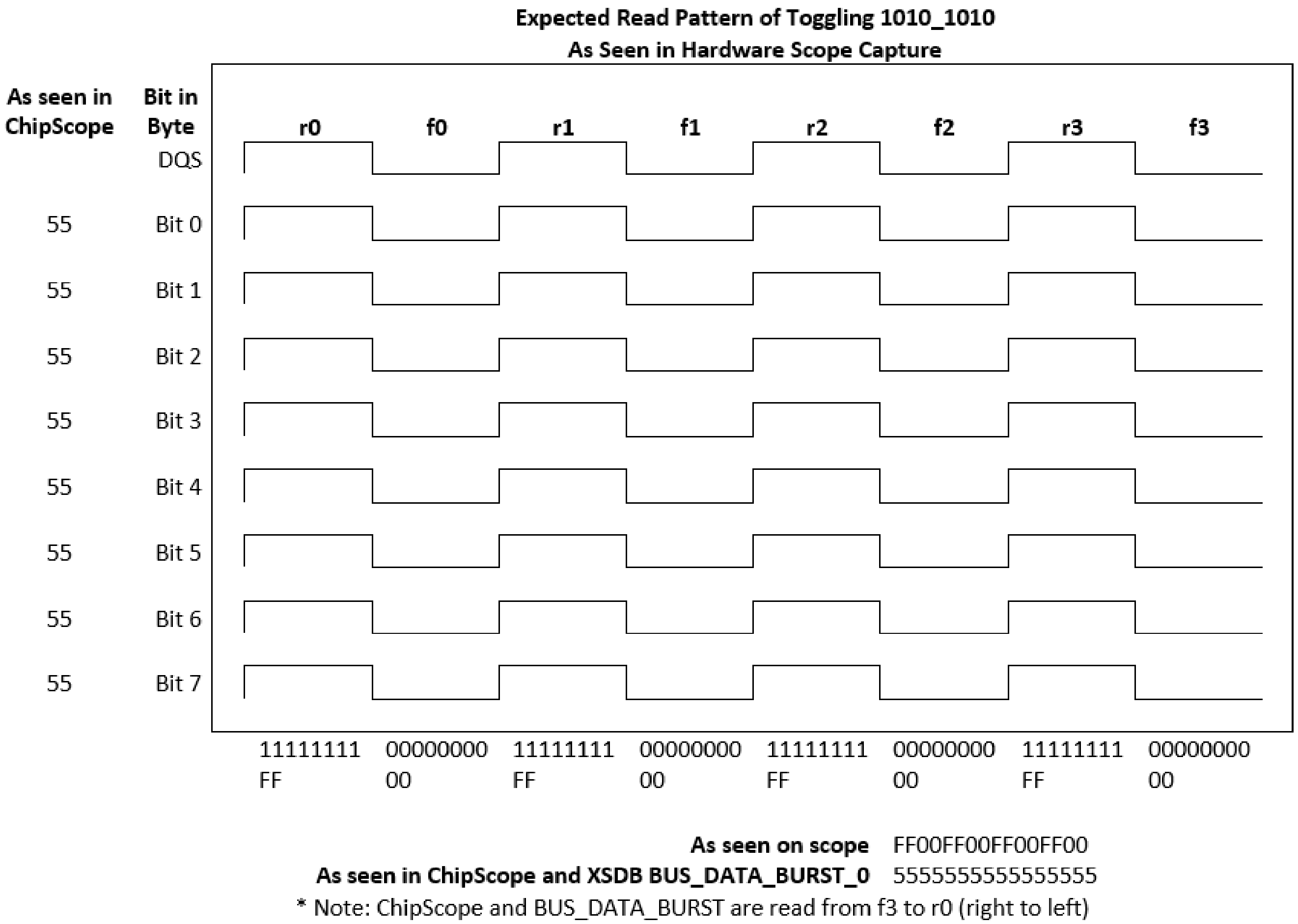


Figure 38-62: **Write DQS-to-DQ Debug Data (XSDB BUS_DATA_BURST, Associated Read Data Saved)**

This is a sample of results for the Write DQS Centering XSDB debug signals:

```
WRITE_DQS_ODELAY_FINAL_BYTE0                string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE1                string true true 010
WRITE_DQS_ODELAY_FINAL_BYTE2                string true true 020
WRITE_DQS_ODELAY_FINAL_BYTE3                string true true 02b
```

```
WRITE_DQS_ODELAY_FINAL_BYTE4                   string true true 00b
WRITE_DQS_ODELAY_FINAL_BYTE5                   string true true 02c
WRITE_DQS_ODELAY_FINAL_BYTE6                   string true true 01b
WRITE_DQS_ODELAY_FINAL_BYTE7                   string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE8                   string true true 016
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE0             string true true 035
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE1             string true true 01d
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE2             string true true 030
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE3             string true true 03a
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE4             string true true 019
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE5             string true true 039
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE6             string true true 028
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE7             string true true 039
WRITE_DQS_TO_DQ_DESKEW_DELAY_BYTE8             string true true 028
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT0               string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT1               string true true 034
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT2               string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT3               string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT4               string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT5               string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT6               string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT7               string true true 02c
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT0               string true true 011
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT1               string true true 00e
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT2               string true true 00d
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT3               string true true 00c
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT4               string true true 00e
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT5               string true true 00e
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT6               string true true 010
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT7               string true true 009
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT0               string true true 023
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT1               string true true 01b
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT2               string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT3               string true true 019
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT4               string true true 019
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT5               string true true 01a
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT6               string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT7               string true true 014
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT0               string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT1               string true true 02a
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT2               string true true 025
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT3               string true true 025
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT4               string true true 028
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT5               string true true 029
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT6               string true true 021
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT7               string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT0               string true true 008
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT1               string true true 005
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT2               string true true 00b
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT3               string true true 008
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT4               string true true 004
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT5               string true true 000
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT6               string true true 009
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT7               string true true 007
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT0               string true true 031
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT1               string true true 02f
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT2               string true true 02e
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT3               string true true 02d
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT4               string true true 030
```

```
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT5                    string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT6                    string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT7                    string true true 02a
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT0                    string true true 020
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT1                    string true true 023
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT2                    string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT3                    string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT4                    string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT5                    string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT6                    string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT7                    string true true 01b
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT0                    string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT1                    string true true 031
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT2                    string true true 028
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT3                    string true true 02a
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT4                    string true true 02d
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT5                    string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT6                    string true true 031
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT7                    string true true 02e
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT0                    string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT1                    string true true 020
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT2                    string true true 017
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT3                    string true true 01c
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT4                    string true true 018
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT5                    string true true 013
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT6                    string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT7                    string true true 012
```

Hardware Measurements

Probe the DQ bit alignment at the memory during writes. Trigger at the start (`cal_r*_status[14]` = R for Rising Edge) and again at the end of per bit deskew (`cal_r*_status[15]` = R for Rising Edge) to view the starting and ending alignments. To look at each byte, add a trigger on the byte using `dbg_cmp_byte`.

Expected Results

Hardware measurements should show the write DQ bits are deskewed at the end of these calibration stages.

- Determine if any bytes completed successfully. The write calibration algorithm sequentially steps through each DQS byte group detecting the capture edges.
- If the incorrect data pattern is detected, determine if the error is due to the write access or the read access. See the Determining If a Data Error is Due to the Write or Read, page 770.

Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the trigger (`cal_r*_status[14]` =R for Rising Edge).

The following simulation examples show how the debug signals should behave during successful Write Per-Bit Deskew:

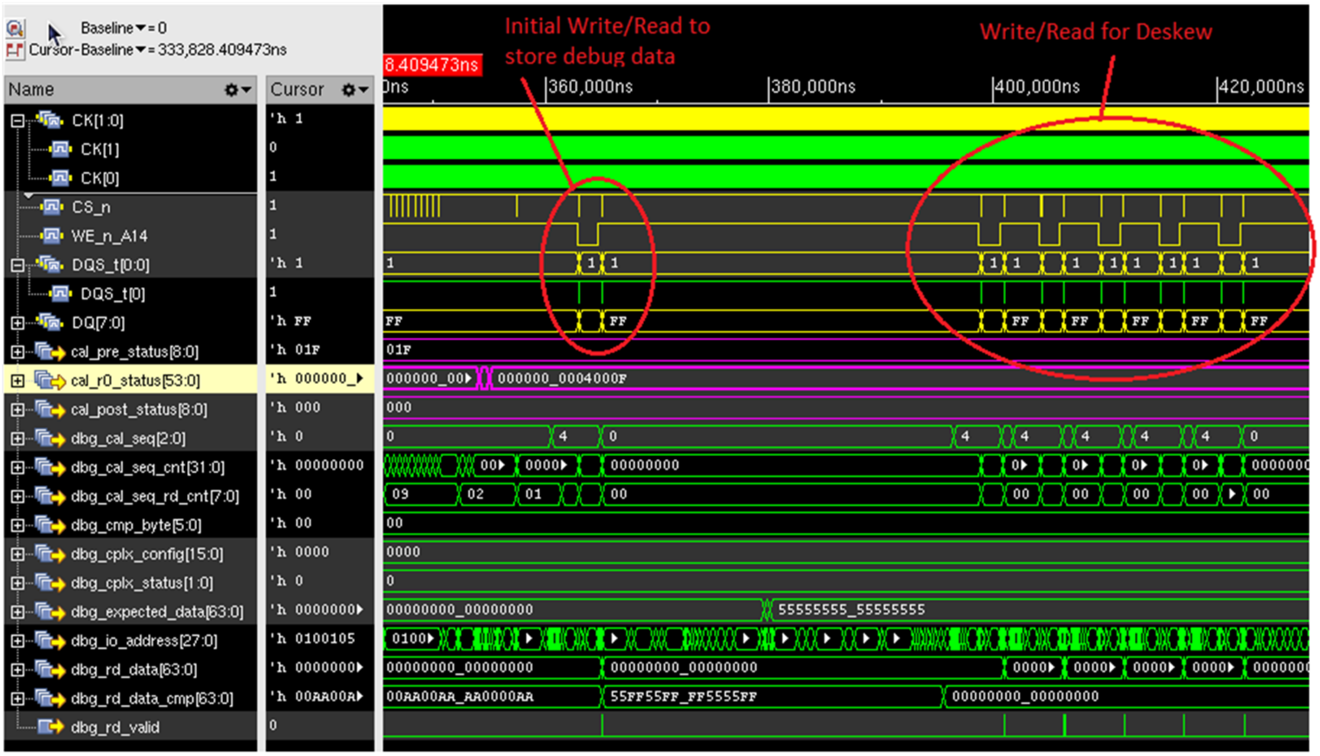


Figure 38-63: **RTL Debug Signals during Write Per-Bit Deskew #1**

Figure 38-64: **RTL Debug Signals during Write Per-Bit Deskew #2**

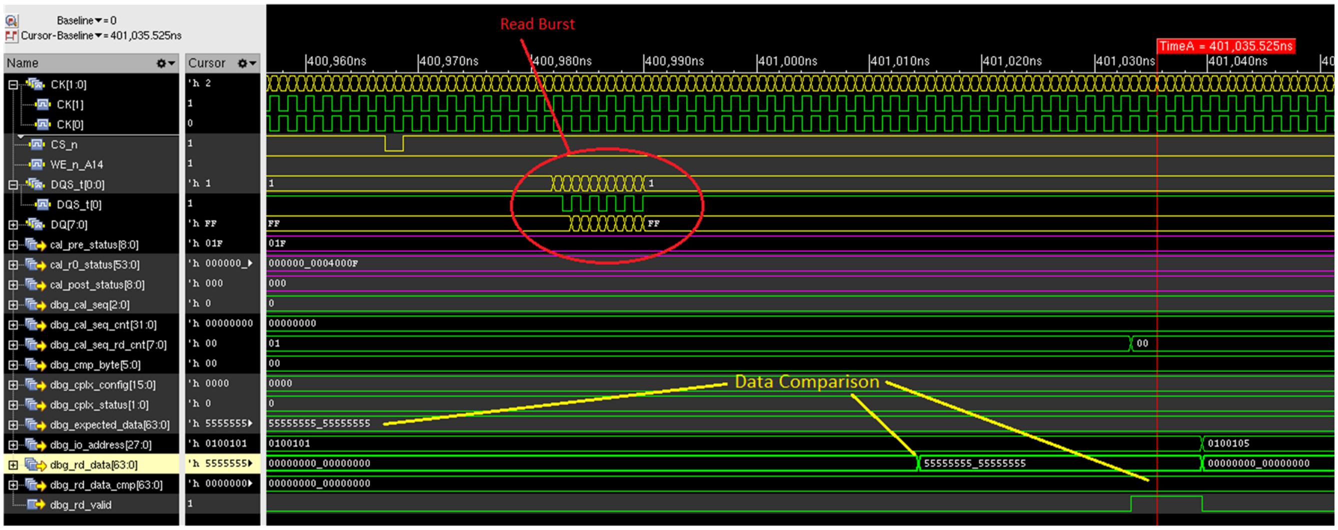


Figure 38-65: **RTL Debug Signals during Write Per-Bit Deskew #3**

Debugging Write DQS Centering Failures

After per-bit write deskew, the next step is to determine the relative center of the DQS in the write data eye and compensate for any error in the TX_DATA_PHASE 90° offset.

1. Issue a set of write and read bursts with the data pattern `10101010` and check the read data. Just as in read write per-bit deskew when issuing a write to the DRAM, the DQS and DQ toggles for eight clock cycles before and after the expected write latency. This is used to ensure the data is written into the DRAM even if the command-to-write data relationship is still unknown.
2. Increment DQ ODELAY taps together until the read data pattern on all DQ bits changes from the expected data pattern `10101010`. The amount of delay required to find the failing point is saved as WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE as shown in Figure 38-66.

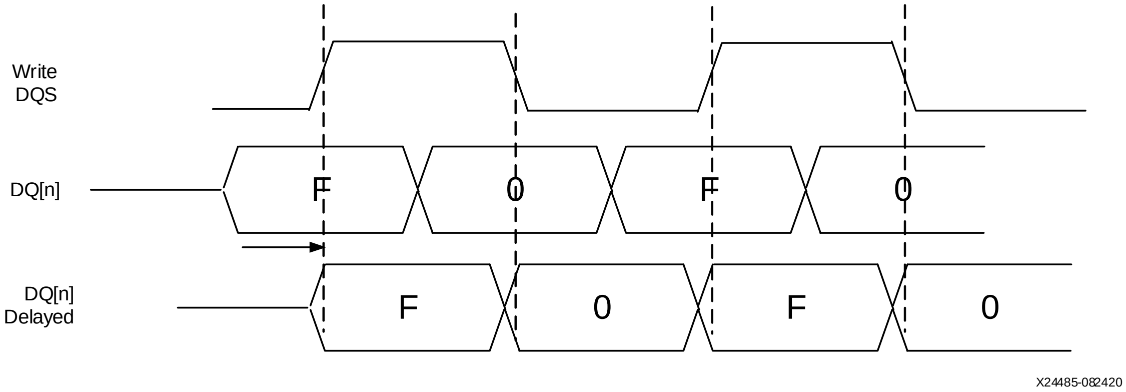


Figure 38-66: **Write DQS Centering – Left Edge**

3. Return DQ ODELAY taps to their original value.

4. Find the right edge of the window by incrementing the DQS ODELAY taps until the data changes from the expected data pattern `10101010`. The amount of delay required to find the failing point is saved as WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE as shown in Figure 38-67.

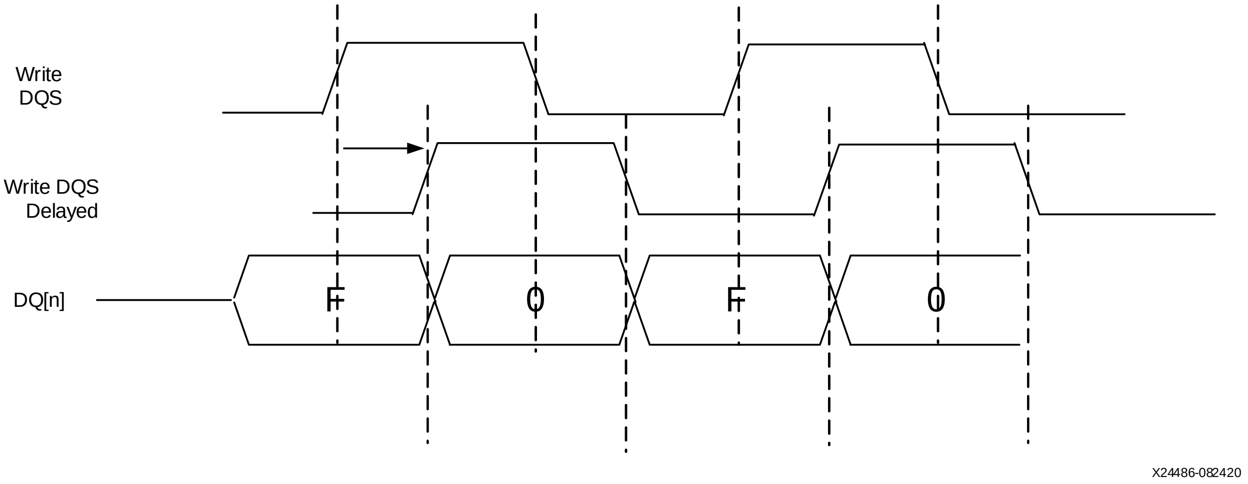


Figure 38-67: **Write DQS Centering – Right Edge**

5. Calculate the center tap location for the DQS ODELAY, based on deskew and left and right edges.

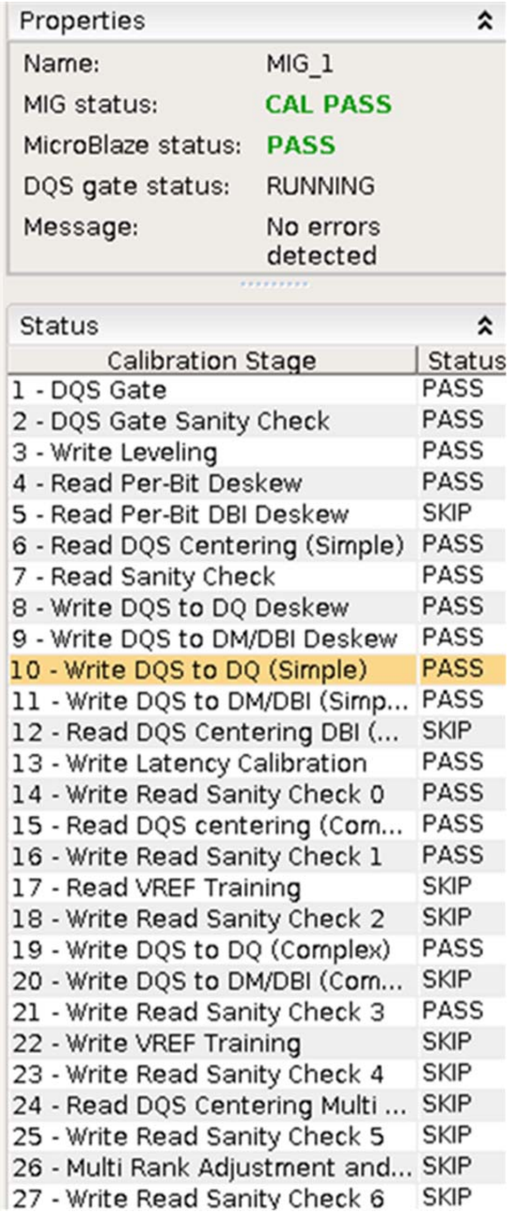
   New DQS delay = deskew – [(dly0 – dly1)/2]

   Where dly0 is the original DQS delay + left margin and dly1 is the original DQS delay + right margin.

The final ODELAY tap setting for DQS is indicated by WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE while the DQ is WRITE_DQS_TO_DQ_DQ_ODELAY. The final computed left and right margin are WRITE_DQS_TO_DQ _MARGIN_LEFT_BYTE and WRITE_DQS_TO_DQ _MARGIN_RIGHT_BYTE.

Debug

To determine the status of Write DQS Centering Calibration, click the **Write DQS to DQ (Simple)** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Properties

Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Status

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simp...	PASS
12 - Read DQS Centering DBI (...	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Com...	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Com...	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi ...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-68: **Memory IP XSDB Debug GUI Example – Write DQS to DQ (Simple)**

The status of Write DQS Centering can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-23. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-23: **DDR_CAL_ERROR Decode for Write DQS Centering Calibration**

Write DQS to DQ DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Byte	N/A	No valid data found	Check BUS_DATA_BURST XSDB field to check what values were returned. Check the alignment of DQS to DQ during a write burst with a scope on the PCB. Check the DQS-to-CK alignment. Check the WRLVL fields in XSDB for a given byte. Check the Write_dqs_to_dq_deskew values.
0x2	Byte	N/A	No valid data found after adjustment	Check what adjustments have been made by analyzing the following: WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE* WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE* WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE* WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE* WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE*_BIT* WRITE_DQ_ODELAY_FINAL_BYTE*_BIT* WRITE_DQS_ODELAY_FINAL_BYTE*_BIT* See how much DQS and DQ have moved from the PRE_ADJUST_MARGIN to the MARGIN values. If the values are reasonable, probe DQS and DQ in hardware after this stage completes looking at the phase alignment between DQS and DQ on the failing byte.
0x3	Byte	N/A	Failed to return to original location after measuring write margin	
0xF	Byte	N/A	Timeout error waiting for read data to return	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-24 shows the signals and values adjusted or used during the Write DQS Centering stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-24: **Signals of Interest for Write DQS Centering**

Signal	Usage	Signal Description
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE*	One per Byte	Left side of the write DQS-to-DQ window measured during calibration before adjustments made.
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT _BYTE*	One per Byte	Right side of the write DQS-to-DQ window measured during calibration before adjustments made.
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE*	One per Byte	Left side of the write DQS-to-DQ window.
WRITE_DQS_TO_DQ_MARGIN_RIGHT _BYTE*	One per Byte	Right side of the write DQS-to-DQ window.
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE*	One per Byte	Final DQS ODELAY value after Write DQS-to-DQ (simple).
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE*_BIT*	One per Bit	Final DQ ODELAY value after Write DQS-to-DQ (simple).
WRITE_DQS_ODELAY_FINAL_BYTE*_BIT*	One per Byte	Final DQS ODELAY value.
WRITE_DQ_ODELAY_FINAL_BYTE*_BIT*	One per Bit	Final DQ ODELAY value.

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figure 38-69 is an example of how the data is converted.

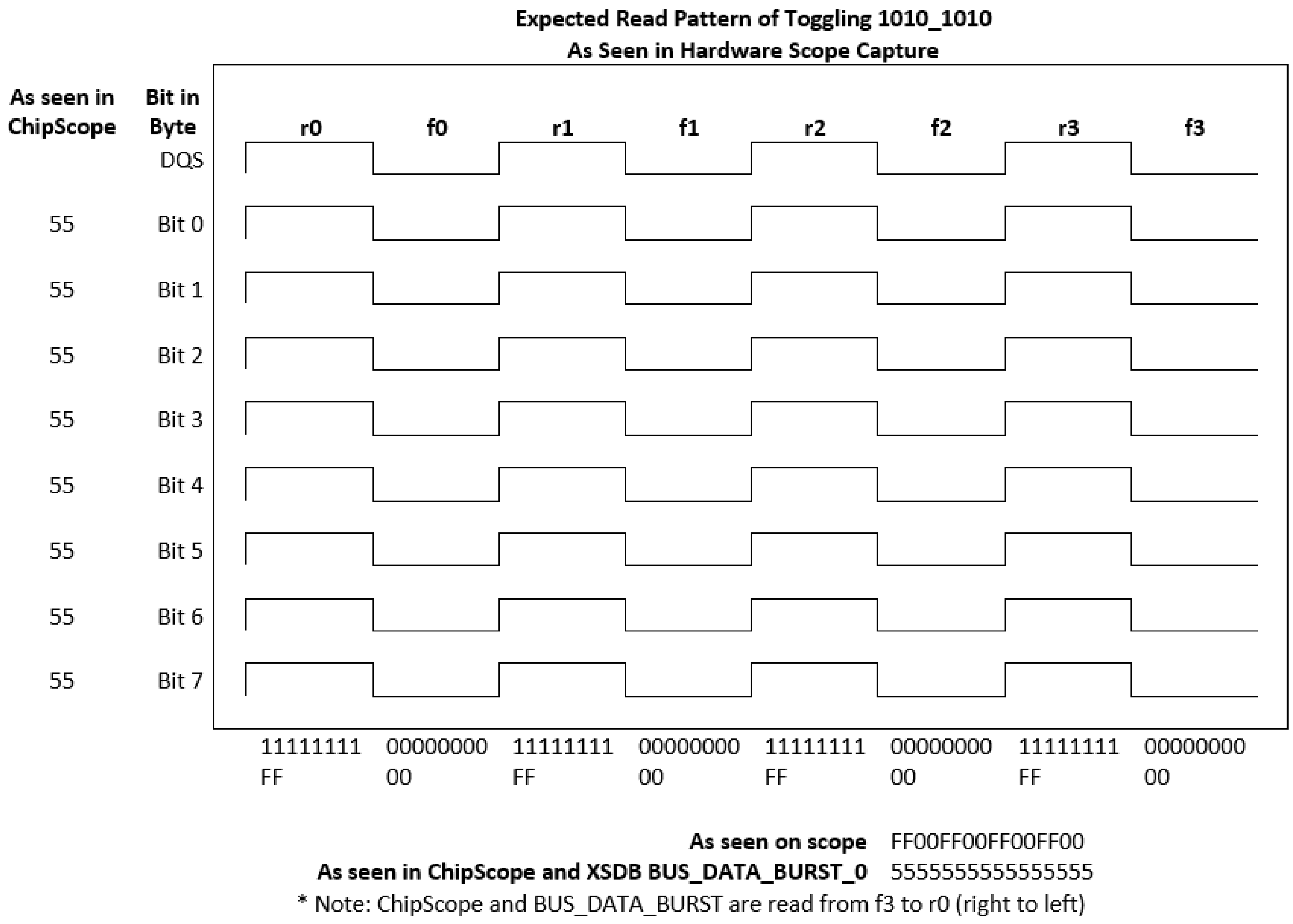


Figure 38-69: **Expected Read Pattern of Toggling 1010_1010**

This is a sample of results for the Write DQS Centering XSDB debug signals:

```
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE0     string true true 063
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE1     string true true 044
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE2     string true true 058
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE3     string true true 065
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE4     string true true 042
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE5     string true true 066
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE6     string true true 057
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE7     string true true 068
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE8     string true true 057
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE0    string true true 056
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE1    string true true 042
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE2    string true true 05a
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE3    string true true 063
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE4    string true true 042
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE5    string true true 05c
```

```
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE6    string true true 048
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE7    string true true 05f
WRITE_DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT_BYTE8    string true true 048
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT0                 string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT1                 string true true 034
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT2                 string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT3                 string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT4                 string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT5                 string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT6                 string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE0_BIT7                 string true true 02c
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT0                 string true true 011
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT1                 string true true 00e
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT2                 string true true 00d
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT3                 string true true 00c
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT4                 string true true 00e
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT5                 string true true 00e
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT6                 string true true 010
WRITE_DQ_ODELAY_FINAL_BYTE1_BIT7                 string true true 009
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT0                 string true true 023
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT1                 string true true 01b
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT2                 string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT3                 string true true 019
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT4                 string true true 019
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT5                 string true true 01a
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT6                 string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE2_BIT7                 string true true 014
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT0                 string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT1                 string true true 02a
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT2                 string true true 025
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT3                 string true true 025
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT4                 string true true 028
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT5                 string true true 029
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT6                 string true true 021
WRITE_DQ_ODELAY_FINAL_BYTE3_BIT7                 string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT0                 string true true 008
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT1                 string true true 005
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT2                 string true true 00b
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT3                 string true true 008
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT4                 string true true 004
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT5                 string true true 000
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT6                 string true true 009
WRITE_DQ_ODELAY_FINAL_BYTE4_BIT7                 string true true 007
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT0                 string true true 031
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT1                 string true true 02f
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT2                 string true true 02e
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT3                 string true true 02d
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT4                 string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT5                 string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT6                 string true true 030
WRITE_DQ_ODELAY_FINAL_BYTE5_BIT7                 string true true 02a
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT0                 string true true 020
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT1                 string true true 023
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT2                 string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT3                 string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT4                 string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT5                 string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT6                 string true true 01d
WRITE_DQ_ODELAY_FINAL_BYTE6_BIT7                 string true true 01b
```

```
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT0                 string true true 033
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT1                 string true true 031
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT2                 string true true 028
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT3                 string true true 02a
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT4                 string true true 02d
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT5                 string true true 02b
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT6                 string true true 031
WRITE_DQ_ODELAY_FINAL_BYTE7_BIT7                 string true true 02e
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT0                 string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT1                 string true true 020
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT2                 string true true 017
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT3                 string true true 01c
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT4                 string true true 018
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT5                 string true true 013
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT6                 string true true 01f
WRITE_DQ_ODELAY_FINAL_BYTE8_BIT7                 string true true 012
WRITE_DQS_ODELAY_FINAL_BYTE0                     string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE1                     string true true 010
WRITE_DQS_ODELAY_FINAL_BYTE2                     string true true 020
WRITE_DQS_ODELAY_FINAL_BYTE3                     string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE4                     string true true 00b
WRITE_DQS_ODELAY_FINAL_BYTE5                     string true true 02c
WRITE_DQS_ODELAY_FINAL_BYTE6                     string true true 01b
WRITE_DQS_ODELAY_FINAL_BYTE7                     string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE8                     string true true 016
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE0                 string true true 02b
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE1                 string true true 010
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE2                 string true true 020
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE3                 string true true 02b
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE4                 string true true 010
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE5                 string true true 02c
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE6                 string true true 01b
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE7                 string true true 02b
WRITE_DQS_TO_DQ_DQS_ODELAY_BYTE8                 string true true 016
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT0             string true true 030
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT1             string true true 031
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT2             string true true 030
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT3             string true true 02d
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT4             string true true 028
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT5             string true true 028
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT6             string true true 030
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE0_BIT7             string true true 029
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT0             string true true 00d
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT1             string true true 00a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT2             string true true 009
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT3             string true true 008
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT4             string true true 00a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT5             string true true 00a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT6             string true true 00c
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE1_BIT7             string true true 005
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT0             string true true 01f
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT1             string true true 017
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT2             string true true 019
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT3             string true true 015
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT4             string true true 015
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT5             string true true 016
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT6             string true true 019
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE2_BIT7             string true true 010
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT0             string true true 028
```

```
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT1             string true true 027
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT2             string true true 022
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT3             string true true 022
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT4             string true true 025
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT5             string true true 026
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT6             string true true 01e
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE3_BIT7             string true true 028
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT0             string true true 008
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT1             string true true 005
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT2             string true true 00b
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT3             string true true 008
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT4             string true true 004
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT5             string true true 000
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT6             string true true 009
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE4_BIT7             string true true 007
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT0             string true true 02c
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT1             string true true 02a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT2             string true true 029
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT3             string true true 028
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT4             string true true 02b
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT5             string true true 02b
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT6             string true true 02b
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE5_BIT7             string true true 025
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT0             string true true 01b
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT1             string true true 01e
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT2             string true true 01a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT3             string true true 01a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT4             string true true 01a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT5             string true true 018
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT6             string true true 018
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE6_BIT7             string true true 016
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT0             string true true 02e
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT1             string true true 02c
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT2             string true true 023
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT3             string true true 025
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT4             string true true 028
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT5             string true true 026
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT6             string true true 02c
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE7_BIT7             string true true 029
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT0             string true true 019
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT1             string true true 01a
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT2             string true true 011
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT3             string true true 016
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT4             string true true 012
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT5             string true true 00d
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT6             string true true 019
WRITE_DQS_TO_DQ_DQ_ODELAY_BYTE8_BIT7             string true true 00c
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE0                string true true 028
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE1                string true true 026
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE2                string true true 02a
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE3                string true true 028
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE4                string true true 028
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE5                string true true 027
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE6                string true true 027
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE7                string true true 02a
WRITE_DQS_TO_DQ_MARGIN_LEFT_BYTE8                string true true 026
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE0               string true true 027
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE1               string true true 028
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE2               string true true 028
```

```
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE3                  string true true 02a
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE4                  string true true 029
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE5                  string true true 029
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE6                  string true true 027
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE7                  string true true 02b
WRITE_DQS_TO_DQ_MARGIN_RIGHT_BYTE8                  string true true 025
```

Hardware Measurements

Probe the DQS to DQ write phase relationship at the memory. DQS should be center aligned to DQ at the end of this stage of calibration. Trigger at the start (**`cal_r*_status[18]`** = R for Rising Edge) and again at the end (**`cal_r*_status[19]`** = R for Rising Edge) of Write DQS Centering to view the starting and ending alignments.

Expected Results

Hardware measurements should show that the write DQ bits are deskewed and that the write DQS are centered in the write DQ window at the end of these calibration stages.

- Look at the individual WRITE_DQS_TO_DQ_DQS_ODELAY and WRITE_DQS_TO_DQ_DQ_ODELAY tap settings for each nibble. The taps should only vary by 0 to 20 taps. See Determining Window Size in ps, page 773 to calculate the write window.
- Determine if any bytes completed successfully. The write calibration algorithm sequentially steps through each DQS byte group detecting the capture edges.
- If the incorrect data pattern is detected, determine if the error is due to the write access or the read access. See Determining If a Data Error is Due to the Write or Read, page 770.
- Both edges need to be found. This is possible at all frequencies because the algorithm uses 90° of ODELAY taps to find the edges.
- To analyze the window size in ps, see Determining Window Size in ps, page 773. As a general rule of thumb, the window size for a healthy system should be ≥ 30% of the expected UI size.

Using the Vivado Hardware Manager and while running the **Memory IP Example Design** with **Debug Signals** enabled, set the trigger (**`cal_r*_status[18]`** = R for Rising Edge). The simulation examples shown in the Debugging Write Per-Bit Deskew Failures > Expected Results section can be used to additionally monitor the expected behavior for Write DQS Centering.

Write Data Mask Calibration

Note: The calibration step is only enabled for the first rank in a multi-rank system.

During calibration the Data Mask (DM) signals are not used, they are deasserted during any writes before/after the required amount of time to ensure they have no impact on the pattern being written to the DRAM. If the DM signals are not used, this step of calibration is skipped.

Two patterns are used to calibrate the DM pin. The first pattern is written to the DRAM with the DM deasserted, ensuring the pattern is written to the DRAM properly. The second pattern overwrites the first pattern at the same address but with the DM asserted in a known position in the burst, as shown in Figure 38-71.

Because this stage takes place before Write Latency Calibration when issuing a write to the DRAM, the DQS and DQ/DM toggles for eight clock cycles before and after the expected write latency. This is used to ensure the data is written into the DRAM even though the command-to-write data relationship is still unknown.

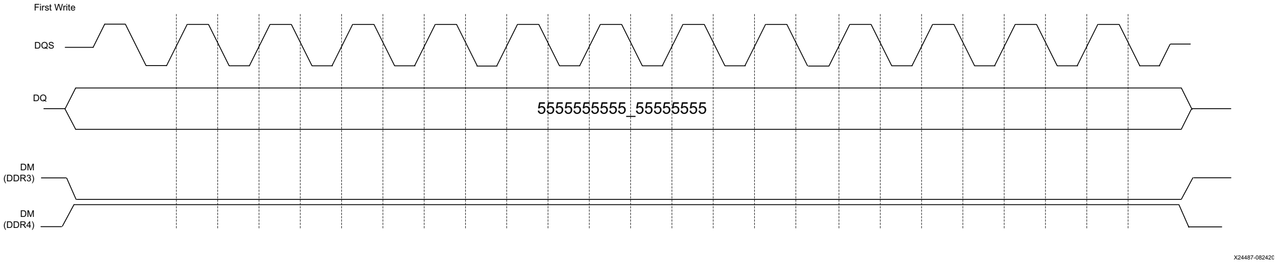


Figure 38-70: **DM Base Data Written**

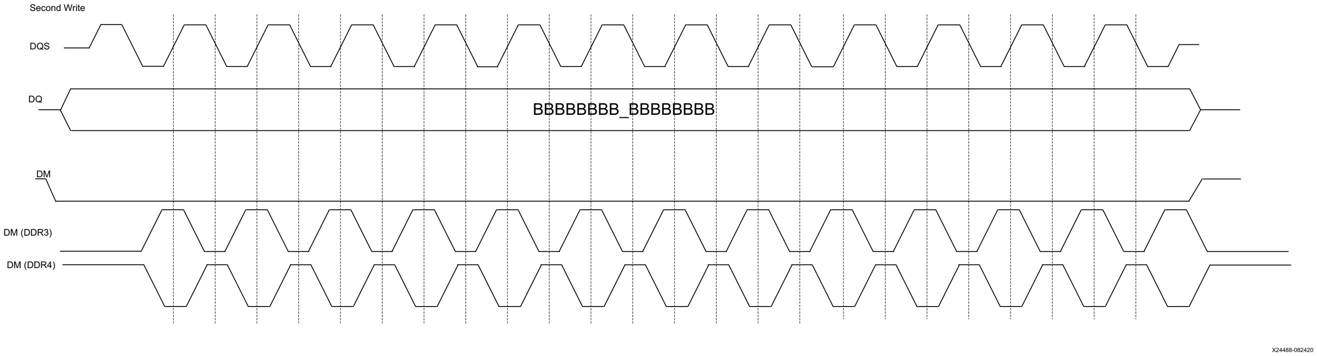


Figure 38-71: **DM Asserted**

The read back data for any given nibble is `5B5B_5B5B`, where the location of the `5` in the burst indicates where the DM is asserted. Because the data is constant during this step, the DQS-to-DQ alignment is not stressed. Only the DQS-to-DM is checked as the DQS and DM phase relationship is adjusted with each other.

Write DQS-to-DM Per-Bit Deskew

This step is similar to Write DQS-to-DQ Per-Bit Deskew but involves the DM instead of the DQ bits. See Write Calibration Overview, page 670 for an in-depth overview of the algorithm. The DQS ODELAY value used to edge align the DQS with the DM is stored as WRITE_DQS_TO_DM_DESKEW_BYTE. The ODELAY value for the DM is stored as WRITE_DQS_TO_DM_DM_ODELAY_BYTE.

Write DQS-to-DM Centering

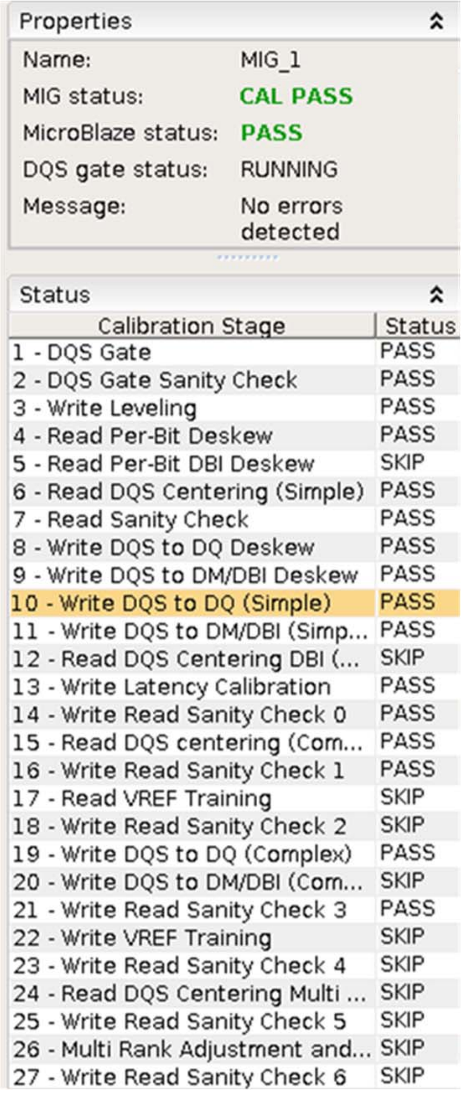
This step is similar to Write DQS-to-DQ Centering but involves the DM instead of the DQ bits. See Write Calibration Overview, page 670 for an in-depth overview of the algorithm. The tap value DM was set at to find the left edge is saved as WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE. The tap value DQS was set at to find the right edge is saved as WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE.

The final DM margin is stored at WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE and WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE.

Because the DQS ODELAY can only hold a single value, compute the aggregate smallest left/right margin between the DQ and DM. The DQS ODELAY value is set in the middle of this aggregate window. The final values of the DQS and DM can be found at WRITE_DQS_ODELAY_FINAL and WRITE_DM_ODELAY_FINAL.

Debug

To determine the status of Write Data Mask Calibration, click the **Write DQS to DM/DBI (Simple)** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Properties

Name:	MIG_1
MIG status:	CAL PASS
MicroBlaze status:	PASS
DQS gate status:	RUNNING
Message:	No errors detected

Status

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simp...	PASS
12 - Read DQS Centering DBI (...	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Com...	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Com...	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi ...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-72: **Memory IP XSDB Debug GUI Example – Write DQS to DM/DMBI (Simple)**

The status of Write Data Mask Calibration can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-25. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-25: **DDR_CAL_ERROR Decode for Write Data Mask Calibration**

Write DQS to DM Deskew DDR_CAL_ ERROR_ CODE	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	Byte	N/A	DQS Deskew Error. Ran out of taps, no valid data found.	Check BUS_DATA_BURST XSDB field to check what values were returned. Check the alignment of DQS to DM during a write burst with a scope on the PCB. Check the DQS-to-CK alignment. Check the WRLVL fields in XSDB for a given byte. Check the signal level of the DM on a write.
0x2	Byte	N/A	DQ (or DM) Deskew Error. Failure point not found (bit only indicated when set to CAL_FULL).	Check for a mapping issue. This usually implies a delay is not moving when it should. Check the connections going to the XIPHY and ensure the correct RIU is selected based on the byte being adjusted.
0xF	Byte	N/A	Timeout error waiting for read data to return.	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-26 shows the signals and values adjusted or used during the Write Data Mask stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-26: **Signals of Interest for Write Data Mask Calibration**

Signal	Usage	Signal Description
WRITE_DQS_TO_DM_DESKEW_BYTE*	One per Byte	ODELAY value required to place DQS into the byte write data valid window during write per-bit deskew.
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE*	One per byte	Left side of the write DQS-to-DM window measured during calibration before adjustments made.
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT _BYTE*	One per Byte	Right side of the write DQS-to-DM window measured during calibration before adjustments made.
WRITE_DQS_TO_DM _MARGIN_LEFT_BYTE*	One per Byte	Left side of the write DQS-to-DM window.
WRITE_DQS_TO_DM _MARGIN_RIGHT _BYTE*	One per Byte	Right side of the write DQS-to-DM window.
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE*	One per Byte	Final DQS ODELAY value after Write DQS-to-DM (simple).
WRITE_DQS_TO_DM_DM_ODELAY_BYTE*_BIT*	One per Bit	Final DM ODELAY value after Write DQS-to-DQ (simple).

Table 38-26: **Signals of Interest for Write Data Mask Calibration *(Cont'd)***

Signal	Usage	Signal Description
WRITE_DQS_ODELAY_FINAL_BYTE*_BIT*	One per Byte	Final DQS ODELAY value.
WRITE_DM_ODELAY_FINAL_BYTE*_BIT*	One per Bit	Final DM ODELAY value.
BUS_DATA_BURST (2014.3+)		During calibration for a byte an example data burst is saved for later analysis in case of failure. BUS_DATA_BURST_0 holds an initial read data burst pattern for a given byte with the starting alignment prior to write DM deskew (TX_DATA_PHASE set to 1 for DQS, 0 for DM and DQ). BUS_DATA_BURST_1 holds a read data burst after write DM deskew and at the start of write DQS-to-DM centering, after TX_DATA_PHASE for DQS is set to 1 and the TX_DATA_PHASE for DQ/DM is set to 1. After a byte calibrates, the example read data saved in the BUS_DATA_BURST registers is cleared. BUS_DATA_BURST_2 and BUS_DATA_BURST_3 are not used.

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figure 38-73 and Figure 38-74 are examples of how the data is converted.

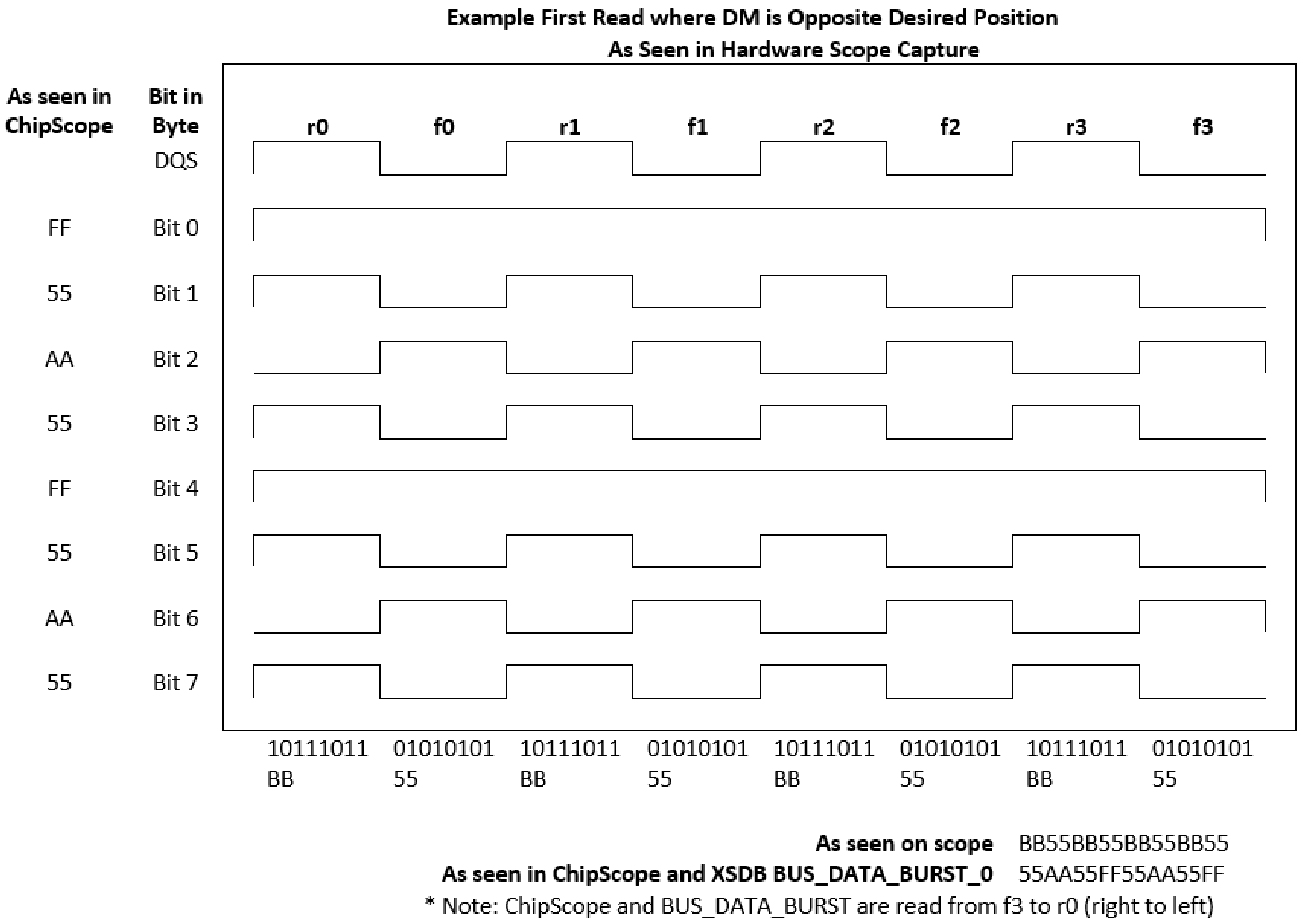


Figure 38-73: **Example First Read Where DM is Opposite Desired Position**

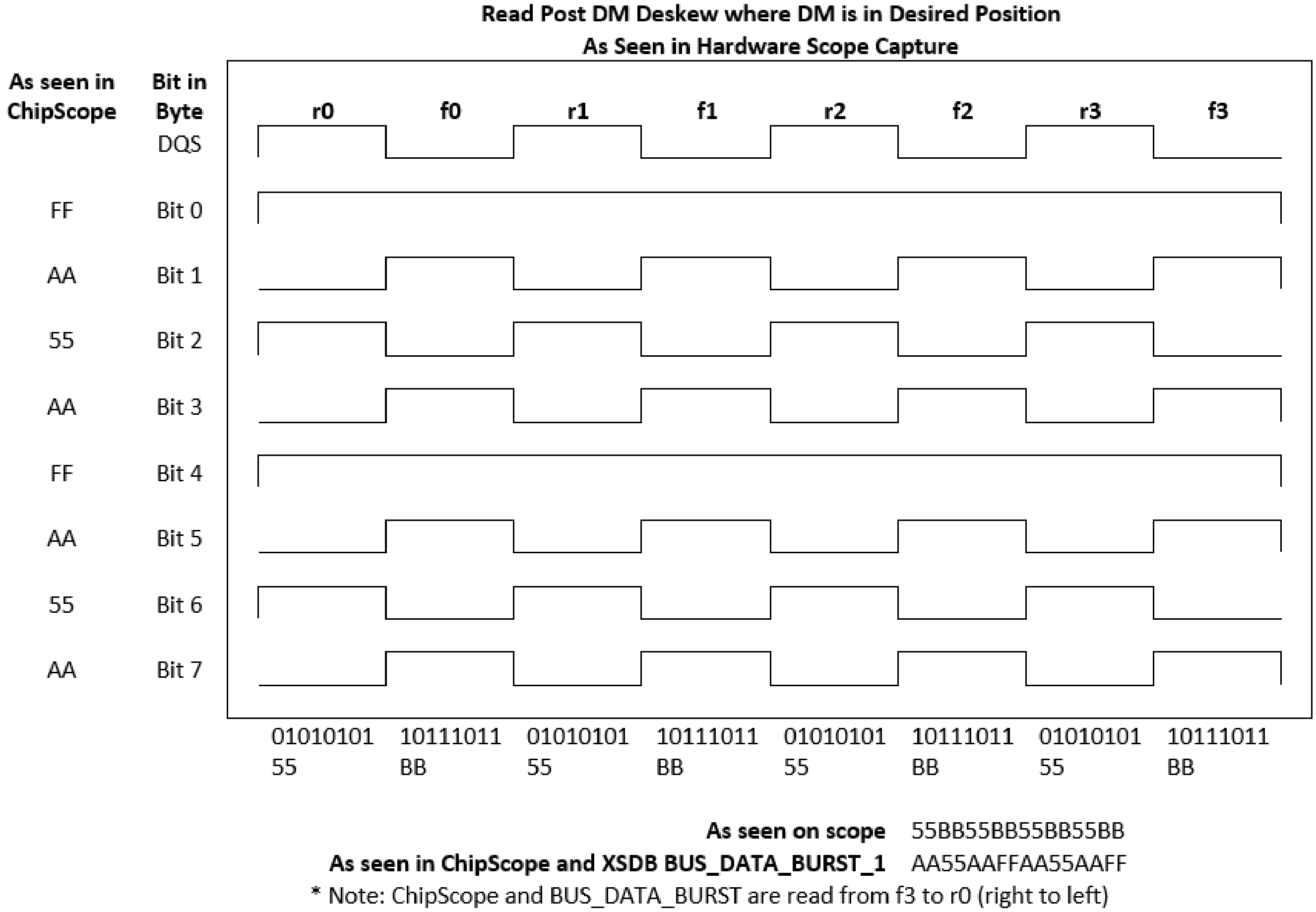


Figure 38-74: **Read Post DM Deskew Where DM is in Desired Position**

This is a sample of results for the Write Data Mask XSDB debug signals:

```
WRITE_DM_ODELAY_FINAL_BYTE0                     string true true 031
WRITE_DM_ODELAY_FINAL_BYTE1                     string true true 01b
WRITE_DM_ODELAY_FINAL_BYTE2                     string true true 02a
WRITE_DM_ODELAY_FINAL_BYTE3                     string true true 036
WRITE_DM_ODELAY_FINAL_BYTE4                     string true true 011
WRITE_DM_ODELAY_FINAL_BYTE5                     string true true 036
WRITE_DM_ODELAY_FINAL_BYTE6                     string true true 029
WRITE_DM_ODELAY_FINAL_BYTE7                     string true true 039
WRITE_DM_ODELAY_FINAL_BYTE8                     string true true 029
WRITE_DQS_ODELAY_FINAL_BYTE0                    string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE1                    string true true 010
WRITE_DQS_ODELAY_FINAL_BYTE2                    string true true 020
WRITE_DQS_ODELAY_FINAL_BYTE3                    string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE4                    string true true 00b
WRITE_DQS_ODELAY_FINAL_BYTE5                    string true true 02c
WRITE_DQS_ODELAY_FINAL_BYTE6                    string true true 01b
WRITE_DQS_ODELAY_FINAL_BYTE7                    string true true 02b
WRITE_DQS_ODELAY_FINAL_BYTE8                    string true true 016
WRITE_DQS_TO_DM_DESKEW_BYTE0                    string true true 035
WRITE_DQS_TO_DM_DESKEW_BYTE1                    string true true 01d
```

```
WRITE_DQS_TO_DM_DESKEW_BYTE2                     string true true 030
WRITE_DQS_TO_DM_DESKEW_BYTE3                     string true true 03a
WRITE_DQS_TO_DM_DESKEW_BYTE4                     string true true 019
WRITE_DQS_TO_DM_DESKEW_BYTE5                     string true true 039
WRITE_DQS_TO_DM_DESKEW_BYTE6                     string true true 028
WRITE_DQS_TO_DM_DESKEW_BYTE7                     string true true 039
WRITE_DQS_TO_DM_DESKEW_BYTE8                     string true true 028
WRITE_DQS_TO_DM_DM_ODELAY_BYTE0                  string true true 031
WRITE_DQS_TO_DM_DM_ODELAY_BYTE1                  string true true 01b
WRITE_DQS_TO_DM_DM_ODELAY_BYTE2                  string true true 02a
WRITE_DQS_TO_DM_DM_ODELAY_BYTE3                  string true true 036
WRITE_DQS_TO_DM_DM_ODELAY_BYTE4                  string true true 011
WRITE_DQS_TO_DM_DM_ODELAY_BYTE5                  string true true 036
WRITE_DQS_TO_DM_DM_ODELAY_BYTE6                  string true true 029
WRITE_DQS_TO_DM_DM_ODELAY_BYTE7                  string true true 039
WRITE_DQS_TO_DM_DM_ODELAY_BYTE8                  string true true 029
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE0                 string true true 02b
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE1                 string true true 015
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE2                 string true true 026
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE3                 string true true 033
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE4                 string true true 013
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE5                 string true true 02e
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE6                 string true true 01d
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE7                 string true true 02e
WRITE_DQS_TO_DM_DQS_ODELAY_BYTE8                 string true true 019
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE0                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE1                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE2                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE3                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE4                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE5                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE6                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE7                string true true 000
WRITE_DQS_TO_DM_MARGIN_LEFT_BYTE8                string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE0               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE1               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE2               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE3               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE4               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE5               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE6               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE7               string true true 000
WRITE_DQS_TO_DM_MARGIN_RIGHT_BYTE8               string true true 000
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE0     string true true 026
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE1     string true true 01e
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE2     string true true 01c
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE3     string true true 019
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE4     string true true 022
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE5     string true true 025
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE6     string true true 023
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE7     string true true 025
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_LEFT_BYTE8     string true true 01e
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE0    string true true 033
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE1    string true true 03f
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE2    string true true 03e
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE3    string true true 03f
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE4    string true true 039
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE5    string true true 036
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE6    string true true 03b
```

```
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE7    string true true 03a
WRITE_DQS_TO_DM_PRE_ADJUST_MARGIN_RIGHT_BYTE8    string true true 041
```

Hardware Measurements

- Probe the DM to DQ bit alignment at the memory during writes. Trigger at the start (**`cal_r*_status[20]`** = R for Rising Edge) and again at the end (**`cal_r*_status[21]`** = R for Rising Edge) of Simple Pattern Write Data Mask Calibration to view the starting and ending alignments.
- Probe the DM to DQ bit alignment at the memory during writes. Trigger at the start (**`cal_r*_status[38]`** = R for Rising Edge) and again at the end (**`cal_r*_status[39]`** = R for Rising Edge) of Complex Pattern Write Data Mask Calibration to view the starting and ending alignments.

The following simulation examples show how the debug signals should behave during successful Write DQS-to-DM Calibration.

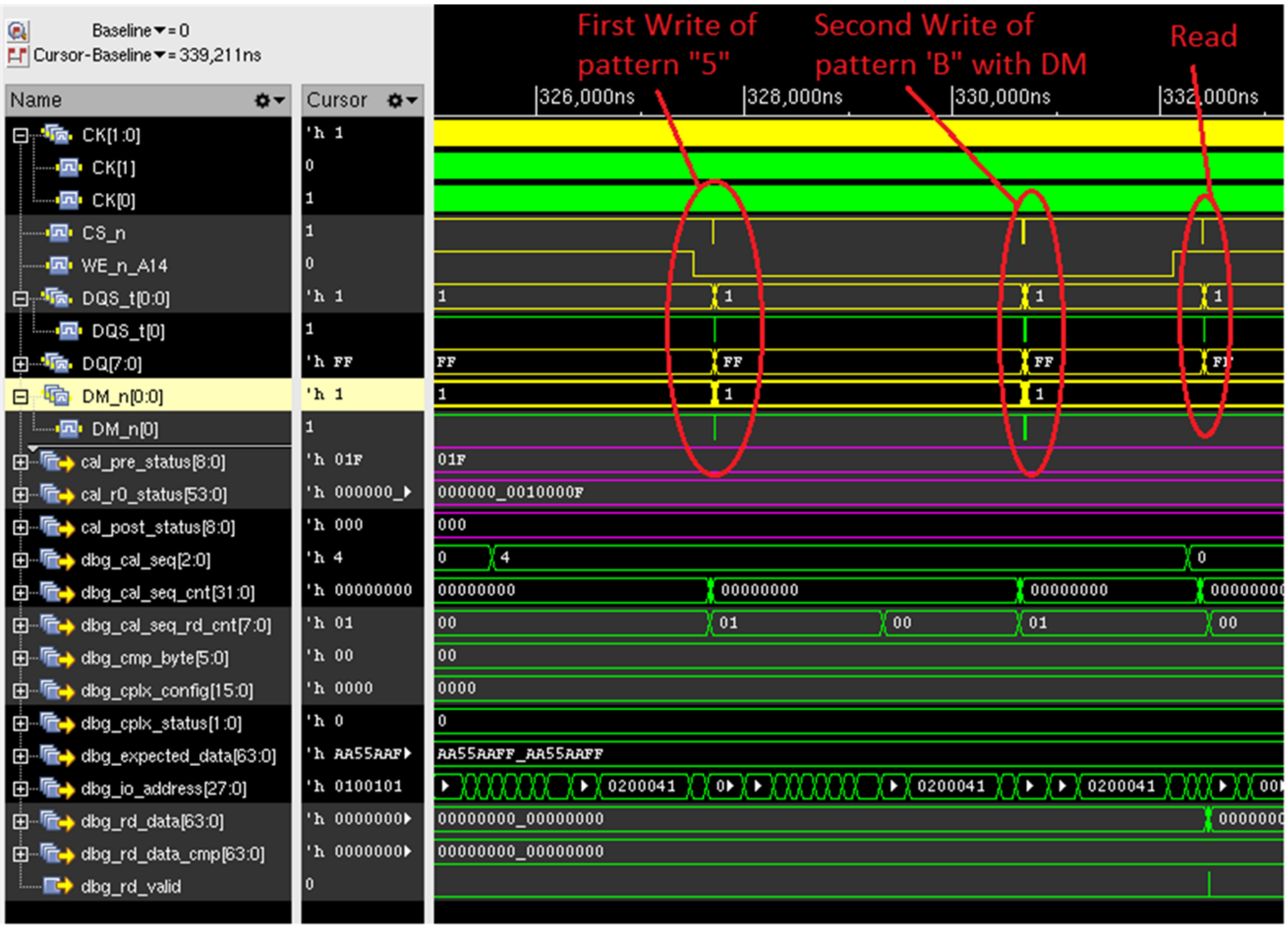


Figure 38-75: **RTL Debug Signals during Write DQS-to-DM Calibration #1**

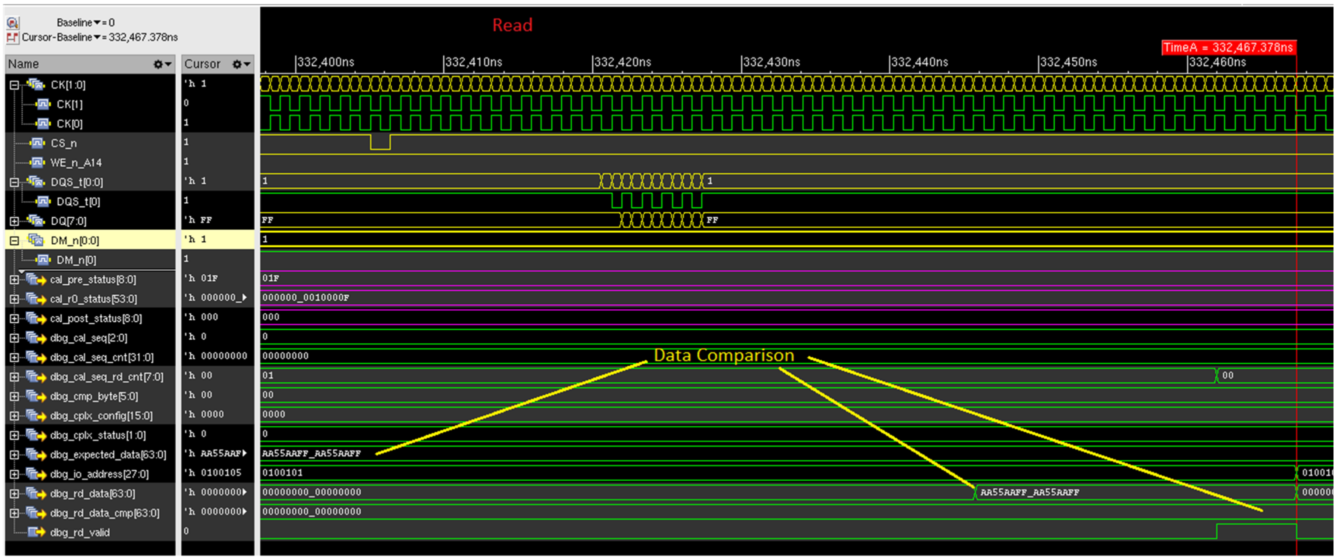


Figure 38-76: **RTL Debug Signals during Write DQS-to-DM Calibration #2**

Expected Results

- Look at the individual WRITE_DQS_TO_DM_DQS_ODELAY and WRITE_DQS_TO_DM_DM_ODELAY tap settings for each nibble. The taps should only vary by 0 to 20 taps. See Determining Window Size in ps, page 773 to calculate the write window.
- Determine if any bytes completed successfully. The write calibration algorithm sequentially steps through each DQS byte group detecting the capture edges.
- If the incorrect data pattern is detected, determine if the error is due to the write access or the read access. See Determining If a Data Error is Due to the Write or Read, page 770.
- Both edges need to be found. This is possible at all frequencies because the algorithm uses 90° of ODELAY taps to find the edges.

Debugging Read DQS Centering with DBI

If the read DBI option is selected for DDR4, the capture of the DBI pin itself must be taken into account when deciding where to position the capture clock in the data valid window. When DBI read is enabled, the DRAM sends back a signal to indicate if the data stored in the array is inverted, so as to save power by not sending back the inverted data on the bus. The receiver inverts the data when the DBI pin is asserted. The DRAM does not support DBI during MPR reads used in Read DQS Centering (simple) calibration, so a pattern must be written into the DRAM before being read out. This stage occurs after the Write DQS-to-DQ/DM stages to ensure the pattern can be written into the DRAM properly.

1. Turn on DBI on the read path (MRS setting in the DRAM and a fabric switch that inverts the read data when the value read from the DBI pin is asserted).

2. Write the pattern `0-F-0-F-0-F-0-F` to the DRAM (extending the data pattern before/after the burst due to this step happening before write latency calibration) to address `0x000`/Bank Group 0.
3. If the nibble does not contain the DBI pin, skip the nibble and go to next nibble.
4. Start from the current setting of PQTR/NQTR, which is the center of the data valid window for the DQ found so far.
5. Issue reads to address `0x000`/Bank Group 0. This is repeated until read DQS Centering with DBI is completed.

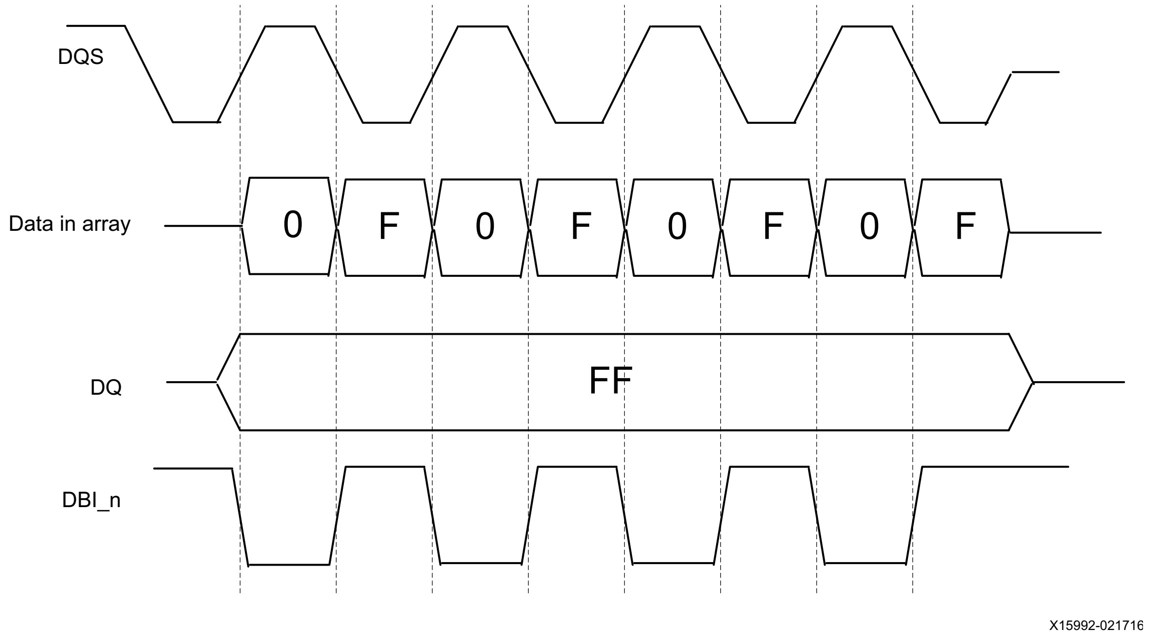


Figure 38-77: **Read DQS Centering with DBI Read Pattern**

6. Find the left edge of read DBI pin. Decrement PQTR/NQTR to find the left edge of the read DBI pin until the data pattern changes from the expected pattern.
7. Find the right edge of the read DBI pin. Compute the aggregate window given the XSDB results for Read DQS Centering (simple) and the new result from DBI. This means take the (largest left + Smallest right)/2 + largest left. This gives the center result for the given nibble + DBI pin (aggregate center).
8. Turn off DBI on the read path (MRS setting in the DRAM and fabric switch).

Debug

To determine the status of Read DQS centering with DBI Calibration, click the **Read DQS Centering DBI (Simple) Calibration** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.

The status of Read DQS Centering DBI (Simple) can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-27. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-27: **DDR_CAL_ERROR Decode for Read DQS Centering with DBI**

Per-Bit DBI Deskew DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Nibble	N/A	No valid data found for a given nibble.	Check the BUS_DATA_BURST fields in XSDB. Check the dbg_rd_data, dbg_rd_data_cmp, and dbg_expected_data signals in the ILA. Check the pinout for the DBI pin. Probe the board and check for the returning pattern to determine if the initial write to the DRAM happened properly, or if it is a read failure. Probe the DBI pin during the read.
0xF	Nibble	N/A	Timeout error waiting for all read data bursts to return.	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-28 describes the signals and values adjusted or used during the Read DQS Centering DBI (Simple) stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-28: **Signals of Interest for Read DQS Centering with DBI**

Signal	Usage	Signal Description
RDLVL_DBI_PQTR_LEFT_RANK_NIBBLE	One per nibble	Read leveling PQTR when left edge of read data valid window is detected during Read DQS Centering DBI (Simple).
RDLVL_DBI_PQTR_RIGHT_RANK_NIBBLE	One per nibble	Read leveling PQTR when right edge of read data valid window is detected during Read DQS Centering DBI (Simple).
RDLVL_DBI_PQTR_CENTER_RANK*_NIBBLE*	One per nibble	Read leveling PQTR center point between right and left during Read DQS Centering DBI (Simple).
RDLVL_DBI_NQTR_LEFT_RANK*_NIBBLE*	One per nibble	Read leveling NQTR when left edge of read data valid window is detected during Read DQS Centering DBI (Simple).
RDLVL_DBI_NQTR_RIGHT_ RANK*_NIBBLE*	One per nibble	Read leveling NQTR when right edge of read data valid window is detected during Read DQS Centering DBI (Simple).

Table 38-28: **Signals of Interest for Read DQS Centering with DBI *(Cont'd)***

Signal	Usage	Signal Description
RDLVL_DBI_NQTR_CENTER_RANK*_NIBBLE*	One per nibble	Read leveling NQTR center point between right and left during Read DQS Centering DBI (Simple).
RDLVL_IDELAY_DBI_RANK*_BYTE*	One per rank per Byte	Read leveling IDELAY delay value for the DBI pin set during DBI deskew.
RDLVL_PQTR_LEFT_RANK*_NIBBLE*	One per rank per nibble	Read leveling PQTR tap position when left edge of read data valid window is detected (simple pattern).
RDLVL_NQTR_LEFT_RANK*_NIBBLE*	One per rank per nibble	Read leveling NQTR tap position when left edge of read data valid window is detected (simple pattern).
RDLVL_PQTR_RIGHT_RANK*_NIBBLE*	One per rank per nibble	Read leveling PQTR tap position when right edge of read data valid window is detected (simple pattern).
RDLVL_NQTR_RIGHT_RANK*_NIBBLE*	One per rank per nibble	Read leveling NQTR tap position when right edge of read data valid window is detected (simple pattern).
RDLVL_PQTR_CENTER_RANK*_NIBBLE*	One per rank per nibble	Read leveling PQTR center tap position found at the end of read DQS centering (simple pattern).
RDLVL_NQTR_CENTER_RANK*_NIBBLE*	One per rank per nibble	Read leveling NQTR center tap position found at the end of read DQS centering (simple pattern).
RDLVL_IDELAY_VALUE_RANK*_BYTE*_BIT*	One per rank per Bit	Read leveling IDELAY delay value found during per bit read DQS centering (simple pattern).
BISC_ALIGN_PQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_ALIGN_NQTR_NIBBLE*	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_PQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_PQTR – BISC_ALIGN_PQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_PQTR – BISC_ALIGN_PQTR).

Table 38-28: **Signals of Interest for Read DQS Centering with DBI *(Cont'd)***

Signal	Usage	Signal Description
BISC_NQTR_NIBBLE*	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_NQTR – BISC_ALIGN_NQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_NQTR – BISC_ALIGN_NQTR).
BUS_DATA_BURST		When a failure occurs during Read DQS centering with DBI, some data is saved to indicate what the data looks like for a byte across some tap settings for a given byte the failure occurred for (DQ IDELAY is not adjusted). See Figure 38-48 for an example of the delays used for the capture: BUS_DATA_BURST_0 holds a single burst of data when PQTR/NQTR set to 0 taps. BUS_DATA_BURST_1 holds a single burst of data when PQTR/NQTR set to 90°. BUS_DATA_BURST_2 holds a single burst of data when PQTR/NQTR set to 180°. BUS_DATA_BURST_3 holds a single burst of data when PQTR/NQTR set to 270°.

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figure 38-78 and Figure 38-79 are examples of how the data is converted.

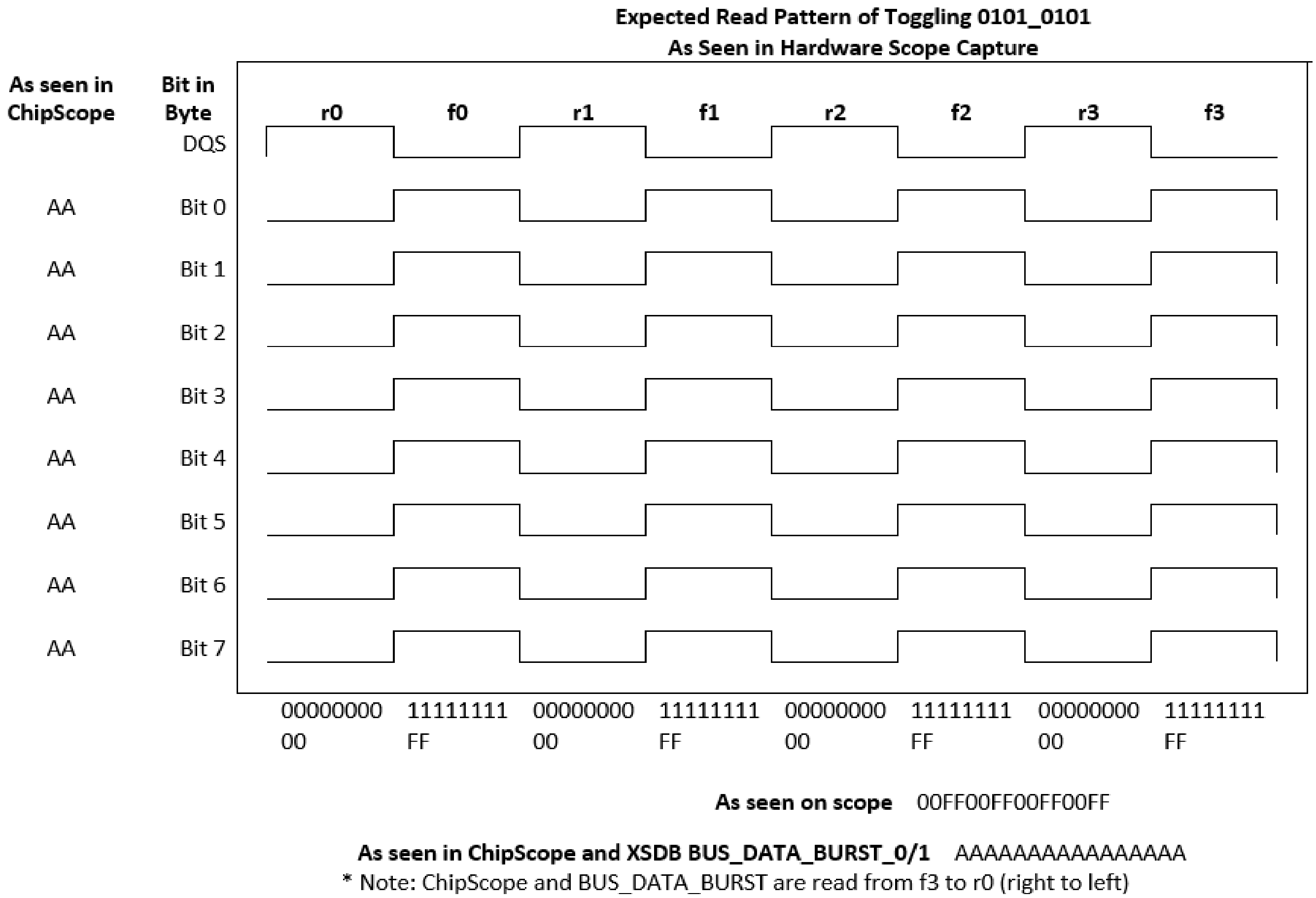


Figure 38-78: **Expected Read Pattern of Toggling 0101_0101**

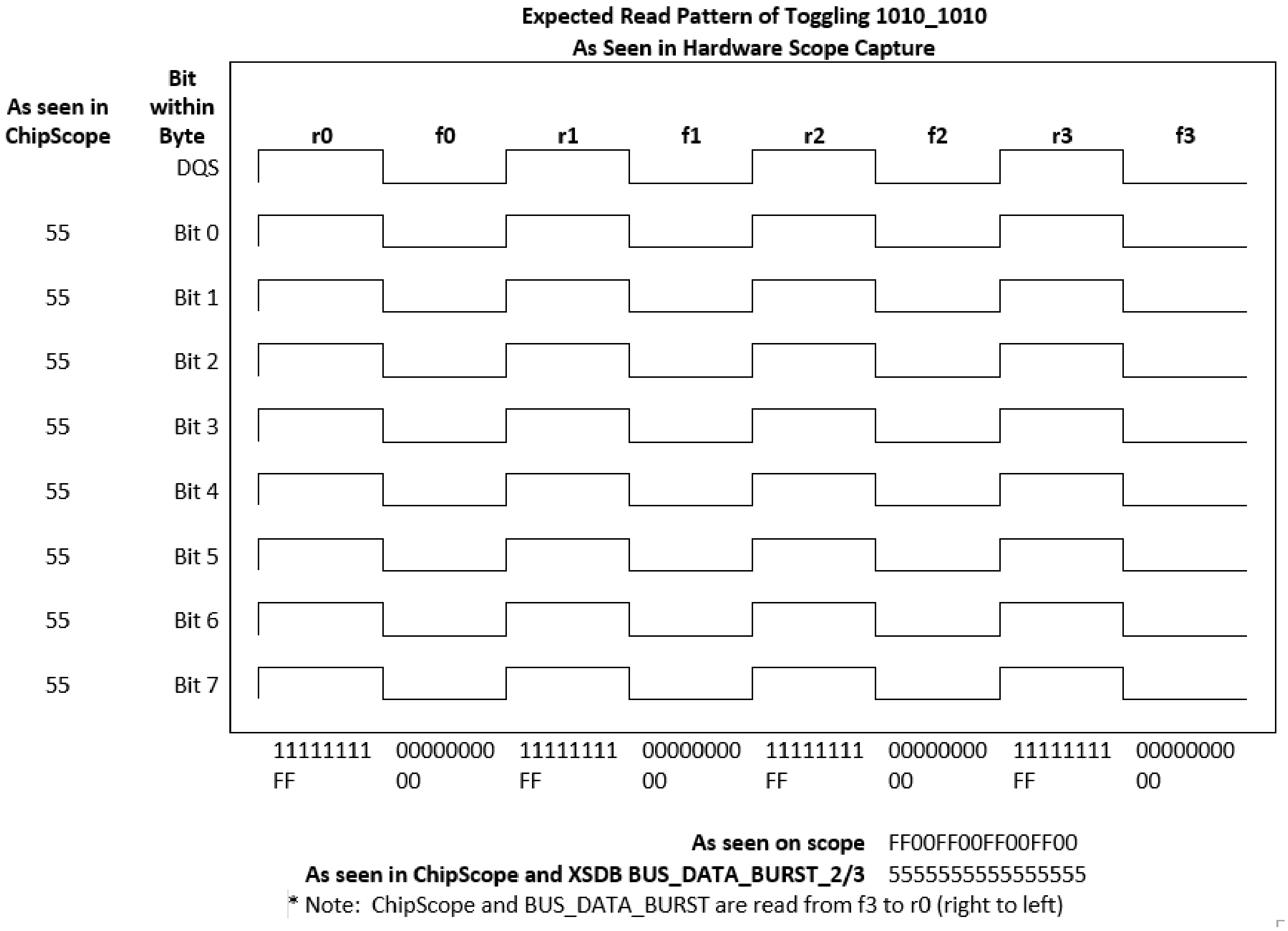


Figure 38-79: **Expected Read Pattern of Toggling 1010_1010**

This is a sample of results for Read DQS Centering DBI (Simple) XSDB debug signals (nibbles that do not contain the DBI pin are skipped and hence the fields are all 0):

```
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE0   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE1   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE2   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE3   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE4   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE5   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE6   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE7   string  true       true     000
RDLVL_DBI_NQTR_CENTER_RANK0_BYTE8   string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE0     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE1     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE2     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE3     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE4     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE5     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE6     string  true       true     000
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE7     string  true       true     001
RDLVL_DBI_NQTR_LEFT_RANK0_BYTE8     string  true       true     000
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE0    string  true       true     058
```

```
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE1    string  true       true     050
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE2    string  true       true     04f
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE3    string  true       true     050
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE4    string  true       true     051
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE5    string  true       true     051
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE6    string  true       true     052
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE7    string  true       true     050
RDLVL_DBI_NQTR_RIGHT_RANK0_BYTE8    string  true       true     04c
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE0   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE1   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE2   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE3   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE4   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE5   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE6   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE7   string  true       true     000
RDLVL_DBI_PQTR_CENTER_RANK0_BYTE8   string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE0     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE1     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE2     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE3     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE4     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE5     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE6     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE7     string  true       true     000
RDLVL_DBI_PQTR_LEFT_RANK0_BYTE8     string  true       true     000
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE0    string  true       true     065
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE1    string  true       true     05f
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE2    string  true       true     05b
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE3    string  true       true     061
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE4    string  true       true     05d
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE5    string  true       true     05e
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE6    string  true       true     05e
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE7    string  true       true     067
RDLVL_DBI_PQTR_RIGHT_RANK0_BYTE8    string  true       true     05e
RDLVL_IDELAY_DBI_FINAL_BYTE0        string  true       true     03b
RDLVL_IDELAY_DBI_FINAL_BYTE1        string  true       true     03a
RDLVL_IDELAY_DBI_FINAL_BYTE2        string  true       true     031
RDLVL_IDELAY_DBI_FINAL_BYTE3        string  true       true     038
RDLVL_IDELAY_DBI_FINAL_BYTE4        string  true       true     034
RDLVL_IDELAY_DBI_FINAL_BYTE5        string  true       true     03a
RDLVL_IDELAY_DBI_FINAL_BYTE6        string  true       true     035
RDLVL_IDELAY_DBI_FINAL_BYTE7        string  true       true     03c
RDLVL_IDELAY_DBI_FINAL_BYTE8        string  true       true     030
RDLVL_IDELAY_DBI_RANK0_BYTE0        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE1        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE2        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE3        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE4        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE5        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE6        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE7        string  true       true     000
RDLVL_IDELAY_DBI_RANK0_BYTE8        string  true       true     000
```

Expected Results

- Look at the window measured during Read DQS Centering (Simple) and compare what is found during Read DQS Centering DBI (Simple). The eye size found should be similar, and the PQTR/NQTR should not move by more than 10 taps typically.
- Determine if any bytes completed successfully. The algorithm sequentially steps through each DQS byte sequentially.

Hardware Measurements

1. Probe the write commands and read commands at the memory:
    - Write = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 1; `act_n` = 1
    - Read = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 0; `act_n` = 1
2. Probe a data pin and DBI pin to check for data being returned from the DRAM.
3. Probe the writes checking the signal level of the write DQS and the write DQ.
4. Probe the DBI pin which should be deasserted during the write burst. The DBI pin should not be asserted since DBI write should be OFF.
5. Probe the read burst after the write and check if the expected data pattern is being returned.
6. Check for floating address pins if the expected data is not returned.
7. Check for any stuck-at level issues on DQ/DBI pins whose signal level does not change. If at all possible probe at the receiver to check termination and signal integrity.
8. Check the DBG port signals and the full read data and comparison result to check the data in general interconnect. The calibration algorithm has RTL logic issue the commands and check the data.
9. Check if the `dbg_rd_valid` aligns with the data pattern or is OFF (which can indicate an issue with DQS gate calibration). Set up a trigger when the error gets asserted to capture signals in the hardware debugger for analysis.
10. Re-check results from DQS gate or other previous calibration stages. Compare passing byte lanes against failing byte lanes for previous stages of calibration. If a failure occurs during simple pattern calibration, check the values found during deskew for example.
11. All of the data comparison for read DQS Centering occurs in the general interconnect, so it can be useful to pull in the debug data in the hardware debugger and take a look at what the data looks like coming back as taps are adjusted, see Figure 38-80. The screen captures are from simulation, with a small burst of five reads. Look at `dbg_rd_data`, `dbg_rd_data_cmp`, and `dbg_rd_valid`.
12. Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the Read DBI Deskew trigger to `cal_r*_status[22]` = R (rising edge). To view each byte, add an additional trigger on `dbg_cmp_byte` and

set to the byte of interest. The following simulation example shows how the debug signals should behave during successful Read DQS Centering with DBI.

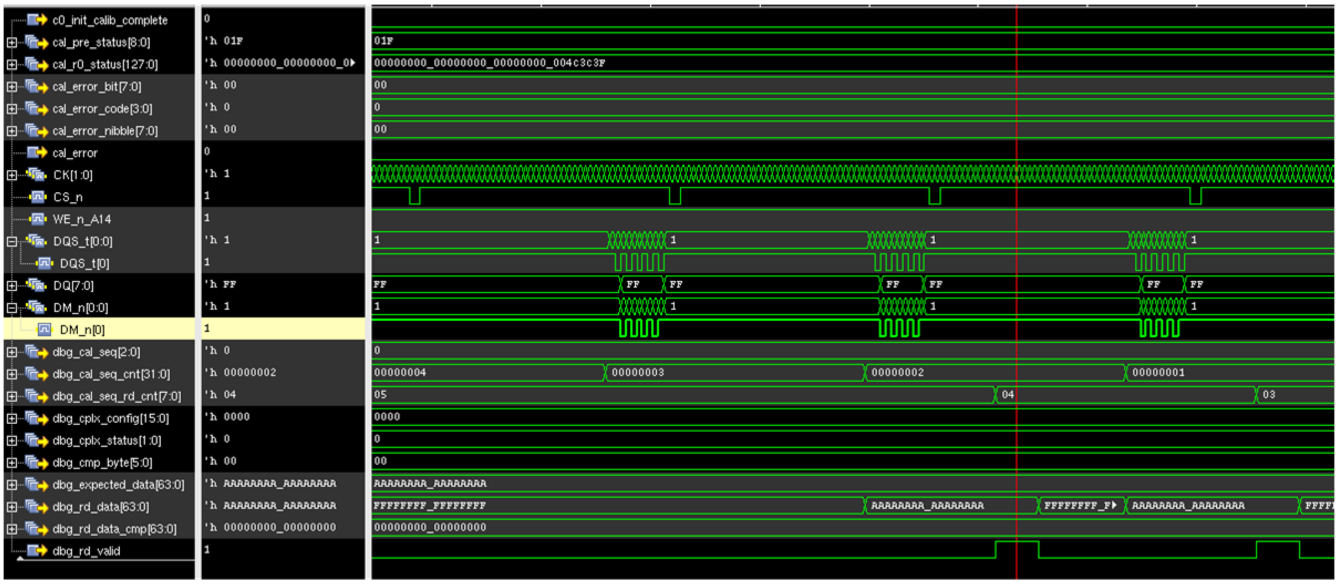


Figure 38-80: **RTL Debug Signals during Read DQS Centering with DBI (No Error)**

13. After failure during this stage of calibration, the design goes into a continuous loop of read commands to allow board probing.

Write Latency Calibration

Write latency calibration is required to align the write DQS to the correct CK edge. During write leveling, the write DQS is aligned to the nearest rising edge of CK. However, this might not be the edge that captures the write command. Depending on the interface type (UDIMM, RDIMM, LRDIMM, or component), the DQS could be up to three CK cycles earlier than, or aligned to the CK edge that captures the write command.

Write latency calibration makes use of the coarse tap in the WL_DLY_RNK of the XIPHY for adjusting the write latency on a per byte basis. Write leveling uses up a maximum of three coarse taps of the XIPHY delay to ensure each write DQS is aligned to the nearest clock edge. Memory Controller provides the write data 1TCK early to the PHY, which is then delayed by write leveling up to one memory clock cycle. This means for the zero PCB delay case of a typical simulation the data would be aligned at the DRAM without additional delay added from write calibration.

Write latency calibration can only account for early data, because in the case where the data arrives late at the DRAM there is no push back on the controller to provide the data earlier. With 16 XIPHY coarse taps available (each tap is 90°), four memory clock cycles of shift are available in the XIPHY with one memory clock used by write leveling. This leaves three memory clocks of delay available for write latency calibration.

Figure 38-81 shows the calibration flow to determine the setting required for each byte.

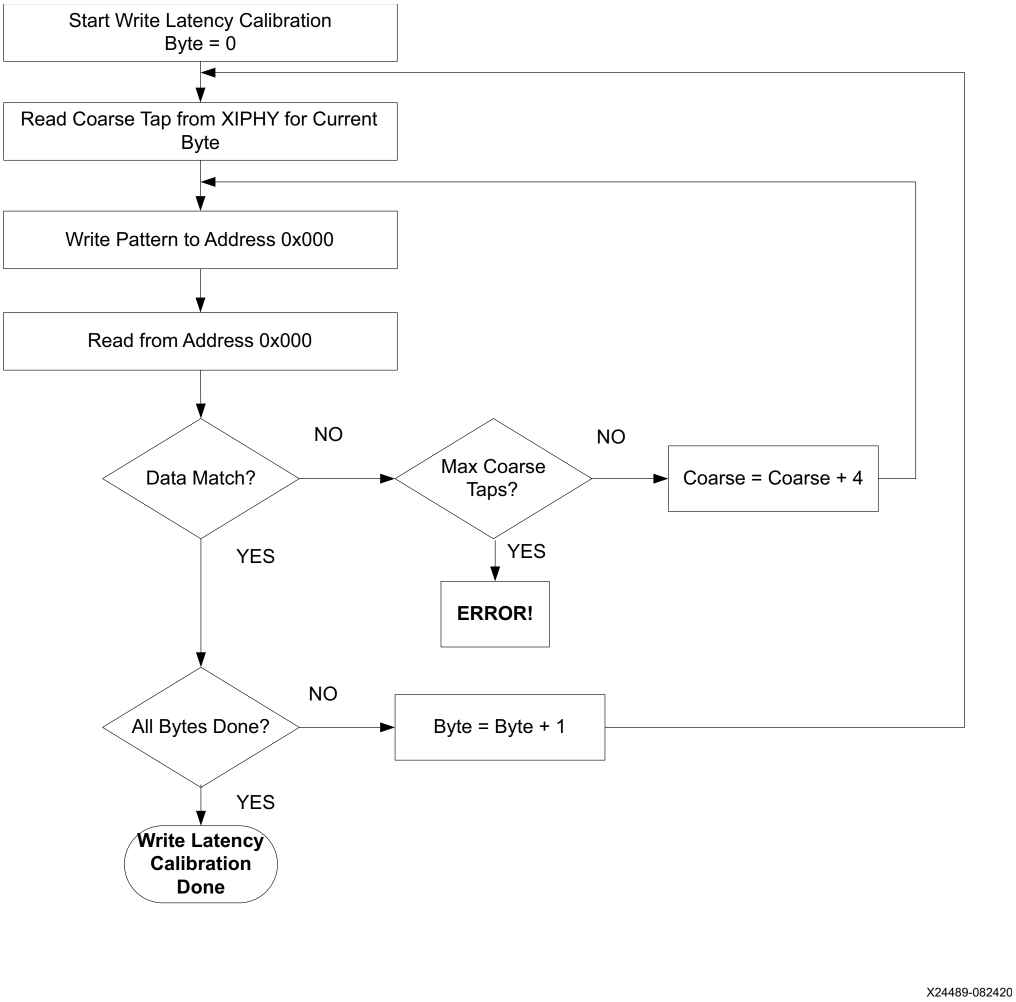


Figure 38-81: **Write Latency Calibration Flow**

The write DQS for the write command is extended for longer than required to ensure the DQS is toggling when the DRAM expects it to clock in the write data. A specific data pattern is used to check when the correct data pattern gets written into the DRAM, as shown in Figure 38-82.

In the example at the start of write latency calibration for the given byte. the target write latency falls in the middle of the data pattern. The returned data would be `55AA9966FFFFFFFF` rather than the expected `FF00AA5555AA9966`. The write DQS and data are delayed using the XIPHY coarse delay and the operation is repeated, until the correct data pattern is found or there are no more coarse taps available. After the pattern is found, the amount of coarse delay required is indicated by WRITE_LATENCY_CALIBRATION_COARSE_Rank_Byte.

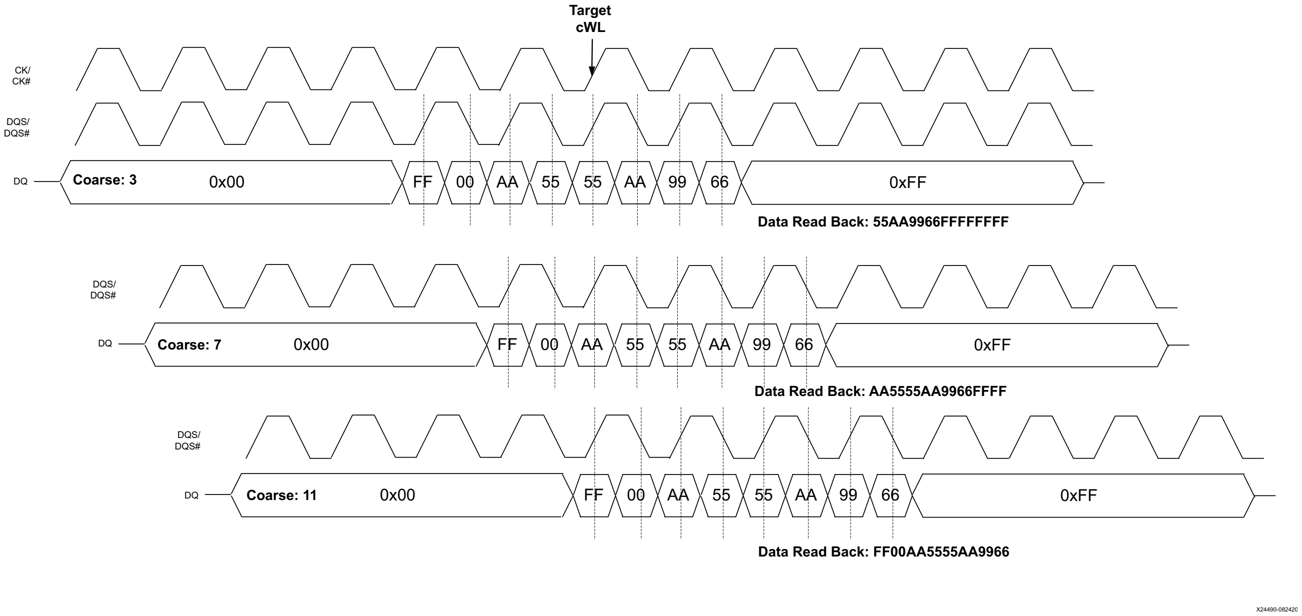
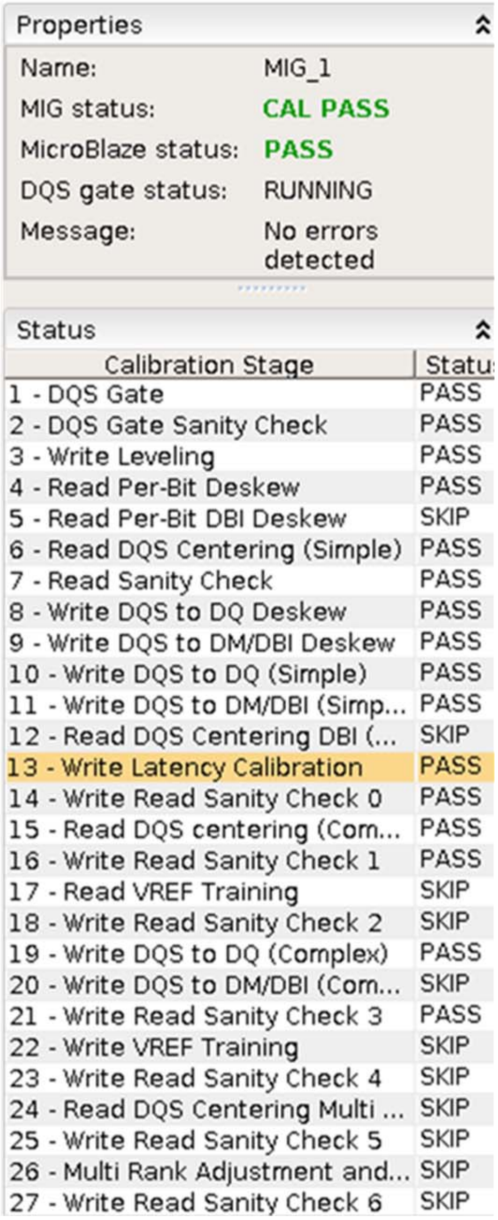


Figure 38-82: **Write Latency Calibration Alignment Example**

- If the data pattern is not found for a given byte, the data pattern found is checked to see if the data at the maximum delay available still arrives too early (indicating not enough adjustment was available in the XIPHY to align to the correct location) or if the first burst with no extra delay applied is already late (indicating at the start the data would need to be pulled back). The following data pattern is checked:
  - Expected pattern on a per-nibble basis: `F0A55A96`
  - Late Data Comparison: `00F0AA55A`
  - Early Data Comparison: `A55A96FF`, `5A96FFFF`, `96FFFFFF`
- If neither of these cases holds true, an attempt is made to try to reclassify the error as either a write or a read failure. A single write burst is sent to the DRAM followed by 20 read bursts. The data from the first read burst is stored for comparison with the remaining 19 read bursts.
- If all the read data matches, the error is classified as a write failure.
- If the data does not match, it is marked as a read failure.

Debug

To determine the status of Write Latency Calibration, click the **Write Latency Calibration** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Properties

Name:	MIG_1
MIG status:	CAL PASS
MicroBlaze status:	PASS
DQS gate status:	RUNNING
Message:	No errors detected

Status

Calibration Stage	Statu
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simp...	PASS
12 - Read DQS Centering DBI (...	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Com...	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Com...	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi ...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and...	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-83: **Memory IP XSDB Debug GUI Example – Write Latency Calibration**

The status of Write Latency Calibration can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-29. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-29: **DDR_CAL_ERROR Decode for Write Latency Calibration**

Write Latency DDR_CAL_ ERROR_ CODE	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	Byte	N/A	Could not find the data pattern given the amount of movement available.	Check BUS_DATA_BURST XSDB data to check which bits failed or what data looked like when failed. Check margin for the byte for earlier stages of calibration. Probe the DQS/DQ signals (and DM if applicable).
0x2	Byte	N/A	Data pattern not found. Data late at the start, instead of "F0A55A96," found "00F0A55A."	Check trace lengths for signals against what is allowed. If other Bytes calibrated properly check the WRITE_LATENCY_CALIBRATION_COARSE setting for them and check how much movement was required to calibrate them. Check that the CAS write latency is set properly during the initialization sequence.
0x3	Byte	N/A	Data pattern not found. Data too early, not enough movement to find pattern. Found pattern of "A55A96FF," "5A96FFFF," or "96FFFFFF."	Check trace lengths for signals against what is allowed. If other Bytes calibrated properly check the WRITE_LATENCY_CALIBRATION_COARSE setting for them and check how much movement was required to calibrate them. Check that the CAS write latency is set properly during the initialization sequence.
0x4	Byte	N/A	Data pattern not found. Multiple reads to the same address resulted in a read mismatch.	Check read data margins from earlier stages of calibration. Check signal integrity during reads on the DQs and DQ. Check BUS_DATA_BURST XSDB data to check which bits failed.
0xF	Byte	N/A	Timeout error waiting for read data to return.	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-30 shows the signals and values adjusted or used during the Write Latency stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** in the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-30: **Signals of Interest for Write Latency Calibration**

Signal	Usage	Signal Description
WRITE_LATENCY_CALIBRATION_COARSE	One per Byte	Number of coarse taps added during Write Latency calibration.
BUS_DATA_BURST (2014.3+)		During calibration for a byte the read data is saved to XSDB for later analysis in case of a failure. BUS_DATA_BURST_0 holds the read burst for at the starting coarse tap value left by write leveling (initial coarse tap setting). BUS_DATA_BURST_1 holds the read burst at initial coarse tap + 4. BUS_DATA_BURST_2 holds the read burst at initial coarse tap + 8. BUS_DATA_BURST_3 holds the read burst at initial coarse tap + 12. After a given byte finishes calibration, the BUS_DATA_BURST registers are cleared to 0 for use by the next byte.

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figure 38-84 to Figure 38-86 show examples of how the data is converted. Because all Fs are written before this expected Write Latency pattern and all 0s after, this pattern can have Fs before and 0s after until Write Latency calibration is completed at which time Figure 38-84 to Figure 38-86 are accurate representation.

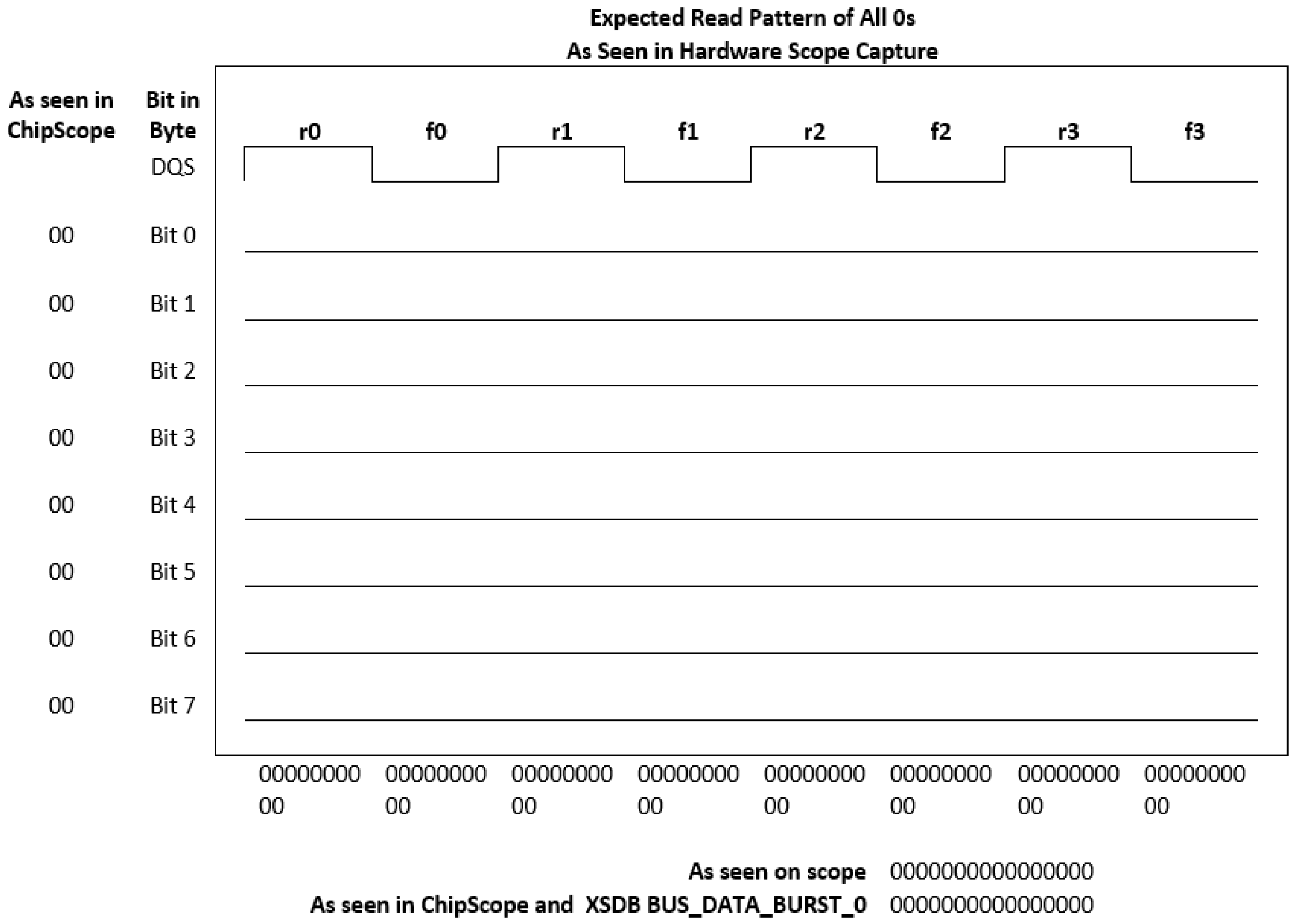


Figure 38-84: **Expected Read Pattern of All 0s**

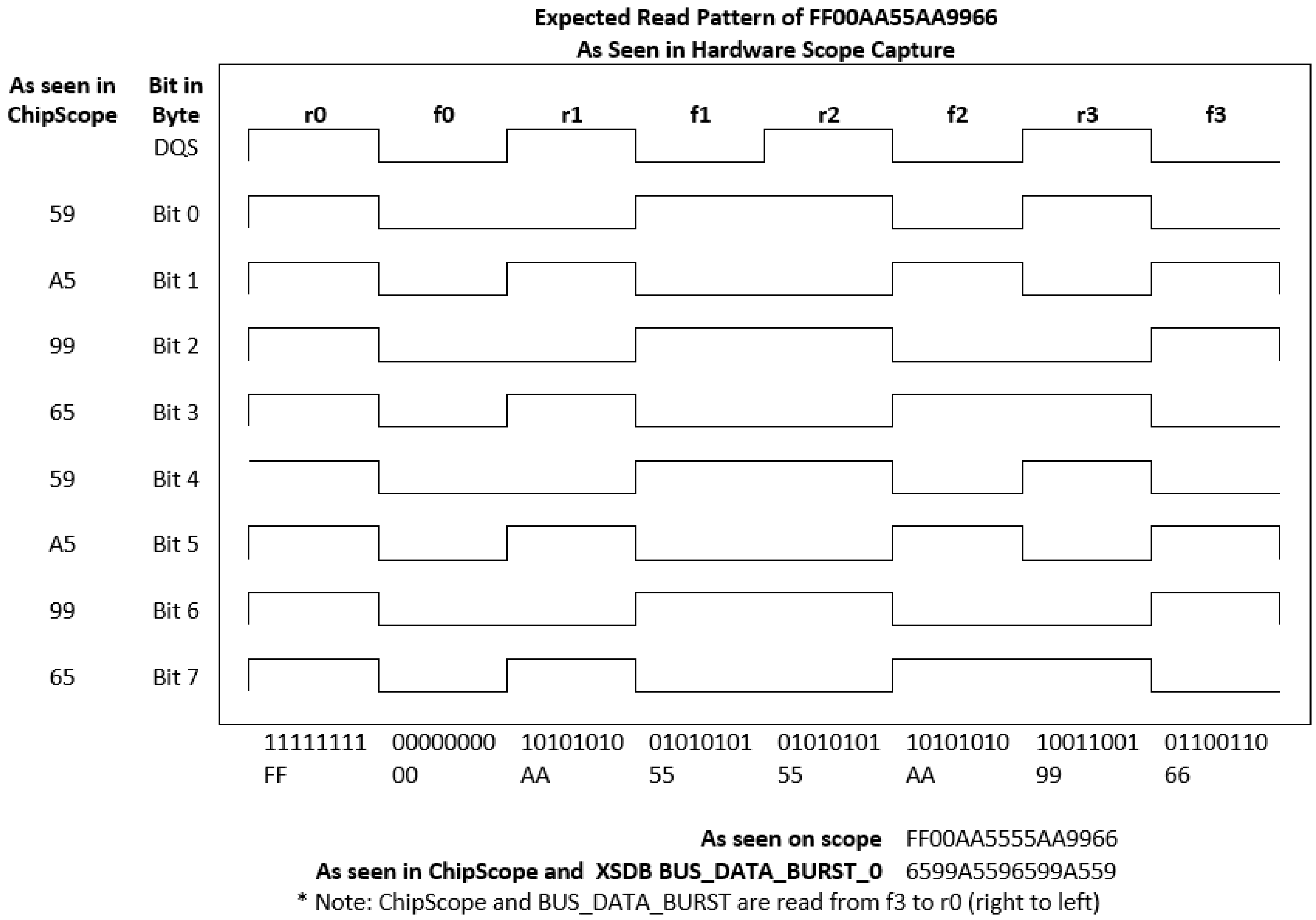


Figure 38-85: **Expected Read Pattern of FF00AA55AA9966**

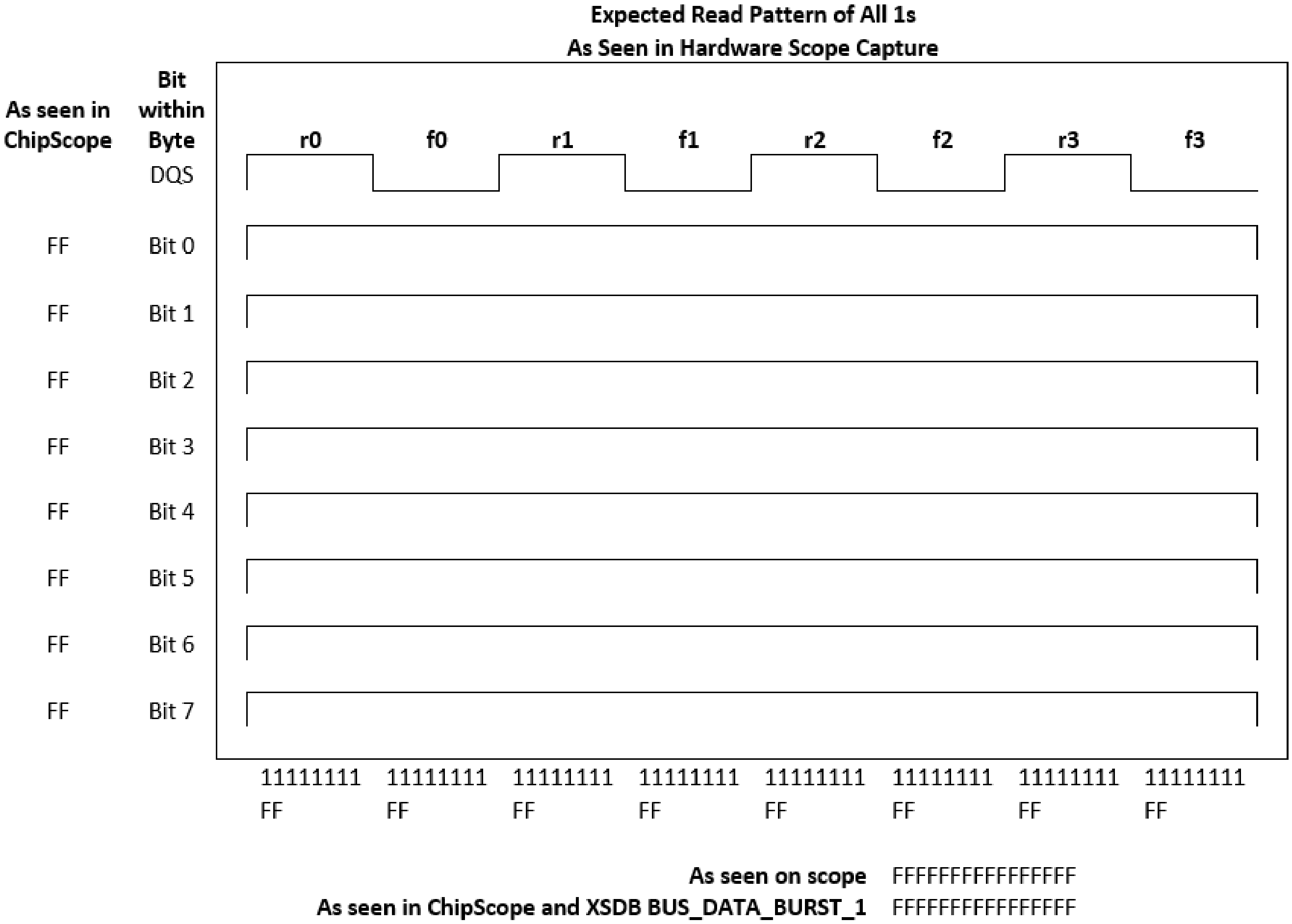


Figure 38-86: **Expected Read Pattern of All 1s**

This is a sample of results for the Write Latency XSDB debug signals:

```
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE0      string true true 003
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE1      string true true 004
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE2      string true true 004
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE3      string true true 004
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE4      string true true 006
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE5      string true true 005
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE6      string true true 005
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE7      string true true 005
WRITE_LATENCY_CALIBRATION_COARSE_RANK0_BYTE8      string true true 005
```

Hardware Measurements

If the design is stuck in the Write Latency stage, the issue could be related to either the write or the read. Determining whether the write or read is causing the failure is critical. The following steps should be completed. For additional details and examples, see the Determining If a Data Error is Due to the Write or Read, page 770 section.

1. To trigger on the start of Write Latency Calibration, set the trigger to (`cal_r*_status[24]` = R for Rising Edge).
2. To trigger on the end of Write Latency Calibration, set the trigger to (`cal_r*_status[25]` = R for Rising Edge). To look at each byte, additionally add a trigger on `dbg_cmp_byte` and set to the byte of interest.
3. To ensure the writes are correct, observe the write DQS to write DQ relationship at the memory using high quality scope and probes. During Write Latency, a write is followed by a read so care needs to be taken to ensure the write is captured. For more information, see the Determining If a Data Error is Due to the Write or Read, page 770 section. If there is a failing bit, determining the write DQS to write DQ relationship for the specific DQ bit is critical. The write ideally has the DQS center aligned in the DQ window. Misalignment between DQS and DQ during Write Calibration points to an issue with Write DQS Centering calibration. Review the Debugging Write DQS Centering Failures, page 680 section.
4. If the DQ-DQS alignment looks correct, next observe the `we_n` to DQS relationship at the memory during a write again using high quality scope and probes. The `we_n` to DQS delay must equal the CAS Write Latency (CWL).
5. Using high quality scope and probes, verify the expected pattern (`FF00AA5555AA9966`) is being written to the DRAM during a write and that the expected pattern is being read back during the first Write Calibration read. If the pattern is correct during write and read at the DRAM, verify the DQS-CK alignment. During Write Calibration, these two signals should be aligned. Write Leveling aligned these two signals which has successfully completed before Write Latency.
6. Probe ODT and `we_n` during a write command. For ODT to be properly powered on in the memory, ODT must assert before the write command.
7. Probe DM to ensure it is held low during calibration. If a board issue exists causing DM to improperly assert, incorrect data can be read back during calibration causing a write calibration failure. An example of a board issue on DM is when DM is not used and tied low at the memory with improper termination.

Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the trigger.

- To trigger on the start of Write Latency Calibration, set the trigger to (`cal_r*_status[24]` = R for Rising Edge).
- To trigger on the end of Write Latency Calibration, set the trigger to (`cal_r*_status[25]` = R for Rising Edge). To look at each byte, additionally add a trigger on `dbg_cmp_byte` and set to the byte of interest.

The following simulation example shows how the debug signals should behave during successful Write Latency Calibration.

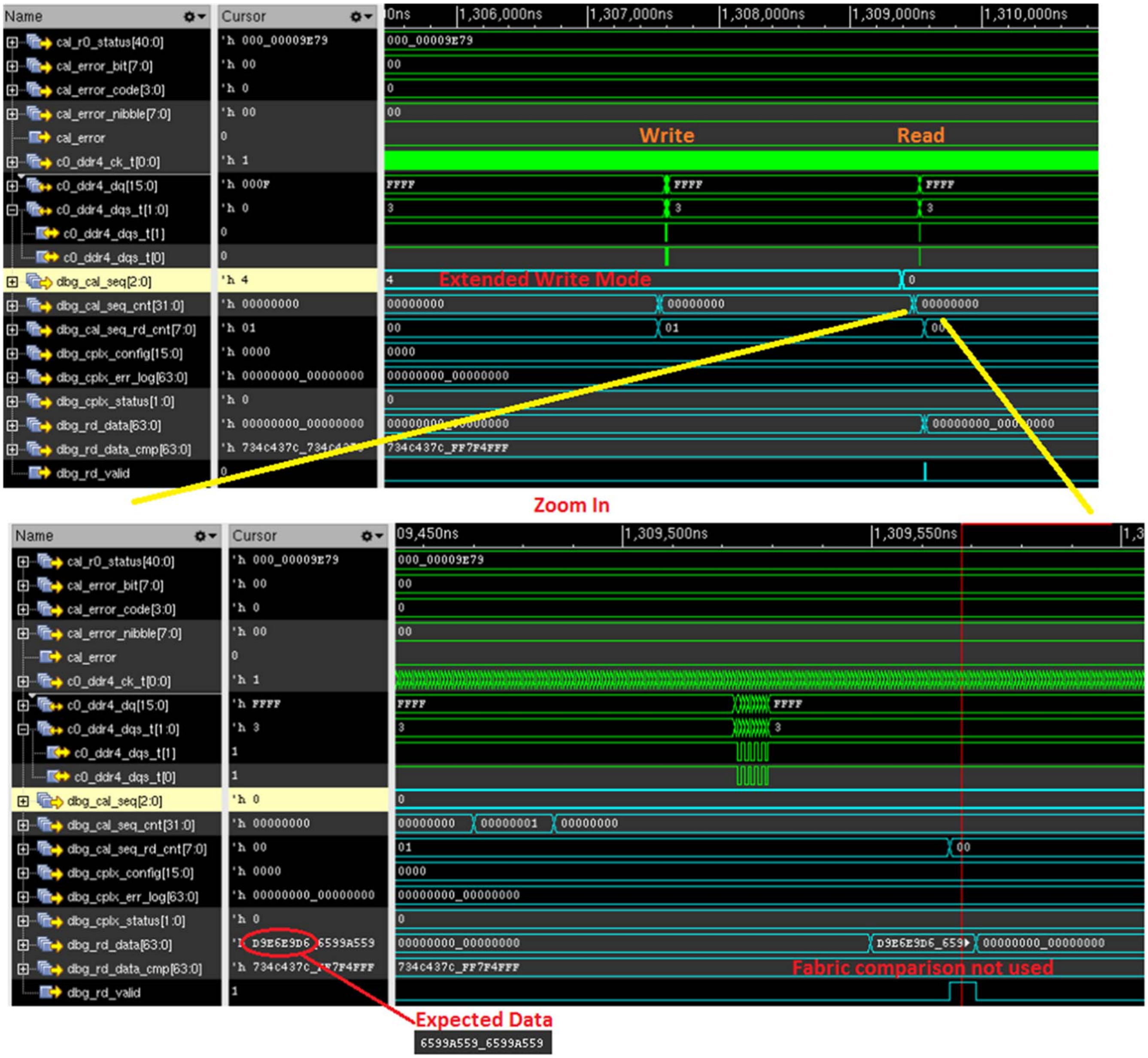


Figure 38-87: **RTL Debug Signals during Write Latency Calibration (x4 Example Shown)**

Expected Results

The expected value on WRITE_LATENCY_CALIBRATION_COARSE is dependent on the starting point set by Write Leveling (which can be 0 to 4). The PCB skew to the SDRAM typically adds up to two memory clock cycles to this starting point where each clock cycle is four coarse taps.

Debugging Read Complex Pattern Calibration Failures

Note: Only enabled for data rates above 1,600 Mb/s.

Complex data patterns are used for advanced read DQS centering for memory systems to improve read timing margin. Long and complex data patterns on both the victim and aggressor DQ lanes impact the size and location of the data eye. The objective of the complex calibration step is to generate the worst case data eye on each DQ lane so that the DQS signal can be aligned, resulting in good setup/hold margin during normal operation with any work load.

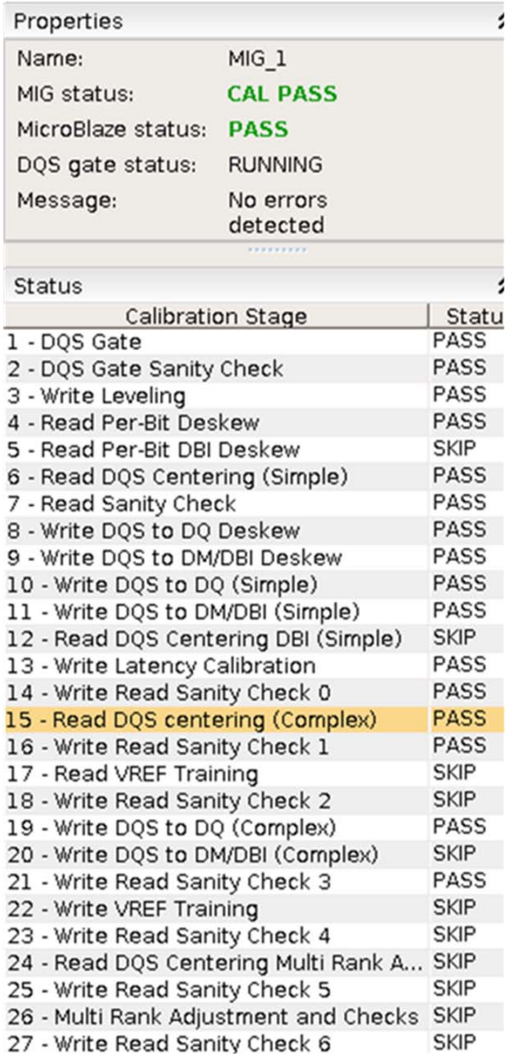
There are two long data patterns stored in a block RAM, one for a victim DQ lane, and an aggressor pattern for all other DQ lanes. These patterns are used to generate write data, as well as expected data on reads for comparison and error logging. Each pattern consists of 157 8-bit chunks or BL8 bursts.

Each DQ lane of 1-byte takes a turn at being the victim. An RTL state machine automatically selects each DQ lane in turn, MUXing the victim or aggressor patterns to the appropriate DQ lanes, issues the read/write transactions, and records errors. The victim pattern is only walked across the DQ lanes of the selected byte to be calibrated, and all other DQ lanes carry the aggressor pattern, including all lanes in un-selected bytes if there is more than 1-byte lane.

Similar steps to those described in Read DQS Centering are performed, with the PQTR/NQTR starting out at the left edge of the simple window found previously. The complex pattern is written and read back. All bits in a nibble are checked to find the left edge of the window, incrementing the bits together as needed or the PQTR/NQTR to find the aggregate left edge. After the left and right edges are found, it steps through the entire data eye.

Debug

To determine the status of Complex Read Leveling Calibration, click the **Read DQS Centering (Complex)** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Properties

Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Status

Calibration Stage	Statu
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	PASS
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank A...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Checks	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-88: **Memory IP XSDB Debug GUI Example – Read DQS Centering (Complex)**

The status of Read Leveling Complex can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-31. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-31: **DDR_CAL_ERROR Decode for Complex Read Leveling**

Read DQS Centering DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Nibble	N/A	No valid data found for a given bit in the nibble	Check if the design meets timing. Check the margin found for the simple pattern for the given nibble/byte. Check if the IDELAY values used for each bit are reasonable to others in the byte. Check the dbg_cplx_config, dbg_cplx_status, dbg_cplx_err_log, dbg_rd_data, and dbg_expected_data during this stage of calibration. Determine if it is a read or a write error by measuring the signals on the bus after the write.
0x2	Nibble	N/A	Could not find the left Edge (error condition) to determine window size	Check the dbg_cplx_config, dbg_cplx_status, dbg_cplx_err_log, dbg_rd_data, and dbg_expected_data and see if the data changes during this stage of calibration.
0xF	Nibble	N/A	Timeout error waiting for read data to return	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-32 shows the signals and values adjusted or used during the Read Leveling Complex stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-32: **Signals of Interest for Complex Pattern Calibration**

Signal	Usage	Signal Description
RDLVL_COMPLEX_PQTR_LEFT_Rank*_Nibble*	One per nibble	Read leveling PQTR tap position when left edge of read data valid window is detected (complex pattern).
RDLVL_COMPLEX_NQTR_LEFT_Rank*_Nibble*	One per nibble	Read leveling NQTR tap position when left edge of read data valid window is detected (complex pattern).
RDLVL_COMPLEX_PQTR_RIGHT_Rank*_Nibble*	One per nibble	Read leveling PQTR tap position when right edge of read data valid window is detected (complex pattern).
RDLVL_COMPLEX_NQTR_RIGHT_Rank*_Nibble*	One per nibble	Read leveling NQTR tap position when right edge of read data valid window is detected (complex pattern).
RDLVL_COMPLEX_PQTR_CENTER_Rank*_Nibble*	One per nibble	Read leveling PQTR center tap position found at the end of read DQS centering (complex pattern).

Table 38-32: **Signals of Interest for Complex Pattern Calibration *(Cont'd)***

Signal	Usage	Signal Description
RDLVL_COMPLEX_NQTR_CENTER_Rank*_Nibble*	One per nibble	Read leveling NQTR center tap position found at the end of read DQS centering (complex pattern).
RDLVL_COMPLEX_IDELAY_Rank*_Bit*	One per Bit	Read leveling IDELAY delay value (complex pattern).
RDLVL_COMPLEX_IDELAY_DBI_Byte*	One per Byte	Reserved

This is a sample of results for Complex Read Leveling using the Memory IP Debug GUI within the Hardware Manager.

Note: Either the "Table" or "Chart" view can be used to look at the calibration windows.

Figure 38-89 and Figure 38-90 are screen captures from 2015.1 and might vary from the current version.

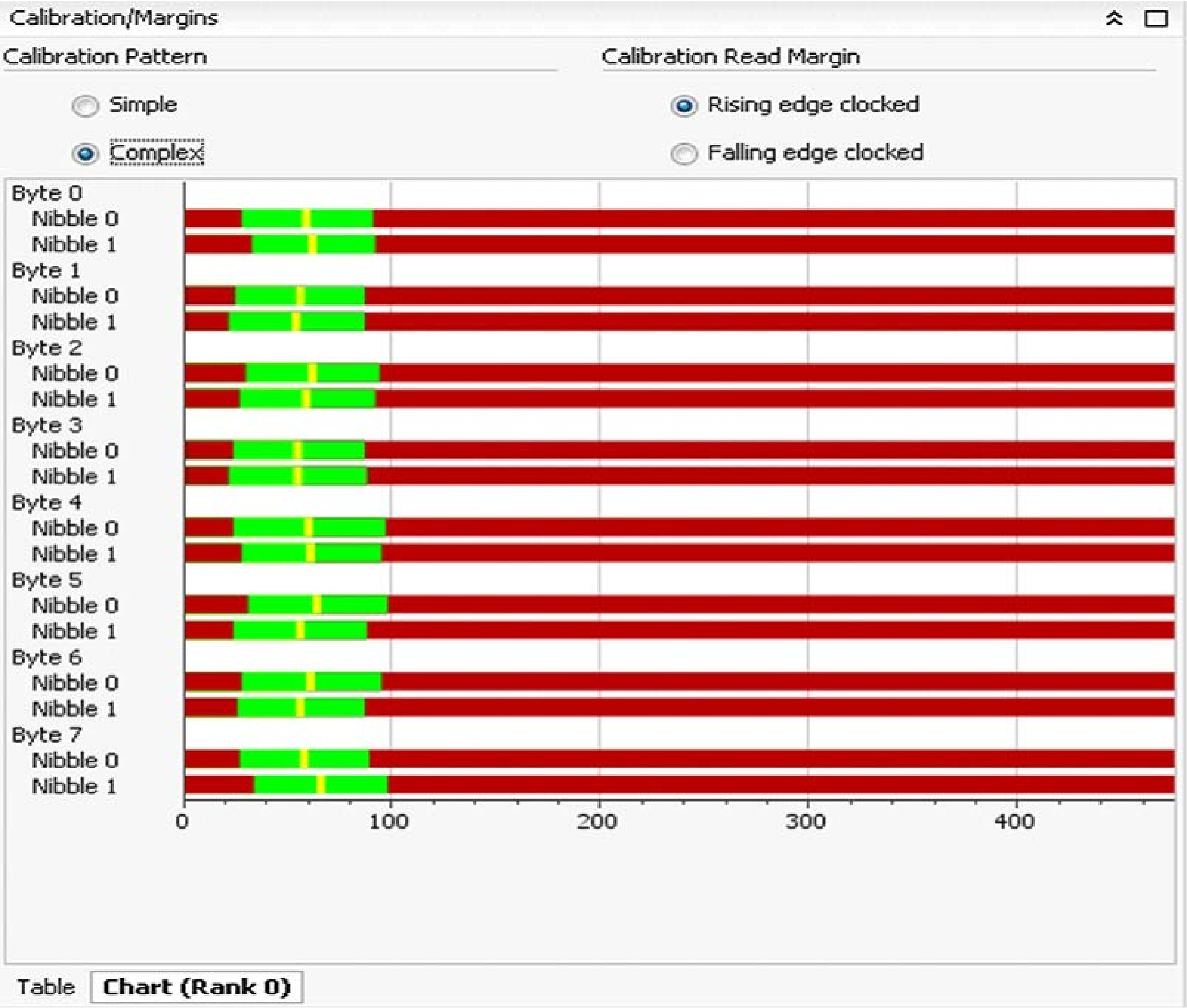


Figure 38-89: **Example of Complex Read Calibration Margin**

This is a sample of results for the Read Leveling Complex XSDB debug signals:

```
RDLVL_COMPLEX_IDELAY_DBI_BYTE0              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE1              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE2              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE3              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE4              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE5              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE6              string true true 000
RDLVL_COMPLEX_IDELAY_DBI_BYTE7              string true true 000
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT0       string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT1       string true true 03e
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT2       string true true 042
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT3       string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT4       string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT5       string true true 03e
```

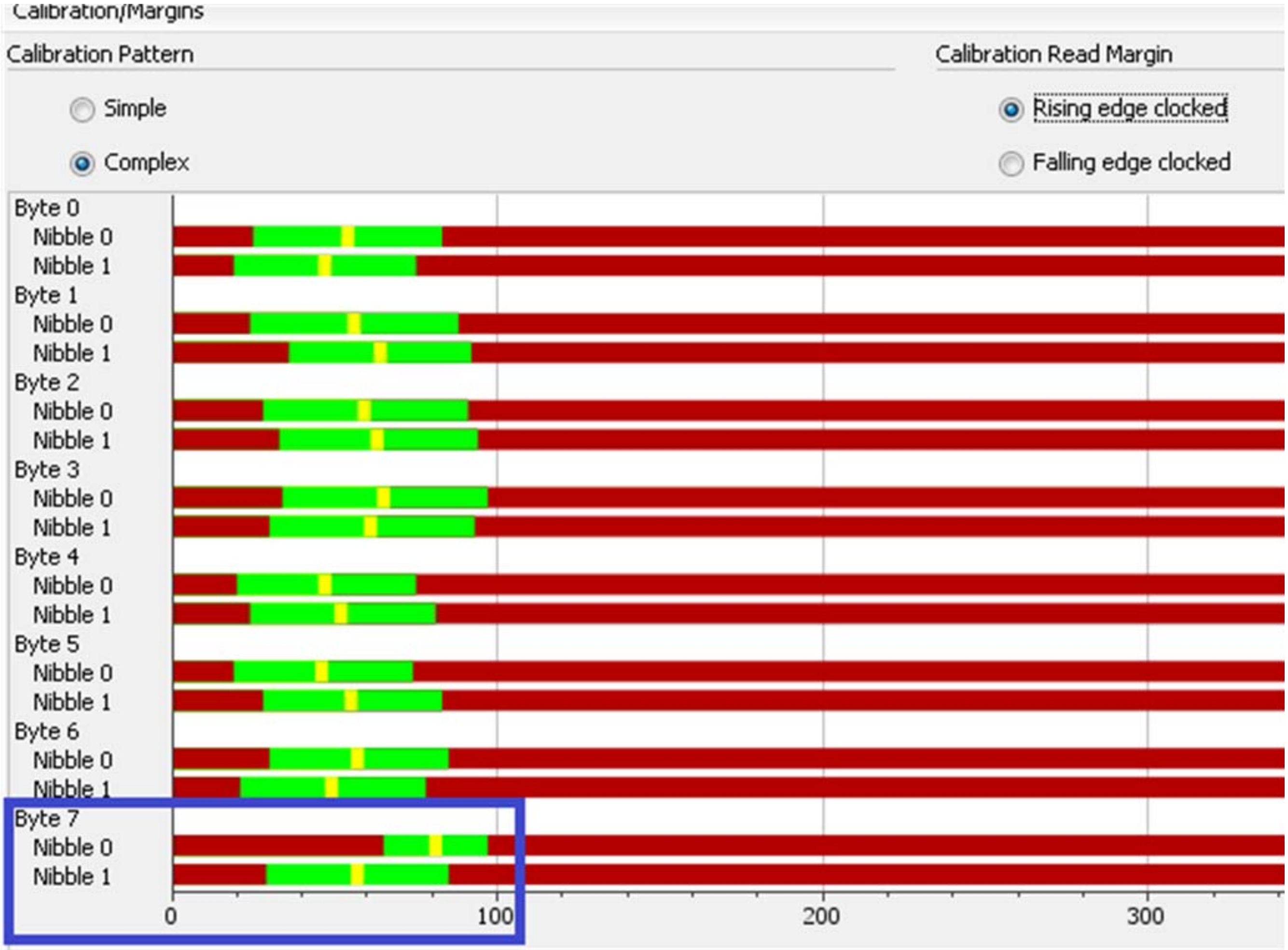
```
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT6          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE0_BIT7          string true true 03e
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT0          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT1          string true true 042
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT2          string true true 03a
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT3          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT4          string true true 03f
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT5          string true true 042
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT6          string true true 03e
RDLVL_COMPLEX_IDELAY_RANK0_BYTE1_BIT7          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT0          string true true 043
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT1          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT2          string true true 047
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT3          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT4          string true true 000
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT5          string true true 03f
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT6          string true true 043
RDLVL_COMPLEX_IDELAY_RANK0_BYTE2_BIT7          string true true 03c
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT0          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT1          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT2          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT3          string true true 03c
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT4          string true true 03e
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT5          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT6          string true true 038
RDLVL_COMPLEX_IDELAY_RANK0_BYTE3_BIT7          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT0          string true true 044
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT1          string true true 045
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT2          string true true 046
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT3          string true true 042
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT4          string true true 046
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT5          string true true 041
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT6          string true true 043
RDLVL_COMPLEX_IDELAY_RANK0_BYTE4_BIT7          string true true 041
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT0          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT1          string true true 048
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT2          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT3          string true true 047
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT4          string true true 03f
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT5          string true true 04c
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT6          string true true 040
RDLVL_COMPLEX_IDELAY_RANK0_BYTE5_BIT7          string true true 048
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT0          string true true 038
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT1          string true true 043
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT2          string true true 038
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT3          string true true 042
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT4          string true true 03b
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT5          string true true 041
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT6          string true true 03d
RDLVL_COMPLEX_IDELAY_RANK0_BYTE6_BIT7          string true true 042
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT0          string true true 044
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT1          string true true 041
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT2          string true true 048
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT3          string true true 043
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT4          string true true 048
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT5          string true true 043
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT6          string true true 049
RDLVL_COMPLEX_IDELAY_RANK0_BYTE7_BIT7          string true true 045
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE0        string true true 03c
```

```
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE1        string true true 041
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE2        string true true 03b
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE3        string true true 038
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE4        string true true 03a
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE5        string true true 039
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE6        string true true 038
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE7        string true true 038
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE8        string true true 03a
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE9        string true true 03f
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE10       string true true 041
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE11       string true true 03a
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE12       string true true 03d
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE13       string true true 039
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE14       string true true 036
RDLVL_COMPLEX_NQTR_CENTER_RANK0_NIBBLE15       string true true 040
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE0          string true true 01a
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE1          string true true 020
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE2          string true true 01c
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE3          string true true 018
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE4          string true true 01a
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE5          string true true 018
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE6          string true true 017
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE7          string true true 017
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE8          string true true 016
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE9          string true true 01d
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE10         string true true 020
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE11         string true true 01a
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE12         string true true 01b
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE13         string true true 018
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE14         string true true 013
RDLVL_COMPLEX_NQTR_LEFT_RANK0_NIBBLE15         string true true 020
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE0         string true true 05f
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE1         string true true 062
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE2         string true true 05b
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE3         string true true 059
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE4         string true true 05b
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE5         string true true 05a
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE6         string true true 059
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE7         string true true 059
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE8         string true true 05e
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE9         string true true 061
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE10        string true true 062
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE11        string true true 05b
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE12        string true true 05f
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE13        string true true 05a
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE14        string true true 05a
RDLVL_COMPLEX_NQTR_RIGHT_RANK0_NIBBLE15        string true true 061
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE0        string true true 03b
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE1        string true true 03e
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE2        string true true 038
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE3        string true true 036
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE4        string true true 03e
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE5        string true true 03b
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE6        string true true 037
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE7        string true true 037
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE8        string true true 03c
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE9        string true true 03d
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE10       string true true 040
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE11       string true true 038
```

```
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE12       string true true 03d
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE13       string true true 038
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE14       string true true 03a
RDLVL_COMPLEX_PQTR_CENTER_RANK0_NIBBLE15       string true true 042
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE0          string true true 01c
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE1          string true true 021
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE2          string true true 019
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE3          string true true 016
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE4          string true true 01e
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE5          string true true 01b
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE6          string true true 018
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE7          string true true 016
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE8          string true true 018
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE9          string true true 01c
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE10         string true true 01f
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE11         string true true 018
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE12         string true true 01c
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE13         string true true 01a
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE14         string true true 01b
RDLVL_COMPLEX_PQTR_LEFT_RANK0_NIBBLE15         string true true 022
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE0         string true true 05b
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE1         string true true 05c
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE2         string true true 057
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE3         string true true 057
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE4         string true true 05e
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE5         string true true 05c
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE6         string true true 057
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE7         string true true 058
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE8         string true true 061
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE9         string true true 05f
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE10        string true true 062
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE11        string true true 058
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE12        string true true 05f
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE13        string true true 057
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE14        string true true 059
RDLVL_COMPLEX_PQTR_RIGHT_RANK0_NIBBLE15        string true true 062
```

Expected Results

- Look at the individual PQTR/NQTR tap settings for each nibble. The taps should only vary by 0 to 20 taps. Use the BISC values to compute the estimated bit time in taps.
  - For example, Byte 7 Nibble 0 in Figure 38-90 is shifted and smaller compared to the remaining nibbles. This type of result is not expected. For this specific example, the SDRAM was not properly loaded into the socket.



Calibration/Margins

Calibration Pattern

○ Simple

● Complex

Calibration Read Margin

● Rising edge clocked

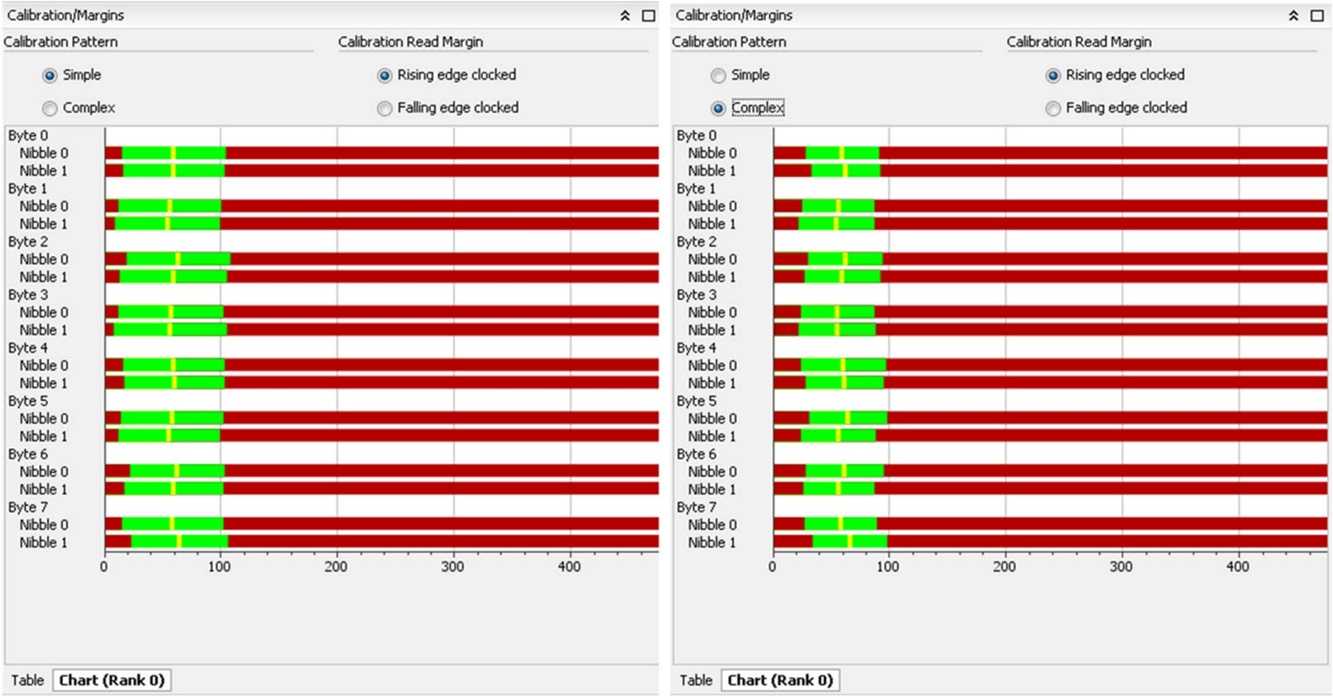
○ Falling edge clocked

Figure 38-90: **Suspicious Calibrated Read Window for Byte 7 Nibble 0**

- Look at the individual IDELAY taps for each bit. The IDELAY taps should only vary by 0 to 20 taps, and is dependent on PCB trace delays. For Deskew the IDELAY taps are typically in the 50 to 70 tap range, while PQTR and NQTR are usually in the 0 to 5 tap range.
- Determine if any bytes completed successfully. The read leveling algorithm sequentially steps through each DQS byte group detecting the capture edges.
- If the incorrect data pattern is detected, determine if the error is due to the write access or the read access. See Determining If a Data Error is Due to the Write or Read, page 770.
- To analyze the window size in ps, see Determining Window Size in ps, page 773. As a general rule of thumb, the window size for a healthy system should be ≥ 30% of the expected UI size.
- Compare read leveling window (read margin size) results from the simple pattern calibration versus the complex pattern calibration. The windows should all shrink but the reduction in window size should shrink relatively across the data byte lanes.

- Use the Memory IP Debug GUI to quickly compare simple versus complex window sizes.

Figure 38-91 is a screen capture from 2015.1 and might vary from the current version.



Calibration/Margins

Calibration Pattern: ● Simple ○ Complex

Calibration Read Margin: ● Rising edge clocked ○ Falling edge clocked

Calibration/Margins

Calibration Pattern: ○ Simple ● Complex

Calibration Read Margin: ● Rising edge clocked ○ Falling edge clocked

Figure 38-91: **Comparing Simple and Complex Read Calibration Windows**

Hardware Measurements

1. Probe the write commands and read commands at the memory:
    - Write = **`cs_n`**= 1; **`ras_n`** = 0; **`cas_n`** = 1; **`we_n`** = 1; **`act_n`** = 1 (DDR4 only)
    - Read = **`cs_n`** = 1; **`ras_n`** = 0; **`cas_n`** = 1; **`we_n`** = 0; **`act_n`** = 1 (DDR4 only)
2. Probe a data pin to check for data being returned from the DRAM.
3. Probe the V_{REF} level at the DRAM (for DDR3).
4. Probe the DM pin which should be deasserted during the write burst (or tied off on the board with an appropriate value resistor).
5. Probe the read burst after the write and check if the expected data pattern is being returned.
6. Check for floating address pins if the expected data is not returned.
7. Check for any stuck-at level issues on DQ pins whose signal level does not change. If at all possible probe at the receiver to check termination and signal integrity.
8. Check the DBG port signals and the full read data and comparison result to check the data in general interconnect. The calibration algorithm has RTL logic to issue the

commands and check the data. Check if the **`dbg_rd_valid`** aligns with the data pattern or is off. Set up a trigger when the error gets asserted to capture signals in the hardware debugger for analysis.

9. Re-check results from previous calibration stages. Compare passing byte lanes against failing byte lanes for previous stages of calibration. If a failure occurs during complex pattern calibration, check the values found during simple pattern calibration for example.
10. All of the data comparison for complex read calibration occur in the general interconnect, so it can be useful to pull in the debug data in the hardware debugger and take a look at what the data looks like coming back as taps are adjusted, see Figure 38-92 and Figure 38-93. Screenshots shown are from simulation, with a small loop count set for the data pattern. Look at **`dbg_rd_data`**, **`dbg_rd_valid`**, and **`dbg_cplx_err_log`**.
11. Using the Vivado Hardware Manager and while running the Memory IP Example Design with **Debug Signals** enabled, set the Read Complex calibration trigger to **`cal_r*_status[28]`** = R (rising edge). To view each byte, add an additional trigger on **`dbg_cmp_byte`** and set to the byte of interest. The following simulation example shows how the debug signals should behave during Read Complex Calibration.

    Figure 38-92 shows the start of the complex calibration data pattern with an emphasis on the **`dbg_cplx_config`** bus shown. The "read start" bit is Bit[0] and the number of loops is set based on Bits[15:9], hence Figure 38-92 shows the start of complex read pattern and the loop count set to 1 (for simulation only). The **`dbg_cplx_status`** goes to 1 to indicate the pattern is in progress. See Table 38-2, page 594 for the list of all debug signals.

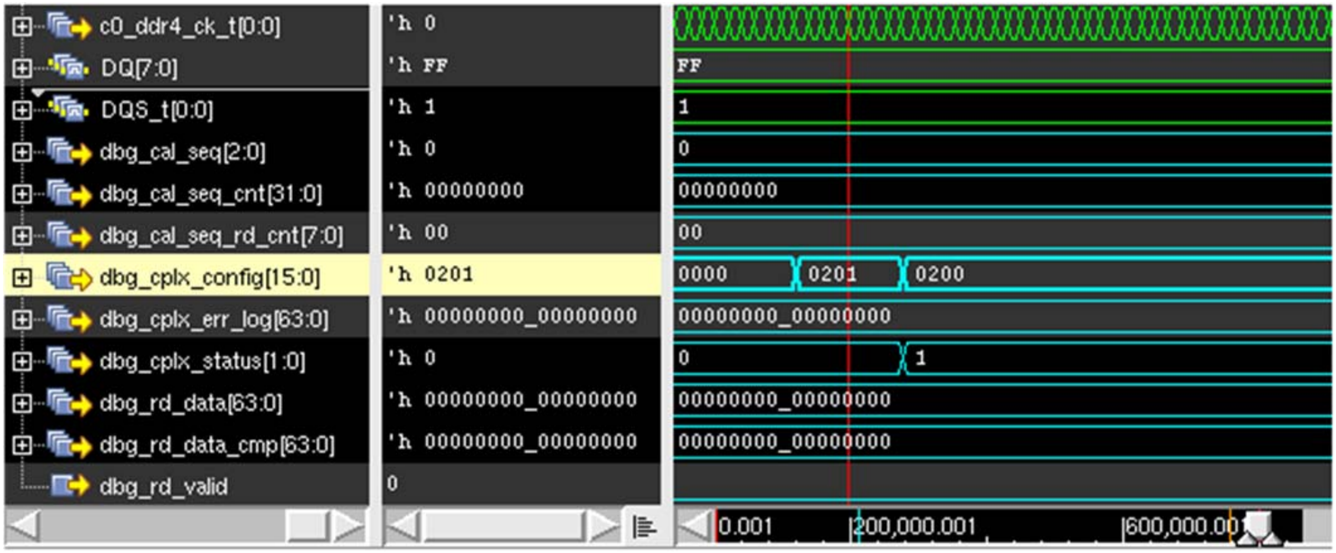


Figure 38-92: **RTL Debug Signals during Read Complex (Start)**

Figure 38-93: **RTL Debug Signals during Read Complex (Writes and Reads)**

12. Analyze the debug signal **`dbg_cplx_err_log`**. This signal shows comparison mismatches on a per-bit basis. When a bit error occurs, signifying an edge of the window has been found, typically a single bit error is shown on **`dbg_cplx_err_log`**. Meaning, all bits of this bus are 0 except for the single bit that had a comparison mismatch which is set to 1. When an unexpected data error occurs during complex read calibration, for example a byte shift, the entire bus would be 1. This is not the expected bit mismatch found in window detection but points to a true read versus write issue. Now, the read data should be compared with the expected (compare) data and the error debugged to determine if it is a read or write issue. Use **`dbg_rd_data`** and **`dbg_rd_dat_cmp`** to compare the received data to the expected data.
13. For more information, see Debugging Data Errors, page 758.
14. After failure during this stage of calibration, the design goes into a continuous loop of read commands to allow board probing.

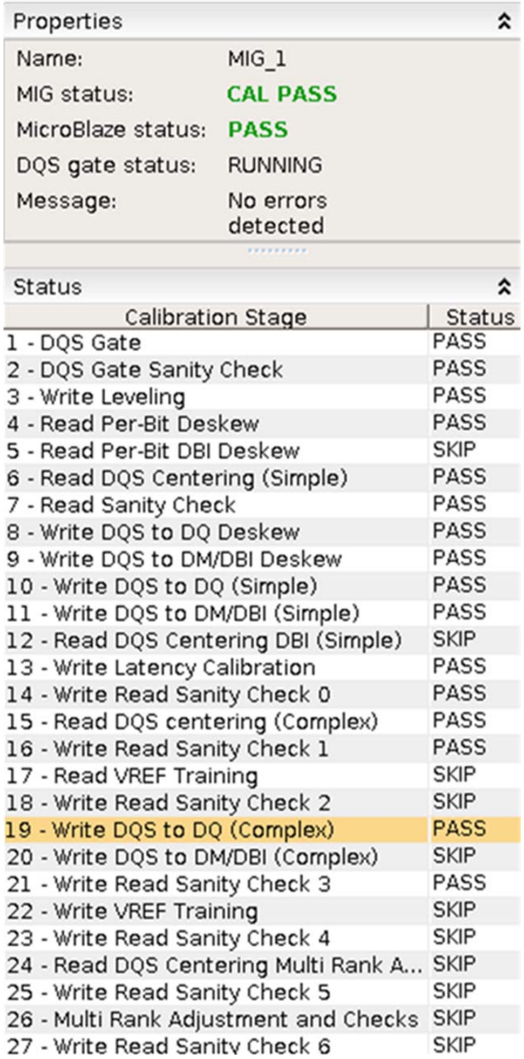
Debugging Write Complex Pattern Calibration Failures

Calibration Overview

The final stage of Write DQS-to-DQ centering that is completed before normal operation is repeating the steps performed during Write DQS-to-DQ centering but with a difficult/complex pattern. The purpose of using a complex pattern is to stress the system for SI effects such as ISI and noise while calculating the write DQS center and write DQ positions. This ensures the write center position can reliably capture data with margin in a true system.

Debug

To determine the status of Write Complex Pattern Calibration, click the **Write DQS to DQ (Complex)** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	PASS
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank A...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Checks	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-94: **Memory IP XSDB Debug GUI Example – Write DQS to DQ (Complex)**

The status of Write Complex Pattern Calibration can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-33. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-33: **DDR_CAL_ERROR Decode for Read Leveling and Write DQS Centering Calibration**

Write DQS to DQ DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Byte	N/A	No valid data found	Check if the design meets timing. Check the margin found for the simple pattern for the given nibble/byte. Check if the ODELAY values used for each bit are reasonable to others in the byte. Check the dbg_cplx_config, dbg_cplx_status, dbg_cplx_err_log, dbg_rd_data, and dbg_expected_data during this stage of calibration. Check the default VREF value being used is correct for the configuration.
0xF	Byte	N/A	Timeout error waiting for read data to return	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-34 shows the signals and values adjusted or used during the Write Complex Pattern stage of calibration. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-34: **Signals of Interest for Complex Pattern Calibration**

Signal	Usage	Signal Description
WRITE_COMPLEX_DQS_TO_DQ_PRE_ADJUST_MARGIN_LEFT_BYTE*	One per Byte	Left side of the write DQS-to-DQ window measured during calibration before adjustments made.
WRITE_ COMPLEX _DQS_TO_DQ_PRE_ADJUST_MARGIN_RIGHT _BYTE*	One per Byte	Right side of the write DQS-to-DQ window measured during calibration before adjustments made.
WRITE_ COMPLEX _DQS_TO_DQ _MARGIN_LEFT_BYTE*	One per Byte	Left side of the write DQS-to-DQ window.
WRITE_ COMPLEX _DQS_TO_DQ _MARGIN_RIGHT _BYTE*	One per Byte	Right side of the write DQS-to-DQ window.
WRITE_ COMPLEX _DQS_TO_DQ_DQS_ODELAY_BYTE*	One per Byte	Final DQS ODELAY value after Write DQS-to-DQ (Complex).
WRITE_ COMPLEX _DQS_TO_DQ_DQ_ODELAY_BYTE*_BIT*	One per Bit	Final DQ ODELAY value after Write DQS-to-DQ (Complex).
WRITE_DQS_ODELAY_FINAL_BYTE*_BIT*	One per Byte	Final DQS ODELAY value.
WRITE_DQ_ODELAY_FINAL_BYTE*_BIT*	One per Bit	Final DQ ODELAY value.

Expected Results

- Look at the individual WRITE_COMPLEX_DQS_TO_DQ_DQS_ODELAY and WRITE_COMPLEX_DQS_TO_DQ_DQ_ODELAY tap settings for each nibble. The taps should only vary by 0 to 20 taps. To calculate the write window, see Determining Window Size in ps, page 773.
- Determine if any bytes completed successfully. The write calibration algorithm sequentially steps through each DQS byte group detecting the capture edges.
- If the incorrect data pattern is detected, determine if the error is due to the write access or the read access. See Determining If a Data Error is Due to the Write or Read, page 770.
- Both edges need to be found. This is possible at all frequencies because the algorithm uses 90° of ODELAY taps to find the edges.
- To analyze the window size in ps, see Determining Window Size in ps, page 773. As a general rule of thumb, the window size for a healthy system should be ≥ 30% of the expected UI size.

Using the Vivado Hardware Manager and while running the Memory IP Example Design with the **Debug Signals** enabled, set the trigger (`cal_r*_status[36]` = R for Rising Edge).

The following simulation example shows how the debug signals should behave during successful Write DQS-to-DQ.

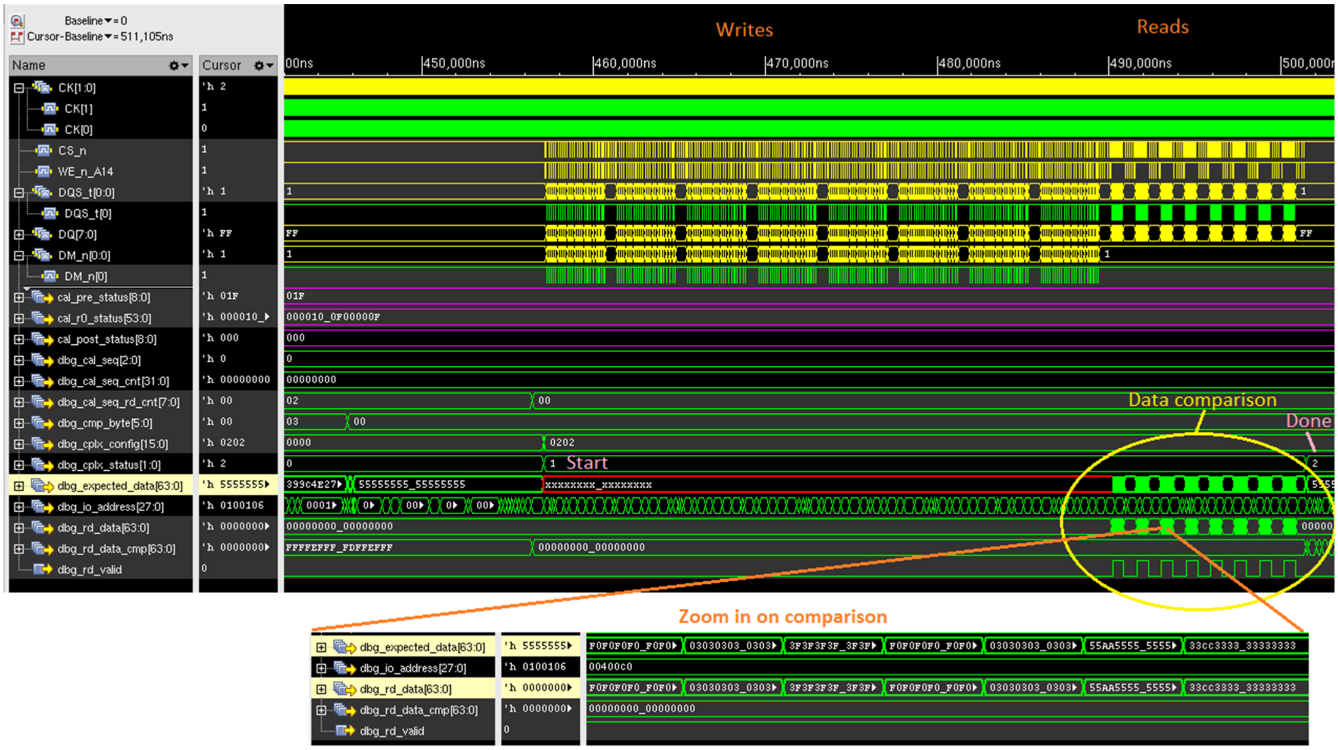


Figure 38-95: **Expected Behavior during Write Complex Pattern Calibration**

Hardware Measurements

1. If the write complex pattern fails, use high quality probes and scope the DQS-to-DQ phase relationship at the memory during a write. Trigger at the start (`cal_r*_status[36]` = R for Rising Edge) and again at the end (`cal_r*_status[37]` = R for Rising Edge) of Write Complex DQS Centering to view the starting and ending alignments. The alignment should be approximately 90°.
2. If the DQS-to-DQ alignment is correct, observe the `we_n`-to-DQS relationship to see if it meets CWL again using `cal_r*_status[25]` = R for Rising Edge as a trigger.
3. For all stages of write/read leveling, probe the write commands and read commands at the memory:
    - Write = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 1; `act_n` = 1 (DDR4 only)
    - Read = `cs_n` = 1; `ras_n` = 0; `cas_n` = 1; `we_n` = 0; `act_n` = 1 (DDR4 only)

Multi-Rank Adjustments and Checks (Multi-Rank Designs Only)

Calibration Overview

For multi-rank designs, previously calibrated positions must be validated and adjusted across each rank within the system. The previously calibrated areas that need further adjustment for multi-rank systems are Read Level, DQS Preamble, and Write Latency. The adjustments are described in the following sections.

Common Read Leveling Settings

Each DQS has a single IDELAY/PQTR/NQTR value that is used across ranks. During Read Leveling Calibration, each rank is allowed to calibrate independently to find the ideal IDELAY/PQTR/NQTR tap positions for each DQS to each separate rank. During the multi-rank checks, the minimum and maximum value found for each DQS IDELAY/PQTR/NQTR positions are checked, the range is computed, and the center point is used as the final setting. For example, if a DQS has a PQTR that sees values of rank0 = 50, rank1 = 50, rank2 = 50, and rank3 = 75, the final value would be 62. This is done to ensure a value can work well across all ranks rather than averaging the values and giving preference to values that happen more frequently.

DQS Gate Adjustment

During DQS gate calibration for multi-rank systems, each rank is allowed to calibrate independently. After all ranks have been calibrated, an adjustment is required before normal operation to ensure fast rank-to-rank switching.

Across all ranks within a byte, the read latency and general interconnect delay (`clb2phy_rd_en`) must match. During the DQS Gate Adjustment stage of calibration, the coarse taps found during DQS Preamble Detection for each rank are adjusted such that a common read latency and `clb2phy_rd_en` can be used. Additionally, the coarse taps have to be within four taps within the same byte lane across all ranks. Table 38-35 shows the DQS Gate adjustment examples.

Table 38-35: **DQS gate Adjustment Examples**

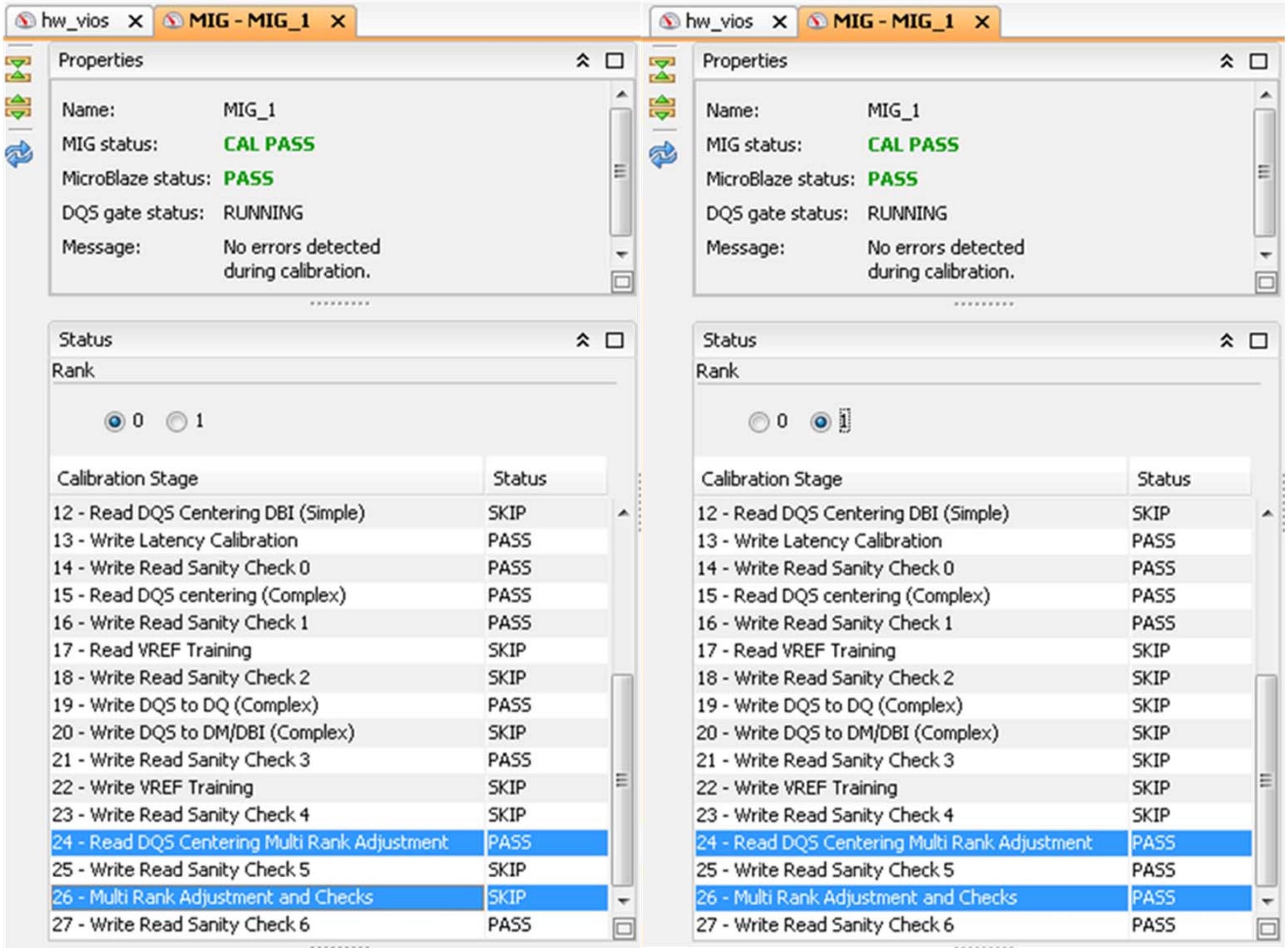
Example	Setting	Calibration		After Multi-Rank Adjustment		
		Rank 0	Rank 1	Rank 0	Rank 1	Result
#1	Read latency	14	15	14	14	Pass
	Coarse taps	8	6	8	10	
#2	Read latency	22	21	21	21	Pass
	Coarse taps	6	9	10	9	
#3	Read latency	10	15	N/A	N/A	Error
	Coarse taps	9	9	N/A	N/A	
#4	Read latency	10	11	10	10	Error
	Coarse taps	6	9	6	13	

Write Latency Check between Ranks

The write leveling and write latency values are calibrated separately for each rank. After all ranks have been calibrated, a check is made to ensure certain XIPHY requirements are met on the write path. The difference in write latency between the ranks is allowed to be 180° (or two XIPHY coarse taps). This is checked during this stage.

Debug

To determine the status of Multi-Rank Adjustments and Checks, click the **Read DQS Centering Multi Rank Adjustment** or **Multi Rank Adjustments and Checks** stage under the **Status** window and view the results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected during calibration.

Rank

● 0 ○ 1

Calibration Stage	Status
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank Adjustment	PASS
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Checks	SKIP
27 - Write Read Sanity Check 6	PASS

Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected during calibration.

Rank

○ 0 ● 1

Calibration Stage	Status
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	SKIP
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	SKIP
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank Adjustment	PASS
25 - Write Read Sanity Check 5	PASS
26 - Multi Rank Adjustment and Checks	PASS
27 - Write Read Sanity Check 6	PASS

Figure 38-96: **Memory IP XSDB Debug GUI Example – Read DQS Centering Multi-Rank Adjustment and Multi-Rank Adjustment and Checks**

The status of Read Level Multi-Rank Adjustment can also be determined by decoding the DDR_CAL_ERROR_0 and DDR_CAL_ERROR_1 results according to Table 38-36. Execute the Tcl commands noted in the XSDB Debug section to generate the XSDB output containing the signal results.

Table 38-36: **DDR_CAL_ERROR Decode for Multi-Rank Adjustments and Checks**

Multi-Rank Adjustments & Checks DDR_CAL_ERROR_CODE	DDR_CAL_ERROR_1	DDR_CAL_ERROR_0	Description	Recommended Debug Steps
0x1	Byte	RIU Nibble	Could not find common setting across ranks for general interconnect read latency setting for given byte. Variance between ranks could not be compensated with coarse taps.	Check PCB Trace lengths against what is allowed. Check the calibration results for DQS_GATE_COARSE, and DQS_GATE_READ_LATENCY for the byte that failed.
0x2	Byte	RIU Nibble	Read skew between ranks for a given byte larger than 360°.	Check PCB Trace lengths against what is allowed. Check the calibration results for DQS_GATE_COARSE and DQS_GATE_READ_LATENCY for the byte that failed.
0x3	Byte	RIU Nibble	Write skew between ranks for a given byte larger than 180°.	Check PCB Trace lengths against what is allowed. Check the calibration results for WRLVL_COARSE_STABLE0 and WRITE_LATENCY_CALIBRATION_COARSE for the byte that failed.

Table 38-37 shows the signals and values adjusted or used during Read Level Multi-Rank Adjustment and Multi-Rank DQS Gate. The values can be analyzed in both successful and failing calibrations to determine the resultant values and the consistency in results across resets. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-37: **Signals of Interest for Multi-Rank Adjustments and Checks**

Signal	Usage	Signal Description
RDLVL_PQTR_FINAL_NIBBLE*	One per nibble	Final Read leveling PQTR tap position from the XIPHY.
RDLVL_NQTR_FINAL_NIBBLE*	One per nibble	Final Read leveling NQTR tap position from the XIPHY.
RDLVL_IDELAY_FINAL_BYTE*_BIT*	One per Bit	Final IDELAY tap position from the XIPHY.
RDLVL_IDELAY_DBI_FINAL_BYTE*	One per Byte	Reserved

Table 38-37: **Signals of Interest for Multi-Rank Adjustments and Checks *(Cont'd)***

Signal	Usage	Signal Description
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE*	One per Byte	Final common general interconnect read latency setting used for a given byte.
MULTI_RANK_DQS_GATE_COARSE_RANK*_BYTE*	One per Rank per Byte	Final RL_DLY_COARSE tap value used for a given byte (might differ from calibrated value).

Expected Results

If no adjustments are required then the MULTI_RANK_* signals can be blank as shown, the field is only populated when a change is made to the values.

```
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE0     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE1     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE2     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE3     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE4     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE5     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE6     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE7     000
MULTI_RANK_DQS_GATE_COARSE_RANK0_BYTE8     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE0     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE1     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE2     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE3     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE4     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE5     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE6     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE7     000
MULTI_RANK_DQS_GATE_COARSE_RANK1_BYTE8     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE0     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE1     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE2     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE3     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE4     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE5     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE6     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE7     000
MULTI_RANK_DQS_GATE_READ_LATENCY_BYTE8     000
```

The Read level Multi-Rank Adjustment changes the values of the "FINAL" fields for the read path. The margin for each individual rank is given in the table and chart but the final value is stored here.

```
RDLVL_IDELAY_FINAL_BYTE0_BIT0            04d
RDLVL_IDELAY_FINAL_BYTE0_BIT1            052
RDLVL_IDELAY_FINAL_BYTE0_BIT2            055
RDLVL_IDELAY_FINAL_BYTE0_BIT3            051
RDLVL_IDELAY_FINAL_BYTE0_BIT4            04f
RDLVL_IDELAY_FINAL_BYTE0_BIT5            04e
RDLVL_IDELAY_FINAL_BYTE0_BIT6            050
RDLVL_IDELAY_FINAL_BYTE0_BIT7            04b
RDLVL_IDELAY_FINAL_BYTE1_BIT0            04d
```

```
RDLVL_IDELAY_FINAL_BYTE1_BIT1            050
RDLVL_IDELAY_FINAL_BYTE1_BIT2            04f
RDLVL_IDELAY_FINAL_BYTE1_BIT3            04c
RDLVL_IDELAY_FINAL_BYTE1_BIT4            050
RDLVL_IDELAY_FINAL_BYTE1_BIT5            051
RDLVL_IDELAY_FINAL_BYTE1_BIT6            052
RDLVL_IDELAY_FINAL_BYTE1_BIT7            04e
RDLVL_IDELAY_FINAL_BYTE2_BIT0            04f
RDLVL_IDELAY_FINAL_BYTE2_BIT1            052
RDLVL_IDELAY_FINAL_BYTE2_BIT2            053
RDLVL_IDELAY_FINAL_BYTE2_BIT3            049
RDLVL_IDELAY_FINAL_BYTE2_BIT4            04f
RDLVL_IDELAY_FINAL_BYTE2_BIT5            052
RDLVL_IDELAY_FINAL_BYTE2_BIT6            04e
RDLVL_IDELAY_FINAL_BYTE2_BIT7            04c
RDLVL_IDELAY_FINAL_BYTE3_BIT0            051
RDLVL_IDELAY_FINAL_BYTE3_BIT1            056
RDLVL_IDELAY_FINAL_BYTE3_BIT2            04c
RDLVL_IDELAY_FINAL_BYTE3_BIT3            04b
RDLVL_IDELAY_FINAL_BYTE3_BIT4            04f
RDLVL_IDELAY_FINAL_BYTE3_BIT5            050
RDLVL_IDELAY_FINAL_BYTE3_BIT6            055
RDLVL_IDELAY_FINAL_BYTE3_BIT7            050
RDLVL_IDELAY_FINAL_BYTE4_BIT0            04b
RDLVL_IDELAY_FINAL_BYTE4_BIT1            04c
RDLVL_IDELAY_FINAL_BYTE4_BIT2            046
RDLVL_IDELAY_FINAL_BYTE4_BIT3            048
RDLVL_IDELAY_FINAL_BYTE4_BIT4            054
RDLVL_IDELAY_FINAL_BYTE4_BIT5            055
RDLVL_IDELAY_FINAL_BYTE4_BIT6            054
RDLVL_IDELAY_FINAL_BYTE4_BIT7            04f
RDLVL_IDELAY_FINAL_BYTE5_BIT0            044
RDLVL_IDELAY_FINAL_BYTE5_BIT1            049
RDLVL_IDELAY_FINAL_BYTE5_BIT2            04a
RDLVL_IDELAY_FINAL_BYTE5_BIT3            045
RDLVL_IDELAY_FINAL_BYTE5_BIT4            04d
RDLVL_IDELAY_FINAL_BYTE5_BIT5            052
RDLVL_IDELAY_FINAL_BYTE5_BIT6            04e
RDLVL_IDELAY_FINAL_BYTE5_BIT7            04b
RDLVL_IDELAY_FINAL_BYTE6_BIT0            03d
RDLVL_IDELAY_FINAL_BYTE6_BIT1            03e
RDLVL_IDELAY_FINAL_BYTE6_BIT2            039
RDLVL_IDELAY_FINAL_BYTE6_BIT3            03c
RDLVL_IDELAY_FINAL_BYTE6_BIT4            053
RDLVL_IDELAY_FINAL_BYTE6_BIT5            052
RDLVL_IDELAY_FINAL_BYTE6_BIT6            04d
RDLVL_IDELAY_FINAL_BYTE6_BIT7            04c
RDLVL_IDELAY_FINAL_BYTE7_BIT0            040
RDLVL_IDELAY_FINAL_BYTE7_BIT1            03f
RDLVL_IDELAY_FINAL_BYTE7_BIT2            040
RDLVL_IDELAY_FINAL_BYTE7_BIT3            03c
RDLVL_IDELAY_FINAL_BYTE7_BIT4            046
RDLVL_IDELAY_FINAL_BYTE7_BIT5            047
RDLVL_IDELAY_FINAL_BYTE7_BIT6            048
RDLVL_IDELAY_FINAL_BYTE7_BIT7            045
RDLVL_IDELAY_FINAL_BYTE8_BIT0            04b
RDLVL_IDELAY_FINAL_BYTE8_BIT1            050
RDLVL_IDELAY_FINAL_BYTE8_BIT2            051
RDLVL_IDELAY_FINAL_BYTE8_BIT3            04e
```

```
    RDLVL_IDELAY_FINAL_BYTE8_BIT4            04a
    RDLVL_IDELAY_FINAL_BYTE8_BIT5            04c
    RDLVL_IDELAY_FINAL_BYTE8_BIT6            04d
    RDLVL_IDELAY_FINAL_BYTE8_BIT7            04a
    RDLVL_NQTR_CENTER_FINAL_NIBBLE0          064
    RDLVL_NQTR_CENTER_FINAL_NIBBLE1          06b
    RDLVL_NQTR_CENTER_FINAL_NIBBLE2          066
    RDLVL_NQTR_CENTER_FINAL_NIBBLE3          06b
    RDLVL_NQTR_CENTER_FINAL_NIBBLE4          062
    RDLVL_NQTR_CENTER_FINAL_NIBBLE5          06c
    RDLVL_NQTR_CENTER_FINAL_NIBBLE6          067
    RDLVL_NQTR_CENTER_FINAL_NIBBLE7          069
    RDLVL_NQTR_CENTER_FINAL_NIBBLE8          065
    RDLVL_NQTR_CENTER_FINAL_NIBBLE9          05d
    RDLVL_NQTR_CENTER_FINAL_NIBBLE10         05d
    RDLVL_NQTR_CENTER_FINAL_NIBBLE11         05c
    RDLVL_NQTR_CENTER_FINAL_NIBBLE12         061
    RDLVL_NQTR_CENTER_FINAL_NIBBLE13         051
    RDLVL_NQTR_CENTER_FINAL_NIBBLE14         054
    RDLVL_NQTR_CENTER_FINAL_NIBBLE15         04f
    RDLVL_NQTR_CENTER_FINAL_NIBBLE16         063
    RDLVL_NQTR_CENTER_FINAL_NIBBLE17         06d
    RDLVL_PQTR_CENTER_FINAL_NIBBLE0          064
    RDLVL_PQTR_CENTER_FINAL_NIBBLE1          06a
    RDLVL_PQTR_CENTER_FINAL_NIBBLE2          066
    RDLVL_PQTR_CENTER_FINAL_NIBBLE3          068
    RDLVL_PQTR_CENTER_FINAL_NIBBLE4          061
    RDLVL_PQTR_CENTER_FINAL_NIBBLE5          06d
    RDLVL_PQTR_CENTER_FINAL_NIBBLE6          067
    RDLVL_PQTR_CENTER_FINAL_NIBBLE7          06c
    RDLVL_PQTR_CENTER_FINAL_NIBBLE8          069
    RDLVL_PQTR_CENTER_FINAL_NIBBLE9          060
    RDLVL_PQTR_CENTER_FINAL_NIBBLE10         061
    RDLVL_PQTR_CENTER_FINAL_NIBBLE11         061
    RDLVL_PQTR_CENTER_FINAL_NIBBLE12         066
    RDLVL_PQTR_CENTER_FINAL_NIBBLE13         056
    RDLVL_PQTR_CENTER_FINAL_NIBBLE14         058
    RDLVL_PQTR_CENTER_FINAL_NIBBLE15         058
    RDLVL_PQTR_CENTER_FINAL_NIBBLE16         061
    RDLVL_PQTR_CENTER_FINAL_NIBBLE17         06b
```

Hardware Measurements

No hardware measurements are available because no command or data are sent to the memory during this stage. Algorithm only goes through previously collected data.

Debugging Write and Read Sanity Checks

Calibration Overview

Throughout calibration, read and write/read sanity checks are performed to ensure that as each stage of calibration completes, proper adjustments and alignments are made allowing writes and reads to be completed successfully. Sanity checks are performed as follows:

- Check for DQS Gate after DQS Preamble Detection
- Read Sanity Check after Read DQS Centering (Simple)
- Write/Read Sanity Check after Write Latency Calibration
- Write/Read Sanity Check after Read DQS Centering (Complex)
- Write/Read Sanity Check after Read V_{REF} Training (Reserved)
- Write/Read Sanity Check after Write DQS-to-DQ Centering (Complex)
- Write/Read Sanity Check after Write V_{REF} Training (Reserved)
- Write/Read Sanity check after Read DQS Centering Multi-Rank Adjustment (For ranks other than the first one)
- Write/Read Sanity check after DQS Gate Multi-Rank Adjustment when there is more than one rank

Each sanity check performed uses a different data pattern to expand the number of patterns checked during calibration.

Table 38-38: **Sanity Check Data Patterns**

Sanity Check Stage	Data Pattern (as stored) – 32 bits, 4 bits concatenated together each as {f3,r3,f2,r2,f1,r1,f0,r0}.	Data on DQ bus (nibble) as would be seen in a simulation or a scope – r0 f0 r1 f1 r2 f2 r3 f3
DQS Gate Sanity Check	0xAAAAAAAA	0F0F_0F0F
Read Sanity Check	0xAAAAAAAA	0F0F_0F0F
Write/Read Sanity Check 0	0x399C4E27	937E_C924
Write/Read Sanity Check 1	0x3587D5DC	E4F1_B837
Write/Read Sanity Check 2	0x919CD315	B254_F02E
Write/Read Sanity Check 3	0x4E2562E5	5AD8_07B1
Write/Read Sanity Check 4	0x2C6C9AAA	03CF_2D43

Table 38-38: **Sanity Check Data Patterns *(Cont'd)***

Sanity Check Stage	Data Pattern (as stored) – 32 bits, 4 bits concatenated together each as {f3,r3,f2,r2,f1,r1,f0,r0}.	Data on DQ bus (nibble) as would be seen in a simulation or a scope – r0 f0 r1 f1 r2 f2 r3 f3
Write/Read Sanity Check 5	Rank = 0 (No sanity check) Rank = 1 (0x75294A2F) Rank = 2 (0x75294A30) Rank = 3 (0x75294A31)	Rank = 0 (No sanity check) Rank = 1 (D397_8DA0) Rank = 2 (C286_9DA0) Rank = 3 (D286_9DA0)
Write/Read Sanity Check 6(1)	Rank = 0 (0xE5742542) Rank = 1 (0xE5742543) Rank = 2 (0xE5752442) Rank = 3 (0xE5752443)	Rank = 0 (A1E0_4ED8) Rank = 1 (B1E0_4ED8) Rank = 2 (C1E0_4ED8) Rank = 3 (D1E0_4ED8)

Notes:

1. For 3DS systems, the Write/Read Sanity Check 6 is repeated for each stack in a given rank. For each stack, the data pattern is adjusted by adding 0x100 to the data pattern (as stored) for the base rank pattern. For example, for rank 0, stack 0 would be data pattern 0xE5742542 as shown in the table, but rank 0, stack 1 the pattern would be 0xE5742642 (and show up as 83E0_4ED8 on the DQ bus).

Data swizzling (bit reordering) is completed within the UltraScale PHY. Therefore, the data visible on BUS_DATA_BURST and a scope in hardware is ordered differently compared to what would be seen in ChipScope. Figures are examples of how the data is converted for the sanity check data patterns.

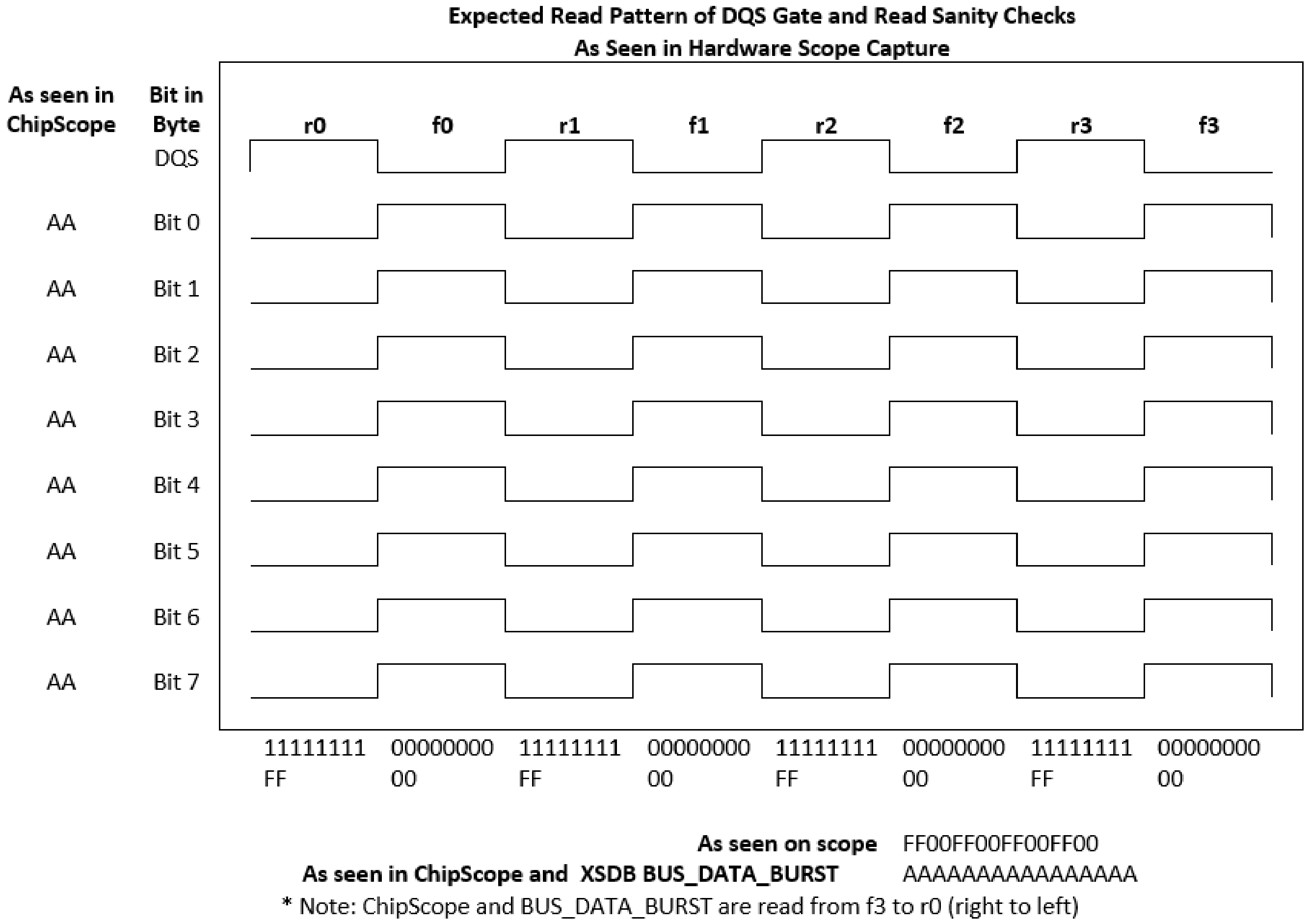
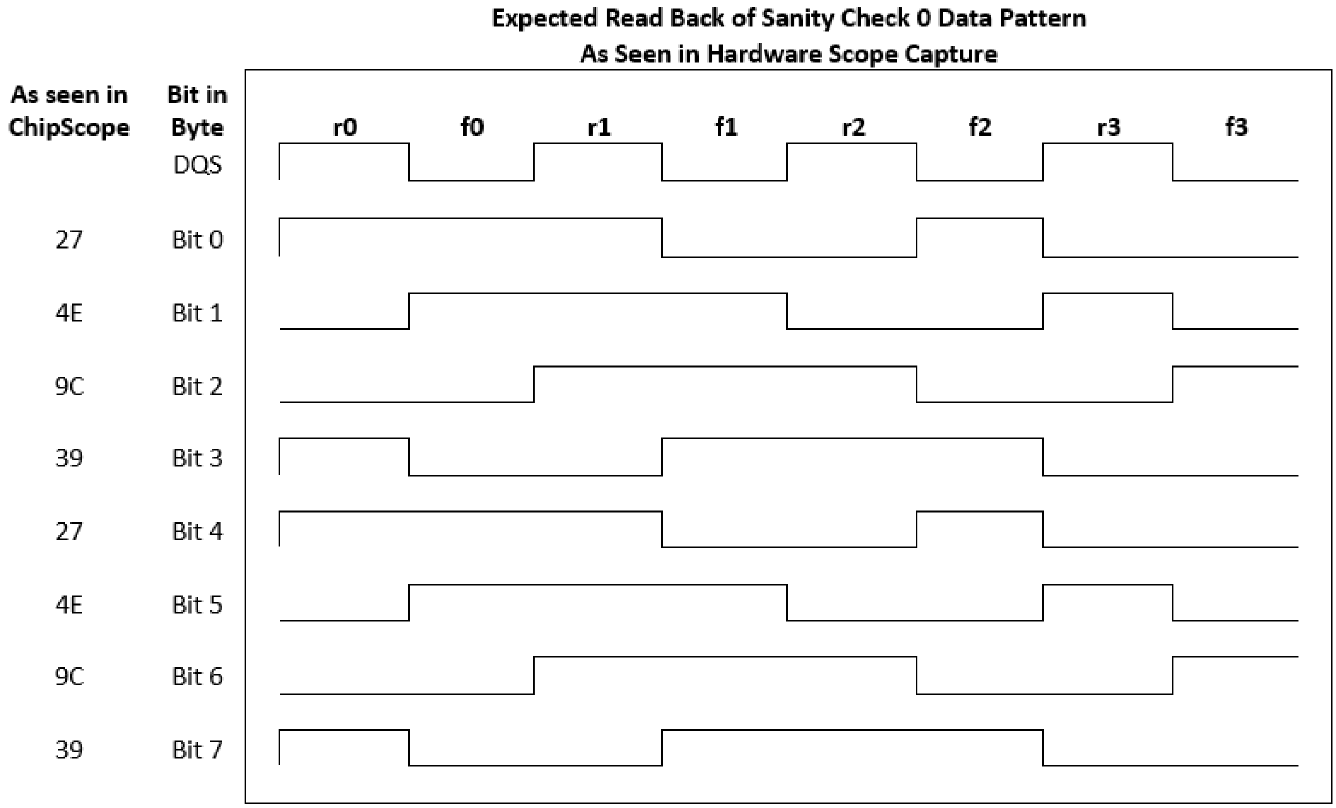


Figure 38-97: **Expected Read Pattern of DQS Gate and Read Sanity Checks**



As seen on scope 993377EECC992244
As seen in ChipScope and XSDB BUS_DATA_BURST 399C4E27399C4E27
* Note: ChipScope and BUS_DATA_BURST are read from f3 to r0 (right to left)

Figure 38-98: **Expected Read Back of Sanity Check 0 Data Pattern**

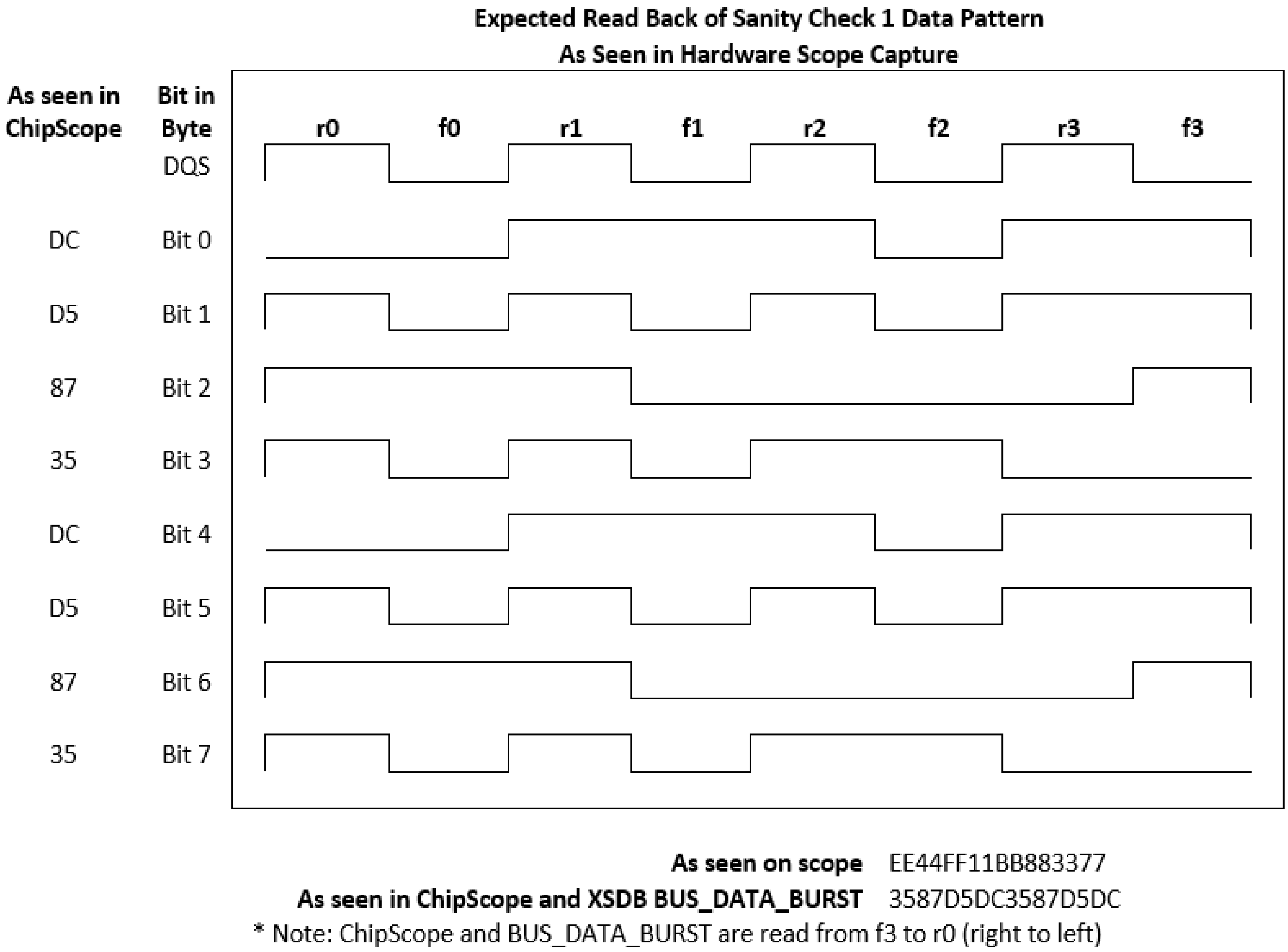
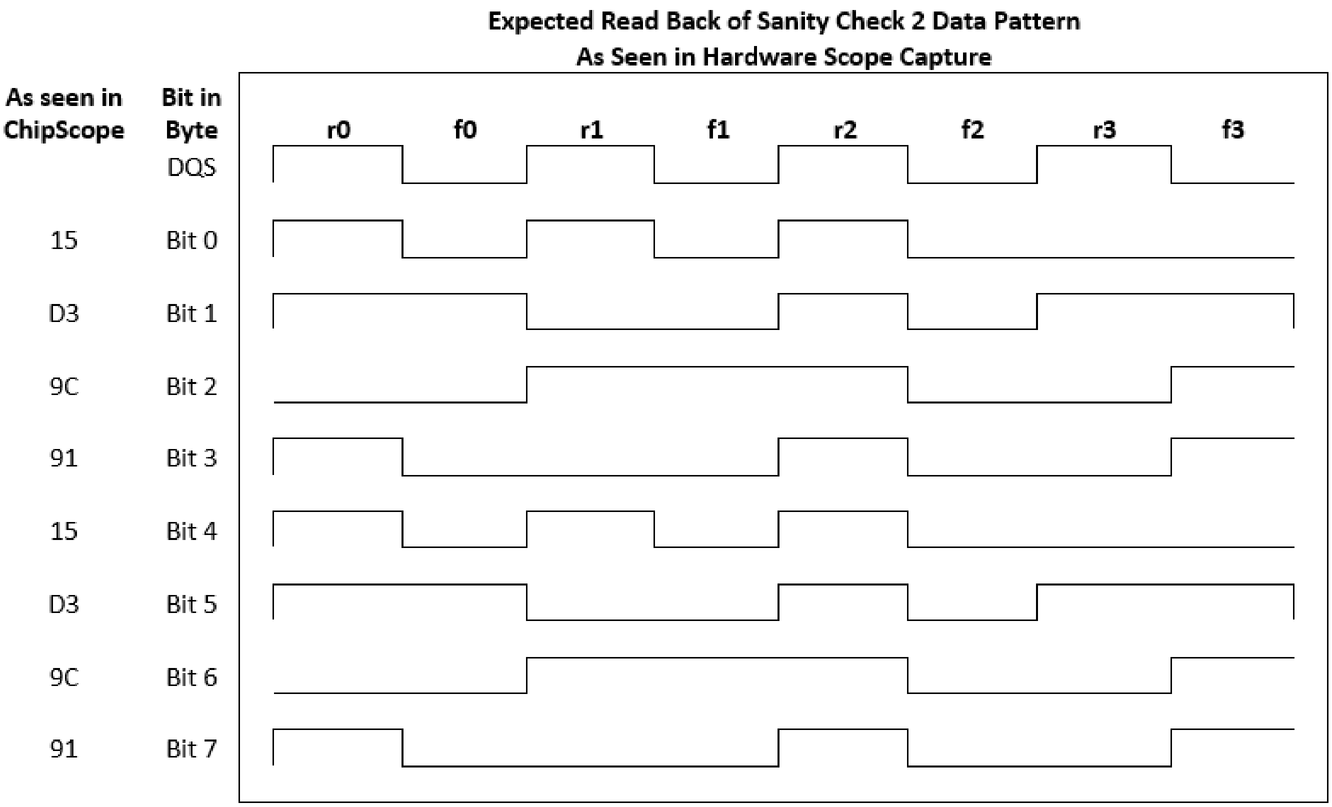


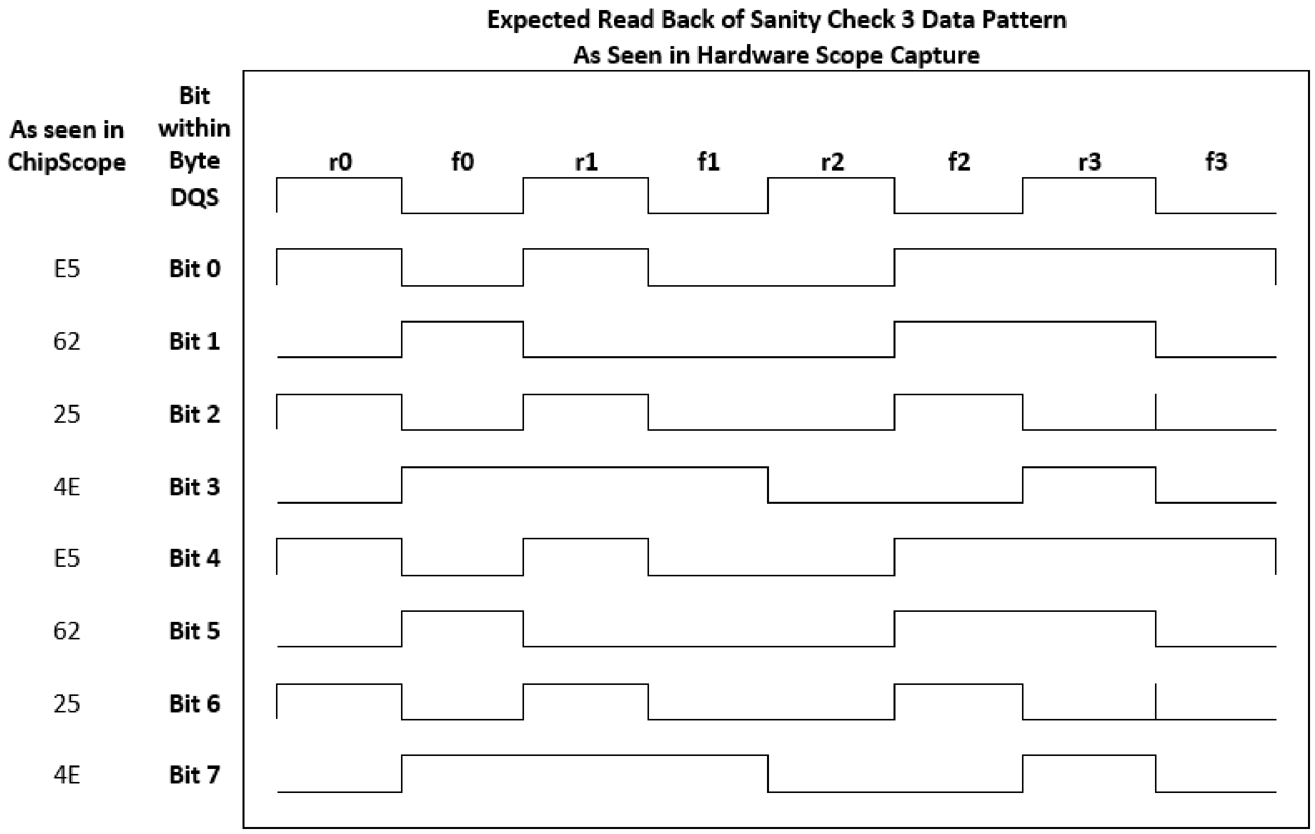
Figure 38-99: **Expected Read Back of Sanity Check 1 Data Pattern**



As seen on scope	BB225544FF0022EE
As seen in ChipScope and XSDB BUS_DATA_BURST	919CD315919CD315

* Note: ChipScope and BUS_DATA_BURST are read from f3 to r0 (right to left)

Figure 38-100: **Expected Read Back of Sanity Check 2 Data Pattern**

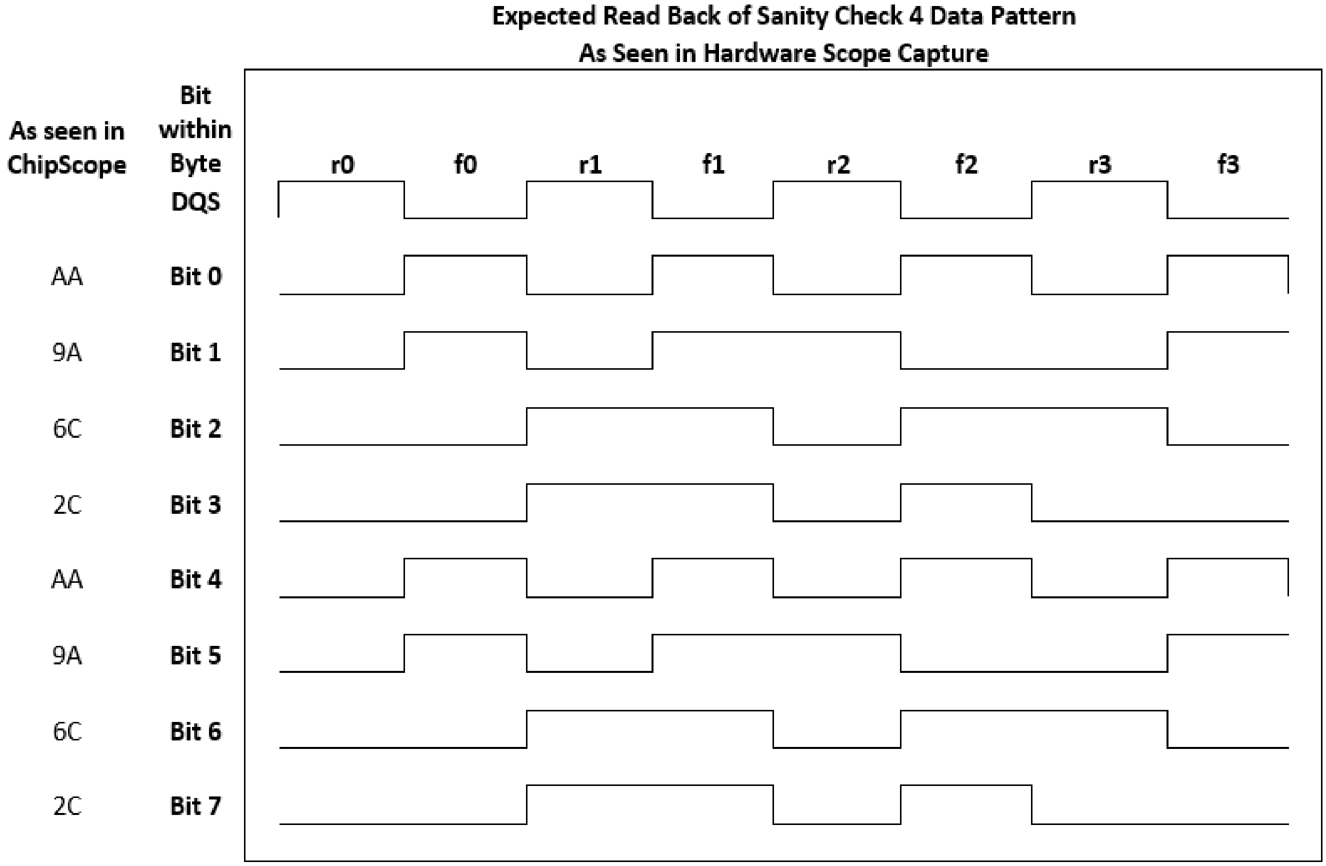


As seen on scope 55AADD880077BB11

As seen in ChipScope and XSDB BUS_DATA_BURST 4E2562E54E2562E5

*Note: ChipScope and BUS_DATA_BURST are read from f3 to r0 (right to left)

Figure 38-101: **Expected Read Back of Sanity Check 3 Data Pattern**



As seen on scope 0033CCFF22DD4433

As seen in ChipScope and XSDB BUS_DATA_BURST 2C6C9AAA2C6C9AAA

*Note: ChipScope and BUS_DATA_BURST are read from f3 to r0 (right to left)

Figure 38-102: **Expected Read Back of Sanity Check 4 Data Pattern**

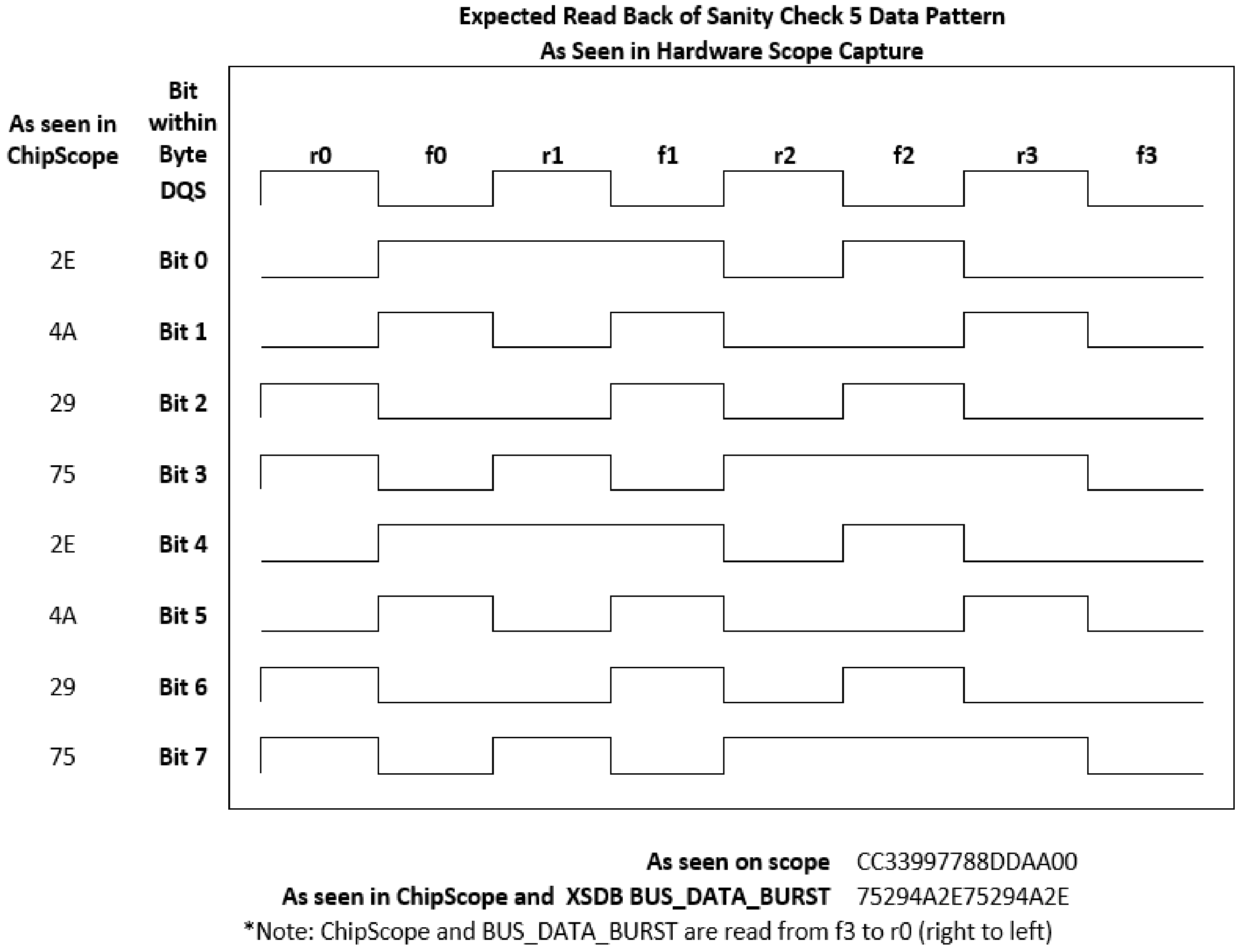
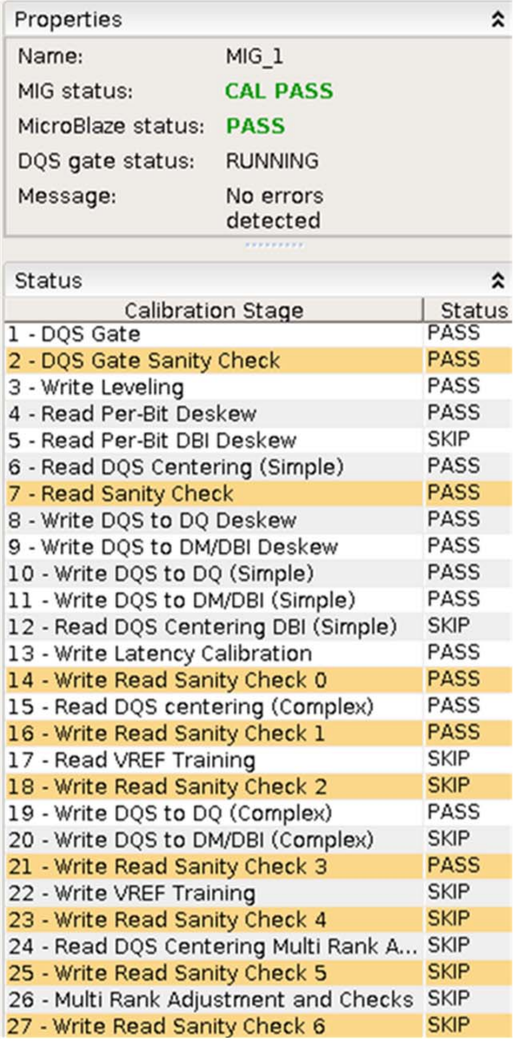


Figure 38-103: **Expected Read Back of Sanity Check 5 Data Pattern**

Debug

To determine the status of each sanity check, analyze the **Memory IP Status** window to view the completion of each check. Click the sanity check of interest to view the specific results within the **Memory IP Properties** window. The message displayed in **Memory IP Properties** identifies how the stage failed or notes if it passed successfully.



Name: MIG_1
MIG status: CAL PASS
MicroBlaze status: PASS
DQS gate status: RUNNING
Message: No errors detected

Calibration Stage	Status
1 - DQS Gate	PASS
2 - DQS Gate Sanity Check	PASS
3 - Write Leveling	PASS
4 - Read Per-Bit Deskew	PASS
5 - Read Per-Bit DBI Deskew	SKIP
6 - Read DQS Centering (Simple)	PASS
7 - Read Sanity Check	PASS
8 - Write DQS to DQ Deskew	PASS
9 - Write DQS to DM/DBI Deskew	PASS
10 - Write DQS to DQ (Simple)	PASS
11 - Write DQS to DM/DBI (Simple)	PASS
12 - Read DQS Centering DBI (Simple)	SKIP
13 - Write Latency Calibration	PASS
14 - Write Read Sanity Check 0	PASS
15 - Read DQS centering (Complex)	PASS
16 - Write Read Sanity Check 1	PASS
17 - Read VREF Training	SKIP
18 - Write Read Sanity Check 2	SKIP
19 - Write DQS to DQ (Complex)	PASS
20 - Write DQS to DM/DBI (Complex)	SKIP
21 - Write Read Sanity Check 3	PASS
22 - Write VREF Training	SKIP
23 - Write Read Sanity Check 4	SKIP
24 - Read DQS Centering Multi Rank A...	SKIP
25 - Write Read Sanity Check 5	SKIP
26 - Multi Rank Adjustment and Checks	SKIP
27 - Write Read Sanity Check 6	SKIP

Figure 38-104: **Memory IP XSDB Debug GUI Example – Write and Read Sanity Checks**

The status of each sanity check can also be determined by decoding DDR_CAL_STATUS_RANK*_* as shown in Table 38-4. Only two possible errors can occur during this stage of calibration, as shown Table 38-39. The data pattern used changes depending on which sanity check stage is run.

Table 38-39: **DDR_CAL_ERROR Decode for Sanity Checks**

Check for DQS Gate DDR_CAL_ ERROR_ CODE	DDR_CAL_ ERROR_1	DDR_CAL_ ERROR_0	Description	Recommended Debug Steps
0x1	nibble	0	Writes to error reg for each nibble that has compare failure. Register for XSDB holds the last nibble that had an error. For 2014.3+ the data and expected data for up to three nibble errors is written to the data burst registers of XSDB. The fourth data burst location holds the array of all the nibble failures to indicate which of all nibbles showed an error.	Check the BUS_DATA_BURST XSDB Fields to determine which nibbles/bits failed. Check margin found during previous stages of calibration for the given byte that failed.
0xF	N/A	N/A	Timeout error waiting for read data to return.	Check the dbg_cal_seq_rd_cnt and dbg_cal_seq_cnt.

Table 38-40 shows the signals and values used to help determine which bytes the error occurred on, as well as to provide some data returned for comparison with the expected data pattern. These values can be found within the **Memory IP Core Properties** within the Hardware Manager or by executing the Tcl commands noted in the XSDB Debug section.

Table 38-40: **Signals of Interest for Sanity Check**

Signal	Usage	Signal Description
BUS_DATA_BURST (2014.3+)		Stored sample data and list of which nibbles had an error. Determine which bytes or bits had a failure. BUS_DATA_BURST_0 (BIT0-BIT3 addresses) stores the received data for the first nibble in which an error occurred. BUS_DATA_BURST_0 (BIT4-BIT7 addresses) stores the expected data pattern. BUS_DATA_BURST_1 (BIT0-BIT3 addresses) stores the received data for the second nibble in which an error occurred. BUS_DATA_BURST_1 (BIT4-BIT7 addresses) stores the expected data pattern. BUS_DATA_BURST_2 (BIT0-BIT3 addresses) stores the received data for the third nibble in which an error occurred. BUS_DATA_BURST_2 (BIT4-BIT7 addresses) stores the expected data pattern. BUS_DATA_BURST_3 stores an array which indicates which nibbles saw an error (indicated by a 1 in that bit location). Each address locations stores an array for up to eight nibbles. For example, BUS_DATA_BURST_3_BIT_0 = 0x3 would indicate nibble 0 and nibble 1 saw an error. BUS_DATA_BURST_3_BIT_0 = 0x14 would indicate nibble 2 and nibble 4 saw an error. BUS_DATA_BURST_3_BIT_1 = 0x5 would indicate nibble 8 and nibble 10 saw an error.

Hardware Measurements

The calibration status bits (`cal_r*_status`) can be used as hardware triggers to capture the write (when applicable) and read command and data on the scope. The entire interface is checked with one write followed by one read command, so any bytes or bits that need to be probed can be checked on a scope. The `cal_r*_status` triggers are as follows for the independent sanity checks:

- Check for DQS Gate after DQS Preamble Detection:
    - Start –> `cal_r*_status[2]` = R for Rising Edge
    - End –> `cal_r*_status[3]` = R for Rising Edge
- Read Sanity Check:
    - Start –> `cal_r*_status[12]` = R for Rising Edge
    - End –> `cal_r*_status[13]` = R for Rising Edge
- Write/Read Sanity Check 0:
    - Start –> `cal_r*_status[26]` = R for Rising Edge
    - End –> `cal_r*_status[27]` = R for Rising Edge
- Write/Read Sanity Check 1:
    - Start –> `cal_r*_status[30]` = R for Rising Edge
    - End –> `cal_r*_status[31]` = R for Rising Edge
- Write/Read Sanity Check 2:
    - Start –> `cal_r*_status[34]` = R for Rising Edge
    - End –> `cal_r*_status[35]` = R for Rising Edge
- Write/Read Sanity Check 3:
    - Start –> `cal_r*_status[40]` = R for Rising Edge
    - End –> `cal_r*_status[41]` = R for Rising Edge
- Write/Read Sanity Check 4:
    - Start –> `cal_r*_status[44]` = R for Rising Edge
    - End –> `cal_r*_status[45]` = R for Rising Edge
- Write/Read Sanity Check 5 (for more than 1 rank):
    - Start –> `cal_r*_status[48]` = R for Rising Edge
    - End –> `cal_r*_status[49]` = R for Rising Edge
- Write/Read Sanity Check 6 (all ranks):
    - Start –> `cal_r*_status[52]` = R for Rising Edge

- End –> `cal_r*_status[53]` = R for Rising Edge

VT Tracking

Tracking Overview

Calibration occurs one time at start-up, at a set voltage and temperature to ensure relation capture of the data, but during normal operation the voltage and temperature can change or drift if conditions change. Voltage and temperature (VT) change can adjust the relationship between DQS and DQ used for read capture and change the time in which the DQS/DQ arrive at the FPGA as part of a read.

DQS Gate Tracking

The arrival of the DQS at the FPGA as part of a read is calibrated at start-up, but as VT changes the time in which the DQS arrives can change. DQS gate tracking monitors the arrival of the DQS with a signal from the XIPHY and makes small adjustments as required if the DQS arrives earlier or later a sampling clock in the XIPHY. This adjustment is recorded as shown in Table 38-41.

Debug

Table 38-41: **Signals of Interest for DQS Tracking**

Signal	Usage	Signal Description
DQS_TRACK_COARSE_BYTE*	One per Byte	Last recorded value for DQS gate coarse setting.
DQS_TRACK_FINE_BYTE*	One per Byte	Last recorded value for DQS gate fine setting.
DQS_TRACK_COARSE_MAX_BYTE*	One per Byte	Maximum coarse tap recorded during DQS gate tracking.
DQS_TRACK_FINE_MAX_BYTE*	One per Byte	Maximum fine tap recorded during DQS gate tracking.
DQS_TRACK_COARSE_MIN_BYTE*	One per Byte	Minimum coarse tap recorded during DQS gate tracking.
DQS_TRACK_FINE_MIN_BYTE*	One per Byte	Minimum fine tap recorded during DQS gate tracking.
BISC_ALIGN_PQTR	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_ALIGN_NQTR	One per nibble	Initial 0° offset value provided by BISC at power-up.

Table 38-41: **Signals of Interest for DQS Tracking *(Cont'd)***

Signal	Usage	Signal Description
BISC_PQTR	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_PQTR – BISC_ALIGN_PQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_PQTR – BISC_ALIGN_PQTR). Useful to know how many fine taps make up a coarse tap to compute amount of DQS gate drift (Average of the P & N values used for computation).
BISC_NQTR	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_NQTR – BISC_ALIGN_NQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_NQTR – BISC_ALIGN_NQTR). Useful to know how many fine taps make up a coarse tap to compute amount of DQS gate drift. (Average of the P & N values used for computation).

Expected Results

```
DQS_TRACK_COARSE_MAX_RANK0_BYTE0                   string true true 007
DQS_TRACK_COARSE_MAX_RANK0_BYTE1                   string true true 006
DQS_TRACK_COARSE_MAX_RANK0_BYTE2                   string true true 007
DQS_TRACK_COARSE_MAX_RANK0_BYTE3                   string true true 007
DQS_TRACK_COARSE_MAX_RANK0_BYTE4                   string true true 008
DQS_TRACK_COARSE_MAX_RANK0_BYTE5                   string true true 008
DQS_TRACK_COARSE_MAX_RANK0_BYTE6                   string true true 008
DQS_TRACK_COARSE_MAX_RANK0_BYTE7                   string true true 008
DQS_TRACK_COARSE_MAX_RANK0_BYTE8                   string true true 008
DQS_TRACK_COARSE_MIN_RANK0_BYTE0                   string true true 006
DQS_TRACK_COARSE_MIN_RANK0_BYTE1                   string true true 006
DQS_TRACK_COARSE_MIN_RANK0_BYTE2                   string true true 007
DQS_TRACK_COARSE_MIN_RANK0_BYTE3                   string true true 007
DQS_TRACK_COARSE_MIN_RANK0_BYTE4                   string true true 008
DQS_TRACK_COARSE_MIN_RANK0_BYTE5                   string true true 008
DQS_TRACK_COARSE_MIN_RANK0_BYTE6                   string true true 008
DQS_TRACK_COARSE_MIN_RANK0_BYTE7                   string true true 007
DQS_TRACK_COARSE_MIN_RANK0_BYTE8                   string true true 007
DQS_TRACK_COARSE_RANK0_BYTE0                       string true true 007
DQS_TRACK_COARSE_RANK0_BYTE1                       string true true 006
DQS_TRACK_COARSE_RANK0_BYTE2                       string true true 007
DQS_TRACK_COARSE_RANK0_BYTE3                       string true true 007
DQS_TRACK_COARSE_RANK0_BYTE4                       string true true 008
DQS_TRACK_COARSE_RANK0_BYTE5                       string true true 008
DQS_TRACK_COARSE_RANK0_BYTE6                       string true true 008
DQS_TRACK_COARSE_RANK0_BYTE7                       string true true 008
DQS_TRACK_COARSE_RANK0_BYTE8                       string true true 007
DQS_TRACK_FINE_MAX_RANK0_BYTE0                     string true true 02d
DQS_TRACK_FINE_MAX_RANK0_BYTE1                     string true true 02d
DQS_TRACK_FINE_MAX_RANK0_BYTE2                     string true true 027
DQS_TRACK_FINE_MAX_RANK0_BYTE3                     string true true 01a
DQS_TRACK_FINE_MAX_RANK0_BYTE4                     string true true 021
DQS_TRACK_FINE_MAX_RANK0_BYTE5                     string true true 020
DQS_TRACK_FINE_MAX_RANK0_BYTE6                     string true true 012
DQS_TRACK_FINE_MAX_RANK0_BYTE7                     string true true 02e
DQS_TRACK_FINE_MAX_RANK0_BYTE8                     string true true 02e
```

```
DQS_TRACK_FINE_MIN_RANK0_BYTE0                   string true true 000
DQS_TRACK_FINE_MIN_RANK0_BYTE1                   string true true 023
DQS_TRACK_FINE_MIN_RANK0_BYTE2                   string true true 01d
DQS_TRACK_FINE_MIN_RANK0_BYTE3                   string true true 00f
DQS_TRACK_FINE_MIN_RANK0_BYTE4                   string true true 019
DQS_TRACK_FINE_MIN_RANK0_BYTE5                   string true true 018
DQS_TRACK_FINE_MIN_RANK0_BYTE6                   string true true 00a
DQS_TRACK_FINE_MIN_RANK0_BYTE7                   string true true 000
DQS_TRACK_FINE_MIN_RANK0_BYTE8                   string true true 000
DQS_TRACK_FINE_RANK0_BYTE0                       string true true 001
DQS_TRACK_FINE_RANK0_BYTE1                       string true true 028
DQS_TRACK_FINE_RANK0_BYTE2                       string true true 022
DQS_TRACK_FINE_RANK0_BYTE3                       string true true 014
DQS_TRACK_FINE_RANK0_BYTE4                       string true true 01d
DQS_TRACK_FINE_RANK0_BYTE5                       string true true 01c
DQS_TRACK_FINE_RANK0_BYTE6                       string true true 00e
DQS_TRACK_FINE_RANK0_BYTE7                       string true true 001
DQS_TRACK_FINE_RANK0_BYTE8                       string true true 02b
BISC_ALIGN_NQTR_NIBBLE0                          string true true 000
BISC_ALIGN_NQTR_NIBBLE1                          string true true 000
BISC_ALIGN_NQTR_NIBBLE2                          string true true 000
BISC_ALIGN_NQTR_NIBBLE3                          string true true 000
BISC_ALIGN_NQTR_NIBBLE4                          string true true 000
BISC_ALIGN_NQTR_NIBBLE5                          string true true 000
BISC_ALIGN_NQTR_NIBBLE6                          string true true 000
BISC_ALIGN_NQTR_NIBBLE7                          string true true 000
BISC_ALIGN_PQTR_NIBBLE0                          string true true 007
BISC_ALIGN_PQTR_NIBBLE1                          string true true 004
BISC_ALIGN_PQTR_NIBBLE2                          string true true 006
BISC_ALIGN_PQTR_NIBBLE3                          string true true 005
BISC_ALIGN_PQTR_NIBBLE4                          string true true 005
BISC_ALIGN_PQTR_NIBBLE5                          string true true 004
BISC_ALIGN_PQTR_NIBBLE6                          string true true 004
BISC_ALIGN_PQTR_NIBBLE7                          string true true 004
BISC_NQTR_NIBBLE0                                string true true 036
BISC_NQTR_NIBBLE1                                string true true 033
BISC_NQTR_NIBBLE2                                string true true 037
BISC_NQTR_NIBBLE3                                string true true 035
BISC_NQTR_NIBBLE4                                string true true 037
BISC_NQTR_NIBBLE5                                string true true 036
BISC_NQTR_NIBBLE6                                string true true 036
BISC_NQTR_NIBBLE7                                string true true 036
BISC_PQTR_NIBBLE0                                string true true 038
BISC_PQTR_NIBBLE1                                string true true 036
BISC_PQTR_NIBBLE2                                string true true 038
BISC_PQTR_NIBBLE3                                string true true 035
BISC_PQTR_NIBBLE4                                string true true 037
BISC_PQTR_NIBBLE5                                string true true 037
BISC_PQTR_NIBBLE6                                string true true 035
BISC_PQTR_NIBBLE7                                string true true 036
```

BISC VT Tracking

The change in the relative delay through the FPGA for the DQS and DQ is monitored in the XIPHY and adjustments are made to the delays to account for the change in resolution of the delay elements. The change in the delays are recorded in the XSDB.

Debug

Table 38-42: **Signals of Interest for DQS Tracking**

Signal	Usage	Signal Description
VT_TRACK_PQTR_NIBBLE*	One per nibble	PQTR position last read during BISC VT Tracking.
VT_TRACK_NQTR_NIBBLE*	One per nibble	NQTR position last read during BISC VT Tracking.
VT_TRACK_PQTR_MAX_NIBBLE*	One per nibble	Maximum PQTR value found during BISC VT Tracking.
VT_TRACK_NQTR_MAX_NIBBLE*	One per nibble	Maximum NQTR value found during BISC VT Tracking.
VT_TRACK_PQTR_MIN_NIBBLE*	One per nibble	Minimum PQTR value found during BISC VT Tracking.
VT_TRACK_NQTR_MIN_NIBBLE*	One per nibble	Minimum NQTR value found during BISC VT Tracking.
RDLVL_PQTR_CENTER_FINAL_NIBBLE*	One per nibble	Final PQTR position found during calibration.
RDLVL_NQTR_CENTER_FINAL_NIBBLE*	One per nibble	Final NQTR position found during calibration.
BISC_ALIGN_PQTR	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_ALIGN_NQTR	One per nibble	Initial 0° offset value provided by BISC at power-up.
BISC_PQTR	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_PQTR – BISC_ALIGN_PQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_PQTR – BISC_ALIGN_PQTR). Useful to know how many fine taps make up a coarse tap to compute amount of DQS gate drift (Average of the P & N values used for computation).
BISC_NQTR	One per nibble	Initial 90° offset value provided by BISC at power-up. Compute 90° value in taps by taking (BISC_NQTR – BISC_ALIGN_NQTR). To estimate tap resolution take (¼ of the memory clock period)/ (BISC_NQTR – BISC_ALIGN_NQTR). Useful to know how many fine taps make up a coarse tap to compute amount of DQS gate drift. (Average of the P & N values used for computation).

Expected Results

To see where the PQTR and NQTR positions have moved since calibration, compare the VT_TRACK_PQTR_NIBBLE* and VT_TRACK_NQTR_NIBBLE* XSDB values to the final calibrated positions which are stored in RDLVL_PQTR_CENTER_FINAL_NIBBLE* and RDLVL_NQTR_CENTER_FINAL_NIBBLE*.

To see how much movement the PQTR and NQTR taps exhibit over environmental changes, monitor:

```
VT_TRACK_PQTR_NIBBLE*
VT_TRACK_NQTR_NIBBLE*
VT_TRACK_PQTR_MAX_NIBBLE*
VT_TRACK_NQTR_MAX_NIBBLE*
VT_TRACK_PQTR_MIN_NIBBLE*
VT_TRACK_NQTR_MIN_NIBBLE*
```

Calibration Times

Calibration time depends on a number of factors, such as:

- General Interconnect Clock Frequency
- Number of DDR Ranks
- Memory Width
- Board Trace Lengths

Table 38-43 gives an example of calibration times for a DDR memory interface.

Table 38-43: **DDR Calibration Times**

Memory Interface	Component Type	Width	Memory Interface Speed (MT/s)	Calibration Time (s)
DDR3	x8 components	72-bit	2,133	0.83
			1,866	1.10
			1,600	0.75
			1,333	1.04
			1,066	1.56
			800	0.84
	Dual-Rank SO-DIMM x8	72-bit	1,600	0.85
			1,333	1.15
	Dual-Rank RDIMM x8	72-bit	1,600	0.94
			1,333	1.28

Table 38-43: **DDR Calibration Times *(Cont'd)***

Memory Interface	Component Type	Width	Memory Interface Speed (MT/s)	Calibration Time (s)
DDR4	X8 components	72-bit	2,400	0.61
			2,133	0.79
			1,866	1.05
			1,600	0.73
			1,333	1.06
	UDIMM x8	72-bit	2,133	0.83
			1,600	0.91
			1,333	1.15
	RDIMM x8	72-bit	2,133	0.93
			1,600	0.79
			1,333	1.22
	Dual-Rank RDIMM x4	72-bit	1,600	0.88
			1,333	1.18

Debugging Data Errors

General Checks

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

Replicating Data Errors Using the Advanced Traffic Generator

When data errors are seen during normal operation, the Memory IP Advanced Traffic Generator (ATG) should be used to replicate the error. The ATG is a verified solution that can be configured to send a wide range of data, address, and command patterns. It additionally presents debug status information for general memory traffic debug post calibration. The ATG stores the write data and compares it to the read data. This allows comparison of expected and actual data when errors occur. This is a critical step in data error debug as this section will go through in detail.

ATG Setup

The default ATG configuration exercises predefined traffic instructions which are included in the `mem_v1_2_tg_instr_bram.sv` module. To move away from the default configuration and use the ATG for data error debug, use the provided VIO and ILA cores that are generated with the example design. For more information, see the Using VIO to Control ATG in Chapter 36, Traffic Generator.

Table 38-44: **General Control**

General Control	I/O	Width	Description
vio_tg_start	I	1	Enable traffic generator to proceed from "START" state to "LOAD" state after calibration completes. If you do not plan to program instruction table NOR PRBS data seed, tie this signal to 1'b1. If you plan to program instruction table OR PRBS data seed, set this bit to 0 during reset. After reset deassertion and done with instruction/seed programming, set this bit to 1 to start traffic generator.
vio_tg_rst	I	1	Reset traffic generator (synchronous reset, level sensitive) If there is outstanding traffic in memory pipeline, assert this signal long enough until all outstanding transactions have completed.
vio_tg_restart	I	1	Restart traffic generator after traffic generation is complete, paused, or stopped with error (level sensitive) If there is outstanding traffic in memory pipeline, assert this signal long enough until all outstanding transactions have completed.
vio_tg_pause	I	1	Pause traffic generator (level sensitive)
vio_tg_err_chk_en	I	1	If enabled, stop upon first error detected. Read test is performed to determine whether "READ" or "WRITE" error occurred. If not enabled, continue traffic without stop.
vio_tg_err_clear	I	1	Clear all errors excluding sticky error bit (positive edge sensitive) Only use this signal when vio_tg_status_state is either TG_INSTR_ERRDONE or TG_INSTR_PAUSE
vio_tg_err_clear_all	I	1	Clear all errors including sticky error bit (positive edge sensitive) Only use this signal when vio_tg_status_state is either TG_INSTR_ERRDONE or TG_INSTR_PAUSE
vio_tg_err_continue	I	1	Continue traffic after error(s) at TG_INSTR_ERRDONE state (positive edge sensitive)

Table 38-45: **Instruction Programming**

Instruction Programming	I/O	Width	Description
vio_tg_direct_instr_en	I	1	0: Traffic Table Mode – Traffic Generator uses traffic patterns programmed in 32-entry traffic table 1: Direct Instruction Mode – Traffic Generator uses current traffic pattern presented at VIO interface
vio_tg_instr_program_en	I	1	Enable instruction table programming (level sensitive)
vio_tg_instr_num	I	5	Instruction number to be programmed

Table 38-45: **Instruction Programming *(Cont'd)***

Instruction Programming	I/O	Width	Description
vio_tg_instr_addr_mode	I	4	Address mode to be programmed. LINEAR = 0; (with user defined start address) PRBS = 1; (PRBS supported range from 8 to 34 based on address width) WALKING1 = 2; WALKING0 = 3; 4:15 Reserved ***Note:*** QDR-IV only supports Linear address with start address equals to 0.
vio_tg_instr_data_mode	I	4	Data mode to be programmed. LINEAR = 0; PRBS = 1; (PRBS supported 8,10,23) WALKING1 = 2; WALKING0 = 3; HAMMER1 = 4; HAMMER0 = 5; Block RAM = 6; CAL_CPLX = 7; (Must be programmed along with victim mode CAL_CPLX) 8:15: Reserved
vio_tg_instr_rw_mode	I	4	0: Read Only (No data check) 1: Write Only (No data check) 2: Write / Read (Read performs after Write and data value is checked against expected write data. For QDR II+ SRAM, one port is used for write and another port is used for read) 3: Write Once and Read forever (Data check on Read data) 4-15: Reserved

vio_tg_instr_rw_submode	I	2	Read/Write sub-mode to be programmed.
			This is a sub-mode option when vio_tg_instr_rw_mode is set to "WRITE_READ" mode.
			This mode is only valid for DDR3/DDR4 and RLDRAM 3. For QDR II+ SRAM and QDR-IV, this mode should be set to 0
			WRITE_READ = 0; // Send all Write commands follow by Read commands defined in the instruction
			WRITE_READ_SIMULTANEOUSLY = 1; // Send Write and Read commands pseudo-randomly. Note that Write is always ahead of Read.

Table 38-45: **Instruction Programming *(Cont'd)***

Instruction Programming	I/O	Width	Description

vio_tg_instr_victim_mode	I	3	Victim mode to be programmed.
			One victim bit could be programmed using global register vio_tg_victim_bit.
			The rest of the bits on signal bus are considered to be aggressors.
			The following program options define aggressor behavior:
			NO_VICTIM = 0;
			HELD1 = 1; // All aggressor signals held at 1
			HELD0 = 2; // All aggressor signals held at 0
			NONINV_AGGR = 3; // All aggressor signals are same as victim
			INV_AGGR = 4; // All aggressor signals are inversion of victim
			DELAYED_AGGR = 5; // All aggressor signals are delayed version of victim (num of cycle of delay is programmed at vio_tg_victim_aggr_delay)
			DELAYED_VICTIM = 6; // Victim signal is delayed version of all aggressors
			CAL_CPLX = 7; Complex Calibration pattern (Must be programed along with Data Mode CAL_CPLX)

vio_tg_instr_victim_aggr_delay	I	5	Define aggressor/victim pattern to be N-delay cycle of victim/aggressor. It is used when victim mode "DELAY_AGGR" or "DELAY VICTIM" mode is used in traffic pattern.

vio_tg_instr_victim_select	I	3	Victim bit behavior programmed.
			VICTIM_EXTERNAL = 0; // Use Victim bit provided in vio_tg_glb_victim_bit
			VICTIM_ROTATE4 = 1; // Victim bit rotates from Bit[0] to Bit[3] for every Nibble
			VICTIM_ROTATE8 = 2; // Victim bit rotates from Bit[0] to Bit[7] for every byte
			VICTIM_ROTATE_ALL = 3; // Victim bit rotates through all bits

vio_tg_instr_num_of_iter	I	32	Number of Read/Write commands to issue (number of issue must be > 0 for each instruction programmed)
vio_tg_instr_m_nops_btw_n_burst_m	I	10	M: Number of NOP cycles in between Read/Write commands at User interface at general interconnect clock. N: Number of Read/Write commands before NOP cycle insertion at User interface at general interconnect clock.
vio_tg_instr_m_nops_btw_n_burst_n	I	32	M: Number of NOP cycles in between Read/Write commands at User interface at general interconnect clock. N: Number of Read/Write commands before NOP cycle insertion at User interface at general interconnect clock.
vio_tg_instr_nxt_instr	I	6	Next instruction to run. To end traffic, next instruction should point at EXIT instruction. 6'b000000-6'b011111 – valid instruction 6'b1????? – EXIT instruction

Table 38-46: **Status Registers**

Status Registers	I/O	Width	Description
vio_tg_status_state	O	4	Traffic Generator state machine state
vio_tg_status_err_bit_valid	O	1	Intermediate error detected Used as trigger to detect read error
vio_tg_status_err_bit	O	APP_DATA_WIDTH	Intermediate error bit mismatch Bitwise mismatch pattern
vio_tg_status_err_addr	O	APP_ADDR_WIDTH	Intermediate error address Address location of failed read
vio_tg_status_exp_bit_valid	O	1	Expected read data valid
vio_tg_status_exp_bit	O	APP_DATA_WIDTH	Expected read data
vio_tg_status_read_bit_valid	O	1	Memory read data valid
vio_tg_status_read_bit	O	APP_DATA_WIDTH	Memory read data
vio_tg_status_first_err_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, first_err_bit_valid is set to 1 when first mismatch error is encountered. This register is not overwritten until vio_tg_err_clear, vio_tg_err_continue, vio_tg_restart is triggered.
vio_tg_status_first_err_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_err_bit_valid is set to 1, error mismatch bit pattern is stored in this register.
vio_tg_status_first_err_addr	O	APP_ADDR_WIDTH	If vio_tg_status_first_err_bit_valid is set to 1, error address is stored in this register.
vio_tg_status_first_exp_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, this represents expected read data valid when first mismatch error is encountered.
vio_tg_status_first_exp_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_exp_bit_valid is set to 1, expected read data is stored in this register.
vio_tg_status_first_read_bit_valid	O	1	If vio_tg_err_chk_en is set to 1, this represents read data valid when first mismatch error is encountered.
vio_tg_status_first_read_bit	O	APP_DATA_WIDTH	If vio_tg_status_first_read_bit_valid is set to 1, read data from memory is stored in this register.
vio_tg_status_err_bit_sticky_valid	O	1	Accumulated error mismatch valid over time. This register is reset by vio_tg_err_clear, vio_tg_err_continue, vio_tg_restart.
vio_tg_status_err_bit_sticky	O	APP_DATA_WIDTH	If vio_tg_status_err_bit_sticky_valid is set to 1, this represents accumulated error bit
vio_tg_status_err_type_valid	O	1	If vio_tg_err_chk_en is set to 1, read test is performed upon the first mismatch error. Read test returns error type of either "READ" or "WRITE" error. This register stores valid status of read test error type.

Table 38-46: **Status Registers *(Cont'd)***

Status Registers	I/O	Width	Description
vio_tg_status_err_type	O	1	If vio_tg_status_err_type_valid is set to 1, this represents error type result from read test. 0 = Write Error 1 = Read Error
vio_tg_status_done	O	1	All traffic programmed completes. ***Note:*** If infinite loop is programmed, vio_tg_status_done does not assert.
vio_tg_status_wr_done	O	1	This signal pulses after a WRITE-READ mode instruction completes
vio_tg_status_watch_dog_hang	O	1	Watchdog hang. This register is set to 1 if there is no READ/WRITE command sent or no READ data return for a period of time (defined in tg_param.vh).
compare_error	O	1	Accumulated error mismatch valid over time. This register resets by vio_tg_err_clear, vio_tg_err_continue, vio_tg_restart.

ATG Debug Programming

The ATG provides three ways for traffic pattern programming:

1. Instruction block RAM (**`mem_v1_2_tg_instr_bram.sv`**)
   - Used for regression with predefined traffic instructions
   - Defines default traffic pattern
   - Override default traffic pattern (re-compilation required)
2. Direct instruction through VIO input
   - Used for quick Debug with SINGLE traffic instruction
   - Reprogram through VIO without re-compilation
3. Program instruction table
   - Used for Debug with MULTIPLE traffic instructions
   - Reprogram through VIO without re-compilation

This document assumes debug using "Direct Instruction through VIO." The same concepts extend to both "Instruction Block RAM" and "Program Instruction Table." "Direct Instruction through VIO" is enabled using **`vio_tg_direct_instr_en`**. After **`vio_tg_direct_instr_en`** is set to 1, all of the traffic instruction fields can be driven by the targeted traffic instruction.

Table 38-47: **VIO Signals**

Signal
vio_tg_instr_addr_mode
vio_tg_instr_data_mode
vio_tg_instr_rw_mode
vio_tg_instr_rw_submode
vio_tg_instr_victim_mode
vio_tg_instr_victim_select
vio_tg_instr_victim_aggr_delay
vio_tg_instr_num_of_iter
vio_tg_instr_m_nops_btw_n_burst_m
vio_tg_instr_m_nops_btw_n_burst_n
vio_tg_instr_nxt_instr

ATG Debug Read/Write Error/First Error Bit/First Error Address

ATG identifies if a traffic error is a Read or Write Error when `vio_tg_err_chk_en` is set to 1. Assume EXP_WR_DATA is the expected write data. After the first traffic error is seen from a read (with a value of EXP_WR_DATA’), ATG issues multiple read commands to the failed memory address. If all reads return data EXP_WR_DATA’, ATG classifies the error as a WRITE_ERROR(0). Otherwise, ATG classifies the error as READ_ERROR(1). ATG also tracks the first error bit, first error address seen.

Example 1: The following VIO setting powers on Read/Write Error Type check.

```
.vio_tg_err_chk_en                 (1'b1), // Powers on Error Type Check
.vio_tg_direct_instr_en            (1'b1), // Powers on Direct Instruction Mode
.vio_tg_instr_num                  (5'b00000),
.vio_tg_instr_addr_mode            (TG_PATTERN_MODE_LINEAR),
.vio_tg_instr_data_mode            (TG_PATTERN_MODE_PRBS),
.vio_tg_instr_rw_mode              (TG_RW_MODE_WRITE_READ),
.vio_tg_instr_rw_submode           (2'b00),
.vio_tg_instr_victim_mode          (TG_VICTIM_MODE_NO_VICTIM),
.vio_tg_instr_victim_select        (3'b000),
.vio_tg_instr_victim_aggr_delay    (5'd0),
.vio_tg_instr_num_of_iter          (32'd1000),
.vio_tg_instr_m_nops_btw_n_burst_m (10'd0),
.vio_tg_instr_m_nops_btw_n_burst_n (32'd10),
.vio_tg_instr_nxt_instr            (6`d0),
```

Figure 38-105 shows a Write Error waveform. When `vio_tg_status_err_type_valid` is 1, `vio_tg_status_err_type` shows a WRITE ERROR (0). When `vio_tg_status_first_err_bit_valid` is 1, the following occurs:

- `vio_tg_status_first_err_bit, 0x8` is the corrupted bit
- `vio_tg_first_err_addr` shows the address with the corrupted data as `0x678`

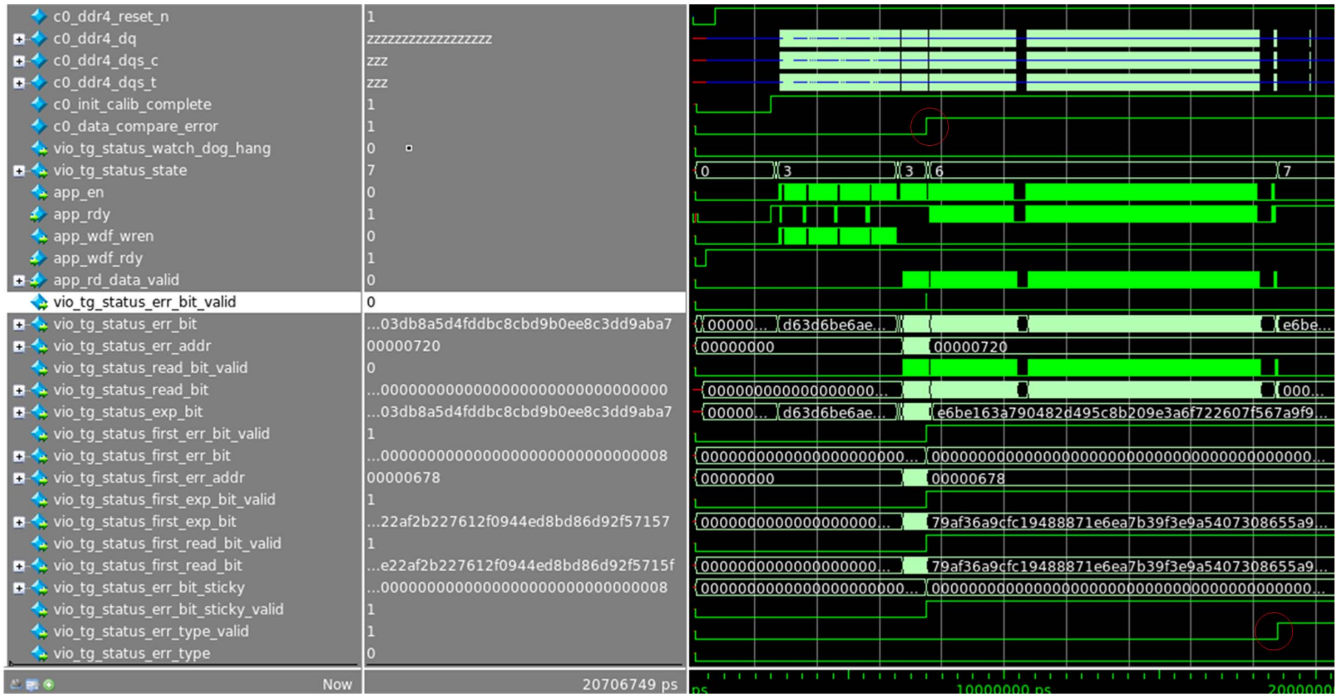


Figure 38-105: **VIO Write Error Waveform**

Figure 38-106 shows a Read Error waveform. When `vio_tg_status_err_type_valid` is 1, `vio_tg_status_err_type` shows a READ ERROR (0). When `vio_tg_status_first_err_bit_valid` is 1, the following occurs:

- `vio_tg_status_first_err_bit, 0x60` is the corrupted bit
- `vio_tg_first_err_addr` shows the address with the corrupted data as `0x1B0`

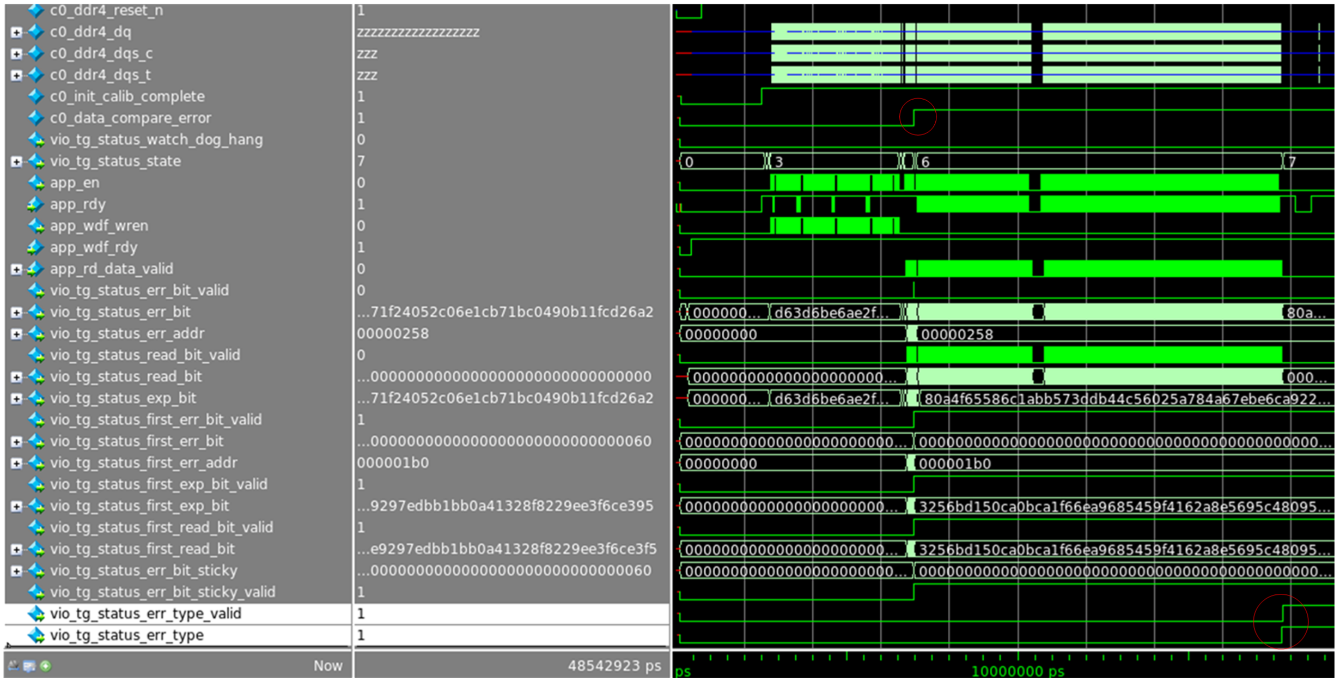


Figure 38-106: **VIO Read Error Waveform**

ATG Debug First Error Bit/First Error Address/Sticky Error Bit

When **`vio_tg_err_chk_en`** is set to 1, ATG stops after the first error. When **`vio_tg_err_chk_en`** is set to 0, ATG does not stop after the first error and would track error continuously using **`vio_tg_status_err_bit_valid`**/ **`vio_tg_status_err_bit`**/**`vio_tg_status_err_addr`**.

The signals **`vio_tg_status_err_bit_sticky_valid`**/ **`vio_tg_status_err_bit_sticky`** accumulate all data bit(s) with error(s) seen.

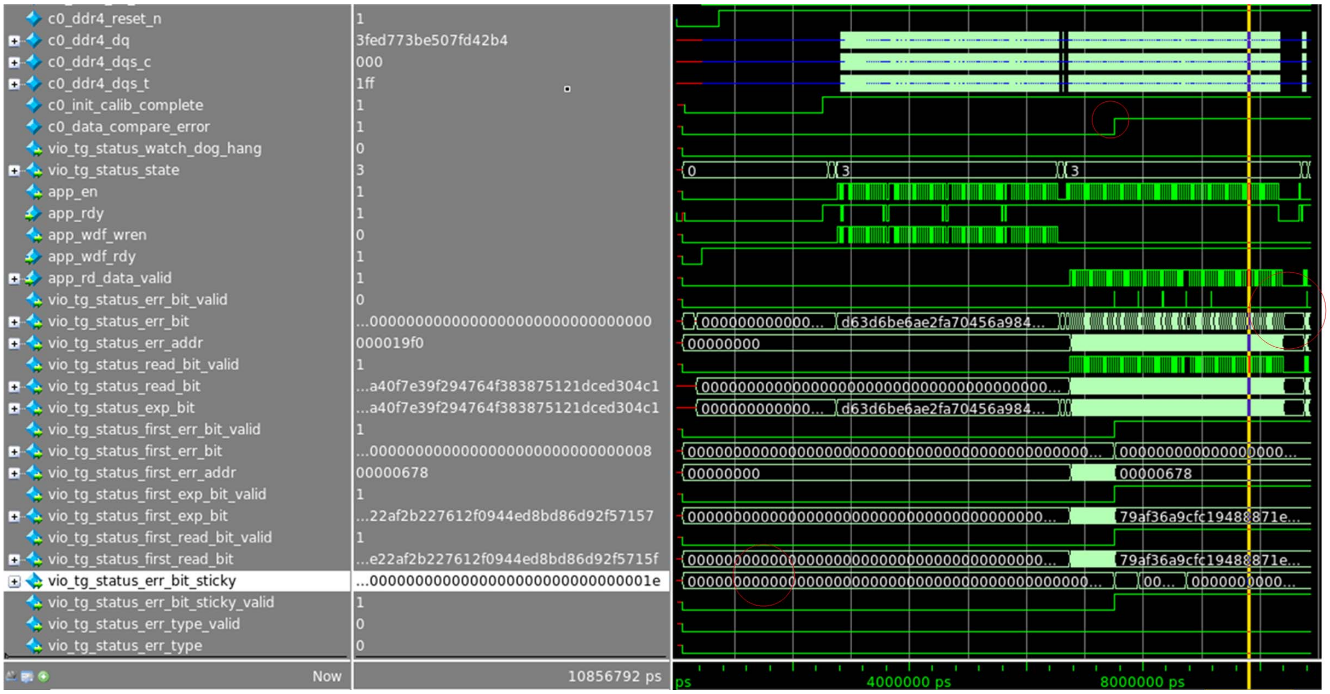
Example 2: The following VIO setting powers off Read/Write Error Type check:

```
.vio_tg_err_chk_en                  (1'b0), // Powers on Error Type Check
.vio_tg_direct_instr_en             (1'b1), // Powers on Direct Instruction Mode
.vio_tg_instr_num                   (5'b00000),
.vio_tg_instr_addr_mode             (TG_PATTERN_MODE_LINEAR),
.vio_tg_instr_data_mode             (TG_PATTERN_MODE_PRBS),
.vio_tg_instr_rw_mode               (TG_RW_MODE_WRITE_READ),
.vio_tg_instr_rw_submode            (2'b00),
.vio_tg_instr_victim_mode           (TG_VICTIM_MODE_NO_VICTIM),
.vio_tg_instr_victim_select         (3'b000),
.vio_tg_instr_victim_aggr_delay     (5'd0),
.vio_tg_instr_num_of_iter           (32'd1000),
.vio_tg_instr_m_nops_btw_n_burst_m  (10'd0),
.vio_tg_instr_m_nops_btw_n_burst_n  (32'd10),
.vio_tg_instr_nxt_instr             (6`d0),
```

Figure 38-107 shows six addresses with read error (note that this is the same example as was used with "Write Error" earlier. "Write Error" is not presented because **`vio_tg_err_chk_en`** is disabled here):

```
vio_tg_status_err_bit_valid is asserted six times.
```

For each assertion, the corresponding bit error is presented at **`vio_tg_status_err_bit`**. After five assertions in **`vio_tg_status_err_bit_valid`** (yellow marker), **`vio_tg_status_err_bit_sticky`** shows bits **`0x1E`** (binary **`11110`**) have bit corruption.



c0_ddr4_reset_n	1
c0_ddr4_dq	3fed773be507fd42b4
c0_ddr4_dqs_c	000
c0_ddr4_dqs_t	1ff
c0_init_calib_complete	1
c0_data_compare_error	1
vio_tg_status_watch_dog_hang	0
vio_tg_status_state	3
app_en	1
app_rdy	1
app_wdf_wren	0
app_wdf_rdy	1
app_rd_data_valid	1
vio_tg_status_err_bit_valid	0
vio_tg_status_err_bit	...00000000000000000000000000000000
vio_tg_status_err_addr	000019f0
vio_tg_status_read_bit_valid	1
vio_tg_status_read_bit	...a40f7e39f294764f383875121dced304c1
vio_tg_status_exp_bit	...a40f7e39f294764f383875121dced304c1
vio_tg_status_first_err_bit_valid	1
vio_tg_status_first_err_bit	...00000000000000000000000000000008
vio_tg_status_first_err_addr	00000678
vio_tg_status_first_exp_bit_valid	1
vio_tg_status_first_exp_bit	...22af2b227612f0944ed8bd86d92f57157
vio_tg_status_first_read_bit_valid	1
vio_tg_status_first_read_bit	...e22af2b227612f0944ed8bd86d92f5715f
vio_tg_status_err_bit_sticky	...0000000000000000000000000000001e
vio_tg_status_err_bit_sticky_valid	1
vio_tg_status_err_type_valid	0
vio_tg_status_err_type	0
Now	10856792 ps

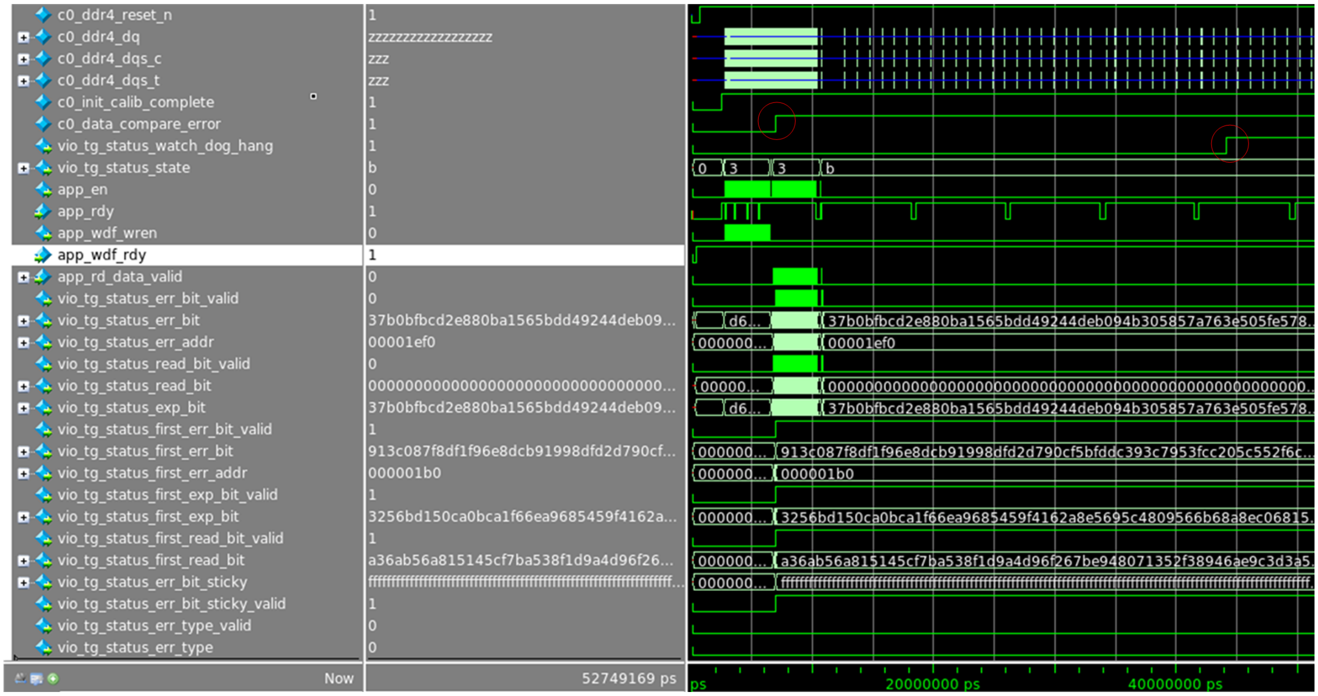
Figure 38-107: **VIO Read Error Waveform**

ATG Debug WatchDog Hang

ATG expects the application interface to accept a command within a certain wait time. ATG also looks for the application interface to return data within a certain wait time after a read command is issued. If either case is violated, ATG flags a WatchDog Hang.

When WatchDogHang is asserted, if **`vio_tg_status_state`** is in "*Wait" states, ATG is waiting for read data return. If **`vio_tg_status_state`** is in "Exe" state, ATG is waiting for application interface to accept the next command.

Example 3: The following example shows that ATG asserts WatchDogHang. This example shares the same VIO control setting as Example 2. In this example, ATG **`vio_tg_status_state`** shows a "DNWait" state. Hence, ATG is waiting for read data return.



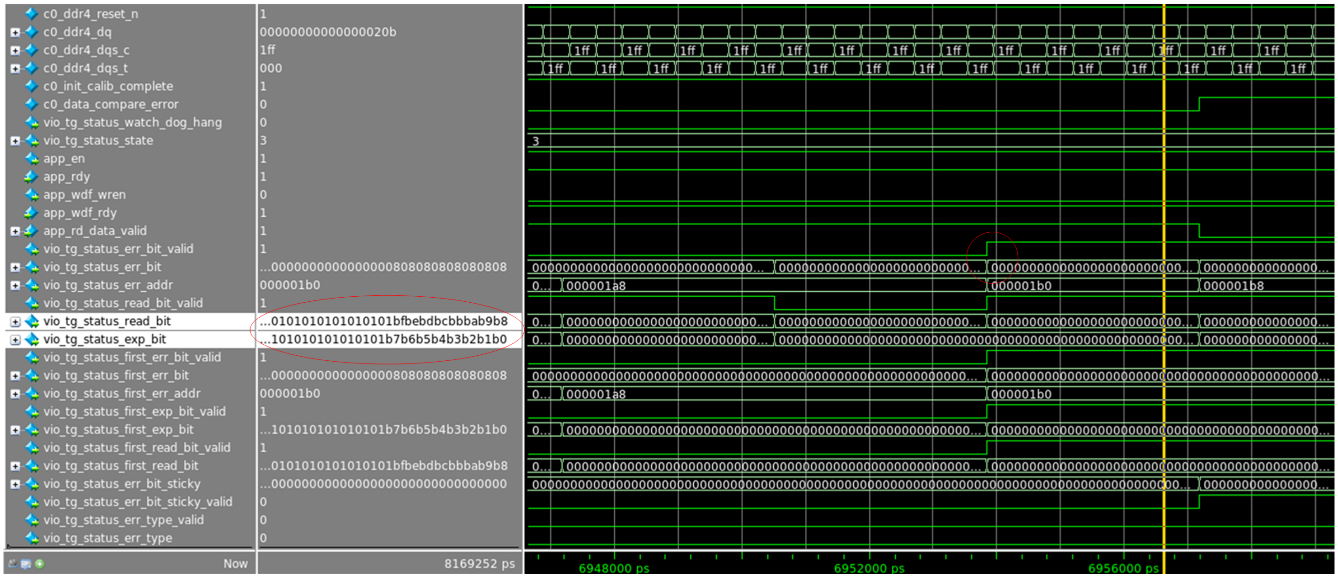
c0_ddr4_reset_n	1
c0_ddr4_dq	zzzzzzzzzzzzzzzzzz
c0_ddr4_dqs_c	zzz
c0_ddr4_dqs_t	zzz
c0_init_calib_complete	1
c0_data_compare_error	1
vio_tg_status_watch_dog_hang	1
vio_tg_status_state	b
app_en	0
app_rdy	1
app_wdf_wren	0
app_wdf_rdy	1
app_rd_data_valid	0
vio_tg_status_err_bit_valid	0
vio_tg_status_err_bit	37b0bfbcd2e880ba1565bdd49244deb09...
vio_tg_status_err_addr	00001ef0
vio_tg_status_read_bit_valid	0
vio_tg_status_read_bit	00000000000000000000000000000000...
vio_tg_status_exp_bit	37b0bfbcd2e880ba1565bdd49244deb09...
vio_tg_status_first_err_bit_valid	1
vio_tg_status_first_err_bit	913c087f8df1f96e8dcb91998dfd2d790cf...
vio_tg_status_first_err_addr	000001b0
vio_tg_status_first_exp_bit_valid	1
vio_tg_status_first_exp_bit	3256bd150ca0bca1f66ea9685459f4162a...
vio_tg_status_first_read_bit_valid	1
vio_tg_status_first_read_bit	a36ab56a815145cf7ba538f1d9a4d96f26...
vio_tg_status_err_bit_sticky	ffffffffffffffffffffffffffffffffffffffffffffffffffff...
vio_tg_status_err_bit_sticky_valid	1
vio_tg_status_err_type_valid	0
vio_tg_status_err_type	0
Now	52749169 ps

Figure 38-108: **ATG Debug Watchdog Hang Waveform**

To further debug, **`vio_tg_instr_data_mode`** is updated to Linear data for better understanding in data return sequence.

```
.vio_tg_err_chk_en                 (1'b0), // Powers on Error Type Check
.vio_tg_direct_instr_en            (1'b1), // Powers on Direct Instruction Mode
.vio_tg_instr_num                  (5'b00000),
.vio_tg_instr_addr_mode            (TG_PATTERN_MODE_LINEAR),
.vio_tg_instr_data_mode            (TG_PATTERN_MODE_LINEAR),
.vio_tg_instr_rw_mode              (TG_RW_MODE_WRITE_READ),
.vio_tg_instr_rw_submode           (2'b00),
.vio_tg_instr_victim_mode          (TG_VICTIM_MODE_NO_VICTIM),
.vio_tg_instr_victim_select        (3'b000),
.vio_tg_instr_victim_aggr_delay    (5'd0),
.vio_tg_instr_num_of_iter          (32'd1000),
.vio_tg_instr_m_nops_btw_n_burst_m (10'd0),
.vio_tg_instr_m_nops_btw_n_burst_n (32'd10),
.vio_tg_instr_nxt_instr            (6`d0),
```

With Linear Data, Figure 38-109 shows that when an error is detected, read data (**`vio_tg_status_read_bit`**) is one request ahead of expected data (**`vio_tg_status_exp_bit`**). One possibility is read command with address `0x1B0` is dropped. Hence the next returned data with read address `0x1B8` is being compared against the expected data of read address `0x1B0`.



c0_ddr4_reset_n	1
c0_ddr4_dq	000000000000000020b
c0_ddr4_dqs_c	1ff
c0_ddr4_dqs_t	000
c0_init_calib_complete	1
c0_data_compare_error	0
vio_tg_status_watch_dog_hang	0
vio_tg_status_state	3
app_en	1
app_rdy	1
app_wdf_wren	0
app_wdf_rdy	1
app_rd_data_valid	1
vio_tg_status_err_bit_valid	1
vio_tg_status_err_bit	...0000000000000000808080808080808
vio_tg_status_err_addr	000001b0
vio_tg_status_read_bit_valid	1
vio_tg_status_read_bit	...0101010101010101bfbebdbcbbbab9b8
vio_tg_status_exp_bit	...1010101010101b7b6b5b4b3b2b1b0
vio_tg_status_first_err_bit_valid	1
vio_tg_status_first_err_bit	...0000000000000000808080808080808
vio_tg_status_first_err_addr	000001b0
vio_tg_status_first_exp_bit_valid	1
vio_tg_status_first_exp_bit	...1010101010101b7b6b5b4b3b2b1b0
vio_tg_status_first_read_bit_valid	1
vio_tg_status_first_read_bit	...0101010101010101bfbebdbcbbbab9b8
vio_tg_status_err_bit_sticky	...0000000000000000000000000000000
vio_tg_status_err_bit_sticky_valid	0
vio_tg_status_err_type_valid	0
vio_tg_status_err_type	0
Now	8169252 ps

Figure 38-109: **ATG Debug Watchdog Hang Waveform with Linear Data**

Isolating the Data Error

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

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

  - Rewriting, rereading, resetting, recalibrating.

The next step is to isolate whether the data corruption is due to writes or reads.

Determining If a Data Error is Due to the Write or Read

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

Some experiments that can help to isolate the issue include:

- If the errors are intermittent, issue a small initial number of writes, followed by continuous reads from those locations. If the reads intermittently yield bad data, there is a potential read issue. If the reads always yield the same (wrong) data, there is a write issue.
- Using high quality probes and scope, capture the write at the memory and the read at the FPGA to view data accuracy, appropriate DQS-to-DQ phase relationship, and signal integrity. To ensure the appropriate transaction is captured on DQS and DQ, look at the initial transition on DQS from 3-state to active. During a Write, DQS does not have a low preamble. During a read, the DQS has a low preamble. The following is an example of a DDR3 Read and a Write to illustrate the difference:

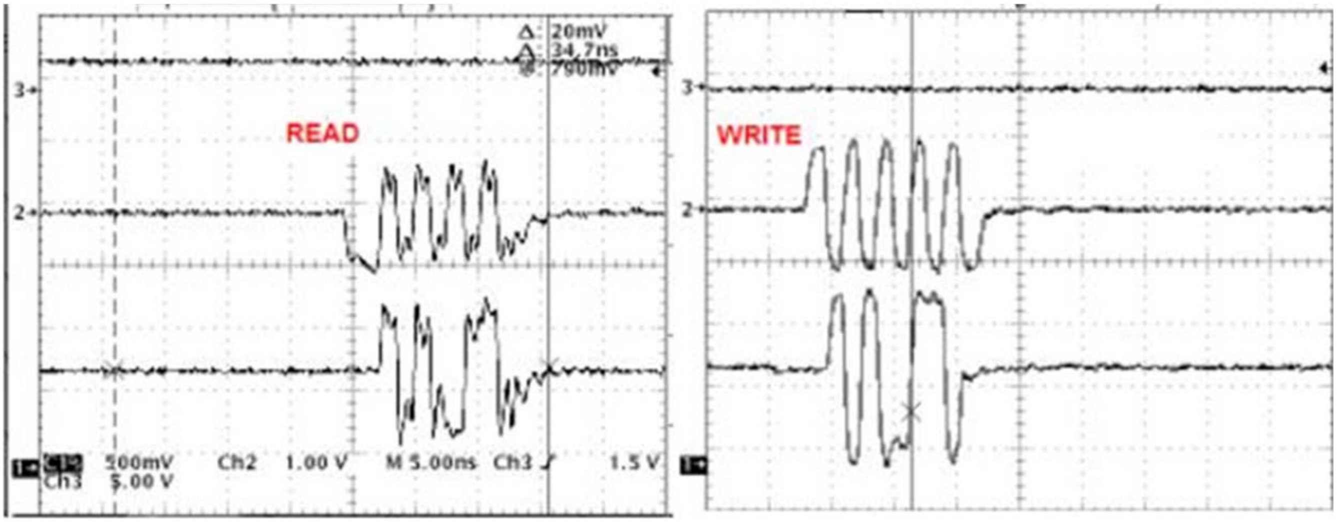
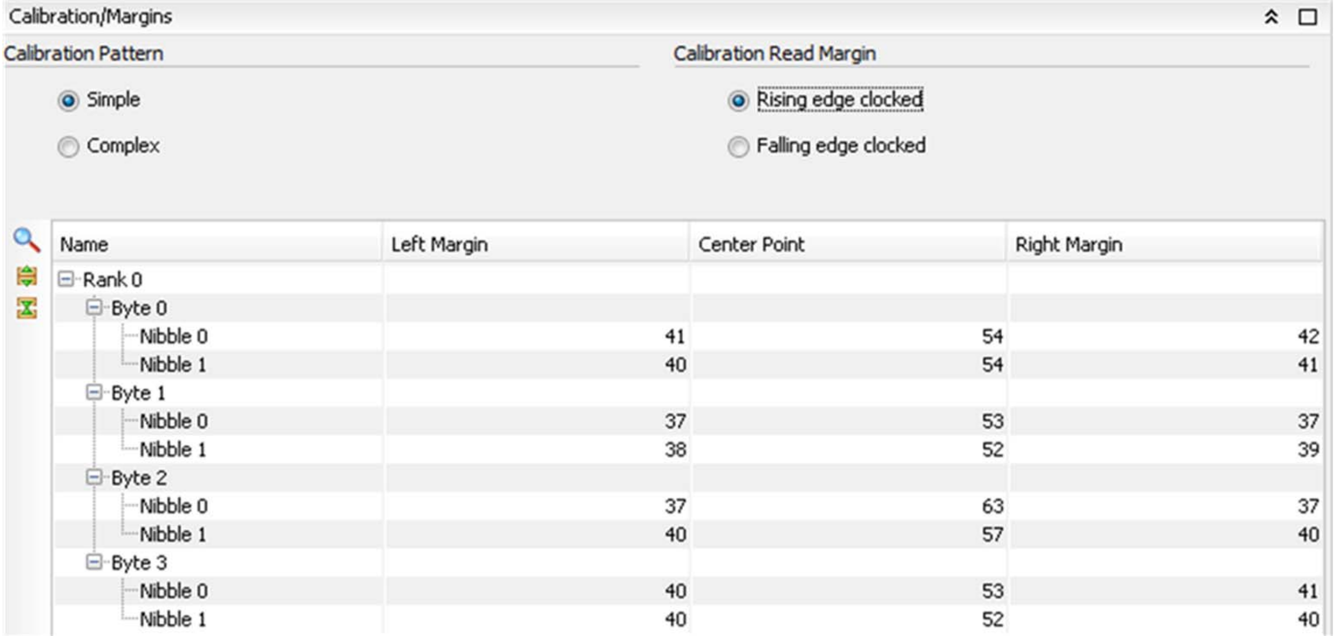


Figure 38-110: **DDR3 Read vs. Write Scope Capture**

- Analyze read timing:
  - Check the PQTR/NQTR values after calibration. Look for variations between PQTR/NQTR values. PQTR/NQTR values should be very similar for DQs in the same DQS group.

Analyzing Read and Write Margin

The XSDB output can be used to determine the available read and write margins during calibration. Starting with 2014.3, an XSDB Memory IP GUI is available through the Hardware Manager to view the read calibration margins for both rising edge clock and failing edge clock. The margins are provided for both simple and complex pattern calibration. The complex pattern results are more representative of the margin expected during post calibration traffic.



Calibration Pattern

● Simple

○ Complex

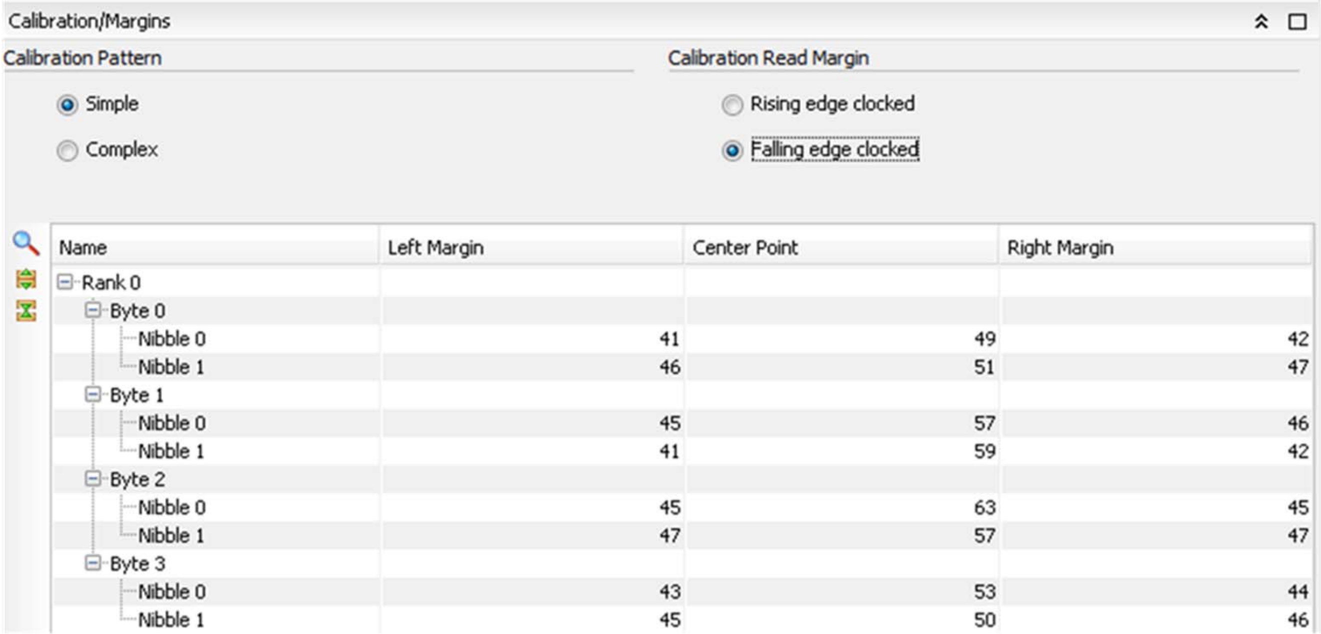
Calibration Read Margin

● Rising edge clocked

○ Falling edge clocked

Name	Left Margin	Center Point	Right Margin
Rank 0			
Byte 0			
Nibble 0	41	54	42
Nibble 1	40	54	41
Byte 1			
Nibble 0	37	53	37
Nibble 1	38	52	39
Byte 2			
Nibble 0	37	63	37
Nibble 1	40	57	40
Byte 3			
Nibble 0	40	53	41
Nibble 1	40	52	40

Figure 38-111: **Calibration Rising Edge Clocked Read Margin**



Calibration/Margins

Calibration Pattern

● Simple

○ Complex

Calibration Read Margin

○ Rising edge clocked

● Falling edge clocked

Name	Left Margin	Center Point	Right Margin
Rank 0			
Byte 0			
Nibble 0	41	49	42
Nibble 1	46	51	47
Byte 1			
Nibble 0	45	57	46
Nibble 1	41	59	42
Byte 2			
Nibble 0	45	63	45
Nibble 1	47	57	47
Byte 3			
Nibble 0	43	53	44
Nibble 1	45	50	46

Figure 38-112: **Calibration Falling Edge Clocked Read Margin**

The following Tcl command can also be used when the Hardware Manager is open to get an output of the window values:

```
report_hw_mig [get_hw_migs]
```

Table 38-48: **Signals of Interest for Read and Write Margin Analysis**

Signal	Usage	Signal Description
MARGIN_CONTROL	Per Interface	Reserved
MARGIN_STATUS	Per Interface	Reserved
RDLVL_MARGIN_PQTR_LEFT_RANK*_BYTE*_BIT*	Per Bit	Number of taps from center of window to left edge.
RDLVL_MARGIN_NQTR_LEFT_RANK*_BYTE*_BIT*	Per Bit	Number of taps from center of window to left edge.
RDLVL_MARGIN_PQTR_RIGHT_RANK*_BYTE*_BIT*	Per Bit	Number of taps from center of window to right edge.
RDLVL_MARGIN_NQTR_RIGHT_RANK*_BYTE*_BIT*	Per Bit	Number of taps from center of window to right edge.
WRITE_DQS_DQ_MARGIN_LEFT_RANK*_BYTE*_BIT*	Per Bit	Number of taps from center of window to left edge.
WRITE_DQS_DQ_MARGIN_RIGHT_RANK*_BYTE*_BIT*	Per Bit	Number of taps from center of window to right edge.

Analyzing Calibration Results

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

Determining Window Size in ps

To determine the window size in ps, first calculate the tap resolution and then multiply the resolution by the number of taps found in the read and/or write window. The tap resolution varies across process (down to variance at each nibble within a part).

However, within a specific process, each tap within the delay chain is the same precise resolution.

1. To compute the 90° offset in taps, take (BISC_PQTR – BISC_ALIGN_PQTR).
2. To estimate tap resolution, take (1/4 of the memory clock period) / (BISC_PQTR – BISC_ALIGN_PQTR).
3. The same then applies for NQTR.

BISC is run on a per nibble basis for both PQTR and NQTR. The write tap results are given on a per byte basis. To use the BISC results to determine the write window, take the average of the BISC PQTR and NQTR results for each nibble. For example, ((BISC_NQTR_NIBBLE0 + BISC_NQTR_NIBBLE1 + BISC_PQTR_NIBBLE0 + BISC_PQTR_NIBBLE1) / 4).

Conclusion

If this document does not help to resolve calibration or data errors, create a WebCase with Xilinx Technical Support (see Technical Support). Attach all of the captured waveforms, XSDB and debug signal results, and the details of your investigation and analysis.

SECTION X: APPENDICES

Upgrading

There are no port or parameter changes for upgrading the Memory IP core in the Vivado Design Suite at this time.

For general information on upgrading the Memory IP, see the “Upgrading IP” section in *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Appendix B

XCKU095/XCVU095 Recommended Memory Pinout Configurations

Introduction

The UltraScale™ devices, XCKU095 and XCVU095, have only one clock region between the two I/O columns in the center of the device which might require special pinout considerations. Other devices in the UltraScale and UltraScale+™ families do not require special pinout considerations because they have two or more clock regions between the I/O columns.

During implementation, a large proportion of the user logic needs to be placed in the center of the device for connectivity and timing reasons. The reduced space between the I/O columns in conjunction with the presence of several Memory Interface IPs, or any large high performance I/O modules, can increase the placement complexity and challenge routing resources. Following the guidelines in this section ensures the most efficient use of available routing resources for faster and predictable timing closure.

For architectural and performance reasons, the memory interface logic needs to be placed in the clock regions located on the right-hand side of the I/Os. The memory interface controller logic is usually placed next to the Address/Command I/Os. A high overall device utilization or user floorplanning constraints in the area next to the Address/Command I/Os can result in reduced available routing resources.

When placing two Memory Interface IPs side-by-side with the Address/Command I/Os located on the same clock region row, several adjacent clock regions become highly utilized which limits the amount of user logic that can cross over or be placed in the same area. When vertically shifted by one or more I/O banks, the potential placement and routing challenges become less common.

In addition to memory interface pin planning, migration to the 2015.3 or later version of the Memory Interface IP helps with timing closure due to updates to the IP clocking and constraints. The next section discusses pinout options for different packages that results in the most efficient use of available routing resources.

Note: Additional design and constraint recommendations are provided in Additional Recommendations section.

Memory Interface Pin Placement

The UltraScale XCKU095 device is available in four different packages and XCVU095 device is available in six different packages. XCVU095 in packages FFVD1517 and FFVC2104 do not require special pin placement because these devices have only one I/O column in the center of the device. Pin placement recommendations to reduce routing challenges for all the relevant packages are listed in this section.

The maximum number of possible 72-bit DDR4 memory interfaces in each package is used to illustrate the pin placement suggestions. This is just an example, the goal is to offset the memory interfaces or at the very least offset the Address/Command I/O banks. The double-headed arrow represents the routing channel that is created by offsetting the Address/Command banks.

XCKU095 FFVA1156 Package

For the XCKU095 in the FFVA1156 package, a pin placement suggestion for two 72-bit DDR4 memory interfaces is shown in Figure B-1. The placement of the Address/Command banks in the horizontally adjacent interfaces A and B is offset by two clock regions to create a routing channel represented by the double-headed arrow.

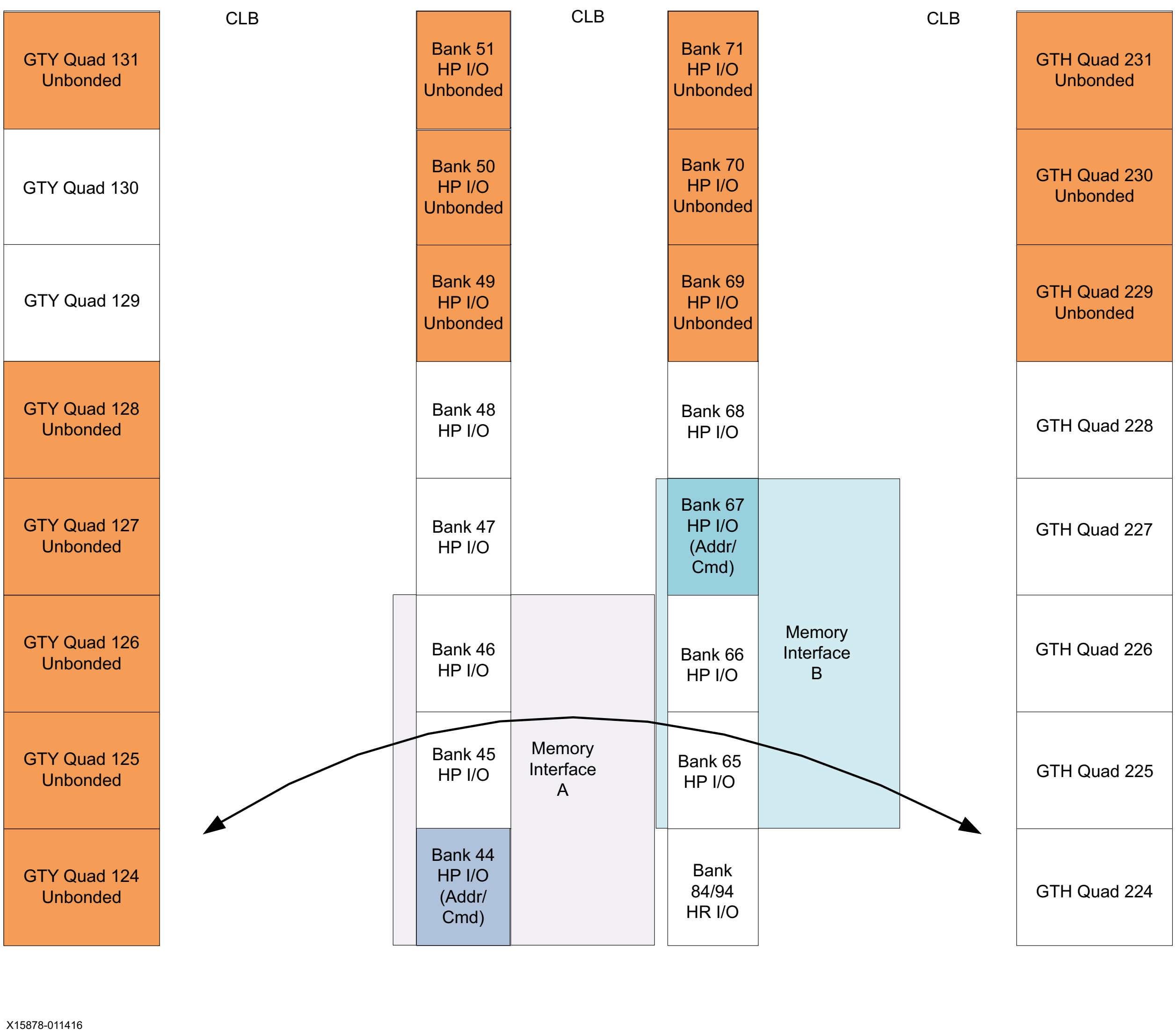


Figure B-1: **XCKU095 FFVA1156 Package Pin Placement**

XCKU095 and XCVU095 in FFVC1517 Package

Both the XCKU095 and XCVU095 are available in the FFVC1517 package. A pin placement recommendation for two 72-bit DDR4 memory interfaces is shown in Figure B-2. The placement of the Address/Command banks in the horizontally adjacent interfaces A and B is offset by two clock regions to create a routing channel represented by the double-headed arrow.

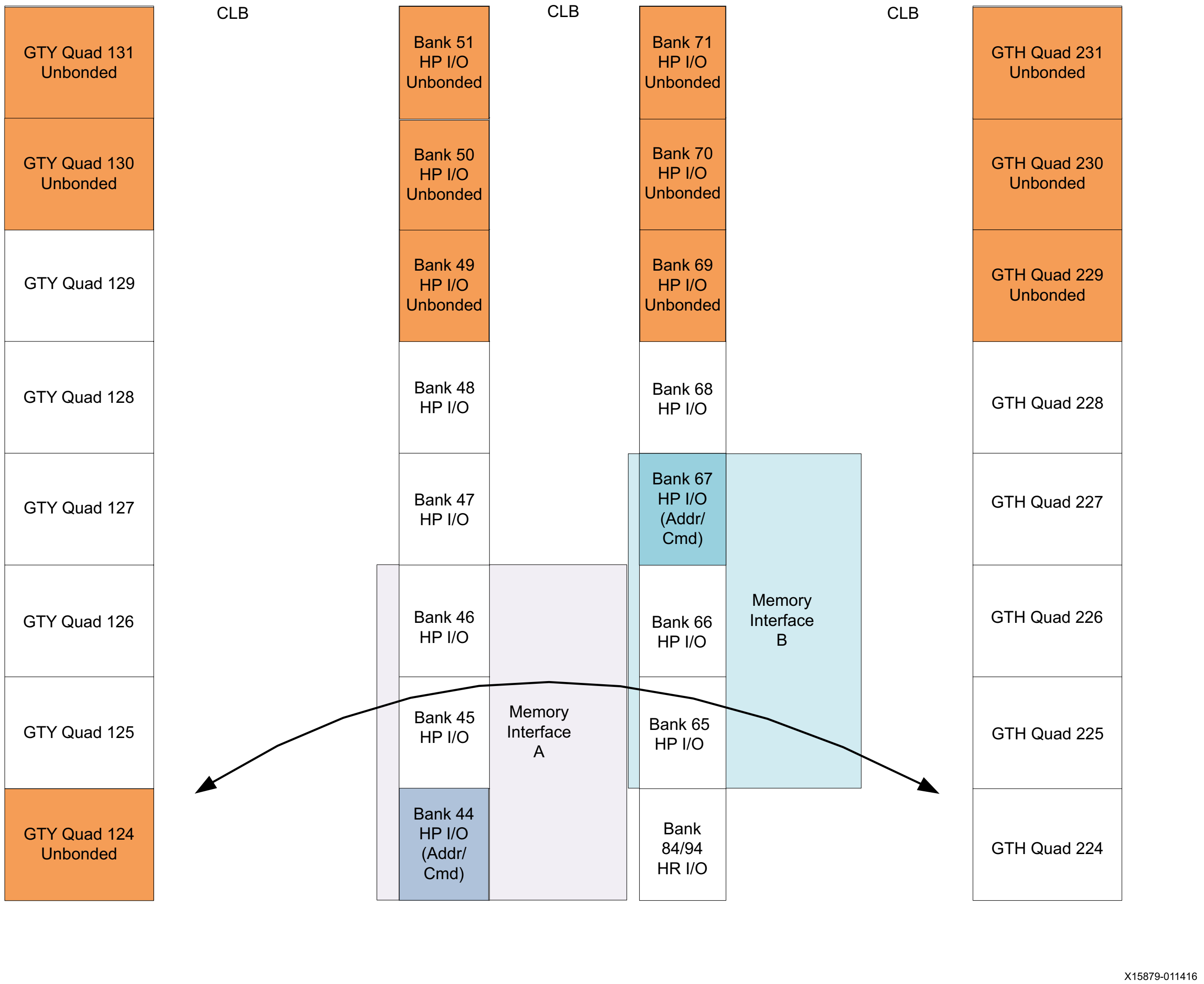


Figure B-2: **FFVC1517 Package Pin Placement**

XCKU095 and XCVU095 in FFVB1760 Package

Both the XCKU095 and XCVU095 are available in the FFVB1760 package. A recommended pin placement with three 72-bit DDR4 memory interfaces is shown in Figure B-3. For this package, Memory Interface A is placed with the Address/Command bank at the top to leverage the unbonded banks in the second column. Memory Interface B and C are offset from each other which creates three routing channels between the two sides of the device.

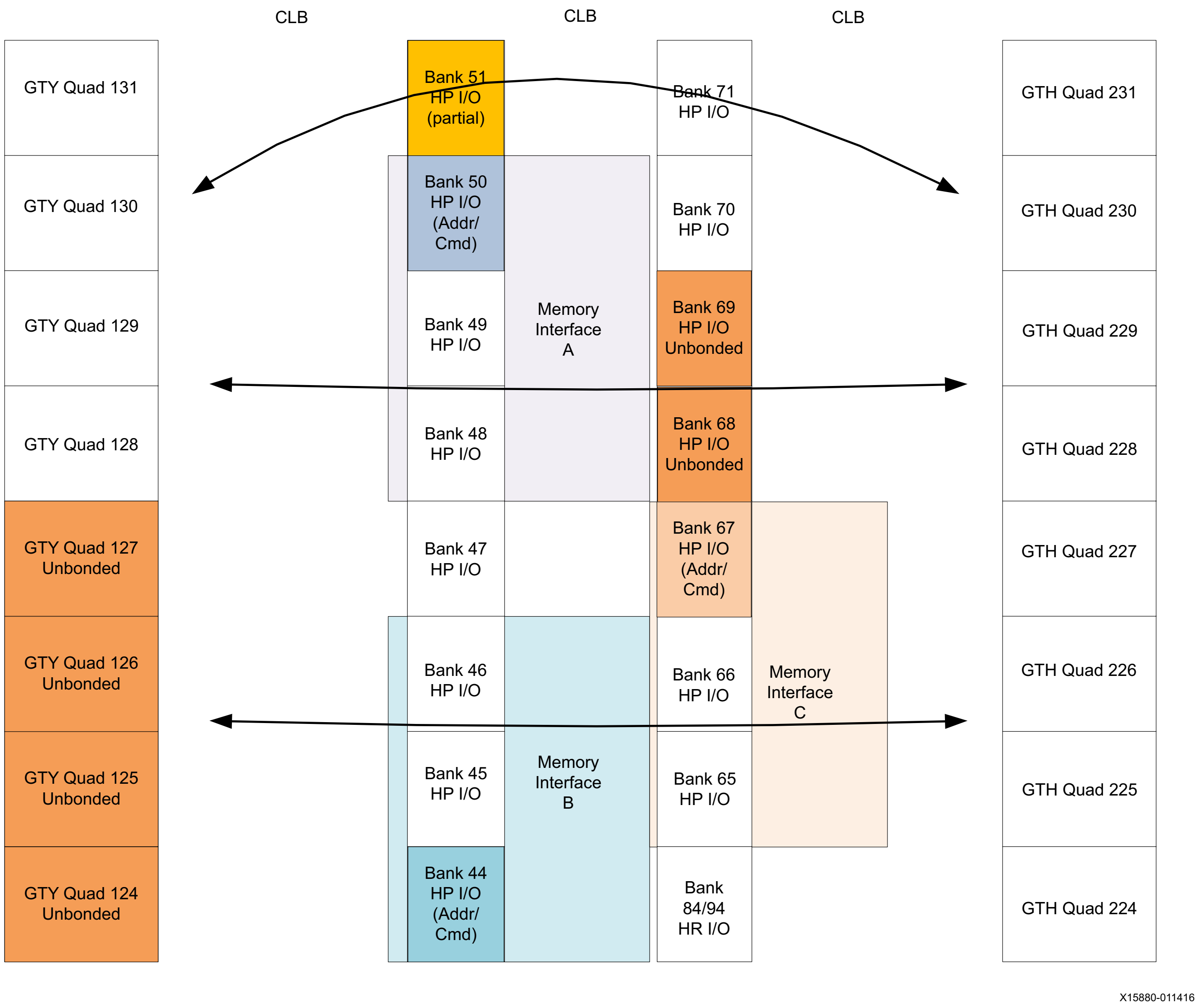


Figure B-3: **FFVB1760 Package Pin Placement**

XCVU095 FFVA2104 Package

For the XCVU095 in the FFVA2104 package, a recommended pin placement with four 72-bit DDR4 memory interfaces is shown in Figure B-4. The strategy for this package was to create a two bank routing channel in the middle of the device. This limited the interfaces to a one bank separation with the horizontally adjacent Memory Interfaces.

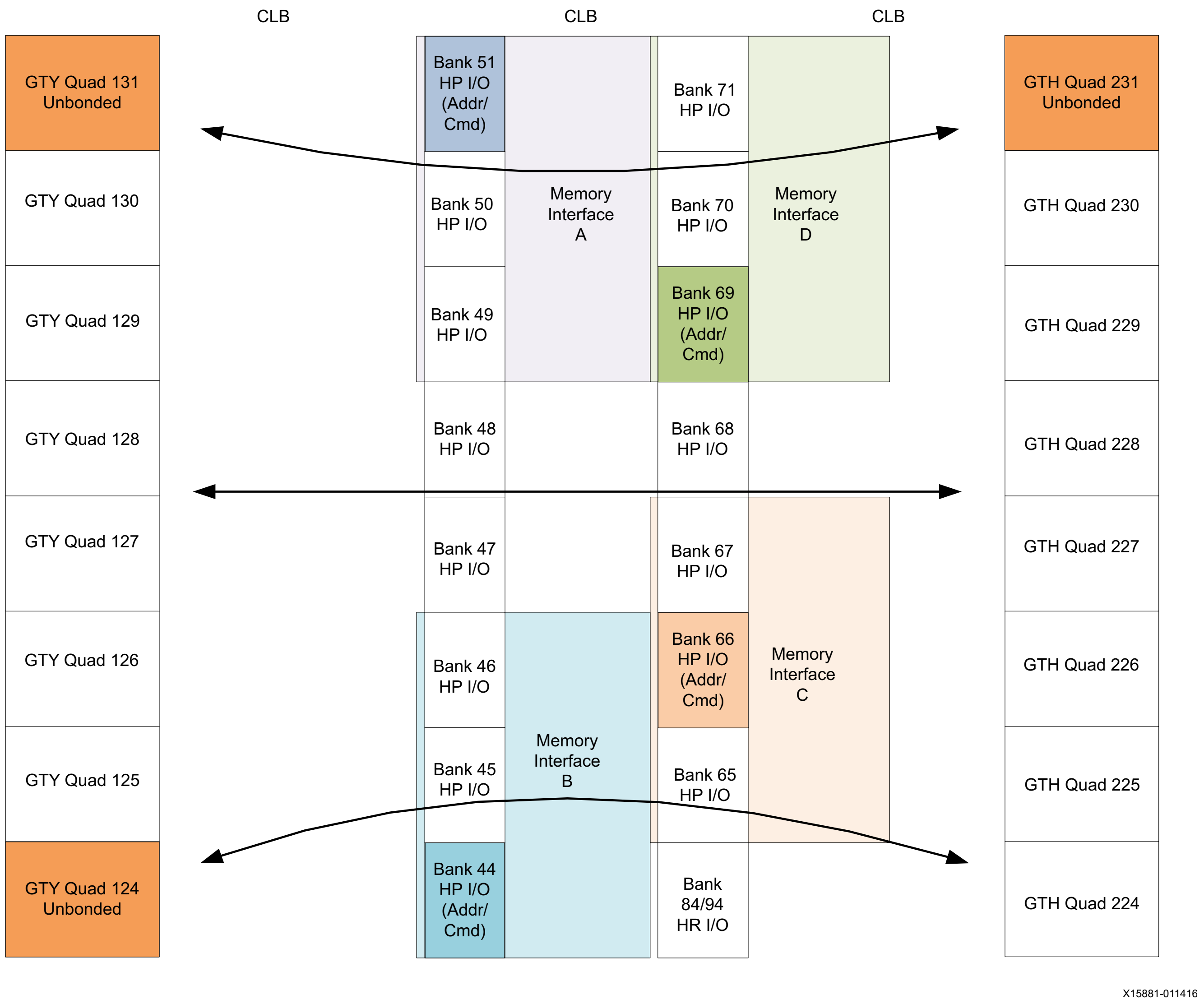


Figure B-4: **XCVU095 FFVA2104 Package Pin Placement**

XCKU095 and XCVU095 in FFVB2104 Package

Both the XCKU095 and XCVU095 are available in the FFVB2104 package. A recommended pin placement with four 72-bit DDR4 memory interfaces is shown in Figure B-5. The strategy for this package was to create a two bank routing channel in the middle of the device. This limited the interfaces to a one bank separation with the horizontally adjacent Memory Interfaces.

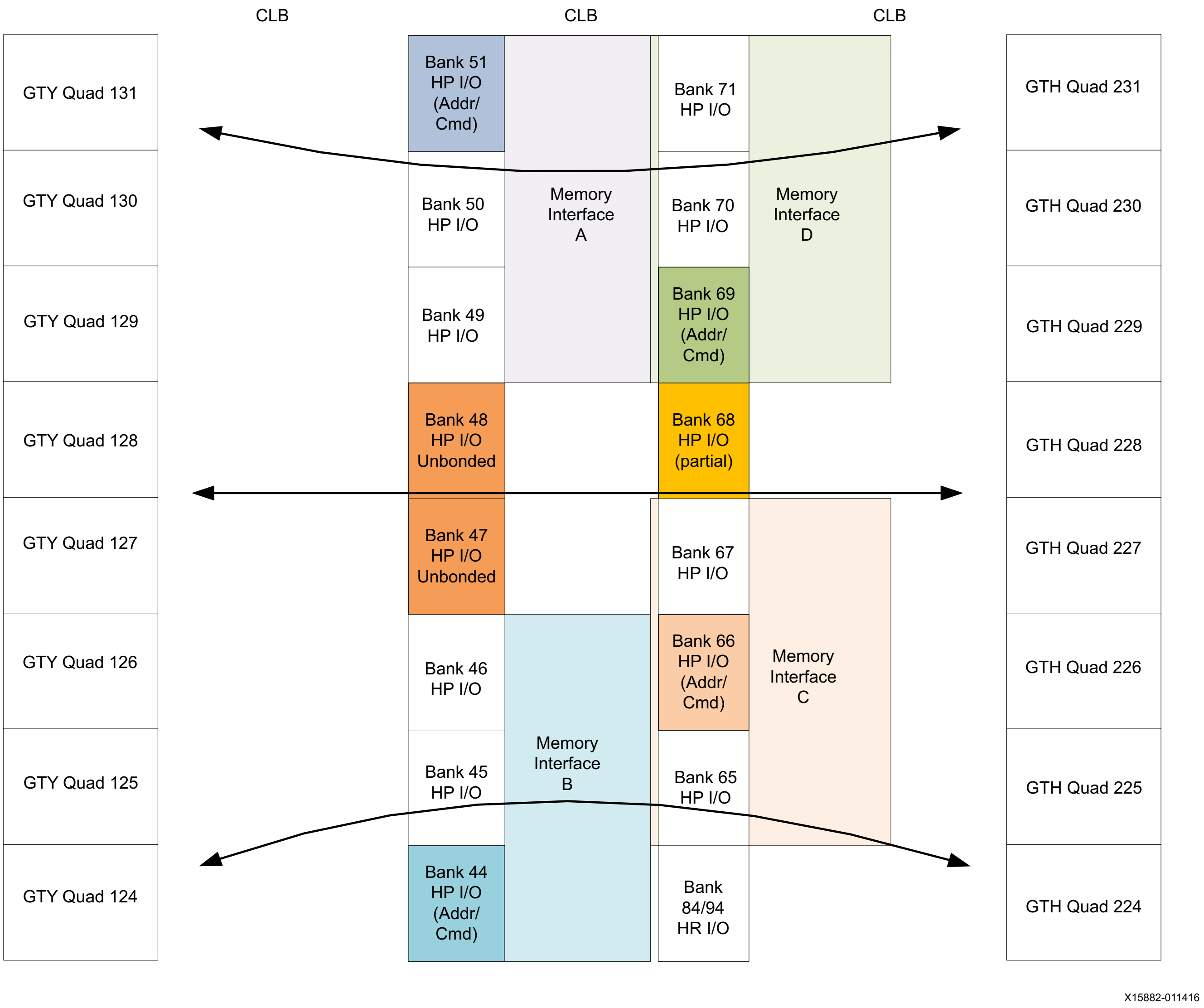


Figure B-5: **FFVB2104 Package Pin Placement**

Additional Recommendations

1. Migrate to 2015.3 or later version of the Vivado Design Suite:
    a. Take advantage of the Quality of Results (QoR) improvements from newer releases.
    b. Upgrade the Memory Interface IPs to benefit from the clocking and constraint improvements.
2. Offset the placement of the Address/Command banks in horizontally adjacent interfaces by at least one clock region to make routing resources available to user logic. The placement of the Address/Command bank within a 72-bit three bank interface depends on whether it is a DIMM or component interface.

    For a component interface, Xilinx recommends placing the Address/Command bank on the outer banks as shown in Figure B-5 for Memory Interface B. This placement enables optimal component placement with fly-by topology as shown in Figure B-6.

    For a DIMM interface, Xilinx recommends placing the Address/Command bank in the center as shown in Figure B-5 for Memory Interface C. This placement enables better PCB routing from the FPGA to the DIMM socket as shown in Figure B-7.

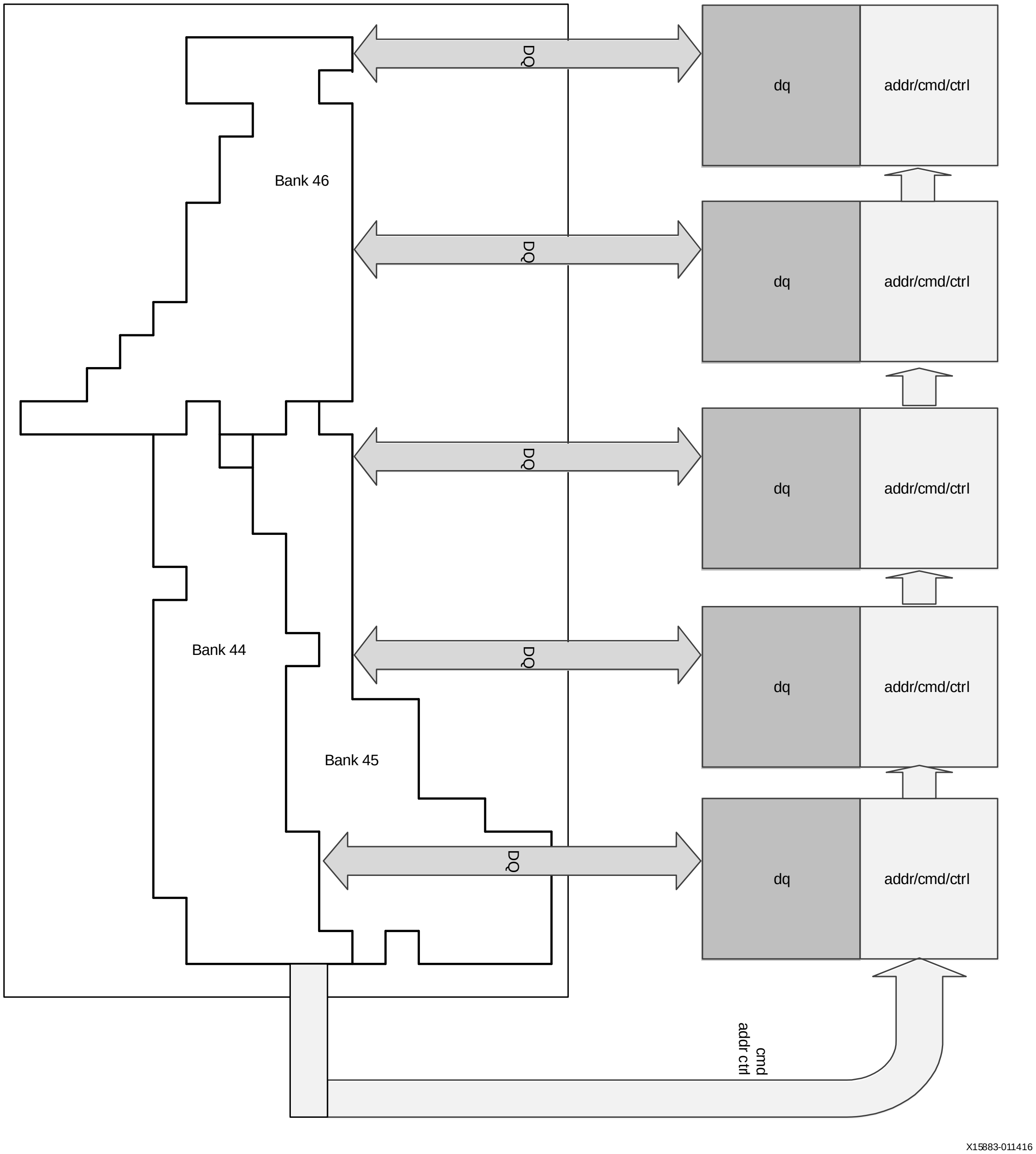


Figure B-6: **Address/Command Placement Recommendation for Five Components with Fly-By Topology**

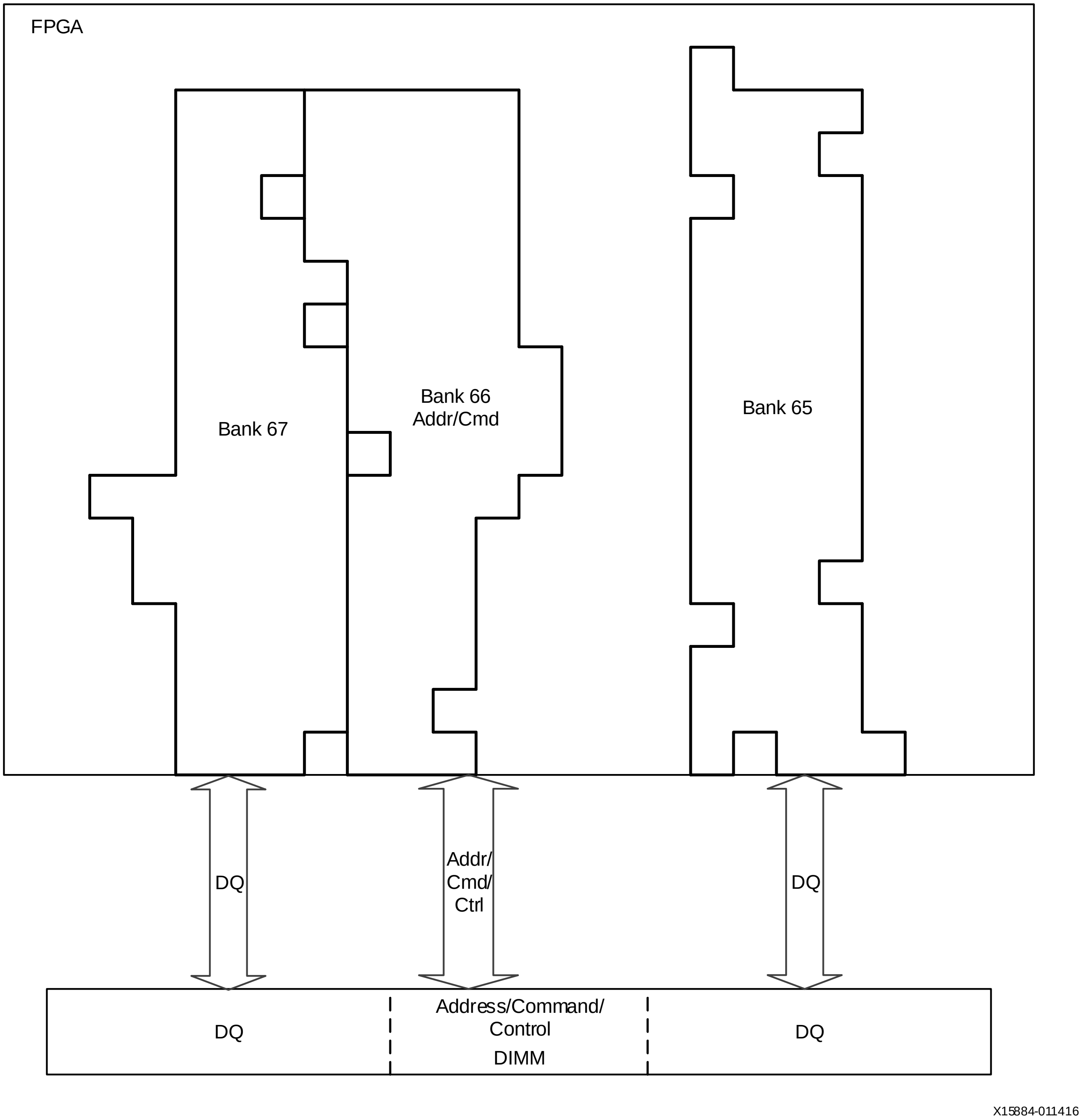


Figure B-7: **Address/Command Bank Placement Recommendation for DIMM**

3. Avoid high device utilization, especially for LUTs as they need space to be spread out in case of high density placement.
4. Design top-level connectivity to minimize crossings over the Memory Interface IPs.
5. Force spreading of memory interface logic placement to a wider area by using the pblock constraints.

a. By default, the memory interface logic is only placed in the clock regions that include the I/O columns.

b. Use two clock region-wide pblock for the Memory Interface IPs located on the right I/O columns.

c. Do not apply this technique to the Memory Interface IPs located on the left I/O columns.

For example, an XCVU095 design with four wide Memory Interface IPs. Only two of them can have their placement relaxed: Memory Interface 2 and Memory Interface 3.

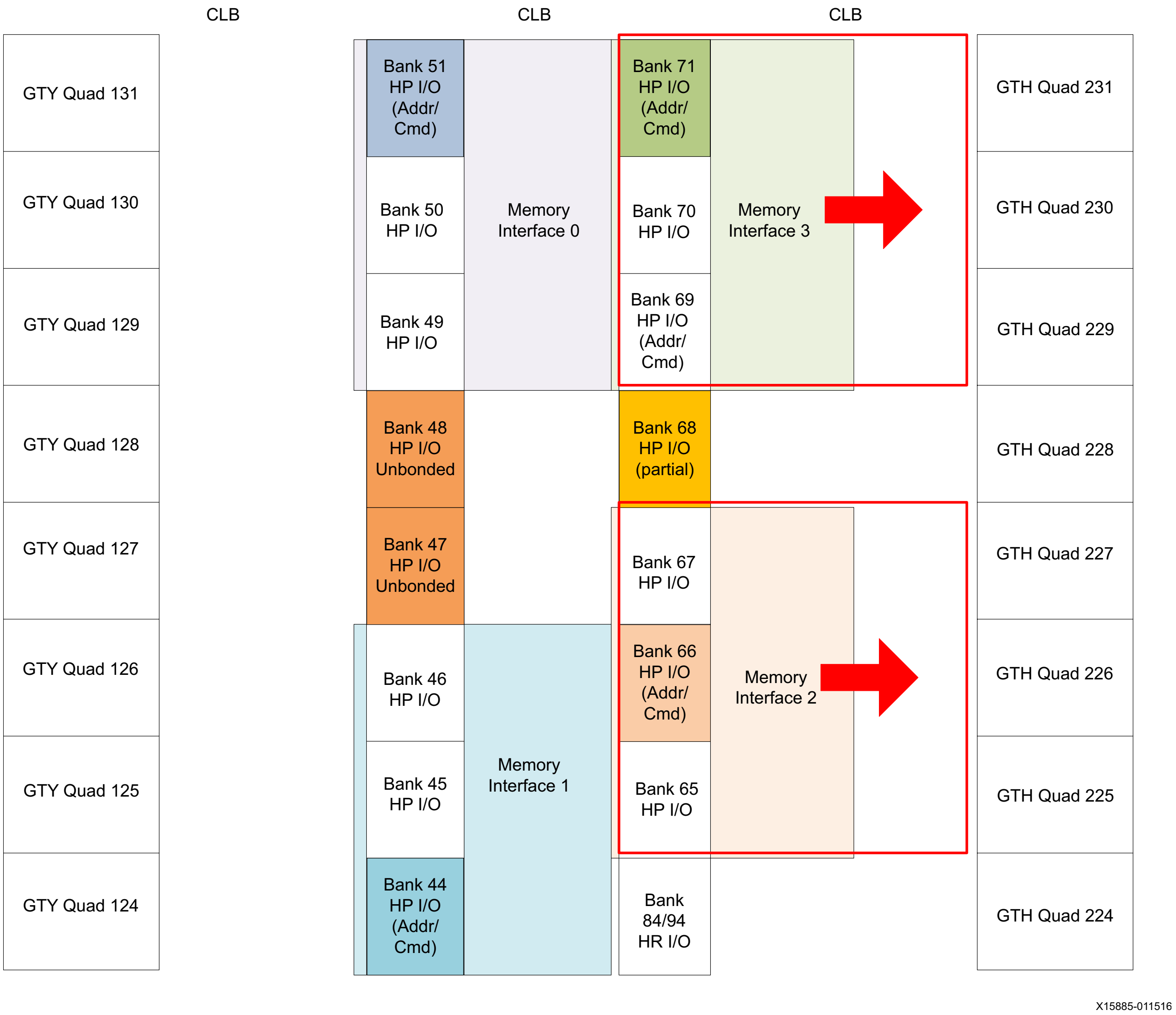


Figure B-8: **Relaxing Memory Interface Placement with pblock Constraints**

The corresponding constraints are as follows:

```
create_pblock MemoryInterface2_pblock
resize_pblock MemoryInterface2_pblock -add CLOCKREGION_X3Y1:CLOCKREGION_X4Y3
add_cells_to_pblock MemoryInterface2_pblock [get_cells a/b/mig_2]

create_pblock MemoryInterface3_pblock
resize_pblock MemoryInterface3_pblock -add CLOCKREGION_X3Y5:CLOCKREGION_X4Y7
add_cells_to_pblock MemoryInterface3_pblock [get_cells a/b/mig_3]
```

6. When migrating, ensure banks selected in one device exist in the target device. See the "Migration between UltraScale Devices and Packages" chapter in the *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583) [Ref 11].

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. JESD79-3F, *DDR3 SDRAM Standard*, JESD79-4, *DDR4 SDRAM Standard,* and JESD209-3C, *LPDDR3 SDRAM Standard*, JEDEC Solid State Technology Association
2. *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892)
3. *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893)
4. *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)
5. *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923)
6. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
7. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
8. *UltraScale Architecture Clocking Resources User Guide* (UG572)
9. *Vivado Design Suite Properties Reference Guide* (UG912)
10. *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187)
11. *UltraScale Architecture PCB Design and Pin Planning User Guide* (UG583)
12. Arm AMBA Specifications
13. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
14. *Vivado Design Suite User Guide: Designing with IP* (UG896)
15. *Vivado Design Suite User Guide: Getting Started* (UG910)
16. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
17. *Vivado Design Suite User Guide: Implementation* (UG904)
18. *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899)
19. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
20. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
21. *UltraScale Maximum Memory Performance Utility* (XTP414)
22. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
23. *Fast Calibration and Daisy Chaining Functions in DDR4 Memory Interfaces Application Note* (XAPP1321)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/20/2022	1.4	**DDR3/DDR4** • Updated AR 71209 in DDR4 SDRAM. • Added note on dm/dbi in DDR4 Pin Rules. • Added note on +: notation in app_addr[APP_ADDR_WIDTH – 1:0]. • Added ECC_TEST important note in AXI4-Lite Slave Control/Status Register Map Detailed Descriptions.
10/22/2021	1.4	**DDR3/DDR4** • Updated clamshell supports the Physical Layer Ping Pong in Clamshell Topology.
08/11/2021	1.4	Editorial updates only. No technical content updates.
06/30/2021	1.4	**DDR3/DDR4** • GB update in Feature Summary. • Updated clock generator in Input Clock Requirement. **LPDDR3** • GB update in Feature Summary. • Updated clock generator in Input Clock Requirement. • Added Pinout Swapping. **QDR II+** • Updated clock generator in Input Clock Requirement. **RLDRAM 3** • Updated clock generator in Input Clock Requirement.
01/21/2021	1.4	• Added Navigating Content by Design Process in each section. **DDR3/DDR4** • Updated Important description in Feature Summary. • Added Important note in DM_DBI Parameter. • Updated description #17 in DDR4 Pin Rules. **Debugging** • Updated description #16 first bullet in General Checks.
06/03/2020	1.4	Added spread spectrum description in all Input Clock Requirement sections.

Date	Version	Revision
10/30/2019	1.4	**DDR3/DDR4** • Updated x4 to x16 components in Feature Summary. • Added LRDIMM support to DDR4 SDRAM. • Added LRDIMM description in Address Parity. • Updated note in Clamshell Topology. • Added PAYLOAD_WIDTH description to Table 4-37. • Added Tandem note in DDR3 Pin Rules and DDR4 Pin Rules. • Added AXI Addressing. • Updated #1 and #2 in VT Tracking. **QDR-IV** • Updated ECC important note in Basic Tab. **RLDRAM 3** • Added Important note in Overview. • Removed Important note in Additional Clocks. • Updated first paragraph and added note in Setting TWTR Check Parameter OFF for RLDRAM 3 Designs.
05/22/2019	1.4	• Added Simulation Language in all Project-Based Simulation Flow Using Vivado Simulator sections. **DDR3/DDR4** • Updated note #3 in Features. • Updated description in Address Parity. • Updated description in Fault Injection. • Added timing parameters for DDR4 in Setting Timing Parameters for DDR4 Non-Custom Memory Parts. • Added note in Simulation. **Debugging** • Updated IBIS description for #11 and updated #15 description in General Checks.
12/05/2018	1.4	**DDR3/DDR4** • Added AXI4 Slave Interface and note in DDR3 SDRAM. • Added AXI4 Slave Interface, note, and recommended note in DDR4 SDRAM. • Added AXI4 Slave Interface Transaction Examples section. • Added important note in DIMM Configurations. • Added Setting Burst Type for PHY_ONLY Designs section. **QDR II+** • Updated qdriip_doff_n in Memory Interface Signals table. **Traffic Generator** • Updated descriptions for M NOPs and N NOPs in Traffic Generator Instruction Options table. • Updated Error Status Registers in Default Traffic Generator Control Connection table.

Date	Version	Revision
04/04/2018	1.4	• Updated Input Clock Requirement section for DDR3, DDR4, LPDDR3, QDR II+, and RLDRAM 3 in Designing with the Core chapters. • Added Recommended DCIUpdateMode note in DDR3, DDR4, LPDDR3, QDR II+, and QDR-IV Pin Rules section in Designing with the Core chapters. • Updated system clock period to 10 in Required Constraints section for DDR3, DDR4, and LPDDR3 in Design Flow Steps chapters. • Removed internal VREF description in Required Constraints in DDR3, DDR4, QDR II+, QDR-IV, and RLDRAM3 Design Flow Steps chapters. **DDR3/DDR4** • Added important note and figure on DBI in DDR4 SDRAM section of Overview chapter. • Updated UltraScale Architecture-Based FPGAs DDR3/DDR4 MIS figure in Overview chapter. • Updated DDR3/DDR4 support for RDIMM in Feature Summary section. • Updated Optional ECC support in DDR4 SDRAM section in Overview chapter. • Updated Vivado Customize IP Dialog Box – Clamshell Topology figure in Core Architecture chapter. • Added match_cycle paragraph in Save Restore section in Core Architecture chapter. • Updated description for app_wdf_mask[APP_MASK_WIDTH – 1:0] in User Interface table in Designing with the Core chapter. • Updated Ping Pong PHY Topology in DDR4 figure in Designing with the Core chapter. • Updated Vivado Customize IP Dialog Box for DDR4 – Basic figure in Design Flow Steps chapter. • Added recommendation note in Simulating the Example Design (Designs with Standard User Interface) section in the Example Design chapter. **LPDDR3** • Updated Input Clock Requirement section in Designing with the Core chapter. • Updated vrp pin description in LPDDR3 Pin Rules section in Designing with the Core chapter.
12/22/2017	1.4	• Added Reduce System Noise During Calibration section in all (except RLDRAM 3) Designing with the Core chapters. **DDR3/DDR4** • Updated DCI data rate description (#5) in DDR4 Pin Rules section. • Added DBI description in DM_DBI Parameter section. **RLDRAM 3** • Added Important note in Additional Clocks section in Designing with the Core chapter.

Date	Version	Revision
10/04/2017	1.4	• Added UltraScale+ references in all Maximum Frequencies sections. **DDR3/DDR4** • Updated maximum density support for DDR4. • Updated UltraScale Architecture-Based FPGAs DDR3/DDR4 Memory Interface Solution figure in Overview chapter. • Updated ROW_BANK_COLUMN and ROW_COLUMN_BANK columns in DDR4 4 Gb (x16) Address Mapping table. • Added DDR3 "BANK_ROW_COLUMN" Mapping and Example tables and DDR3 "ROW_BANK_COLUMN" Mapping and Example tables. • Updated signal names in User Interface Ports Description for Save and Restore table. **LPDDR3** • Updated UltraScale Architecture-Based FPGAs LPDDR3 Memory Interface Solution figure in Overview chapter. • Updated cs_n, ck_c, and ck_t in bank 2 for 32-Bit LPDDR3 Interface Contained in Two Banks table. • Added #13 to LPDDR3 Pin Rules section. **QDR II+** • Updated High-Level Block Diagram of QDR II+ Interface Solution figure in Overview chapter. **Traffic Generator** • Updated Number of instruction iteration description in Traffic Generator Instruction Options table.
06/07/2017	1.4	**DDR3/DDR4** • Updated ddr3/4_ecc_single[7:0] and ddr3/4_ecc_multiple[7:0] descriptions in DDR3 ECC Operation Signal Direction Description and DDR4 ECC Operation Signal Direction Description tables. • Updated GUIs for Vivado Customize IP Dialog Box for DDR3 and DDR4 Advanced Options. **Debugging** • Added export to spreadsheet description in Memory IP Debug GUI Usage section in Debugging appendix.

Date	Version	Revision
04/05/2017	1.4	• Added MicroBlaze MCS ECC in all Core Architecture chapters. • Added system reset pin description to all Pin Rules section in Designing with the Core chapter. • Updated Advanced Options figure in Design Flow Steps chapters. • Removed Synplify Pro Black Box sections in all Example Design chapters. • Added LPDDR3 IP section. **DDR3/DDR4** • Updated Notes in DDR3 and DDR4 section in Overview chapter. • Added 3DS component support in DDR4 SDRAM section in Overview chapter. • Updated the Physical Layer bullet in Overview chapter. • Added CRC for write and 2T timing not supported in DDR4 Feature Summary. • Added Read and Write VREF Calibration section in Core Architecture chapter. • Added note in ECC in Core Architecture chapter. • Added SSTL15 in DDR3 Pin Rules section in Designing with the Core chapter. • Added LVCMOS12 in DDR4 Pin Rules section in Designing with the Core chapter. • Added DDR4 4 Gb (x16) app_addr Mapping Options table in Designing with the Core chapter. • Updated C_S_AXI_SUPPORTS_NARROW_BURST description in AXI4 Slave Interface Parameters table. • Updated description in app_en in User Interface table. • Added note in s_axi_awlock and s_axi_arlock rows in AXI4 Slave Interface Signals table. • Updated Example 2 code in SLOT0_FUNC_CS section. • Updated a and b description in Simulating the Performance Traffic Generator section.
		QDR II+ • Updated description in qdriip_qvld in Memory Interface Signals table. **Traffic Generator** • Added important note in Advanced Traffic Generator section.

Date	Version	Revision
11/30/2016	1.3	• Updated Advanced Clocking Tab GUIs in Design Flow Steps chapters. • Updated SIM_MODE description in all Simulation Speed sections. • Added PFD formula in M and D Support for Reference Input Clock Speed sections. **DDR3/DDR4** • Added Memory Settings in Core Architecture section. • Added Note in the Resets section. • Updated SIM_MODE description in PHY Only Parameters table. • Updated code in SLOT0_CONFIG and SLOT0_FUNC_CS sections. • Added PFD formula in M and D Support for Reference Input Clock Speed section. • Updated stimulus memory description in Modules for Performance Traffic Generator table. • Added 3DS part description in Stimulus Pattern section.
		QDR II+ • Added important note in Resets section. • Added Calibration Sequence descriptions in PHY section. **RLDRAM 3** • Added dm description in #2 in RLDRAM 3 Pin Rules section.

Date	Version	Revision
10/05/2016	1.3	• Added density support in all Feature Summary sections. • Added Reset Sequence sections. • Updated Reset sections. • Added M and D Support for Reference Input Clock Speed sections. • Updated all Design Flow Steps and Example Design GUIs. **DDR3/DDR4** • Updated UltraScale Architecture-Based FPGAs DDR3/DDR4 Memory Interface Solution figure. • Updated DDR Bus Efficiency table. • Added SODIMMs and ECC features in Feature Summary. • Updated maintenance block description in Memory Controller section. • Added ECC Port Description section in ECC section. • Updated ECC Block Diagram in ECC Module section. • Updated MicroBlaze in PHY Module table. • Updated Address Parity section. • Added 0x9 to DQS Gate description in Error Signal Descriptions table. • Added description and updated Status Port Bits title in XSDB Status Signal Descriptions table. • Updated END_ADDR0/1 description in Save Restore section. • Added Clamshell Topology and Migration sections. • Updated User Interface table.
		• Added Group to DDR4 ROW_COLUMN_LRANK_BANK and DDR4 ROW_LRANK_COLUMN_BANK tables. • Added Payload width in app_wdf_data[APP_DATA_WIDTH – 1:0] section. • Updated description in Read Priority (RD_PRI_REG) section. • Updated ECC Control Register Map table. • Added Important Note in Pin and Bank Rules section. • Updated bit names in Correctable Error First Failing Address [63:32] Register table.

Date	Version	Revision
Continued		- Updated bit names in Uncorrectable Error First Failing Address [31:0] Register table. - Updated 0x38 to 0x28 Group_FSM column in DDR3/DDR4 4 Gb (x8) app_addr Mapping Options table. - Updated description in VT Tracking. - Updated R_{TT} default value in Vivado IDE Parameter to User Parameter Relationship table. - Updated Test Bench chapter. **QDR II+** - Updated BUFGs and Clock Roots section. **QDR-IV** - Updated BUFGs and Clock Roots section. **RLDRAM 3** - Updated bits in Feature Summary section. - Updated description in Memory Controller section. - Added Important Note in Pin and Bank Rules section and #16 description. **Traffic Generator** - Updated Advanced Traffic Generator section. **Multiple IP Cores** - Added Important note in Sharing of a Bank section. **Debugging** - Added steps in Understanding Calibration Warnings (Cal_warning) section. - Updated Signal Width to 127 in cal_r*_status in DDR3/DDR4 Debug Signals Used in Vivado Design Suite Debug Feature table. - Updated Debugging Data Errors section.

Date	Version	Revision
06/08/2016	1.2	• Added bullet description in all GCIO Requirements sections. • Updated XDC syntax CLOCK_DEDICATED_ROUTE codes in all Designing with the Core chapters. • Added information on clock input specs in all Pin Rules sections. **DDR3/DDR4** • Updated Save Restore section. • Updated XDC syntax CLOCK_DEDICATED_ROUTE codes in Designing with the Core chapter. • Updated description in Round-Robin section. • Added DDR4 x16 Address Map table and Address Map Graph in Performance section. • Removed QoS description in Read Priority (RD_PRI_REG) section. • Updated C_S_AXI_CTRL_ADDR_WIDTH allowable value in AXI4-Lite Slave Control/Status Register Parameters table. **QDR II+** • Added optional internal V_{REF} sample constraint in Required Constraints section. **QDR-IV** • Added optional internal V_{REF} sample constraint in Required Constraints section. **RLDRAM 3** • Updated User Address Width for 576 Mb and 1.125 Gb table.
		Debugging • Added description to WRLVL_ODELAY_STABLE1_RANK*_BYTE* in Expected Results section. • Updated DDR Calibration Times table.

Date	Version	Revision
04/14/2016	1.2	• Fixed QDR-IV link. • Added QDR-IV, 3DS, and LRDIMM support. • Added (four DQ per DQS per x4 devices) description in DDR3/DDR4 Feature Summary section. • Added important notes relating to specific core versions to all Overview sections. • Added references to all Maximum Frequencies section. • Updated Synplify Pro Black Box sections in all Example Design chapters. • Added Recommended Pinout Configurations appendix. • Updated all Resets section. **DDR3/DDR4** • Updated Fig. 3-6: PHY Overall Initialization and Calibration Sequence. • Added Read DBI Per-bit Deskew section. • Added Read DQS Centering (DBI) section, • Added DDR4 LRDIMM Memory Initialization and Calibration Sequence and Save Restore sections in Core Architecture chapter. • Updated DDR3 and DDR4 Pin Rules sections. • Updated DM_DBI description in PHY Only table. • Updated Command and Address and Write Data tables. • Added Setting Timing Options section. • Updated Fig 4-9: User Mode Ports on DRAM Command Bus Timing Diagram. • Updated Design Flow Steps chapter. • Updated Ping Pong Overview section. • Updated AXI4 Slave Interface Parameters table.Updated AXI4 Slave Interface Signals table. • Added description in AXI4-Lite Slave Control/Status Register Interface Block section. • Updated AXI4-Lite Slave Control/Status Register Parameters table. • Updated List of New I/O Signals table. • Updated PHY Only Parameters table. • Updated PHY section with content moved to Debugging section. • Added Ping Pong PHY section in Designing with the Core chapter. • Updated Design Flow Steps chapter.

Date	Version	Revision
Continued		**QDR II+** • Added Pin Swapping section in Designing with the Core chapter. • Updated Design Flow Steps chapter. **QDRIV** • Added new section. **RLDRAM 3** • Updated Core Architecture chapter. • Added Pin Swapping section in Designing with the Core chapter. • Updated User Address Bit Allocation Based on RLDRAM 3 Configuration. • Updated and added sys_clk_p/n and sys_rst to Table 25-2: User Interface Request Signals. • Added rld_qvld signal in Table 25-5: Physical Interface Signals table. • Updated Design Flow Steps chapter. • Updated Example Design section. • Updated 100 writes and reads description in Test Bench chapter. **Traffic Generator** • Updated vio_tg_err_clear and vio_tg_err_clear_all descriptions in Default Traffic Generator Control Connection table. Added new paragraph in Traffic Error Detection section. **Debugging** • Updated General Checks section. • Updated Table 31-1: Memory IP Configuration XSDB Parameters. • Updated Table 31-8: DDR3/DDR4 DDR Warning Code Decoding. • Updated Debugging DDR3/DDR4 Designs section with content moved to PHY section. • Updated Table 31-9: DDR_CAL_ERROR Decode for DQS Preamble Detection Calibration. • Updated Table 31-21: DDR_CAL_ERROR Decode for Write DQS Centering Calibration. • Added Debugging Read Per-Bit DBI Deskew Failures section. • Updated Table 31-46: Sanity Check Data Patterns.

Date	Version	Revision
11/18/2015	1.1	• Added support for UltraScale+ families. • Added Input Clock Period Jitter in Requirements section. • Updated Resource Utilization sections. • Updated Customizing and Generating the Core figures. • Updated User Parameters section. **DDR3/DDR4** • Added Efficiency and Latency Measurements in Performance section. • Added Address Parity section in Core Architecture. • Added Important note on DFI-compliant in PHY section. • Updated description in Bus Utilization section. • Added ROW_COLUMN_BANK_INTLV description in app_addr[ADDR_WIDTH – 1:0] section. • Added description in app_autoprecharge section. • Added description in Address Map section. • Added description in Autoprecharge section. • Updated Important note in Basic Tab section. **QDR II+** • Updated PHY Overall Initialization and Calibration Sequence figure. • Updated Read Clock (CQ/CQ#) Allocation section. **Debugging** • Added Note in XSDB Debug section. • Updated Data Mask trace impedance. • Added 0x2 and 0x3 descriptions in DDR_CAL_ERROR Decode for Write DQS Centering Calibration table.

Date	Version	Revision
09/30/2015	1.0	• Reset core version to v1.0. • Updated BUFGCE_DIV to BUFG. • Added description to all TXPLL sections. • Added Reference Input Clock Period option in all Sharing of Input Clock Source (sys_clk_p) sections. • Added TXPLL Usage and Additional Clocks sections to all interfaces. • Updated Customizing and Generating the Core figures. • Added note in all Non-Project-Based Simulation sections. **DDR3/DDR4** • Added DDR3L (1.35V), dual slot support and quad-rank support in Feature Summary. • Updated PHY Block Diagram and table. • Updated Fig. 3-6: PHY Overview Initialization and Cal. Seq. • Updated Error Signals Descriptions table. • Updated Read and Write V_{REF} Calibration sections. • Added description to app_wdf_data[APP_DATA_WIDTH – 1:0] and app_wdf_mask[APP_MASK_WIDTH – 1:0] sections. • Added Pin Swapping section. • Updated Pin and Bank Rules section. • Added description in DQS Gate section. • Updated descriptions for SLOT0_CONFIG, SLOT1_CONFIG, and SLOT0_FUNC_CS in PHY Only Parameters table. • Added DIMM Configurations section. • Added description in Basic Tab section. • Updated figures in Customizing and Generating the Core section. • Updated Simulating the Performance Traffic Generator section. **QDR II+** • Added description in Customizing and Generating the Core section.
		RLDRAM 3 • Added interface widths to Supported Configurations table. • Updated signal for user_rd_valid[CMD_PER_CLK – 1:0] in User Interface Request Signals table. • Added RLDRAM 3 Address Width table. • Updated PHY Overall Initialization and Calibration Sequence. • Added description in Customizing and Generating the Core section. • Updated description in Required Constraints section. **Traffic Generator** • Updated chapter. **Multiple IP** • Updated MMCM Constraints section. **Debugging** • Updated Hardware Debug section.

Date	Version	Revision
06/24/2015	7.1	• Updated all Resource Utilization sections. • Added clocking reference in all Requirements sections. • Updated description in all Resets section. • Updated all Clocking sections. • Updated all CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation sections. **DDR3/DDR4** • Added x4 devices are not supported in AXI note in Feature Summary section. • Updated Fig. 3-6: PHY Overall Initialization and Calibration Sequence. • Added Table 3-4: Pre-Calibration XSDB Status Signal Description. • Updated Table 3-5: XSDB Status Signal Description • Added Table 3-6: Post-Calibration XSDB Status Signal Description. • Updated Read per-bit Deskew description in Table 3-6: Error Signal Descriptions. • Updated description in Write DQS-to-DQ Centering section. • Added Read DQS Centering (Complex) and Write DQS-to-DQ Centering (Complex) sections. • Added Notes to Write DQS-to-DQ, Write DQS-to-DM, Write DQS-to-DQ Centering (Complex), Read V_{REF}, and Read DQS Centering (Complex). • Added Read V_{REF} and Write V_{REF} Calibrations section. • Updated letter b and c descriptions in DDR3 Pin Rules section. • Updated AXI4-Lite Slave Control/Status Register Map Detailed Descriptions. • Added description in Project-Based Simulation Flow Using Vivado Simulator section.
		QDR II+ • Added HSTL_I I/O standard support in Feature Summary. • Added description to the Memory Initialization bullet in Overview section. **RLDRAM 3** • Updated description in Required Constraints section. • Updated Fig. 17-4: PHY Overall Initialization and Calibration Sequence. • Updated description d. in RLDRAM 3 Pin Rules. **Traffic Generator** • Updated Advanced Traffic Generator section. **Debugging Appendix** • Added AR: 60305 in General Checks section.

Date	Version	Revision
04/01/2015	7.0	- Updated Supported User Interface and added #3 footnote in IP Facts table. - Updated Application Interface description in the Overview chapter. - Updated descriptions and added BACKBONE description in all Clocking sections. - Added sys_rst and dbg_clk references throughout book. - Added Simulation Flow and Simulation Speed to all sections. - Added Project-Based Simulation Flow Using Vivado Simulator to all sections. - Added CLOCK_DEDICATED_ROUTE Constraints and BUFG Instantiation to all sections. **DDR3/DDR4** - Updated Fig. 1-1: UltraScale Architecture-Based FPGAs Memory Interface Solution. - Updated Feature Summary section. - Updated Memory Controller section. - Updated Group Machines section. - Updated DQS section. - Updated parameters in Write Leveling section. - Updated and added Important note in Read DQS Centering section. - Updated Read Leveling Multi-Rank Adjustment, Multi-Rank Adjustments and Checks, and added Write Latency Multi-Rank Check. - Updated Write Per-bit Deskew section. - Updated Write DQS-to-DM section. - Updated Table 3-5: Error Signal Descriptions. - Updated Table 3-6: Examples of DQS Gate Multi-Rank Adjustment (2 Ranks). - Updated DDR3 and DDR4 Pin Rules sections.
		- Added Pin Mapping for x4 RDIMMs. - Added app_ref_req, app_ref_ack, app_zq_req, and app_zq_ack in Table 4-7: User Interface. - Updated Write Path section. - Added Performance section. - Added descriptions for app_ref_req, app_ref_ack, app_zq_req, and app_zq_ack. - Added Maintenance Commands section. - Updated Table 4-16: AXI4 Slave Interface Parameters. - Added dbg_clk to Table 4-17: AXI4 Slave Interface Signals. - Updated Time Division Multiplexing (TDM), Round-Robin, and Read Priority (RD_PRI_REG) sections.

Date	Version	Revision
Continued		• Updated Table 4-76: Read Data to Table 4-77: PHY Only Parameters. • Updated to 11 writes in Multiple Writes and Reads with Same Address to Page Wrap During Writes sections. • Added Minimum Write CAS Command Spacing and System Considerations for CAS Command Spacing sections. • Updated the Design Flow Steps chapter. **QDR II+** • Updated Feature Summary. **RLDRAM 3** • Added User Interface Allocation section. • Added User Address Bit Allocation Based on RLDRAM 3 Configuration section. • Added description to Interfacing with the Core through the User Interface section. **Traffic Generator** • Added Traffic Generator section. **Multiple IP** • Added Multiple IP section. **Migrating and Upgrading Appendix** • Added link to UG973 and description in Migrating and Upgrading chapter. **Debugging Appendix** • Added description in General Checks section.

Date	Version	Revision
11/19/2014	6.1	**QDR II+** • Added interface calibration in Feature Summary section. • Updated description #2 in Sharing of Input Clock Source (sys_clk_p) section. • Added read data pins description and cross-ref to system clock pins description in QDR II+ Pin Rules section. • Added vrp description in QDR II+ Pin Rules section. • Updated User Parameters table. Updated GUIs in Example Design chapter. **DDR3/DDR4** • Updated Fig. 1-1: UltraScale Architecture-Based FPGAs Memory Interface Solution. • Added interface calibration in Feature Summary section. • Updated RIU code in Overall PHY Architecture section. • Updated description #2 in Sharing of Input Clock Source (sys_clk_p) section. • Added ECC description in Datapath section and ECC section. • Updated resetn, input clock description, and added x4 Part Contained in One Bank tables in DDR3 and DDR4 Pin Rules sections. • Added app_raw_not_ecc in Table 4-5: User Interface. • Updated descriptions in app_cmd[2:0] section. • Updated Fig. 4-2 and Fig. 4-6 to Fig. 4-8. • Added examples for DRAM clock in Write Path section. • Added PHY Only section in Protocol Description. • Updated R_{TT} (nominal)-ODT default values in Table 5:1: Vivado IDE Parameter to User Parameter Relationship. • Updated GUIs in Customizing and Generating the Core section. • Updated User Parameters table. • Updated GUIs in Example Design chapter.
		RLDRAM 3 • Added interface calibration in Feature Summary section. • Updated Table 15-1: Supported Configurations and removed support for Read Latency in Feature Summary. • Added CMD_PER_CLK description in Memory Controller section. • Updated description #2 in Sharing of Input Clock Source (sys_clk_p) section. • Updated input clock description in RLDRAM 3 Pin Rules section. • Added note in Interfacing with the Core through the User Interface section. • Updated Fig. 18-2: Multiple Commands for user_cmd Signal. • Updated User Parameters table. • Updated GUIs in Example Design chapter. • Updated description in Simulating the Example Design (Designs with Standard User Interface) section.

Date	Version	Revision
10/01/2014	6.0	**DDR3/DDR4** • Updated Standards section. • Updated Feature Summary section. • Updated description in Memory Initialization and Calibration Sequence section. • Updated Overall PHY Architecture section. • Updated Fig. 3-4: PHY Overall Initialization and Calibration Sequence. • Added new calibration status descriptions in Memory Initialization and Calibration Sequence section. • Added DQS Gate, Write Leveling, Read Leveling, Read Sanity Check, Write DQS-to-DQ, Write Latency Calibration, Write/Read Sanity Check, Write DQS-to-DM, and Multi-Rank Adjustment sections. • Updated DDR3/DDR4 Pin Rules section. • Added AXI4 Slave Interface in Protocol Description section. • Added Multiple IP Cores and Sharing of Input Clock Source in Clocking section. • Removed Special Designation column in Table 4-1: 16-Bit Interface Contained in One Bank and Table 4-2: 32-Bit Interface Contained in Two Banks. • Added app_autoprecharge to Table 4-3: User Interface. • Added app_autoprecharge section. • Updated app_rdy section. • Updated ref_req and zq_req sections. • Updated Table 5-1: Vivado IDE Parameter to User Parameter Relationship. • Updated note description in Required Constraints section. • Updated description in Simulation section. • Updated GUIs in Example Design chapter.

Date	Version	Revision
Continued		**QDR II+** • Updated Feature Summary section. • Updated Table 9-1: Device Utilization – Kintex UltraScale FPGAs. • Updated Fig. 10-3: PHY Overall Initialization and Calibration Sequence. • Updated MicroBlaze description in Overall PHY Architecture section. • Updated Memory Initialization and Calibration Sequence section. • Updated Resets section. • Deleted Special Designation column in Table 11-1: 18-Bit QDR II+ Interface Contained in Two Banks. • Added Multiple IP Cores and Sharing of Input Clock Source in Clocking section. • Updated Protocol Description section. • Updated Simulation section. • Updated description in Simulating the Example Design (Designs with Standard User Interface) section. • Updated GUIs in Example Design chapter. **RLDRAM 3** • Added Configuration table in Feature Summary section. • Updated Memory Initialization bullet in Overview chapter. • Added description to burst support in Feature Summary section. • Updated Table 16-1: Device Utilization – Kintex UltraScale FPGAs. • Updated Memory Controller section. • Updated Overall PHY Architecture section. • Updated Memory Initialization and Calibration Sequence section. • Added Multiple IP Cores and Sharing of Input Clock Source in Clocking section. • Added data mask description to RLDRAM 3 Pin Rules section. • Updated GUIs in Example Design chapter. **Appendix** Added Migrating Appendix.

Date	Version	Revision
06/04/2014	5.0	• Removed PCB sections and added link to UG583. • Global replace BUFGCE to BUFGCE_DIV. **DDR3/DDR4** • Updated CAS cycle description in DDR3 Feature Summary. • Updated descriptions in Native Interface section. • Updated Control Path section. • Updated Read and Write Coalescing section. • Updated Reordering section. • Updated DDR4 x16 parts in Group Machines section. • Updated Fig. 3-3: PHY Block Diagram. • Updated Table 3-1: PHY Modules. • Updated module names in Overall PHY Architecture section. • Updated Fig. 3-4: PHY Overall Initialization and Calibration Sequence. • Added description to Memory Initialization and Calibration Sequence section. • Added SSI rule in Clocking section. • Added SSI rule and updated Address and ck descriptions in DDR3/DDR4 Pin Rules sections. • Added Important Note for calibration stage in DDR3/DDR4 Pinout Examples sections. • Updated signal descriptions in Table 4-3: User Interface. • Added new content in app_addr[ADDR_WIDTH – 1:0] section. • Updated Write Path section. • Updated Native Interface section. • Added Important Note relating to Data Mask in Controller Options section. • Added PHY Only section. • Updated Fig. 5-1 to 5-8 in Customizing and Generating the Core section. • Added User Parameters section in Design Flow Steps chapter. • Updated I/O Standard and Placement section. • Added Synplify Black Box Testing section in Example Design chapter.

Date	Version	Revision
Continued		**QDR II+** • Updated Read Latency in Feature Summary section. • Updated Fig. 10-2: PHY Block Diagram and Table 17-1: PHY Modules. • Updated Table 11-2: User Interface. • Added SSI rule in Clocking section. • Added Important Note for calibration stage in QDR II+ Pinout Examples section. • Added SSI rule in QDR II+ Pin Rules section. • Updated I/O Standard and Placement section. • Added User Parameters section in Design Flow Steps chapter. • Updated the descriptions in Simulating the Example Design (Designs with Standard User Interface) section. • Added Synplify Black Box Testing section in Example Design chapter. **RLDRAM 3** • Added 18 bits in Feature Summary section. • Updated Fig. 17-4: PHY Block Diagram. • Updated module names in Table 17-1: PHY Modules. • Updated module names in Overall PHY Architecture section. • Added SSI rule in Clocking section. • Updated c) and d) descriptions and added SSI rule in RLDRAM 3 Pin Rules section. • Updated Table 18-2: User Interface Request Signals. • Updated Fig. 18-2: Multiple Commands for user_cmd Signal. • Added Important Note for calibration stage in RLDRAM 3 Pinout Examples section. • Updated I/O Standard and Placement section. • Added User Parameters section in Design Flow Steps chapter. • Updated Test Bench chapter.

Date	Version	Revision
04/02/2014	5.0	• Added Verilog Test Bench in IP Facts table. **DDR3/DDR4** • Added Overview chapter. • Updated component support to 80 bits in Feature Summary section. • Updated DDR Device Utilization tables. • Updated DDR Clocking section. • Updated x4 DRAM to Four Component DRAM Configuration in Designing with the Core chapter. • Updated Important note in PCB Guidelines for DDR3 and DDR4 Overview sections. • Updated Important note in Reference Stack-Up for DDR3 and DDR4 sections. • Updated trace length descriptions in DDR3 and DDR4 sections. • Added V_{TT} Terminations guideline in Generic Routing Guideline for DDR3 and DDR4 sections. • Removed Limitations section. • Added V_{REF} note in Required Constraints section. • Updated new figures in Design Flow Steps chapter. • Added new descriptions in Example Design chapter. • Added new description in Test Bench chapter. **QDR II+ SRAM** • Added new QDR II+ section. **RLDRAM 3** • Added Overview chapter. • Added new Clocking section. • Added new descriptions in Example Design chapter. **Appendix** • Updated Debug Appendix.
12/18/2013	4.2	Initial Xilinx release.

Please Read: Important Legal Notices

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS “XA” IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE (“SAFETY APPLICATION”) UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD (“SAFETY DESIGN”). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2013–2022 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.