



# **External Memory Interface Handbook**

---

## **Volume 3: Reference Material**



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)

EMI\_RM-3.0

Document last updated for Altera Complete Design Suite version:

Document publication date:

12.1

November 2012

© 2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



<b>Chapter Revision Dates .....</b>	<b>xi</b>
-------------------------------------	-----------

## Section I. Functional Descriptions

### Chapter 1. Functional Description—UniPHY

Block Description .....	1-1
I/O Pads .....	1-2
Reset and Clock Generation .....	1-2
Dedicated Clock Networks .....	1-3
Address and Command Datapath .....	1-3
Write Datapath .....	1-4
Leveling Circuitry .....	1-5
Read Datapath .....	1-7
Sequencer .....	1-8
Nios II-Based Sequencer .....	1-8
RTL-based Sequencer .....	1-13
Shadow Registers .....	1-14
Description .....	1-14
Operation .....	1-15
DLL Offset Control Block .....	1-16
Interfaces .....	1-16
AFI .....	1-17
The Memory Interface .....	1-17
The DLL and PLL Sharing Interface .....	1-18
About PLL Simulation .....	1-19
The OCT Sharing Interface .....	1-20
UniPHY Signals .....	1-21
PHY-to-Controller Interfaces .....	1-24
Using a Custom Controller .....	1-28
Using a Vendor-Specific Memory Model .....	1-29
AFI 3.0 Specification .....	1-29
Implementation .....	1-29
Bus Width and AFI Ratio .....	1-29
AFI Parameters .....	1-30
Parameters Affecting Bus Width .....	1-30
AFI Signals .....	1-32
Clock and Reset Signals .....	1-32
Address and Command Signals .....	1-32
Write Data Signals .....	1-34
Read Data Signals .....	1-35
Calibration Status Signals .....	1-35
Tracking Management Signals .....	1-36
Register Maps .....	1-37
UniPHY Register Map .....	1-37
Controller Register Map .....	1-39
Ping Pong PHY .....	1-39
Feature Description .....	1-39
Architecture .....	1-40

---

Ping Pong Gasket .....	1-41
Calibration .....	1-42
Operation .....	1-42
Efficiency Monitor and Protocol Checker .....	1-42
Efficiency Monitor .....	1-42
Protocol Checker .....	1-43
Read Latency Counter .....	1-43
Using the Efficiency Monitor and Protocol Checker .....	1-43
Avalon CSR Slave and JTAG Memory Map .....	1-43
UniPHY Calibration Stages .....	1-45
Overview .....	1-45
Calibration Stages .....	1-46
Assumptions .....	1-46
Memory Initialization .....	1-47
Stage 1: Read Calibration Part One—DQS Enable Calibration and DQ/DQS Centering .....	1-47
Guaranteed Write .....	1-48
DQS Enable Calibration .....	1-49
Centering DQ/DQS .....	1-50
Stage 2: Write Calibration Part One .....	1-51
Stage 3: Write Calibration Part Two—DQ/DQS Centering .....	1-53
Stage 4: Read Calibration Part Two—Read Latency Minimization .....	1-53
Read Latency Tuning .....	1-53
Calibration Signals .....	1-53
Calibration Time .....	1-54
Document Revision History .....	1-55

## Chapter 2. Functional Description—ALTMEMPHY

Block Description .....	2-2
Calibration .....	2-3
Address and Command Datapath .....	2-3
Arria II GX Devices .....	2-3
Clock and Reset Management .....	2-5
Clock Management .....	2-5
Reset Management .....	2-7
Read Datapath .....	2-8
Arria II GX Devices .....	2-8
ALTMEMPHY Signals .....	2-10
PHY-to-Controller Interfaces .....	2-16
Using a Custom Controller .....	2-23
Preliminary Steps .....	2-23
Design Considerations .....	2-23
Clocks and Resets .....	2-23
Calibration Process Requirements .....	2-24
Other Local Interface Requirements .....	2-24
Address and Command Interfacing .....	2-24
Handshake Mechanism Between Read Commands and Read Data .....	2-24
Handshake Mechanism Between Write Commands and Write Data .....	2-25
Partial Writes .....	2-26
Controller Register Map .....	2-26
ALTMEMPHY Calibration Stages .....	2-27
Enter Calibration (s_reset) .....	2-29
Initialize PHY (s_phy_initialize) .....	2-29
Initialize DRAM .....	2-29
Initialize DRAM Power Up Sequence (s_int_dram) .....	2-29

Program Mode Registers for Calibration (s_prog_mr) .....	2-29
Write Header Information in the internal RAM (s_write_ihi) .....	2-30
Load Training Patterns .....	2-30
Write Block Training Pattern (s_write_btp) .....	2-30
Write More Training Patterns (s_write_mtp) .....	2-31
Test More Pattern Writes .....	2-31
Calibrate Read Resynchronization Phase .....	2-33
Initialize Read Resynchronisation Phase Calibration (s_rrp_reset) .....	2-34
Calibrate Read Resynchronization Phase (s_rrp_sweep) .....	2-34
Calculate Read Resynchronization Phase (s_rrp_seek) .....	2-34
Calculate Read Data Valid Window (s_rdv) .....	2-34
Advertise Write Latency (s_was) .....	2-35
Calculate Read Latency (s_adv_rlat) .....	2-35
Output Write Latency (s_adv_wlat) .....	2-36
Calibrate Postamble (s_poa) .....	2-36
Set Up Address and Command Clock Cycle .....	2-37
Write User Mode Register Settings (s_prep_customer_mr_setup) .....	2-37
Voltage and Temperature Tracking .....	2-37
Setup the Mimic Window (s_tracking_setup) .....	2-38
Perform Tracking (s_tracking) .....	2-38
Document Revision History .....	2-38

### Chapter 3. Functional Description—Hard Memory Interface

Hard Memory Interface .....	3-1
High-Level Feature Description .....	3-1
Multi-Port Front End (MPFE) .....	3-2
Fabric Interface .....	3-2
Operation Ordering .....	3-3
Multi-port Scheduling .....	3-3
Port Scheduling .....	3-3
DRAM Burst Scheduling .....	3-4
DRAM Power Saving Modes .....	3-5
MPFE Signal Descriptions .....	3-5
Hard Memory Controller .....	3-7
Clocking .....	3-7
DRAM Interface .....	3-8
ECC .....	3-8
Controller ECC .....	3-8
Bonding of Memory Controllers .....	3-8
Data Return Bonding .....	3-9
FIFO Ready .....	3-9
Bonding Latency Impact .....	3-9
Bonding Controller Usage .....	3-9
Hard PHY .....	3-10
Interconnections .....	3-10
Clock Domains .....	3-10
Hard Sequencer .....	3-11
Hard Memory Interface Implementation Guidelines .....	3-11
MPFE Setup Guidelines .....	3-11
Soft Memory Interface to Hard Memory Interface Migration Guidelines .....	3-12
Pin Connections .....	3-12
Software Interface Preparation .....	3-12
Latency .....	3-13
Bonding Interface Guidelines .....	3-13

---

Document Revision History .....	3-14
<b>Chapter 4. Functional Description—HPS Memory Controller</b>	
Features of the SDRAM Controller Subsystem .....	4-1
SDRAM Controller Subsystem Block Diagram and System Integration .....	4-2
SDRAM Controller .....	4-2
DDR PHY .....	4-2
SDRAM Controller Subsystem Interfaces .....	4-3
MPU Subsystem Interface .....	4-3
L3 Interconnect Interface .....	4-3
CSR Interface .....	4-3
FPGA-to-HPS SDRAM Interface .....	4-3
Memory Controller Architecture .....	4-4
MPFE .....	4-5
Command Block .....	4-5
Write Data Block .....	4-6
Read Data Block .....	4-6
Single-Port Controller .....	4-6
Command Generator .....	4-6
Timer Bank Pool .....	4-6
Arbiter .....	4-7
Rank Timer .....	4-7
Write Data Buffer .....	4-7
ECC Block .....	4-7
AFI Interface .....	4-7
CSR Interface .....	4-7
Functional Description of the SDRAM Controller Subsystem .....	4-7
MPFE Operational Behavior .....	4-7
Operation Ordering .....	4-7
Multiport Scheduling .....	4-8
SDRAM Burst Scheduling .....	4-9
Clocking .....	4-10
Single-Port Controller Operational Behavior .....	4-10
SDRAM Interface .....	4-10
ECC .....	4-11
Interleaving Options .....	4-12
AXI-Exclusive Support .....	4-14
Memory Protection .....	4-14
SDRAM Power Management .....	4-16
DDR PHY .....	4-17
Clocks .....	4-17
Resets .....	4-18
Initialization .....	4-18
Protocol Details .....	4-18
SDRAM Controller Subsystem Programming Model .....	4-21
Initialization .....	4-21
Timing Parameters .....	4-22
SDRAM Controller Address Map and Register Definitions .....	4-22
Using EMI-Related HPS Features in SoC Devices .....	4-22
Architecture .....	4-22
Configuration .....	4-22
Simulation .....	4-23
Document Revision History .....	4-24

## Chapter 5. Functional Description—HPC II Controller

Memory Controller Architecture .....	5-1
Avalon-ST Input Interface .....	5-2
AXI to Avalon-ST Converter .....	5-2
Handshaking .....	5-3
Command Channel Implementation .....	5-3
Data Ordering .....	5-3
Burst Types .....	5-3
Backpressure Support .....	5-4
Command Generator .....	5-4
Timing Bank Pool .....	5-4
Arbiter .....	5-4
Arbitration Rules .....	5-4
Rank Timer .....	5-5
Read Data Buffer .....	5-5
Write Data Buffer .....	5-5
ECC Block .....	5-5
AFI Interface .....	5-5
CSR Interface .....	5-5
Controller Features Descriptions .....	5-5
Data Reordering .....	5-5
Pre-emptive Bank Management .....	5-6
Quasi-1T and Quasi-2T .....	5-6
User Autoprecharge Commands .....	5-6
Half-Rate Bridge .....	5-6
Address and Command Decoding Logic .....	5-7
Low-Power Logic .....	5-7
User-Controlled Self-Refresh .....	5-7
Automatic Power-Down with Programmable Time-Out .....	5-8
ODT Generation Logic .....	5-8
DDR2 SDRAM .....	5-8
DDR3 SDRAM .....	5-9
Burst Merging .....	5-10
ECC .....	5-10
Partial Writes .....	5-11
Partial Bursts .....	5-12
External Interfaces .....	5-13
Clock and Reset Interface .....	5-13
Avalon-ST Data Slave Interface .....	5-13
AXI Data Slave Interface .....	5-13
Enabling the AXI Interface .....	5-13
Controller-PHY Interface .....	5-20
Memory Side-Band Signals .....	5-20
Self-Refresh (Low Power) Interface .....	5-20
User-Controlled Refresh Interface .....	5-21
Configuration and Status Register (CSR) Interface .....	5-21
Top-Level Signals Description .....	5-22
Controller Register Map .....	5-29
Sequence of Operations .....	5-33
Write Command .....	5-33
Read Command .....	5-33
Read-Modify-Write Command .....	5-34
Document Revision History .....	5-34

---

## **Chapter 6. Functional Description—QDR II Controller**

Block Description .....	6-1
Avalon-MM Slave Read and Write Interfaces .....	6-1
Command Issuing FSM .....	6-2
AFI .....	6-2
Avalon-MM and Memory Data Width .....	6-2
Signal Description .....	6-2
Avalon-MM Slave Read Interface .....	6-3
Avalon-MM Slave Write Interface .....	6-3
Document Revision History .....	6-4

## **Chapter 7. Functional Description—RLDRAM II Controller**

Block Description .....	7-1
Avalon-MM Slave Interface .....	7-1
Write Data FIFO Buffer .....	7-2
Command Issuing FSM .....	7-2
Refresh Timer .....	7-2
Timer Module .....	7-2
AFI .....	7-2
User-Controlled Features .....	7-2
Error Detection Parity .....	7-2
User-Controlled Refresh .....	7-3
Avalon-MM and Memory Data Width .....	7-3
Signal Description .....	7-3
Avalon-MM Slave Interface .....	7-3
Document Revision History .....	7-4

## **Chapter 8. Functional Description—RLDRAM 3 PHY-Only IP**

Block Description .....	8-1
Features .....	8-1
RLDRAM III AFI Protocol .....	8-2
Document Revision History .....	8-3

## **Chapter 9. Functional Description—Example Designs**

Synthesis Example Design .....	9-1
Simulation Example Design .....	9-3
Traffic Generator and BIST Engine .....	9-5
Read and Write Generation .....	9-6
Individual Read and Write Generation .....	9-6
Block Read and Write Generation .....	9-6
Address and Burst Length Generation .....	9-6
Sequential Addressing .....	9-6
Random Addressing .....	9-6
Sequential and Random Interleaved Addressing .....	9-7
Traffic Generator Signals .....	9-7
Traffic Generator Add-Ons .....	9-7
User Refresh Generator .....	9-7
Traffic Generator Timeout Counter .....	9-8
Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys .....	9-8
Creating the Qsys System .....	9-8
Notes on Configuring UniPHY IP in Qsys .....	9-9
Document Revision History .....	9-10

## Section II. UniPHY Reference

### Chapter 10. Introduction to UniPHY IP

Release Information .....	10-2
Device Family Support .....	10-2
Features .....	10-3
Unsupported Features .....	10-5
System Requirements .....	10-5
MegaCore Verification .....	10-5
Resource Utilization .....	10-5
DDR2, DDR3, and LPDDR2 SDRAM Controllers with UniPHY .....	10-5
QDR II and QDR II+ SRAM Controllers with UniPHY .....	10-12
RLDRAM II Controller with UniPHY .....	10-13
Document Revision History .....	10-14

### Chapter 11. Latency for UniPHY IP

DDR2, DDR3, and LPDDR2 .....	11-2
QDR II and QDR II+ .....	11-3
RLDRAM II .....	11-4
RLDRAM 3 .....	11-4
Variable Controller Latency .....	11-4
Document Revision History .....	11-5

### Chapter 12. Timing Diagrams for UniPHY IP

DDR2 and DDR3 Timing Diagrams .....	12-1
QDR II and QDR II+ Timing Diagrams .....	12-10
RLDRAM II Timing Diagrams .....	12-15
LPDDR2 Timing Diagrams .....	12-19
RLDRAM 3 Timing Diagrams .....	12-23
Document Revision History .....	12-27

### Chapter 13. UniPHY External Memory Interface Debug Toolkit

Introduction .....	13-1
Architecture .....	13-1
Communication .....	13-1
Calibration and Report Generation .....	13-2
Setup and Use .....	13-2
General Workflow .....	13-3
Linking the Project to a Device .....	13-3
Establishing Communication to Connections .....	13-4
Reports .....	13-4
Summary Report .....	13-4
Calibration Report .....	13-5
Margin Report .....	13-5
Operational Considerations .....	13-5
Specifying a Particular JDI File .....	13-6
PLL Status .....	13-6
Margining Reports .....	13-6
Group Masks .....	13-6
Troubleshooting .....	13-6
EMIF On-Chip Debug Toolkit .....	13-8
Introduction .....	13-8
Access Protocol .....	13-8

---

Command Codes Reference .....	13–10
Header Files .....	13–10
Generating UniPHY IP With the Debug Port .....	13–10
Example C Code for Accessing Debug Data .....	13–11
Document Revision History .....	13–14

## **Chapter 14. Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers**

Upgrading from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY Designs 14–1	
Generating Equivalent Design .....	14–1
Replacing the ALTMEMPHY Datapath with UniPHY Datapath .....	14–2
Resolving Port Name Differences .....	14–2
Creating OCT Signals .....	14–4
Running Pin Assignments Script .....	14–4
Removing Obsolete Files .....	14–4
Simulating your Design .....	14–4
Document Revision History .....	14–6

## **Section III. ALTMEMPHY Reference**

### **Chapter 15. Introduction to ALTMEMPHY IP**

Release Information .....	15–2
Device Family Support .....	15–3
Features .....	15–4
ALTMEMPHY Megafunction .....	15–4
High-Performance Controller II .....	15–4
Unsupported Features .....	15–6
MegaCore Verification .....	15–6
Resource Utilization .....	15–6
System Requirements .....	15–8
Installation and Licensing .....	15–9
Free Evaluation .....	15–9
OpenCore Plus Time-Out Behavior .....	15–10
Document Revision History .....	15–10

### **Chapter 16. Latency for ALTMEMPHY IP**

Latency Stages .....	16–2
Document Revision History .....	16–5

### **Chapter 17. Timing Diagrams for ALTMEMPHY IP**

DDR and DDR2 High-Performance Controllers II .....	17–1
Half-Rate Read .....	17–2
Half-Rate Write .....	17–4
Full-Rate Read .....	17–6
Full-Rate Write .....	17–8
DDR3 High-Performance Controller II .....	17–9
Half-Rate Read (Burst-Aligned Address) .....	17–10
Half-Rate Write (Burst-Aligned Address) .....	17–12
Half-Rate Read (Non Burst-Aligned Address) .....	17–14
Half-Rate Write (Non Burst-Aligned Address) .....	17–16
Half-Rate Read With Gaps .....	17–18
Half-Rate Write With Gaps .....	17–19
Half-Rate Write Operation (Merging Writes) .....	17–20

Write-Read-Write-Read Operation .....	17–22
Document Revision History .....	17–24

## Chapter 18. ALTMEMPHY External Memory Interface Debug Toolkit

Debug Toolkit Overview .....	18–1
Install the Debug Toolkit .....	18–2
Modify the Example Top-Level File to use the Debug Toolkit .....	18–2
Verify the Design .....	18–3
Regenerate the IP .....	18–4
Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project .....	18–4
Add Additional Signals .....	18–5
Add alt_jtagavalon.v to your Quartus II Project Settings Files List .....	18–7
Recompile your Quartus II Test Design .....	18–7
Program Hardware with Debug Enabled .sof .....	18–7
Use the Debug Toolkit .....	18–8
Interpret the Results .....	18–9
Calibration Successful .....	18–9
Calibration Fails .....	18–13
Save the Calibration Results .....	18–13
Understand the Checksum and Failure Code .....	18–15
Document Revision History .....	18–16



The chapters in this document, External Memory Interface Handbook, Volume 3: Reference Material, were revised on the following dates. Where chapters or groups of chapters are available separately, part numbers are listed.

- Chapter 1. Functional Description—UniPHY  
Revised: November 2012  
Part Number: *EMI\_RM\_001-3.1*
- Chapter 2. Functional Description—ALTMEMPHY  
Revised: November 2012  
Part Number: *EMI\_RM\_002-3.3*
- Chapter 3. Functional Description—Hard Memory Interface  
Revised: November 2012  
Part Number: *EMI\_RM\_003-2.1*
- Chapter 4. Functional Description—HPS Memory Controller  
Revised: November 2012  
Part Number: *EMI\_RM\_017-1.0*
- Chapter 5. Functional Description—HPC II Controller  
Revised: November 2012  
Part Number: *EMI\_RM\_004-2.1*
- Chapter 6. Functional Description—QDR II Controller  
Revised: November 2012  
Part Number: *EMI\_RM\_005-3.3*
- Chapter 7. Functional Description—RLDRAM II Controller  
Revised: November 2012  
Part Number: *EMI\_RM\_006-3.3*
- Chapter 8. Functional Description—RLDRAM 3 PHY-Only IP  
Revised: November 2012  
Part Number: *EMI\_RM\_018-1.0*
- Chapter 9. Functional Description—Example Designs  
Revised: November 2012  
Part Number: *EMI\_RM\_007-1.3*
- Chapter 10. Introduction to UniPHY IP  
Revised: November 2012  
Part Number: *EMI\_RM\_008-2.1*
- Chapter 11. Latency for UniPHY IP  
Revised: November 2012  
Part Number: *EMI\_RM\_009-2.1*

- Chapter 12. Timing Diagrams for UniPHY IP  
Revised: November 2012  
Part Number: *EMI\_RM\_010-2.1*
- Chapter 13. UniPHY External Memory Interface Debug Toolkit  
Revised: November 2012  
Part Number: *EMI\_RM\_011-2.2*
- Chapter 14. Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers  
Revised: November 2012  
Part Number: *EMI\_RM\_012-2.3*
- Chapter 15. Introduction to ALTMEMPHY IP  
Revised: November 2012  
Part Number: *EMI\_RM\_013-1.2*
- Chapter 16. Latency for ALTMEMPHY IP  
Revised: November 2012  
Part Number: *EMI\_RM014-1.2*
- Chapter 17. Timing Diagrams for ALTMEMPHY IP  
Revised: November 2012  
Part Number: *EMI\_RM\_015-1.3*
- Chapter 18. ALTMEMPHY External Memory Interface Debug Toolkit  
Revised: November 2012  
Part Number: *EMI\_RM\_016-1.2*

This section provides functional descriptions of the major external memory interface components.

This section includes the following chapters:

- Chapter 1, Functional Description—UniPHY
- Chapter 2, Functional Description—ALTMEMPHY
- Chapter 3, Functional Description—Hard Memory Interface
- Chapter 4, Functional Description—HPS Memory Controller
- Chapter 5, Functional Description—HPC II Controller
- Chapter 6, Functional Description—QDR II Controller
- Chapter 7, Functional Description—RLDRAM II Controller
- Chapter 8, Functional Description—RLDRAM 3 PHY-Only IP
- Chapter 9, Functional Description—Example Designs



For information about the revision history for chapters in this section, refer to “Document Revision History” in each individual chapter.



This chapter describes the UniPHY layer of the external memory interface, and includes related information.

The major sections of this chapter are as follows:

- Block Description
- Interfaces
- UniPHY Signals
- PHY-to-Controller Interfaces
- Using a Custom Controller
- Using a Vendor-Specific Memory Model
- AFI 3.0 Specification
- Register Maps
- Ping Pong PHY
- Efficiency Monitor and Protocol Checker
- UniPHY Calibration Stages

## Block Description

This section describes the major functional units of the UniPHY layer, which include the following:

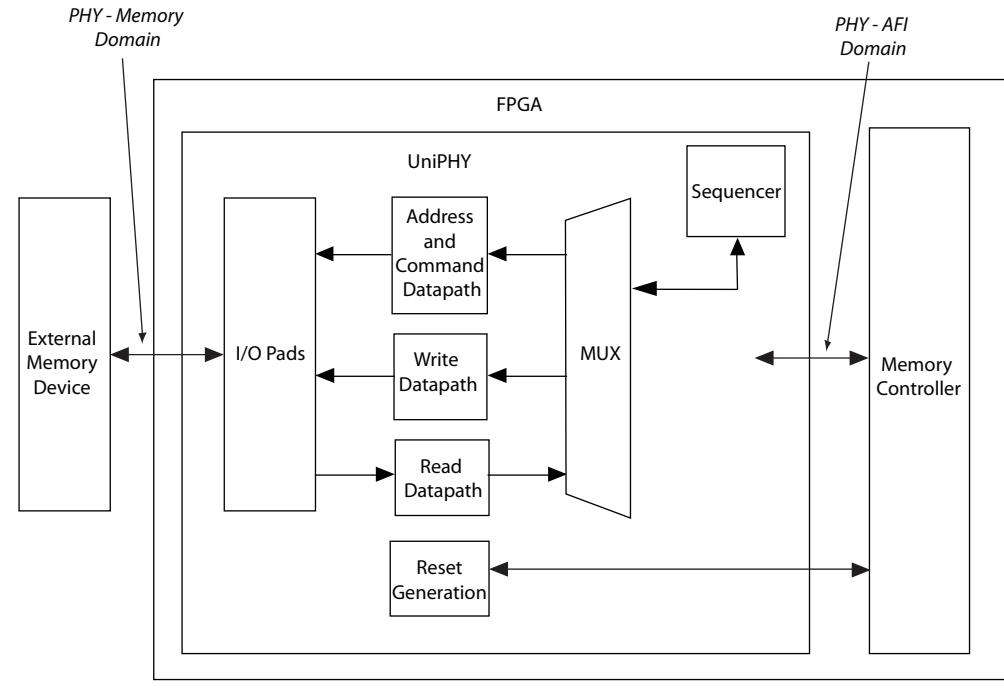
- Reset and Clock Generation
- Address and Command Datapath
- Write Datapath
- Read Datapath
- Sequencer

©2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Figure 1-1 shows the PHY block diagram; individual blocks are discussed in the text.

**Figure 1-1. PHY Block Diagram**



## I/O Pads

The I/O pads contain all the I/O instantiations.

## Reset and Clock Generation

At a high level, clocks in the PHY can be classified into two domains: the PHY-memory domain and the PHY-AFI domain. The PHY-memory domain interfaces with the external memory device and always operate at full-rate. The PHY-AFI domain interfaces with the memory controller and can be a full-rate, half-rate, or quarter-rate clock, based on the controller in use.

The number of clock domains in a memory interface can vary depending on its configuration; for example:

- At the PHY-memory boundary, separate clocks may exist to generate the memory clock signal, the output strobe, and to output write data, as well as address and command signals. These clocks include `pll_dq_write_clk`, `pll_write_clk`, `pll_mem_clk`, and `pll_addr_cmd_clk`. These clocks are phase-shifted as required to achieve the desired timing relationships between memory clock, address and command signals, output data, and output strobe.
- For quarter-rate interfaces, additional clock domains such as `pll_hr_clock` are required to convert signals between half-rate and quarter-rate.

- For high-performance memory interfaces using Arria V, Cyclone V, or Stratix V devices, additional clocks may be required to handle transfers between the device core and the I/O periphery for timing closure. For core-to-periphery transfers, the latch clock is `p11_c2p_write_clock`; for periphery-to-core transfers, it is `p11_p2c_read_clock`. These clocks are automatically phase-adjusted for timing closure during IP generation, but can be further adjusted in the parameter editor. If the phases of these clocks are zero, the Fitter may remove these clocks during optimization.

Also, high-performance interfaces using a Nios II-based sequencer require two additional clocks, `p11_avl_clock` for the Nios II processor, and `p11_config_clock` for clocking the I/O scan chains during calibration.

For a complete list of clocks in your memory interface, compile your design and run the **Report Clocks** command in the TimeQuest Timing Analyzer.

## Dedicated Clock Networks

The UniPHY layer employs three types of dedicated clock networks:

- Global clock network
- Dual-regional clock network
- PHY clock network (applicable to Arria V, Cyclone V, and Stratix V devices, and later)

The PHY clock network is a dedicated high-speed, low-skew, balanced clock tree designed for high-performance external memory interface. For device families that support the PHY clock network, UniPHY always uses the PHY clock network for all clocks at the PHY-memory boundary.

For families that do not support the PHY clock network, UniPHY uses either dual-regional or global clock networks for clocks at the PHY-memory boundary. During generation, the system selects dual-regional or global clocks automatically, depending on whether a given interface spans more than one quadrant. UniPHY does not mix the usage of dual-regional and global clock networks for clocks at the PHY-memory boundary; this ensures that timing characteristics of the various output paths are as similar as possible.

The `<variation_name>.pin_assignments.tcl` script creates the appropriate clock network type assignment. The use of the PHY clock network is specified directly in the RTL code, and does not require an assignment.

The UniPHY uses an active-low, asynchronous assert and synchronous de-assert reset scheme. The global reset signal resets the PLL in the PHY and the rest of the system is held in reset until after the PLL is locked.

## Address and Command Datapath

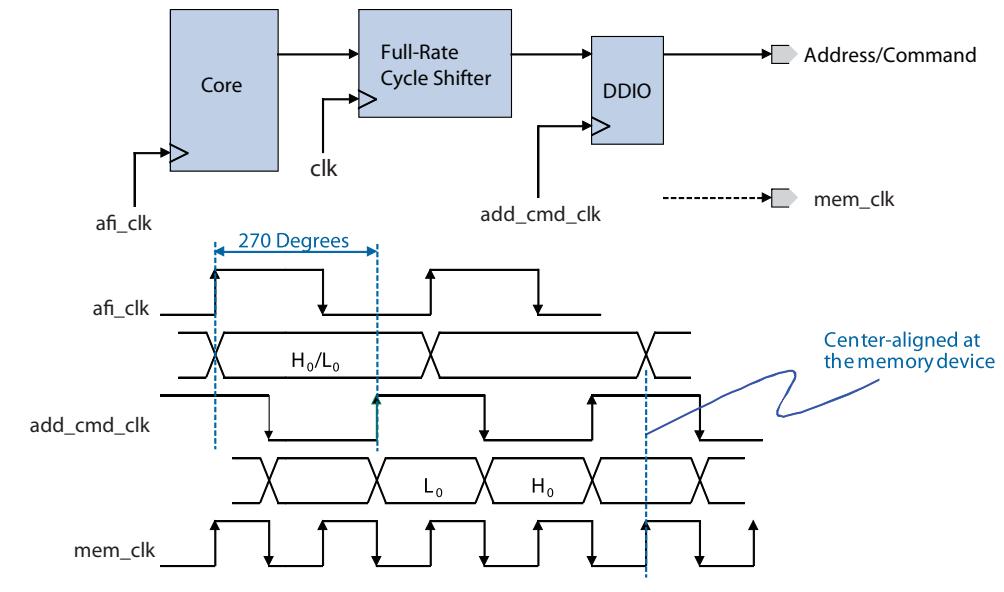
The memory controller controls the read and write addresses and commands to meet the memory specifications. The PHY is indifferent to address or command—that is, it performs no decoding or other operations—and the circuitry is the same for both. In full-rate and half-rate interfaces, address and command is full rate, while in quarter-rate interfaces, address and command is half rate.

Address and command signals are generated in the Altera PHY interface (AFI) clock domain and sent to the memory device in the address and command clock domain. The double-data rate input/output (DDIO) stage converts the half-rate signals into full-rate signals, when the AFI clock runs at half-rate. For quarter-rate interfaces, additional DDIO stages exist to convert the address and command signals in the quarter-rate AFI clock domain to half-rate.

The address and command clock is offset with respect to the memory clock to balance the nominal setup and hold margins at the memory device (center-alignment). In the example of [Figure 1–2](#), this offset is 270 degrees. The Fitter can further optimize margins based on the actual delays and clock skews. In half-rate and quarter-rate designs, the full-rate cycle shifter blocks can perform a shift measured in full-rate cycles to implement the correct write latency; without this logic, the controller would only be able to implement even write latencies as it operates at half the speed. The full-rate cycle shifter is clocked by either the AFI clock or the address and command clock, depending on the PHY configuration, to maximize timing margins on the path from the AFI clock to the address and command clock.

[Figure 1–2](#) illustrates the address and command datapath.

**Figure 1–2. Address and Command Datapath (Half-rate example shown)**

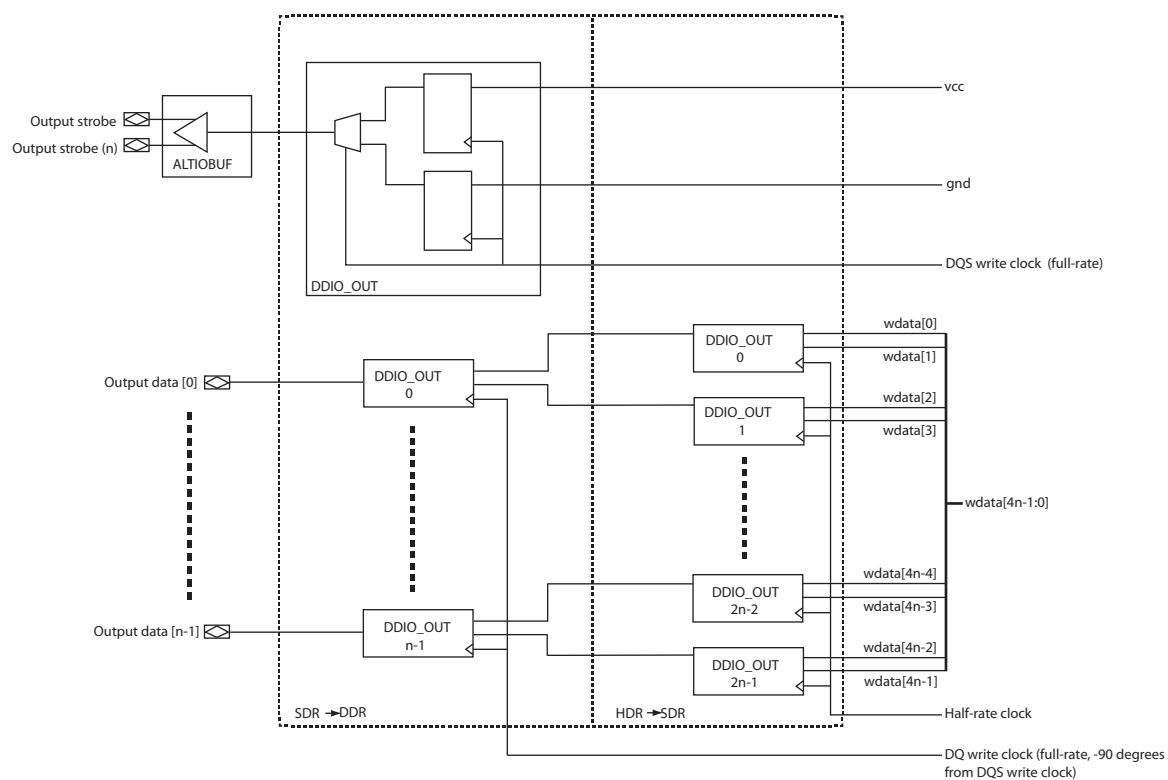


## Write Datapath

The write datapath passes write data from the memory controller to the I/O. The write data valid signal from the memory controller generates the output enable signal to control the output buffer. For memory protocols with a bidirectional data bus, it also generates the dynamic termination control signal, which selects between series (output mode) and parallel (input mode) termination.

Figure 1–3 illustrates a simplified write datapath of a typical half-rate interface. The full-rate DQS write clock is sent to a DDIO\_OUT cell. The output of DDIO\_OUT feeds an output buffer which creates a pair of pseudo differential clocks that connects to the memory. In full-rate mode, only the SDR-DDR portion of the path is used; in half-rate mode, the HDR-SDR circuitry is also required. The use of DDIO\_OUT in both the output strobe and output data generation path ensures that their timing characteristics are as similar as possible. The `<variation_name>_pin_assignments.tcl` script automatically specifies the logic option that associates all data pins to the output strobe pin. The Fitter treats the pins as a DQS/DQ pin group.

**Figure 1–3. Write Datapath**



## Leveling Circuitry

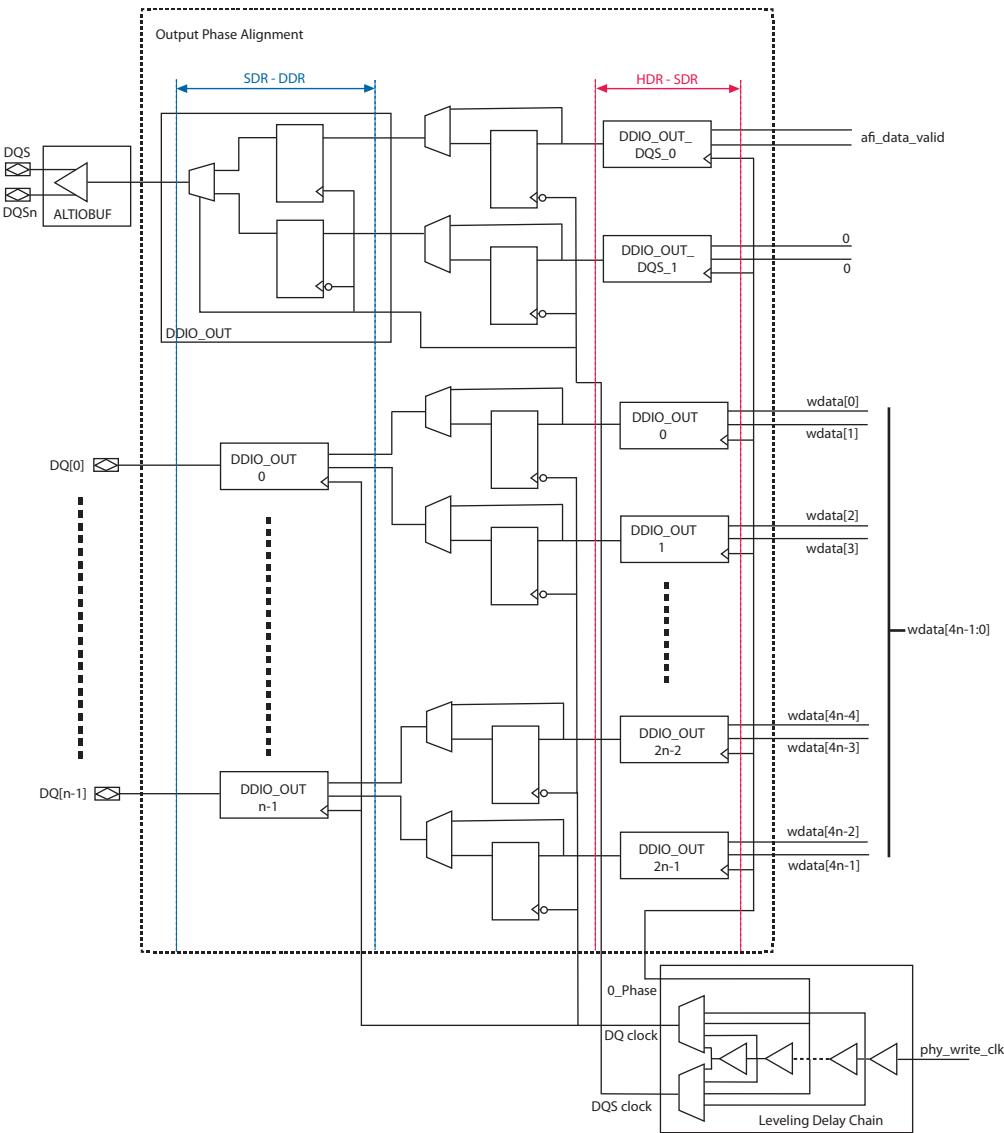
Leveling circuitry is dedicated I/O circuitry to provide calibration support for fly-by address and command networks. For DDR3, leveling is always invoked, whether the interface targets a DIMM or a single component. For DDR3 implementations at higher frequencies, a fly-by topology is recommended for optimal performance. For DDR2, leveling circuitry is invoked automatically for frequencies above 240 MHz; no leveling is used for frequencies below 240 MHz.

For DDR2 at frequencies below 240 MHz, you should use a tree-style layout. For frequencies above 240 MHz, you can choose either a leveled or balanced-T or Y topology, as the leveled PHY calibrates to either implementation. Regardless of protocol, for devices without a levelling block—such as Arria II GZ, Arria V, and Cyclone V—a balanced-T PCB topology for address/command/clock must be used because fly-by topology is not supported.

For details about leveling delay chains, consult the memory interfaces hardware section of the device handbook for your FPGA.

**Figure 1–4** shows the write datapath for a leveling interface. The full-rate PLL output clock `phy_write_clk` goes to a leveling delay chain block which generates all other periphery clocks that are needed. The data signals that generate DQ and DQS signals pass to an output phase alignment block. The output phase alignment block feeds an output buffer which creates a pair of pseudo differential clocks that connect to the memory. In full-rate designs, only the SDR-DDR portion of the path is used; in half-rate mode, the HDR-SDR circuitry is also required. The use of `DDIO_OUT` in both the output strobe and output data generation paths ensures that their timing characteristics are as similar as possible. The `<variation_name>_pin_assignments.tcl` script automatically specifies the logic option that associates all data pins to the output strobe pin. The Quartus II Fitter treats the pins as a DQS/DQ pin group.

**Figure 1–4. Write Datapath for a Leveling Interface**



## Read Datapath

Figure 1–5 shows the read datapath. This section describes the blocks and flow in the read datapath.

For all protocols, the DQS logic block delays the strobe by 90 degrees to center-align the rising strobe edge within the data window. For DDR2, DDR3, and LPDDR2 protocols, the logic block also performs strobe gating, holding the DQS enable signal high for the entire period that data is received. One DQS logic block exists for each data group.

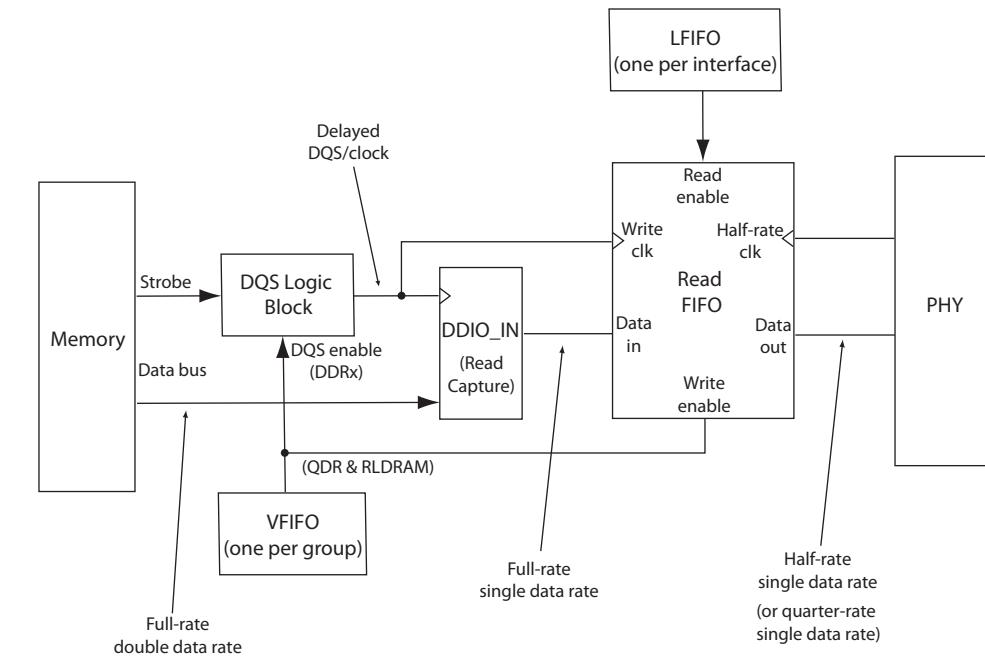
One VFIFO buffer exists for each data group. For DDR2, DDR3, and LPDDR2 protocols, the VFIFO buffer generates the DQS enable signal, which is delayed (by an amount determined during calibration) to align with the incoming DQS signal. For QDR and RLDRAM protocols, the output of the VFIFO buffer serves as the write enable signal for the Read FIFO buffer, signaling when to begin capturing data.

DDIO\_IN receives data from memory at double-data rate and passes data on to the Read FIFO buffer at single-data rate.

The Read FIFO buffer temporarily holds data read from memory; one Read FIFO buffer exists for each data group. For half-rate interfaces, the Read FIFO buffer converts the full-rate, single data-rate input to a half-rate, single data-rate output which is then passed to the PHY core logic. In the case of a quarter-rate interface, soft logic in the PHY performs an additional conversion from half-rate single data rate to quarter-rate single data rate.

One LFIFO buffer exists for each memory interface; the LFIFO buffer generates the read enable signal for all Read FIFO blocks in an interface. The read enable signal is asserted when the Read FIFO blocks have buffered sufficient data from the memory to be read. The timing of the read enable signal is determined during calibration.

**Figure 1–5. Read Datapath**



## Sequencer

Depending on the combination of protocol and IP architecture in your external memory interface, you may have either an RTL-based sequencer or a Nios® II-based sequencer. This section discusses the Nios II-based sequencer and the RTL-based sequencer.

-  **Table 1-3** in Volume 1 of this handbook shows the sequencer support for different protocol-architecture combinations.
-  Be aware that RTL-based sequencer implementations and Nios II-based sequencer implementations can have different pin requirements. You may not be able to migrate from an RTL-based sequencer to a Nios II-based sequencer and maintain the same pinout.
  -  For information on pin planning, refer to [Planning Pin and FPGA Resources](#) in volume 2 of the *External Memory Interface Handbook*.

### Nios II-Based Sequencer

The DDR2, DDR3, and LPDDR2 controllers with UniPHY employ a Nios II-based sequencer that is parameterizable and is dynamically generated at run time. The Nios II-based sequencer is also available with the QDR II and RLDRAM II controllers.

#### Function

The sequencer enables high-frequency memory interface operation by calibrating the interface to compensate for variations in setup and hold requirements caused by transmission delays.

The UniPHY converts the double-data rate interface of high-speed memory devices to a full-rate or half-rate interface for use within an FPGA. To compensate for slight variations in data transmission to and from the memory device, double-data rate is usually center-aligned with its strobe signal; nonetheless, at high speeds, slight variations in delay can result in setup or hold time violations. The sequencer implements a calibration algorithm to determine the combination of delay and phase settings necessary to maintain center-alignment of data and clock signals, even in the presence of significant delay variations. Programmable delay chains in the FPGA I/Os then implement the calculated delays to ensure that data remains centered. Calibration also applies settings to the FIFO buffers within the PHY to minimize latency and ensures that the read valid signal is generated at the appropriate time.

When calibration is completed, the sequencer returns control to the memory controller.

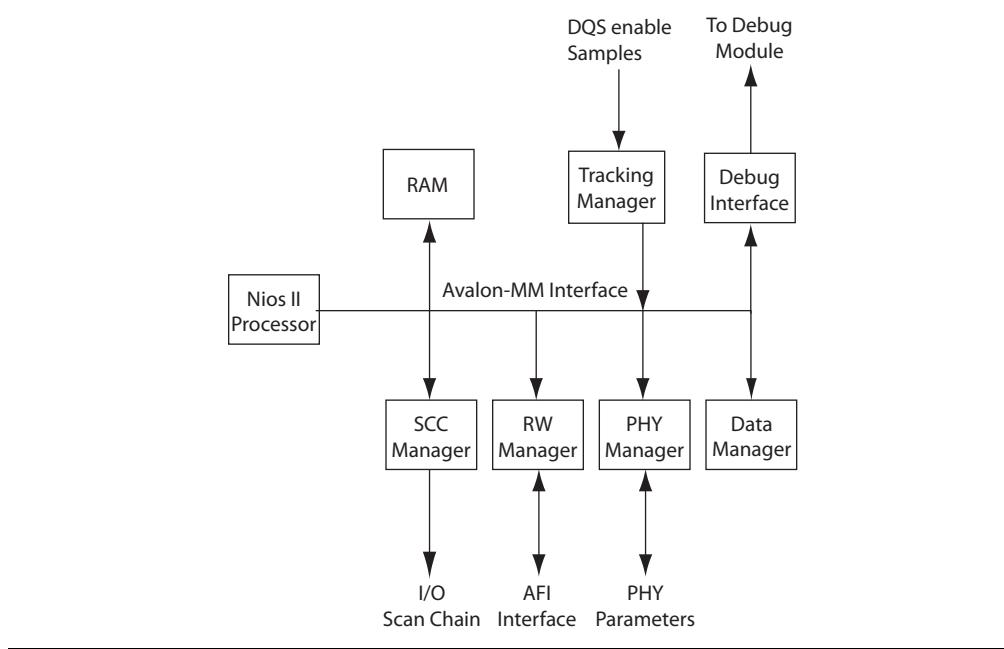
 For more information about calibration, refer to [UniPHY Calibration Stages](#).

## Architecture

Figure 1–6 shows the sequencer block diagram. The sequencer is composed of a Nios II processor and a series of hardware-based component managers, connected together by an Avalon bus. The Nios II processor performs the high-level algorithmic operations of calibration, while the component managers handle the lower-level timing, memory protocol, and bit-manipulation operations.

The high-level calibration algorithms are specified in C code, which is compiled into Nios II code that resides in the FPGA RAM blocks. The debug interface provides a mechanism for interacting with the various managers and for tracking the progress of the calibration algorithm, and can be useful for debugging problems that arise within the PHY. The various managers are specified in RTL and implement operations that would be slow or inefficient if implemented in software.

**Figure 1–6. Sequencer Block Diagram**



The C code that defines the calibration routines is available for your reference in the `\<name>_s0_software` subdirectory. Altera does not recommend that you modify this C code.

### SCC Manager

The scan chain control (SCC) manager allows the sequencer to set various delays and phases on the I/Os that make up the memory interface. The latest Altera device families provide dynamic delay chains on input, output, and output enable paths which can be reconfigured at runtime. The SCC manager provides the calibration routines access to these chains to add delay on incoming and outgoing signals. A master on the Avalon-MM interface may require the maximum allowed delay setting on input and output paths, and may set a particular delay value in this range to apply to the paths.

The SCC manager implements the Avalon-MM interface and the storage mechanism for all input, output, and phase settings. It contains circuitry that configures a DQ- or DQS-configuration block. The Nios II processor may set delay, phases, or register settings; the sequencer scans the settings serially to the appropriate DQ or DQS configuration block.

### RW Manager

The read write (RW) manager encapsulates the protocol to read and write to the memory device through the Altera PHY Interface (AFI). It provides a buffer that stores the data to be sent to and read from memory, and provides the following commands:

- Write configuration—configures the memory for use. Sets up burst lengths, read and write latencies, and other device specific parameters.
- Refresh—initiates a refresh operation at the DRAM. The command does not exist on SRAM devices. The sequencer also provides a register that determines whether the RW manager automatically generates refresh signals.
- Enable or disable multi-purpose register (MPR)—for memory devices with a special register that contains calibration specific patterns that you can read, this command enables or disables access to the register.
- Activate row—for memory devices that have both rows and columns, this command activates a specific row. Subsequent reads and writes operate on this specific row.
- Precharge—closes a row before you can access a new row.
- Write or read burst—writes or reads a burst length of data.
- Write guaranteed—writes with a special mode where the memory holds address and data lines constant. Altera guarantees this type of write to work in the presence of skew, but constrains to write the same data across the entire burst length.
- Write and read back-to-back—performs back-to-back writes or reads to adjacent banks. Most memory devices have strict timing constraints on subsequent accesses to the same bank, thus back-to-back writes and reads have to reference different banks.
- Protocol-specific initialization—a protocol-specific command required by the initialization sequence.

### PHY Manager

The PHY Manager provides access to the PHY for calibration, and passes relevant calibration results to the PHY. For example, the PHY Manager sets the VFIFO and LFIFO buffer parameters resulting from calibration, signals the PHY when the memory initialization sequence finishes, and reports the pass/fail status of calibration.

### Data Manager

The Data Manager stores parameterization-specific data in RAM, for the software to query.

### Tracking Manager

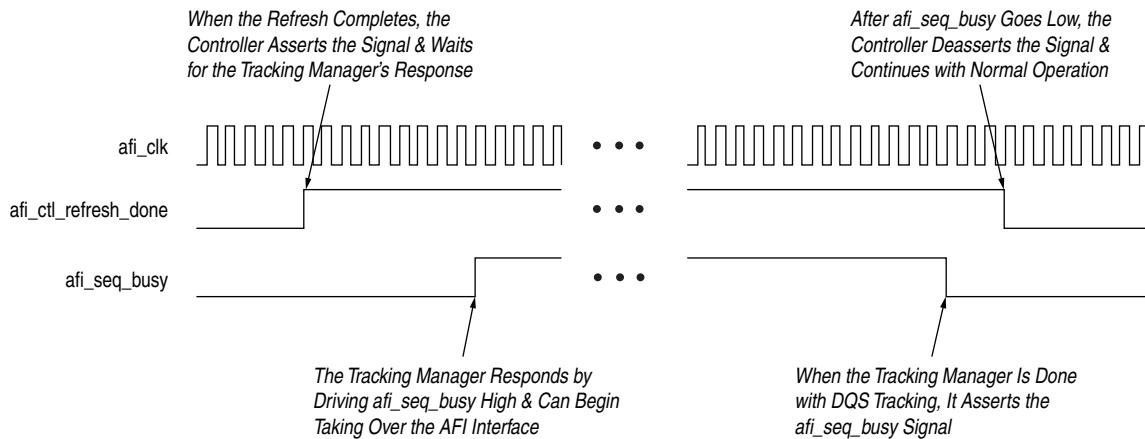
The Tracking Manager detects the effects of voltage and temperature variations that can occur on the memory device over time resulting in reduced margins, and adjusts the DQS enable delay as necessary to maintain adequate operating margins.

The Tracking Manager briefly assumes control of the AFI interface after each memory refresh cycle, issuing a read routine to the RW Manager, and then sampling the DQS tracking. Ideally, the falling edge of the DQS enable signal would align to the last rising edge of the raw DQS signal from the memory device. The Tracking Manager determines whether the DQS enable signal is leading or trailing the raw DQS signal.

Each time a refresh occurs, the Tracking Manager takes a sample of the raw DQS signal; any adjustments of the DQS enable signal occur only after sufficient samples of raw DQS have been taken. When the Tracking Manager determines that the DQS enable signal is either leading or lagging the raw DQS signal, it adjusts the DQS enable appropriately.

Figure 1–7 shows the Tracking manager signals.

**Figure 1–7. Tracking Manager Signals**



Some notes on Tracking Manager operation:

- The time taken by the Tracking Manager is arbitrary; if the period taken exceeds the refresh period, the Tracking Manager handles memory refresh.
- afi\_seq\_busy should go high fewer than 10 clock cycles after afi\_ctl\_refresh\_done or afi\_ctl\_long\_idle is asserted.
- afi\_refresh\_done should deassert fewer than 10 clock cycles after afi\_seq\_busy deasserts.
- afi\_ctl\_long\_idle causes the Tracking Manager to execute an algorithm different than periodic refresh; use afi\_ctl\_long\_idle when a long session has elapsed without a periodic refresh.

- The Tracking Manager is instantiated into the sequencer system when DQS Enable Tracking is turned on. [Table 1-1](#) summarizes configurations supporting DQS tracking.

**Table 1-1. Configurations Supporting DQS Tracking**

Device Family	Protocol	Frequency
Arria V Arria V GZ	LPDDR2	All frequencies.
	DDR3 (single rank)	More than 450 MHz for speed grade 5, or more than 534 MHz.
Cyclone V		
Stratix V		

- If you do not want to use DQS tracking, you can disable it (at your risk), by opening the Verilog file `<variant_name>.if0_c0.v` in an editor, and changing the value of the `USE_DQS_TRACKING` parameter from 1 to 0.

## Nios II Processor

The Nios II processor manages the calibration algorithm; the Nios II processor is unavailable after calibration is completed.

The same calibration algorithm supports all device families, with some differences. The following sections describe the calibration algorithm for DDR3 SDRAM on Stratix III devices. Calibration algorithms for other protocols and families are a subset and significant differences are pointed out when necessary. As the algorithm is fully contained in the software of the sequencer (in the C code) enabling and disabling specific steps involves turning flags on and off.

Calibration consists of the following stages:

- Initialize memory.
- Calibrate read datapath.
- Calibrate write datapath.
- Run diagnostics.

### Initialize Memory

Calibration must initialize all memory devices before they can operate properly. The sequencer performs this memory initialization stage when it takes control of the PHY at startup.

### Calibrate Read Datapath

Calibrating the read datapath comprises the following steps:

- Calibrate DQS enable cycle and phase.
- Perform read per-bit deskew to center the strobe signal within data valid window.
- Reduce LFIFO latency.

### Calibrate Write Datapath

Calibrating the write datapath involves the following steps:

- Center align DQS with respect to DQ.
- Align DQS with `mem_clk`.

### Test Diagnostics

The sequencer estimates the read and write margins under noisy conditions, by sweeping input and output DQ and DQS delays to determine the size of the data valid windows on the input and output sides. The sequencer stores this information in the local memory and you can access it through the debugging interface.

When the diagnostic test finishes, control of the PHY interface passes back to the controller and the sequencer issues a pass or fail signal.

### RTL-based Sequencer

The RTL-based sequencer is available for QDR II and RLDRAM II interfaces; it is a state machine that processes the calibration algorithm. The sequencer assumes control of the interface at reset (whether at initial startup or when the IP is reset) and maintains control throughout the calibration process. The sequencer relinquishes control to the memory controller only after successful calibration. [Table 1–2](#) lists the major states in the RTL-based sequencer.

**Table 1–2. Sequencer States (Part 1 of 2)**

State	Description
RESET	Remain in this state until reset is released.
LOAD_INIT	Load any initialization values for simulation purposes.
STABLE	Wait until the memory device is stable.
WRITE_ZERO	Issue write command to address 0.
WAIT_WRITE_ZERO	Write all 0xAs to address 0.
WRITE_ONE	Issue write command to address 1.
WAIT_WRITE_ONE	Write all 0x5s to address 1.
<b>Valid Calibration States</b>	
V_READ_ZERO	Issue read command to address 0 (expected data is all 0xAs).
V_READ_NOP	This state represents the minimum number of cycles required between 2 back-to-back read commands. The number of NOP states depends on the burst length.
V_READ_ONE	Issue read command to address 1 (expected data is all 0x5s).
V_WAIT_READ	Wait for read valid signal.
V_COMPARE_READ_ZERO_READ_ONE	Parameterizable number of cycles to wait before making the read data comparisons.
V_CHECK_READ_FAIL	When a read fails, the write pointer (in the AFI clock domain) of the valid FIFO buffer is incremented. The read pointer of the valid FIFO buffer is in the DQS clock domain. The gap between the read and write pointers is effectively the latency between the time when the PHY receives the read command and the time valid data is returned to the PHY.
V_ADD_FULL_RATE	Advance the read valid FIFO buffer write pointer by an extra full rate cycle.
V_ADD_HALF_RATE	Advance the read valid FIFO buffer write pointer by an extra half rate cycle. In full-rate designs, equivalent to V_ADD_FULL_RATE.
V_READ_FIFO_RESET	Reset the read and write pointers of the read data synchronization FIFO buffer.
V_CALIB_DONE	Valid calibration is successful.

**Table 1–2. Sequencer States (Part 2 of 2)**

<b>State</b>	<b>Description</b>
<b>Latency Calibration States</b>	
L_READ_ONE	Issue read command to address 1 (expected data is all 0x5s).
L_WAIT_READ	Wait for read valid signal from read datapath. Initial read latency is set to a predefined maximum value.
L_COMPARE_READ_ONE	Check returned read data against expected data. If data is correct, go to L_REDUCE_LATENCY; otherwise go to L_ADD_MARGIN.
L_REDUCE_LATENCY	Reduce the latency counter by 1.
L_READ_FLUSH	Read from address 0, to flush the contents of the read data resynchronization FIFO buffer.
L_WAIT_READ_FLUSH	Wait until the whole FIFO buffer is flushed, then go back to L_READ and try again.
L_ADD_MARGIN	Increment latency counter by 3 (1 cycle to get the correct data, 2 more cycles of margin for run time variations). If latency counter value is smaller than predefined ideal condition minimum, then go to CALIB_FAIL.
CALIB_DONE	Calibration is successful.
CALIB_FAIL	Calibration is not successful.

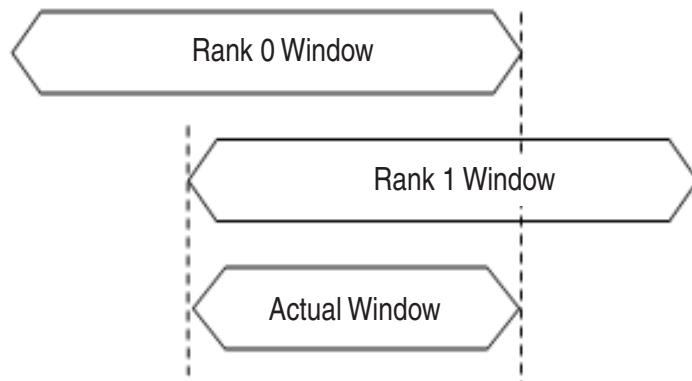
## Shadow Registers

Shadow registers are a hardware feature of Arria V GZ and Stratix V devices that enables high-speed multi-rank calibration for DDR3 quarter-rate and half-rate memory interfaces, up to 667MHz for dual-rank interfaces and 533MHz for quad-rank interfaces.

### Description

Prior to the introduction of shadow registers, the data valid window of a multi-rank interface was calibrated to the overlapping portion of the data valid windows of the individual ranks. The resulting data valid window for the interface would be smaller than the individual data valid windows, limiting overall performance.

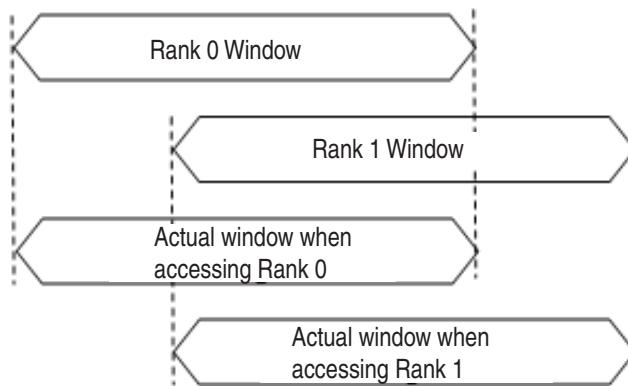
[Figure 1–8](#) illustrates the calibration of overlapping data valid windows, without shadow registers.

**Figure 1–8. Calibration of Overlapping Data Valid Windows, without Shadow Registers**

Shadow registers allow the sequencer to calibrate each rank separately and fully, and then to save the calibrated settings for each rank in its own set of shadow registers, which are part of the IP scan chains. During a rank-to-rank switch, the rank-specific set of calibration settings are restored just-in-time to optimize the data valid window for each rank.

**Figure 1–9** illustrates how the use of rank-specific calibration settings results in a data valid window appropriate for the current rank.

**Figure 1–9. Rank-Specific Calibration Settings, with Shadow Registers**



The shadow registers and their associated rank-switching circuitry are part of the device I/O periphery hardware.

## Operation

The sequencer calibrates each rank individually and stores the resulting configuration in shadow registers, which are part of the IP scan chains. UniPHY then selects the appropriate configuration for the rank in use, switching between configurations as necessary. Calibration results for deskew delay chains are stored in the shadow registers. For DQS enable/disable, delay chain configurations come directly from the FPGA core.

Arria V GZ and Stratix V devices provide two sets of shadow registers, allowing for full support of a dual-rank interface, and support of additional ranks at reduced frequency.

- For a 2-rank interface, each rank is fully calibrated separately.
- For a 4-rank interface, the first set of shadow registers stores calibration data for the first and second rank, and the second set of shadow registers stores calibration data for the third and fourth ranks.

## DLL Offset Control Block

For designs generated with HardCopy compatibility enabled, the UniPHY layer includes DLL offset control blocks, which allow adjustment of the DLL control word for modifying the DQS delay chains to compensate for variations in silicon.

A DLL offset control block can feed only one side of the chip, therefore the IP instantiates two DLL offset control blocks for each DLL, to permit control of resources on two sides of the chip, as in the case of wraparound designs. In non-wraparound designs, where the second DLL offset control block is not needed, the second control block remains unused and disappears during synthesis.

One complication of the dual DLL offset control block process, is that at generation time it is impossible to know where resources are going to be placed, so the system automatically connects the output of the first DLL offset control block to all DQS groups. In the case of a wraparound design, you must modify the RTL code after pin placement, to connect the second DLL offset control block.

To connect the second DLL offset control block, follow these steps:

1. Open the file *<variation\_name>\_new\_io\_pads.v* in an editor.
2. Find a line of a form similar to the following:

```
.dll_offsetdelay_in((i < 0) ?
hc_dll_config_dll_offset_ctrl_offsetctrlout :
hc_dll_config_dll_offset_ctrl_offsetctrlout),
```

This line connects the output of the first DLL offset control block (*hc\_dll\_config\_dll\_offset\_ctrl\_offsetctrlout*) to the offset control input of the DQS delay chain, for every DQS delay chain where *i* < 0.

3. Modify the above line to connect the second DLL offset control block. The following example shows the correct syntax to connect groups 0 to 3 to the output of the first DLL offset control block, and groups 4 and above to the output of the second DLL offset control block:

```
.dll_offsetdelay_in((i < 4) ?
hc_dll_config_dll_offset_ctrl_offsetctrlout :
hc_dll_config_dll_offset_ctrl_b_offsetctrlout),
```

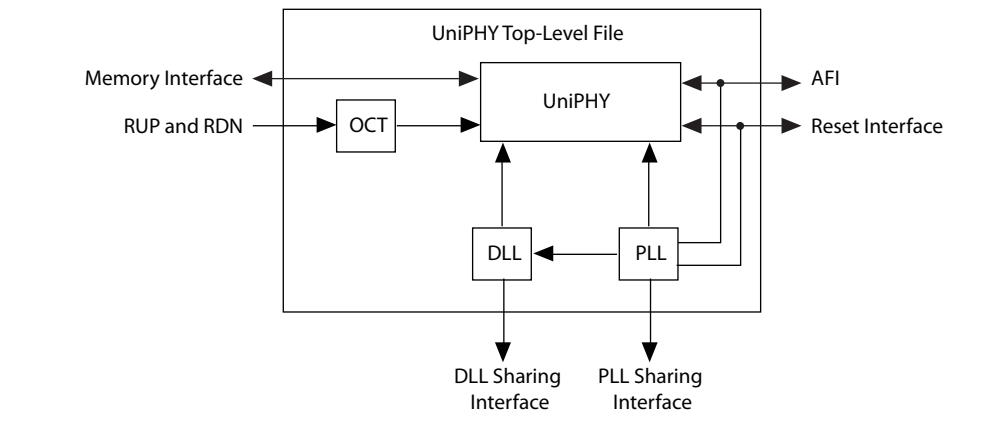
## Interfaces

Figure 1–10 shows the major blocks of the UniPHY and how it interfaces with the external memory device and the controller.



Instantiating the delay-locked loop (DLL) and the phase-locked loop (PLL) on the same level as the UniPHY eases DLL and PLL sharing.

**Figure 1–10. UniPHY Interfaces with the Controller and the External Memory**



The following interfaces are on the UniPHY top-level file:

- AFI
- Memory interface
- DLL sharing interface
- PLL sharing interface
- OCT interface

## AFI

The UniPHY datapath uses the Altera PHY interface (AFI). The AFI is in a simple connection between the PHY and controller. The AFI is based on the DDR PHY interface (DFI) specification, with some calibration-related signals not used and some additional Altera-specific sideband signals added.

For more information about the AFI, refer to [AFI 3.0 Specification](#).

## The Memory Interface

For more information on the memory interface, refer to “[UniPHY Signals](#)” on page 1–21.

## The DLL and PLL Sharing Interface

You can generate the UniPHY memory interface and configure it to share its PLL, DLL, or both interfaces. By default, a UniPHY memory interface variant contains a PLL and DLL; the PLL produces a variety of required clock signals derived from the reference clock, and the DLL produces a delay codeword. In this case the PLL sharing mode is "No sharing". A UniPHY variant can be configured as a PLL Master and/or DLL Master, in which case the corresponding interfaces are exported to the UniPHY top-level and can be connected to an identically configured UniPHY variant PLL Slave and/or DLL Slave. The UniPHY slave variant is instantiated without a PLL and/or DLL, which saves device resources.

-  For Arria II GX, Arria II GZ, Stratix III, and Stratix IV devices, the PLL and DLL must both be shared at the same time—their sharing modes must match. This restriction does not apply to Arria V, Arria V GZ, Cyclone V, or Stratix V devices.
-  For devices with hard memory interface components onboard, you cannot share PLL or DLL resources between soft and hard interfaces.

To share PLLs or DLLs, follow these steps:

1. To create a PLL or DLL master, create a UniPHY memory interface IP core. To make the PLL and/or DLL interface appear at the top-level in the core, on the **PHY Settings** tab in the parameter editor, set the **PLL Sharing Mode** and/or **DLL Sharing Mode** to **Master**.
  2. To create a PLL or DLL slave, create a second UniPHY memory interface IP core. To make the PLL and/or DLL interface appear at the top-level in the core, on the **PHY Settings** tab set the **PLL Sharing Mode** and/or **DLL Sharing Mode** to **Slave**.
  3. Connect the PLL and/or DLL sharing interfaces by following the appropriate step, below:
    - **For cores generated with Megawizard Plug-in Manager:** connect the PLL and/or DLL interface ports between the master and slave cores in your wrapper RTL. When using PLL sharing, connect the afi\_clk, afi\_half\_clk, and afi\_reset\_n outputs from the UniPHY PLL master to the afi\_clk, afi\_half\_clk, and afi\_reset\_in inputs on the UniPHY PLL slave
    - **For cores generated with Qsys:** connect the PLL and/or DLL interface in the Qsys GUI. When using PLL sharing, connect the afi\_clk, afi\_half\_clk, and afi\_reset interfaces from the UniPHY PLL master to the afi\_clk\_in, afi\_half\_clk\_in, and afi\_reset\_in interfaces on the UniPHY PLL slave.
- Qsys supports only one-to-one conduit connections in the patch panel. To share a PLL from a Uniphy PLL master with multiple slaves, you should replicate the number of PLL sharing conduit interfaces in the Qsys patch panel by choosing **Number of PLL sharing interfaces** in the parameter editor.



You may connect a slave UniPHY instance to the clocks from a user-defined PLL instead of from a UniPHY master. The general procedure for doing so is as follows:

- a. Make a template, by generating your IP with **PLL Sharing Mode** set to **No Sharing**, and then compiling the example project to determine the frequency and phases of the clock outputs from the PLL.
- b. Generate an external PLL using the MegaWizard flow, with the equivalent output clocks.
- c. Generate your IP with **PLL Sharing Mode** set to **Slave**, and connect the external PLL to the PLL sharing interface.

You must be very careful when connecting clock signals to the slave. Connecting to clocks with frequency or phase different than what the core expects may result in hardware failure.



The signal `dll_pll_locked` is an internal signal from the PLL to the DLL which ensures that the DLL remains in reset mode until the PLL becomes locked. This signal is not available for use by customer logic.



The PLL and DLL sharing interfaces are available when using SOPC Builder, however the PLL and/or DLL interfaces cannot be connected using the SOPC Builder GUI interface. To complete master-slave connections when using SOPC Builder, you must edit the generated RTL code manually.

## About PLL Simulation

PLL frequencies may differ between the synthesis and simulation file sets. In either case the achieved PLL frequencies and phases are calculated and reported in real time in the parameter editor.

For the simulation file set, clocks are specified in the RTL, not in units of frequency but by the period in picoseconds, thus avoiding clock drift due to picosecond rounding error.

For the synthesis file set, there are two mechanisms by which clock frequencies are specified in the RTL, based on the target device family:

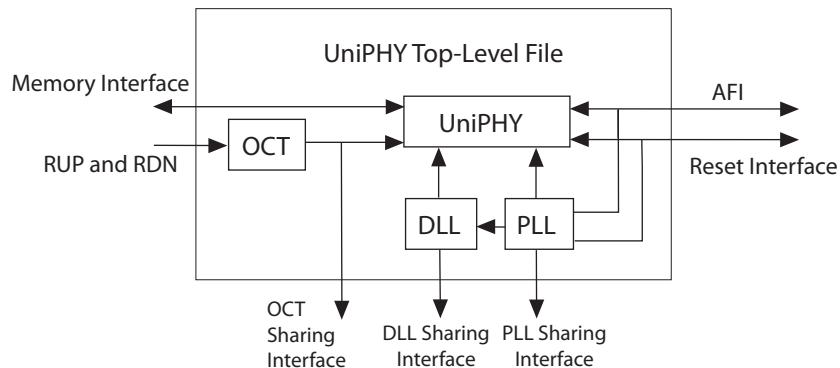
- For Arria V, Arria V GZ, Cyclone V, and Stratix V, clock frequencies are specified in megahertz.
- For Arria II GX, Arria II GZ, Stratix III, and Stratix IV, clock frequencies are specified by integer multipliers and divisors. For these families, the real simulation model—as opposed to the default abstract simulation model—also uses clock frequencies specified by integer ratios.

## The OCT Sharing Interface

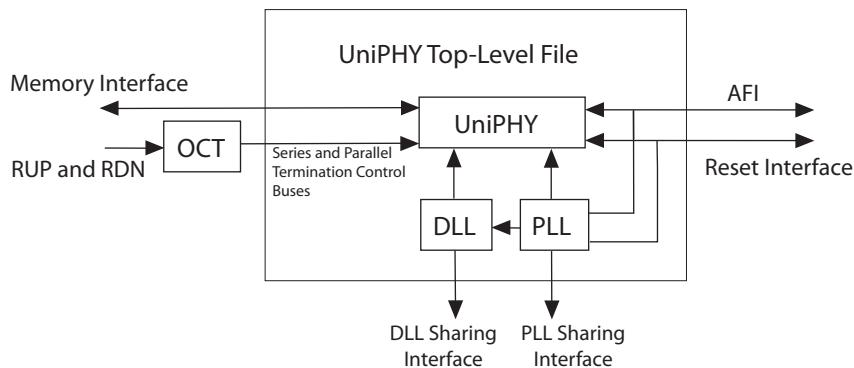
By default, the UniPHY IP generates the required OCT control block at the top-level RTL file for the PHY. If you want, you can instantiate this block elsewhere in your code and feed the required termination control signals into the IP core by turning off **Master for OCT Control Block** on the **PHY Settings** tab. If you turn off **Master for OCT Control Block**, you must instantiate the OCT control block or use another UniPHY instance as a master, and ensure that the parallel and series termination control bus signals are connected to the PHY.

[Figure 1–11](#) and [Figure 1–12](#), respectively, show the PHY architecture with and without Master for OCT Control Block.

**Figure 1–11. PHY Architecture with Master for OCT Control Block**



**Figure 1–12. PHY Architecture without Master for OCT Control Block**



The OCT sharing interface is available when using SOPC Builder, however the OCT sharing interface cannot be connected using the SOPC Builder parameter editor. To complete master-slave connections when using SOPC Builder, you must edit the generated RTL code manually.

If you generate a QDR II or RLDRAM II slave IP core, you must modify the pin assignment script to allow the fitter to correctly resolve the OCT termination block name in the OCT master core.

To modify the pin assignment script for QDR II or RLDRAM II slaves, follow these steps:

1. In a text editor, open your system's Tcl pin assignments script file, as follows:
  - For systems generated with the MegaWizard Plug-In Manager:  
Open the *<IP core name>/<slave core name>\_p0\_pin\_assignments.tcl* file.
  - For systems generated with Qsys or SOPC Builder:  
Open the *<HDL Path>/<submodules>/<slave core name>\_p0\_pin\_assignments.tcl* file.
2. Search for the following line:
  - `set ::master_corename "_MASTER_CORE_"`
3. Replace `_MASTER_CORE_` with the instance name of the UniPHY master to which the slave is connected. The instance name is determined from the pin assignments file name, as follows:
  - For systems generated with Qsys or SOPC Builder, the instance name is the *<master core name>* component of the pins assignments file name:  
*<HDL path>/<submodules>/<master core name>\_p0\_pin\_assignments.tcl*
  - For systems generated with the MegaWizard Plug-in Manager, the instance name is the *<master core name>* component of the pins assignments file name:  
*<IP core name>/<master core name>\_p0\_pin\_assignments.tcl*

## UniPHY Signals

This section describes the UniPHY signals.

**Table 1–3** lists the clock and reset signals.

**Table 1–3. Clock and Reset Signals**

Name	Direction	Width	Description
pll_ref_clk	Input	1	PLL reference clock input.
global_reset_n	Input	1	Active low global reset for PLL and all logic in the PHY, which causes a complete reset of the whole system. Minimum pulse width is 40ns.
soft_reset_n	Input	1	Holding soft_reset_n low holds the PHY in a reset state. However it does not reset the PLL, which keeps running. It also holds the afi_reset_n output low. Mainly for use by SOPC Builder.

**Table 1–4** lists the DDR2 and DDR3 SDRAM interface signals.

**Table 1–4. DDR2 and DDR3 SDRAM Interface Signals (Part 1 of 2)**

Name	Direction	Width	Description
mem_ck, mem_ck_n	Output	MEM_CK_WIDTH	Memory clock.
mem_cke	Output	MEM_CLK_EN_WIDTH	Clock enable.
mem_cs_n	Output	MEM_CHIP_SELECT_WIDTH	Chip select..
mem_cas_n	Output	MEM_CONTROL_WIDTH	Column address strobe.

**Table 1–4. DDR2 and DDR3 SDRAM Interface Signals (Part 2 of 2)**

Name	Direction	Width	Description
mem_ras_n	Output	MEM_CONTROL_WIDTH	Row address strobe.
mem_we_n	Output	MEM_CONTROL_WIDTH	Write enable.
mem_a	Output	MEM_ADDRESS_WIDTH	Address.
mem_ba	Output	MEM_BANK_ADDRESS_WIDTH	Bank address.
mem_dqs, mem_dqs_n	Bidirectional	MEM_DQS_WIDTH	Data strobe.
mem_dq	Bidirectional	MEM_DQ_WIDTH	Data.
mem_dm	Output	MEM_DM_WIDTH	Data mask.
mem_odt	Output	MEM_ODT_WIDTH	On-die termination.
mem_reset_n (DDR3 only)	Output	1	Reset
mem_ac_parity (DDR3 only, RDIMM/LRDIMM only)	Output	MEM_CONTROL_WIDTH	Address/command parity bit. (Even parity, per the RDIMM spec, JESD82-29A.)
mem_err_out_n (DDR3 only, RDIMM/LRDIMM only)	Input	MEM_CONTROL_WIDTH	Address/command parity error.



For information about the AFI signals, refer to [AFI 3.0 Specification](#).

For information about top-level HardCopy migration signals, refer to [HardCopy Design Migration Guidelines](#) in volume 2 of the *External Memory Interface Handbook*.

**Table 1–5** lists parameters relating to UniPHY signals.

**Table 1–5. UniPHY Parameters (Part 1 of 2)**

Parameter Name	Description
AFI_RATIO	AFI_RATIO is 1 in full-rate designs. AFI_RATIO is 2 for half-rate designs. AFI_RATIO is 4 for quarter-rate designs.
MEM_IF_DQS_WIDTH	The number of DQS pins in the interface.
MEM_ADDRESS_WIDTH	The address width of the specified memory device.
MEM_BANK_WIDTH	The bank width of the specified memory device.
MEM_CHIP_SELECT_WIDTH	The chip select width of the specified memory device.
MEM_CONTROL_WIDTH	The control width of the specified memory device.
MEM_DM_WIDTH	The DM width of the specified memory device.
MEM_DQ_WIDTH	The DQ width of the specified memory device.
MEM_READ_DQS_WIDTH	The READ DQS width of the specified memory device.
MEM_WRITE_DQS_WIDTH	The WRITE DQS width of the specified memory device.
OCT_SERIES_TERM_CONTROL_WIDTH	—
OCT_PARALLEL_TERM_CONTROL_WIDTH	—
AFI_ADDRESS_WIDTH	The AFI address width, derived from the corresponding memory interface width.
AFI_BANK_WIDTH	The AFI bank width, derived from the corresponding memory interface width.

**Table 1–5. UniPHY Parameters (Part 2 of 2)**

<b>Parameter Name</b>	<b>Description</b>
AFI_CHIP_SELECT_WIDTH	The AFI chip select width, derived from the corresponding memory interface width.
AFI_DATA_MASK_WIDTH	The AFI data mask width.
AFI_CONTROL_WIDTH	The AFI control width, derived from the corresponding memory interface width.
AFI_DATA_WIDTH	The AFI data width.
AFI_DQS_WIDTH	The AFI DQS width.
DLL_DELAY_CTRL_WIDTH	The DLL delay output control width.
NUM_SUBGROUP_PER_READ_DQS	A read datapath parameter for timing purposes.
QVLD_EXTRA_FLOP_STAGES	A read datapath parameter for timing purposes.
READ_VALID_TIMEOUT_WIDTH	A read datapath parameter; calibration fails when the timeout counter expires.
READ_VALID_FIFO_WRITE_ADDR_WIDTH	A read datapath parameter; the write address width for half-rate clocks.
READ_VALID_FIFO_READ_ADDR_WIDTH	A read datapath parameter; the read address width for full-rate clocks.
MAX_LATENCY_COUNT_WIDTH	A latency calibration parameter; the maximum latency count width.
MAX_READ_LATENCY	A latency calibration parameter; the maximum read latency.
READ_FIFO_READ_ADDR_WIDTH	—
READ_FIFO_WRITE_ADDR_WIDTH	—
MAX_WRITE_LATENCY_COUNT_WIDTH	A write datapath parameter; the maximum write latency count width.
INIT_COUNT_WIDTH	An initialization sequence.
MRSC_COUNT_WIDTH	A memory-specific initialization parameter.
INIT_NOP_COUNT_WIDTH	A memory-specific initialization parameter.
MRS_CONFIGURATION	A memory-specific initialization parameter.
MRS_BURST_LENGTH	A memory-specific initialization parameter.
MRS_ADDRESS_MODE	A memory-specific initialization parameter.
MRS_DLL_RESET	A memory-specific initialization parameter.
MRS_IMP_MATCHING	A memory-specific initialization parameter.
MRS_ODT_EN	A memory-specific initialization parameter.
MRS_BURST_LENGTH	A memory-specific initialization parameter.
MEM_T_WL	A memory-specific initialization parameter.
MEM_T_RL	A memory-specific initialization parameter.
SEQ_BURST_COUNT_WIDTH	The burst count width for the sequencer.
VCALIB_COUNT_WIDTH	The width of a counter that the sequencer uses.
DOUBLE_MEM_DQ_WIDTH	—
HALF_AFI_DATA_WIDTH	—
CALIB_REG_WIDTH	The width of the calibration status register.
NUM_AFI_RESET	The number of AFI resets to generate.

## PHY-to-Controller Interfaces

This section describes the typical modules that are connected to the UniPHY and the port name prefixes each module uses. Also this section describes using a custom controller and describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI PHY interface includes an administration block that configures the memory for calibration and performs necessary accesses to mode registers that configure the memory as required.

For half-rate designs, the address and command signals in the UniPHY are asserted for one `mem_clk` cycle (1T addressing), such that there are two input bits per address and command pin in half-rate designs. If you require a more conservative 2T addressing (where signals are asserted for two `mem_clk` cycles), drive both input bits (of the address and command signal) identically in half-rate designs.

Figure 1-13 shows the half-rate write operation.

**Figure 1-13. Half-Rate Write with Word-Aligned Data**

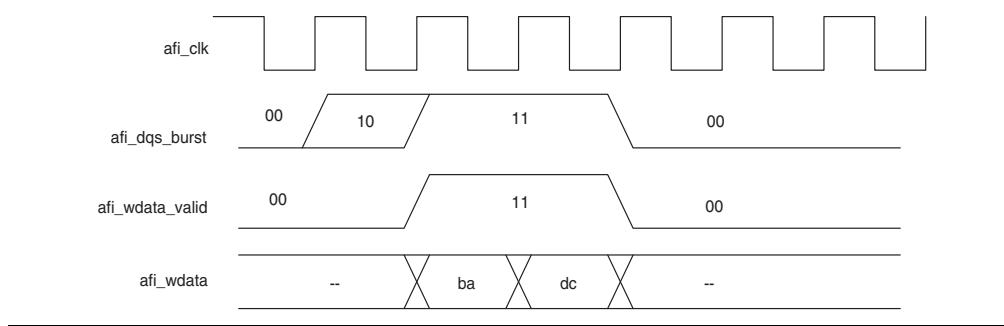
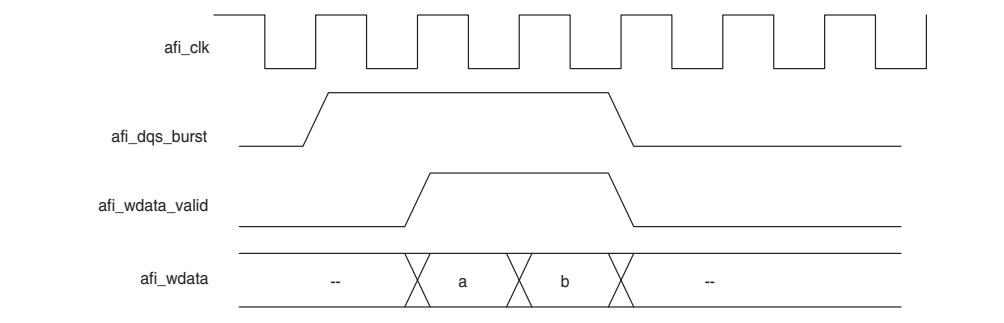


Figure 1-14 shows a full-rate write.

**Figure 1-14. Full-Rate Write**



After calibration is completed, the sequencer sends the write latency in number of clock cycles to the controller.

Figure 1-15 and Figure 1-16 show writes and reads, where the IP core writes data to and reads from the same address. In each example, `afi_rdata` and `afi_wdata` are aligned with controller clock (`afi_clk`) cycles. All the data in the bit vector is valid at once.

The AFI has the following conventions:

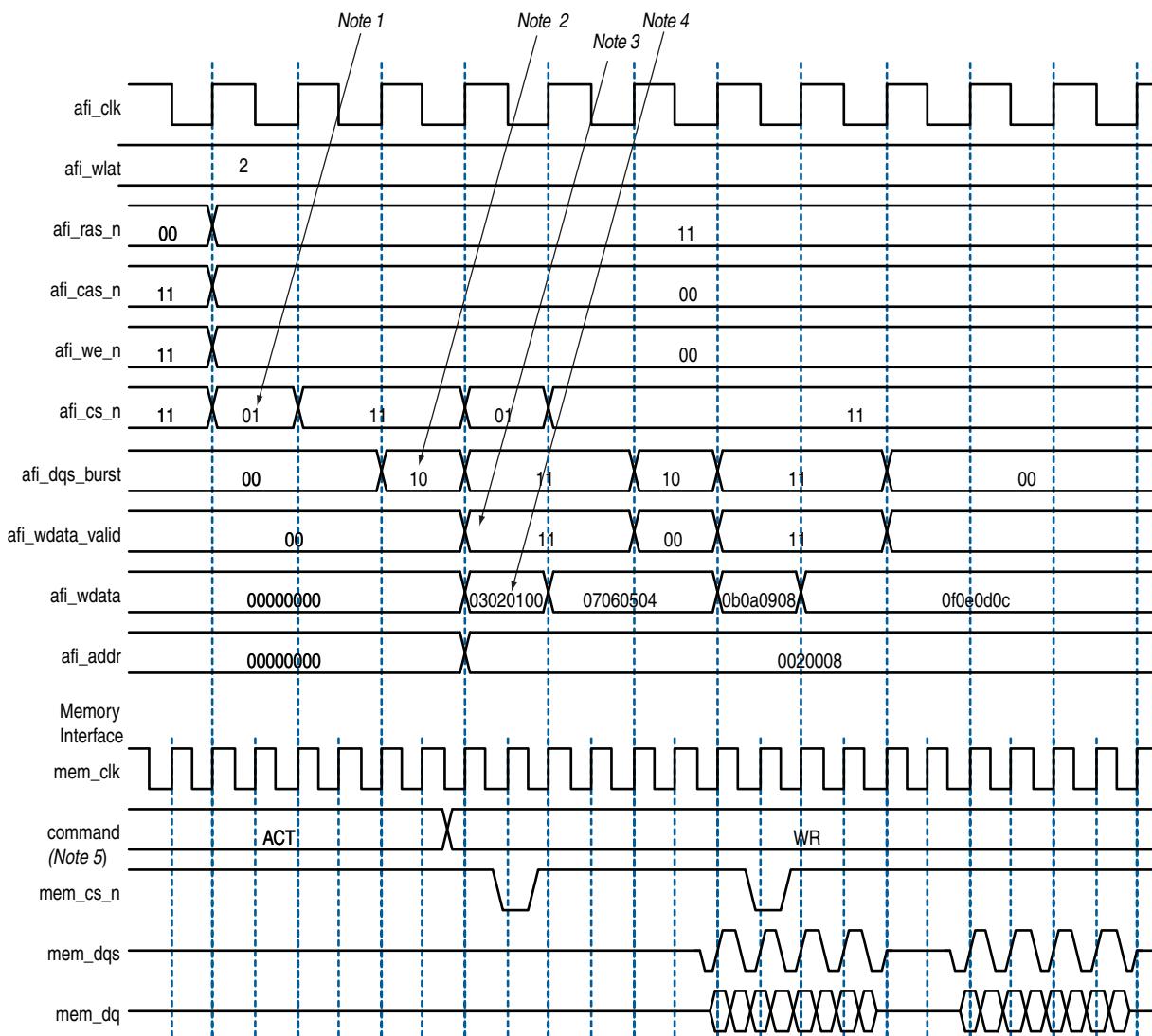
- With the AFI, high and low signals are combined in one signal, so for a single chip select (afi\_cs\_n) interface, afi\_cs\_n[1:0], location 0 appears on the memory bus on one mem\_clk cycle and location 1 on the next mem\_clk cycle.

 This convention is maintained for all signals so for an 8 bit memory interface, the write data (afi\_wdata) signal is afi\_wdata[31:0], where the first data on the DQ pins is afi\_wdata[7:0], then afi\_wdata[15:8], then afi\_wdata[23:16], then afi\_wdata[31:24].

- Spaced reads and writes have the following definitions:
  - Spaced writes—write commands separated by a gap of one controller clock (afi\_clk) cycle.
  - Spaced reads—read commands separated by a gap of one controller clock (afi\_clk) cycle.

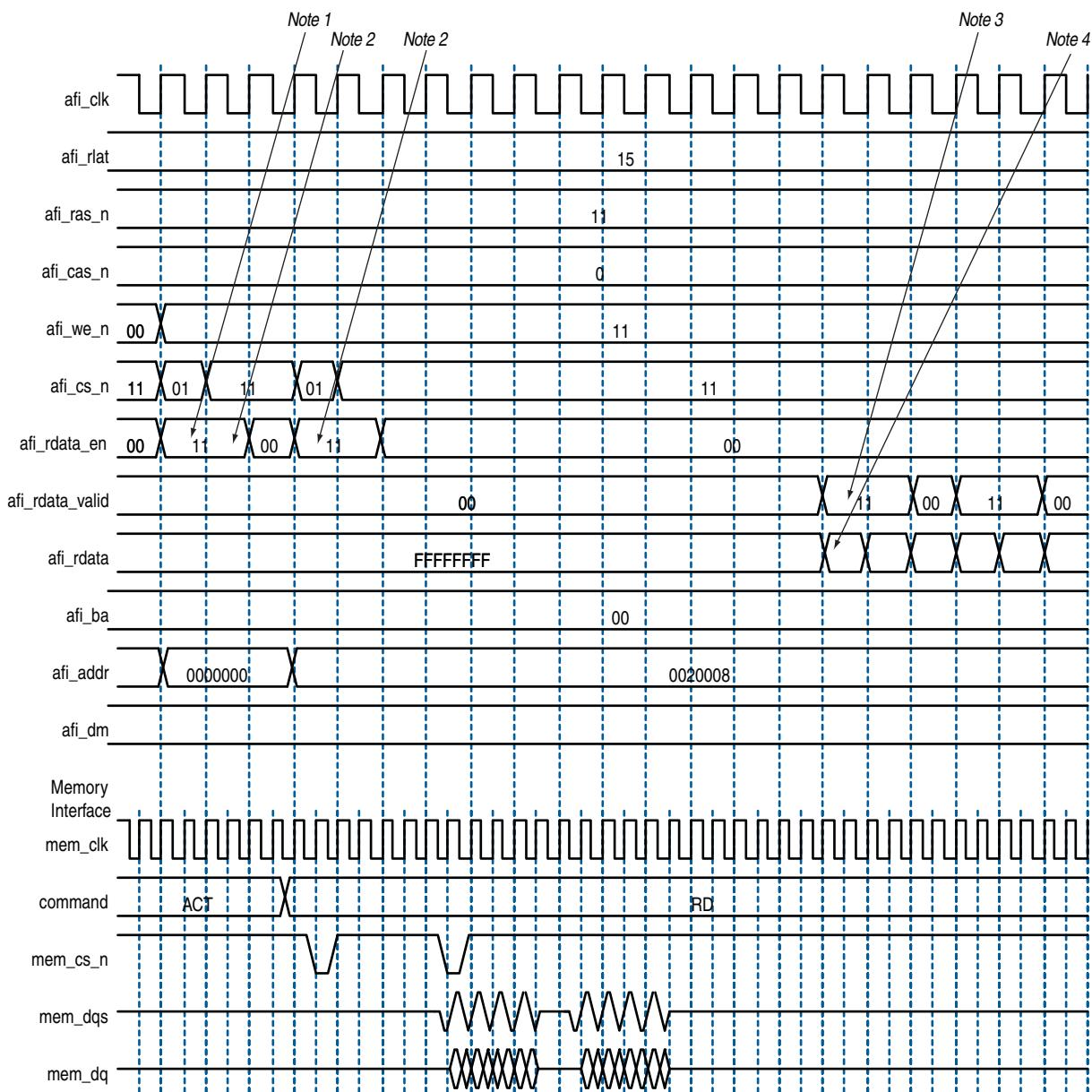
Figure 1–15 and Figure 1–16 assume the following general points:

- The burst length is four.
- An 8-bit interface with one chip select.
- The data for one controller clock (afi\_clk) cycle represents data for two memory clock (mem\_clk) cycles (half-rate interface).

**Figure 1-15. Word-Aligned Writes****Notes to Figure 1-15:**

- (1) To show the even alignment of afi\_cs\_n, expand the signal (this convention applies for all other signals).
- (2) The afi\_dqs\_burst must go high one memory clock cycle before afi\_wdata\_valid. Compare with the word-unaligned case.
- (3) The afi\_wdata\_valid is asserted two afi\_clk controller clock (afi\_clk) cycles after chip select (afi\_cs\_n) is asserted. The afi\_wlat indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive afi\_cs\_n and then wait afi\_wlat (two in this example) afi\_clocks before driving afi\_wdata\_valid.
- (4) Observe the ordering of write data (afi\_wdata). Compare this to data on the mem\_dq signal.
- (5) In all waveforms a command record is added that combines the memory pins ras\_n, cas\_n and we\_n into the current command that is issued. This command is registered by the memory when chip select (mem\_cs\_n) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

**Figure 1–16. Word-Aligned Reads**

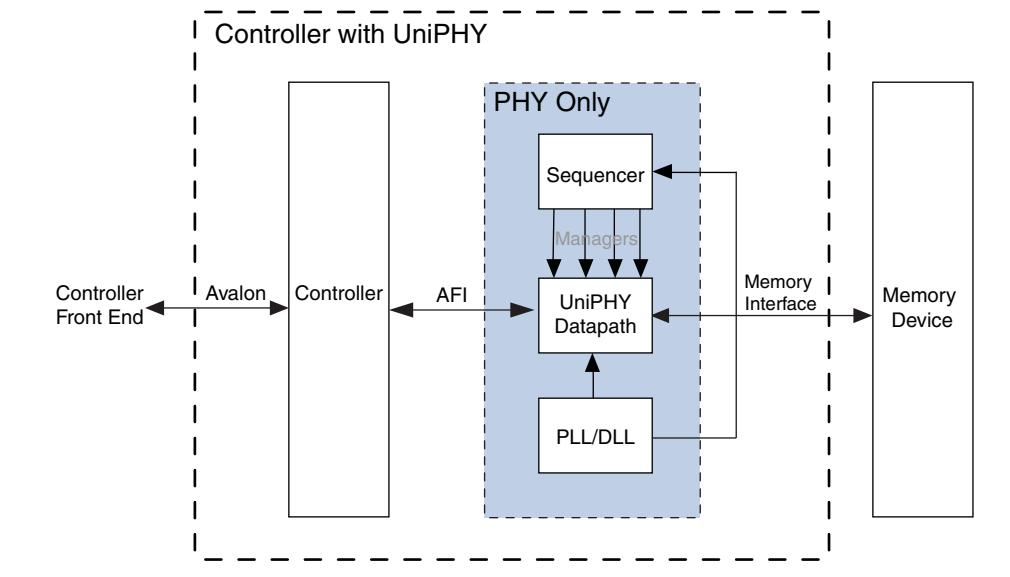


## Using a Custom Controller

By default, the UniPHY-based external memory interface IP cores are delivered with both the PHY and the memory controller integrated, as depicted in [Figure 1-17](#).

If you want to use your own custom controller with the UniPHY PHY, check the **Generate PHY only** box on the **PHY Settings** tab of the parameter editor and generate the IP. The resulting top-level IP consists of only the sequencer, UniPHY datapath, and PLL/DLL — the shaded area in [Figure 1-17](#).

**Figure 1-17. Memory Controller with UniPHY**



The AFI interface is exposed at the top-level of the generated IP core; you can connect the AFI interface to your custom controller.

When you enable **Generate PHY only**, the generated example designs include the memory controller appropriately instantiated to mediate read/write commands from the traffic generator to the PHY-only IP.

- ☞ For information on the AFI protocol, refer to the [AFI 3.0 Specification](#).
- ☞ For information on the example designs, refer to [Traffic Generator and BIST Engine](#) in chapter 7 of this section.

## Using a Vendor-Specific Memory Model

You can replace the Altera-supplied memory model with a vendor-specific memory model. In general, you may find vendor-specific models to be standardized, thorough, and well supported, but sometimes more complex to setup and use.

If you do want to replace the Altera-supplied memory model with a vendor-supplied memory model, observe the following guidelines:

- Ensure that the vendor-supplied memory model that you have is correct for your memory device.
- Disconnect all signals from the default memory model and reconnect them to the vendor-supplied memory model.
- If you intend to run simulation from the Quartus II software, ensure that the .qip file points to the vendor-supplied memory model.

 For related information, refer to [Simulating Memory IP](#) in volume 2 of the *External Memory Interface Handbook*.

## AFI 3.0 Specification

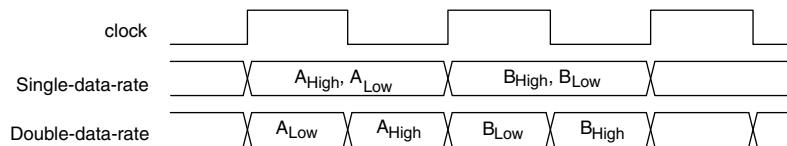
The Altera AFI interface defines communication between the controller and physical layer (PHY) in the external memory interface. The following sections describe the AFI implementation and the AFI signals.

### Implementation

The AFI interface is a single-data-rate interface, meaning that data is transferred on the rising edge of each clock cycle. Most memory interfaces, however, operate at double-data-rate, transferring data on both the rising and falling edges of the clock signal. If the AFI interface is to directly control a double-data-rate signal, two single-data-rate bits must be transmitted on each clock cycle; the PHY then sends out one bit on the rising edge of the clock and one bit on the falling edge.

The AFI convention is to send the low part of the data first and the high part second, as shown in [Figure 1-18](#).

**Figure 1-18. Single versus Double Data Rate Transfer**



### Bus Width and AFI Ratio

In cases where the AFI clock frequency is one-half or one-quarter of the memory clock frequency, the AFI data must be twice or four times as wide, respectively, as the corresponding memory data. The ratio between AFI clock and memory clock frequencies is referred to as the AFI ratio. (A half-rate AFI interface has an AFI ratio of 2, while a quarter-rate interface has an AFI ratio of 4.)

In general, the width of the AFI signal depends on the following three factors:

- The size of the equivalent signal on the memory interface. For example, if  $a[15:0]$  is a DDR3 address input and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_addr` bus will be 16-bits wide.
- The data rate of the equivalent signal on the memory interface. For example, if  $d[7:0]$  is a double-data-rate QDR II input data bus and the AFI clock runs at the same speed as the memory interface, the equivalent `afi_write_data` bus will be 16-bits wide.
- The AFI ratio. For example, if `cs_n` is a single-bit DDR3 chip select input and the AFI clock runs at half the speed of the memory interface, the equivalent `afi_cs_n` bus will be 2-bits wide.

The following formula summarizes the three factors described above:

$$\text{AFI\_width} = \text{memory\_width} * \text{signal\_rate} * \text{AFI\_RATE\_RATIO}$$



The above formula is a general rule, but not all signals obey it. For definite signal-size information, refer to the specific table.

## AFI Parameters

[Table 1–6](#) through [Table 1–14](#) list the AFI parameters grouped according to their functions.

Not all parameters are used for all protocols.

### Parameters Affecting Bus Width

The following parameters affect the width of AFI signal buses. Parameters prefixed by `MEM_IF_` refer to the signal size at the interface between the PHY and memory device.

**Table 1–6. Ratio Parameters**

Parameter Name	Description
<code>AFI_RATE_RATIO</code>	The ratio between the AFI clock frequency and the memory clock frequency. For full-rate interfaces this value is 1, for half-rate interfaces the value is 2, and for quarter-rate interfaces the value is 4.
<code>DATA_RATE_RATIO</code>	The number of data bits transmitted per clock cycle. For single-data rate protocols this value is 1, and for double-data rate protocols this value is 2.
<code>ADDR_RATE_RATIO</code>	The number of address bits transmitted per clock cycle. For single-data rate address protocols this value is 1, and for double-data rate address protocols this value is 2.

**Table 1–7. Memory Interface Parameters**

Parameter Name	Description
MEM_IF_ADDR_WIDTH	The width of the address bus on the memory device(s).
MEM_IF_BANKADDR_WIDTH	The width of the bank address bus on the interface to the memory device(s). Typically, the $\log_2$ of the number of banks.
MEM_IF_CS_WIDTH	The number of chip selects on the interface to the memory device(s).
MEM_IF_WRITE_DQS_WIDTH	The number of DQS (or write clock) signals on the write interface. For example, the number of DQS groups.
MEM_IF_CLK_PAIR_COUNT	The number of CK/CK# pairs.
MEM_IF_DQ_WIDTH	The number of DQ signals on the interface to the memory device(s). For single-ended interfaces such as QDR II, this value is the number of D or Q signals.
MEM_IF_DM_WIDTH	The number of data mask pins on the interface to the memory device(s).
MEM_IF_READ_DQS_WIDTH	The number of DQS signals on the read interface. For example, the number of DQS groups.

**Table 1–8. Derived AFI Parameters**

Parameter Name	Derivation Equation
AFI_ADDR_WIDTH	MEM_IF_ADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO
AFI_BANKADDR_WIDTH	MEM_IF_BANKADDR_WIDTH * AFI_RATE_RATIO * ADDR_RATE_RATIO
AFI_CONTROL_WIDTH	AFI_RATE_RATIO * ADDR_RATE_RATIO
AFI_CS_WIDTH	MEM_IF_CS_WIDTH * AFI_RATE_RATIO
AFI_DM_WIDTH	MEM_IF_DM_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO
AFI_DQ_WIDTH	MEM_IF_DQ_WIDTH * AFI_RATE_RATIO * DATA_RATE_RATIO
AFI_WRITE_DQS_WIDTH	MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO
AFI_LAT_WIDTH	6
AFI_RLAT_WIDTH	AFI_LAT_WIDTH
AFI_WLAT_WIDTH	AFI_LAT_WIDTH * MEM_IF_WRITE_DQS_WIDTH
AFI_CLK_PAIR_COUNT	MEM_IF_CLK_PAIR_COUNT
AFI_WRANK_WIDTH	Number of ranks * MEM_IF_WRITE_DQS_WIDTH * AFI_RATE_RATIO
AFI_RRANK_WIDTH	Number of ranks * MEM_IF_READ_DQS_WIDTH * AFI_RATE_RATIO

## AFI Signals

The following tables list the AFI signals grouped according to their functions.

In each table, the **direction** column denotes the direction of the signal relative to the PHY. For example, a signal defined as an output passes out of the PHY to the controller. The AFI specification does not include any bidirectional signals.

Not all signals are used for all protocols.

### Clock and Reset Signals

The AFI interface provides up to two clock signals and an asynchronous reset signal.

**Table 1–9. Clock and Reset Signals**

Signal Name	Direction	Width	Description
afi_clk	Output	1	Clock with which all data exchanged on the AFI bus is synchronized. In general, this clock is referred to as full-rate, half-rate, or quarter-rate, depending on the ratio between the frequency of this clock and the frequency of the memory device clock.
afi_half_clk	Output	1	Clock signal that runs at half the speed of the afi_clk. The controller uses this signal when the half-rate bridge feature is in use. This signal is optional.
afi_reset_n	Output	1	Asynchronous reset output signal. You must synchronize this signal to the clock domain in which you use it.

### Address and Command Signals

The address and command signals encode read/write/configuration commands to send to the memory device. The address and command signals are single-data rate signals.

**Table 1–10. Address and Command Signals (Part 1 of 2)**

Signal Name	Direction	Width	Description
afi_ba	Input	AFI_BANKADDR_WIDTH	Bank address.
afi_cke	Input	AFI_CLK_EN_WIDTH	Clock enable.
afi_cs_n	Input	AFI_CS_WIDTH	Chip select signal. (The number of chip selects may not match the number of ranks; for example, RDIMMs and LRDIMMs require a minimum of 2 chip select signals for both single-rank and dual-rank configurations. Consult your memory device datasheet for information about chip select signal width.)
afi_ras_n	Input	AFI_CONTROL_WIDTH	RAS# (for DDR2 and DDR3 memory devices.)

**Table 1–10. Address and Command Signals (Part 2 of 2)**

Signal Name	Direction	Width	Description
afi_we_n	Input	AFI_CONTROL_WIDTH	WE# (for DDR2, DDR3, and RLDRAM II memory devices.)
afi_cas_n	Input	AFI_CONTROL_WIDTH	CAS# (for DDR2 and DDR3 memory devices.)
afi_ref_n	Input	AFI_CONTROL_WIDTH	REF# (for RLDRAM II memory devices.)
afi_RST_n	Input	AFI_CONTROL_WIDTH	RESET# (for DDR3 memory devices.)
afi_odt	Input	AFI_CLK_EN_WIDTH	On-die termination signal for DDR2 and DDR3 memory devices. (Do not confuse this memory device signal with the FPGA's internal on-chip termination signal.)
afi_mem_clk_disable	Input	AFI_CLK_PAIR_COUNT	When this signal is asserted, mem_clk and mem_clk_n are disabled. This signal is used in low-power mode.
afi_wps_n	Output	AFI_CS_WIDTH	WPS (for QDR II/II+ memory devices.)
afi_rps_n	Output	AFI_CS_WIDTH	RPS (for QDR II/II+ memory devices.)

## Write Data Signals

The signals described in this section control the data, data mask, and strobe signals passed to the memory device during write operations.

**Table 1-11. Write Data Signals**

Signal Name	Direction	Width	Description
afi_dqs_burst	Input	AFI_WRITE_DQS_WIDTH	Controls the enable on the strobe (DQS) pins for DDR2, DDR3, and LPDDR2 memory devices. When this signal is asserted, mem_dqs and mem_dqsn are driven. This signal must be asserted before afi_wdata_valid to implement the write preamble, and must be driven for the correct duration to generate a correctly timed mem_dqs signal.
afi_wdata_valid	Input	AFI_WRITE_DQS_WIDTH	Write data valid signal. This signal controls the output enable on the data and data mask pins.
afi_wdata	Input	AFI_DQ_WIDTH	Write data signal to send to the memory device at double-data rate. This signal controls the PHY's mem_dq output.
afi_dm	Input	AFI_DM_WIDTH	Data mask. This signal controls the PHY's mem_dm signal for DDR2, DDR3, LPDDR2, and RLDRAM II memory devices.)
afi_bws_n	Input	AFI_DM_WIDTH	Data mask. This signal controls the PHY's mem_bws_n signal for QDR II/II+ memory devices.
afi_wrrank	Input	AFI_WRANK_WIDTH	Shadow register signal. Signal indicating the rank to which the controller is writing, so that the PHY can switch to the appropriate setting. Signal timing is identical to afi_dqs_burst; that is, afi_wrrank must be asserted at the same time as afi_dqs_burst, and must be of the same duration.

## Read Data Signals

The signals described in this section control the data sent from the memory device during read operations.

**Table 1-12. Read Data Signals**

Signal Name	Direction	Width	Description
afi_rdata_en	Input	AFI_RATE_RATIO	Read data enable. Indicates that the memory controller is currently performing a read operation. This signal is held high only for cycles of relevant data (read data masking).  If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).
afi_rdata_en_full	Input	AFI_RATE_RATIO	Read data enable full. Indicates that the memory controller is currently performing a read operation. This signal is held high for the entire read burst.  If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).
afi_rdata	Output	AFI_DQ_WIDTH	Read data from the memory device. This data is considered valid only when afi_rdata_valid is asserted by the PHY.
afi_rdata_valid	Output	AFI_RATE_RATIO	Read data valid. When asserted, this signal indicates that the afi_rdata bus is valid.  If this signal is aligned to even clock cycles, it is possible to use 1-bit even in half-rate mode (i.e., AFI_RATE=2).
afi_rrank	Input	AFI_RRANK_WIDTH	Shadow register signal. Signal indicating the rank from which the controller is reading, so that the PHY can switch to the appropriate setting. Must be asserted at the same time as afi_rdata_en when issuing a read command, and once asserted, must remain unchanged until the controller issues a new read command to another rank.

## Calibration Status Signals

The PHY instantiates a sequencer which calibrates the memory interface with the memory device and some internal components such as read FIFOs and valid FIFOs. The sequencer reports the results of the calibration process to the controller through the AFI interface. This section describes the calibration status signals.

**Table 1-13. Calibration Status Signals**

<b>Signal Name</b>	<b>Direction</b>	<b>Width</b>	<b>Description</b>
afi_cal_success	Output	1	Asserted to indicate that calibration has completed successfully.
afi_cal_fail	Output	1	Asserted to indicate that calibration has failed.
afi_cal_req	Input	1	Effectively a synchronous reset for the sequencer. When this signal is asserted, the sequencer returns to the reset state; when this signal is released, a new calibration sequence begins.
afi_wlat	Output	AFI_WLAT_WIDTH	The required write latency in afi_clk cycles, between address/command and write data being issued at the PHY/controller interface. The afi_wlat value can be different for different groups; each group's write latency can range from 0 to 63. If write latency is the same for all groups, only the lowest 6 bits are required.
afi_rlat	Output	AFI_RLAT_WIDTH	The required read latency in afi_clk cycles between address/command and read data being returned to the PHY/controller interface. Values can range from 0 to 63.

## Tracking Management Signals

When tracking management is enabled, the sequencer can take control over the AFI interface at given intervals, and issue commands to the memory device to track the internal DQS Enable signal alignment to the DQS signal returning from the memory device. The tracking management portion of the AFI interface provides a means for the sequencer and the controller to exchange handshake signals.

**Table 1-14. Tracking Management Signals**

<b>Signal Name</b>	<b>Direction</b>	<b>Width<sup>a</sup></b>	<b>Description</b>
afi_ctl_refresh_done	Input	MEM_IF_CS_WIDTH	Handshaking signal from controller to tracking manager, indicating that a refresh has occurred and waiting for a response.
afi_seq_busy	Output	MEM_IF_CS_WIDTH	Handshaking signal from sequencer to controller, indicating when DQS tracking is in progress.
afi_ctl_long_idle	Input	MEM_IF_CS_WIDTH	Handshaking signal from controller to tracking manager, indicating that it has exited low power state without a periodic refresh, and waiting for response.

## Register Maps

[Table 1–15](#) lists the overall register mapping for the DDR2, DDR3, and LPDDR2 SDRAM Controllers with UniPHY.

**Table 1–15. Register Map**

Address	Description
<b>UniPHY Register Map</b>	
0x001	Reserved.
0x004	UniPHY status register 0.
0x005	UniPHY status register 1.
0x006	UniPHY status register 2.
0x007	UniPHY memory initialization parameters register 0.
<b>Controller Register Map</b>	
0x100	Reserved.
0x110	Controller status and configuration register.
0x120	Memory address size register 0.
0x121	Memory address size register 1.
0x122	Memory address size register 2.
0x123	Memory timing parameters register 0.
0x124	Memory timing parameters register 1.
0x125	Memory timing parameters register 2.
0x126	Memory timing parameters register 3.
0x130	ECC control register.
0x131	ECC status register.
0x132	ECC error address register.

## UniPHY Register Map

The UniPHY register map allows you to control the memory components' mode register settings. [Table 1–16](#) lists the register map for UniPHY.

**Table 1–16. UniPHY Register Map (Part 1 of 3)**

Address	Bit	Name	Default	Access	Description
0x001	15:0	Reserved.	0	—	Reserved for future use.
	31:16	Reserved.	0	—	Reserved for future use.
0x002	15:0	Reserved.	0	—	Reserved for future use.
	31:16	Reserved.	0	—	Reserved for future use.

**Table 1–16. UniPHY Register Map (Part 2 of 3)**

<b>Address</b>	<b>Bit</b>	<b>Name</b>	<b>Default</b>	<b>Access</b>	<b>Description</b>
0x004	0	SOFT_RESET	—	Write only	Initiate a soft reset of the interface. This bit is automatically deasserted after reset.
	23:1	Reserved.	0	—	Reserved for future use.
	24	AFI_CAL_SUCCESS	—	Read only	Reports the value of the UniPHY afi_cal_success. Writing to this bit has no effect.
	25	AFI_CAL_FAIL	—	Read only	Reports the value of the UniPHY afi_cal_fail. Writing to this bit has no effect.
	26	Reserved.	0	—	Reserved for future use.
	31:27	Reserved.	0	—	Reserved for future use.
0x005	7:0	Reserved.	0	—	Reserved for future use.
	15:8	Reserved.	0	—	Reserved for future use.
	23:16	Reserved.	0	—	Reserved for future use.
	31:24	Reserved.	0	—	Reserved for future use.
0x006	7:0	INIT FAILING STAGE	—	Read only	Initial failing error stage of calibration. Only applicable if AFI_CAL_FAIL=1.
	15:8	INIT FAILING SUBSTAGE	—	Read only	Initial failing error substage of calibration. Only applicable if AFI_CAL_FAIL=1.
	23:16	INIT FAILING GROUP	—	Read only	Initial failing error group of calibration. Only applicable if AFI_CAL_FAIL=1.  Returns failing DQ pin instead of failing group, if: INIT FAILING STAGE=1 and INIT FAILING SUBSTAGE=3.  Or INIT FAILING STAGE=4 and INIT FAILING SUBSTAGE=1.
	31:24	Reserved.	0	—	Reserved for future use.
0x007	31:0	DQS_DETECT	—	Read only	Identifies if DQS edges have been identified for each of the groups. Each bit corresponds to one DQS group.
0x008 (DDR2)	1:0	RTT_NOM	—	Read only	Rtt (nominal) setting of the DDR2 Extended Mode Register used during memory initialization.
	31:2	Reserved.	0	—	Reserved for future use.
0x008 (DDR3)	2:0	RTT_NOM	—	—	Rtt (nominal) setting of the DDR3 MR1 mode register used during memory initialization.
	4:3	Reserved.	0	—	Reserved for future use.
	6:5	ODS	—	—	Output driver impedance control setting of the DDR3 MR1 mode register used during memory initialization.
	8:7	Reserved.	0	—	Reserved for future use.
	10:9	RTT_WR	—	—	Rtt (writes) setting of the DDR3 MR2 mode register used during memory initialization.
	31:11	Reserved.	0	—	Reserved for future use.

**Table 1–16. UniPHY Register Map (Part 3 of 3)**

Address	Bit	Name	Default	Access	Description
0x008 (LPDDR 2)	3:0	DS			Driver impedance control for MR3 during initialization.
	31:4	Reserved.			Reserved for future use.

## Controller Register Map

The controller register map allows you to control the memory controller settings.

For information on the controller register map, refer to “[Controller Register Map](#)” on page 5–29 of this volume.

## Ping Pong PHY

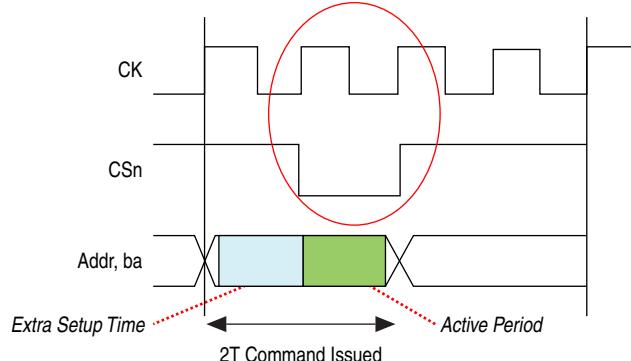
Ping Pong PHY is an implementation of UniPHY that allows two memory interfaces to share address and command buses through time multiplexing. Compared to having two independent interfaces, Ping Pong PHY uses fewer pins and less logic, while maintaining equivalent throughput.

The Ping Pong PHY supports only quarter-rate configurations of the DDR3 protocol on Arria V GZ and Stratix V devices.

## Feature Description

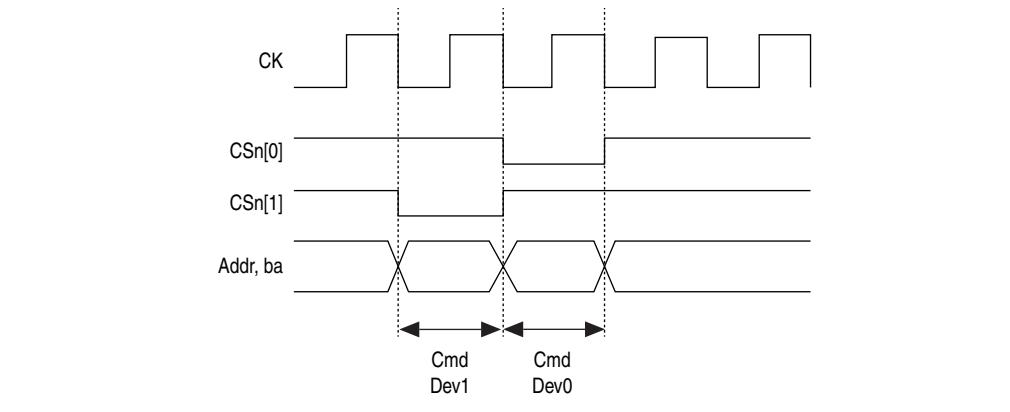
In conventional UniPHY, the address and command buses of a DDR3 quarter-rate interface use 2T time—meaning that they are issued for two full-rate clock cycles, as illustrated in [Figure 1–19](#).

**Figure 1–19. 2T Command Timing**



With the Ping Pong PHY, address and command signals from two independent controllers are multiplexed onto shared buses by delaying one of the controller outputs by one full-rate clock cycle. The result is 1T timing, with a new command being issued on each full-rate clock cycle. [Figure 1-20](#) shows address and command timing for the Ping Pong PHY.

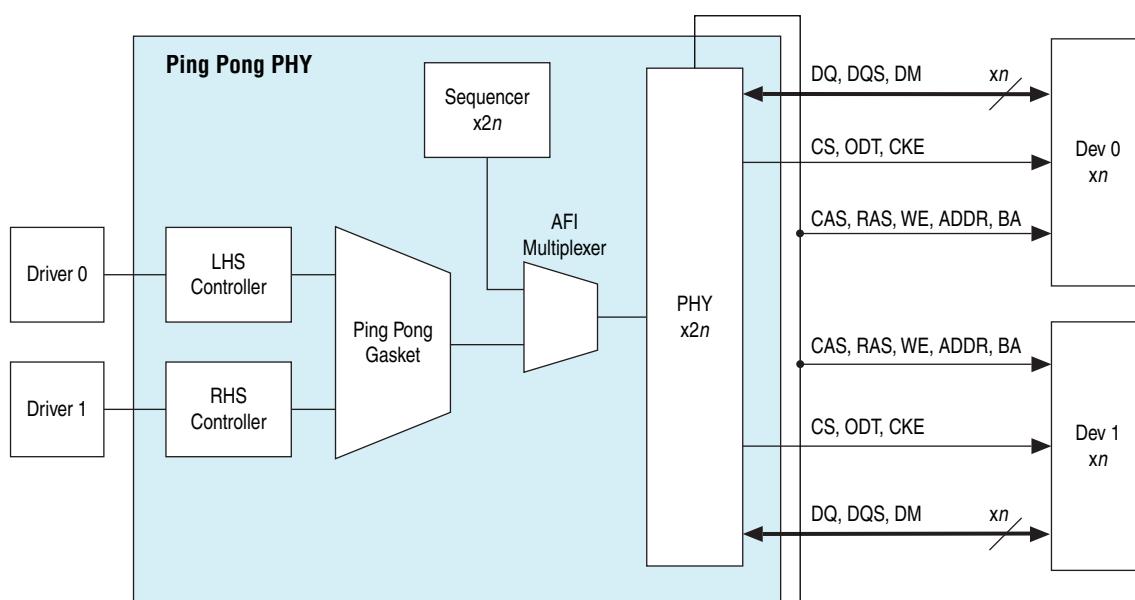
**Figure 1–20. 1T Command Timing Use by Ping Pong PHY**



## Architecture

[Figure 1-21](#) shows a top-level block diagram of the Ping Pong PHY. Functionally, the IP looks like two independent memory interfaces. The two controller blocks are referred to as right-hand side (RHS) and left-hand side (LHS), respectively. A gasket block located between the controllers and the PHY merges the AFI signals. The PHY is double data width and supports both memory devices. The sequencer is the same as with regular UniPHY, and calibrates the entire double-width PHY.

**Figure 1–21. Ping Pong PHY Architecture**



## Ping Pong Gasket

Figure 1–22 shows the gasket architecture. The gasket delays and remaps quarter-rate signals so that they are correctly time-multiplexed at the full-rate PHY output. The gasket also merges address and command buses ahead of the PHY.

AFI interfaces at the input and output of the gasket provide compatibility with the PHY and with memory controllers.

**Figure 1–22. Ping Pong PHY Gasket Architecture**

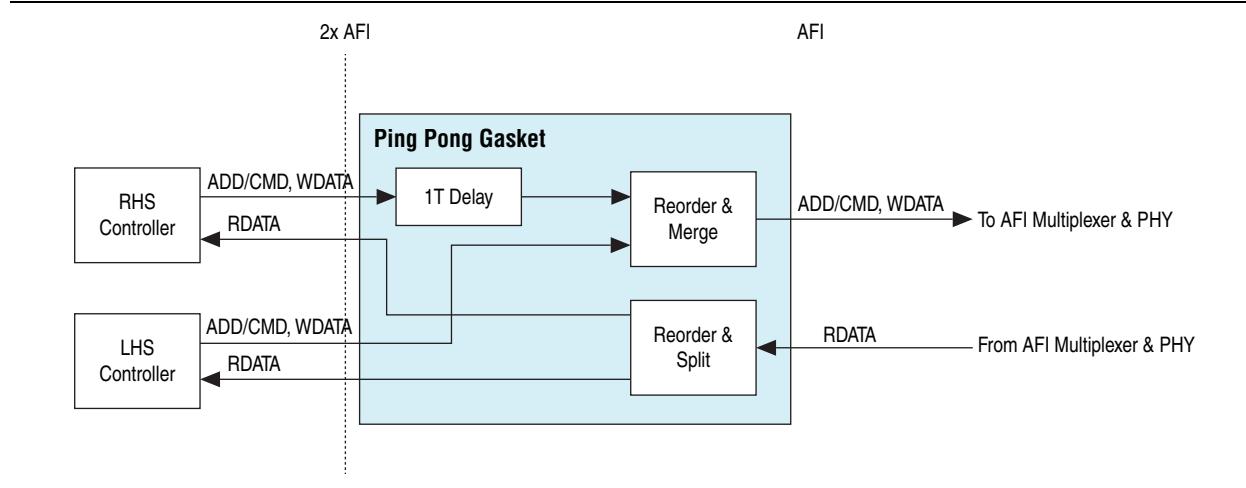


Table 1–17 shows how the gasket processes key AFI signals.

**Table 1–17. Key AFI Signals Processed by Ping Pong PHY Gasket (Part 1 of 2)**

Signal	Direction (Width multiplier)	Description	Gasket Conversions
cas, ras, we, addr, ba	Controller (1x) to PHY (1x)	Address and command buses shared between devices.	Delay RHS by 1T; merge.
cs, odt, cke	Controller (1x) to PHY (2x)	Chip select, on-die termination, and clock enable, one per device.	Delay RHS by 1T; reorder, merge.
wdata, wdata_valid, dqs_burst, dm	Controller (1x) to PHY (2x)	Write datapath signals, one per device.	Delay RHS by 1T; reorder, merge.
rdata_en_rd, rdata_en_rd_full	Controller (1x) to PHY (2x)	Read datapath enable signals indicating controller performing a read operation, one per device.	Delay RHS by 1T.
rdata_rdata_valid	PHY (2x) to Controller (1x)	Read data, one per device.	Reorder; split.
cal_fail, cal_success, seq_busy, wlat, rlat	PHY (1x) to Controller (1x)	Calibration result, one per device.	Pass through.

**Table 1-17. Key AFI Signals Processed by Ping Pong PHY Gasket (Part 2 of 2)**

<b>Signal</b>	<b>Direction (Width multiplier)</b>	<b>Description</b>	<b>Gasket Conversions</b>
rst_n, mem_clk_disable, ctl_refresh_done, ctl_long_idle	Controller (1x) to PHY (1x)	Reset and DQS tracking signals, one per PHY.	AND (&)
cal_req, init_req	Controller (1x) to PHY (1x)	Controller to sequencer requests.	OR ( )
wrank, rrank	Controller (1x) to PHY (2x)	Shadow register support.	Delay RHS by 1T; reorder; merge.

## Calibration

The sequencer treats the Ping Pong PHY as a regular interface of double the width. For example, in the case of two x16 devices, the sequencer calibrates both devices together as a x32 interface. The sequencer chip select signal fans out to both devices so that they are treated as a single interface. The VFIFO calibration process is unchanged. For LFIFO calibration, the LFIFO buffer is duplicated for each interface and the worst-case read datapath delay of both interfaces is used.

## Operation

To use the Ping Pong PHY, you configure a single memory interface according to your requirements, and then select the **Enable Ping Pong PHY** option in the **Advanced PHY Options** section of the **PHY Settings** tab in the DDR3 parameter editor.

The Quartus II software then replicates the interface, resulting in two memory controllers and a shared PHY, with the gasket block inserted between the controllers and PHY. The system makes the necessary modifications to top-level component connections, as well as the PHY read and write datapaths, and the AFI mux, without further input from you.

## Efficiency Monitor and Protocol Checker

The Efficiency Monitor and Protocol Checker is a feature available with the DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY and the RLDRAM II Controller with UniPHY. The Efficiency Monitor and Protocol Checker allows measurement of traffic efficiency on the Avalon-MM bus between the controller and user logic, measures read latencies, and checks the legality of Avalon commands passed from the master. The following sections describe the parts of this feature.

### Efficiency Monitor

The Efficiency Monitor reports read and write throughput on the controller input, by counting command transfers and wait times, and making that information available to the External Memory Interface Toolkit via an Avalon slave port. This information may be useful to you when experimenting with advanced controller settings, such as command look ahead depth and burst merging.

## Protocol Checker

The Protocol Checker checks the legality of commands on the controller's input interface against Altera's Avalon interface specification, and sets a flag in a register on an Avalon slave port if an illegal command is detected.

## Read Latency Counter

The Read Latency Counter measures the minimum and maximum wait times for read commands to be serviced on the Avalon bus. Each read command is time-stamped and placed into a FIFO buffer upon arrival, and latency is determined by comparing that timestamp to the current time when the first beat of the returned read data is provided back to the master.

## Using the Efficiency Monitor and Protocol Checker

To include the Efficiency Monitor and Protocol Checker when you generate your IP core, on the **Diagnostics** tab in the parameter editor, turn on **Enable the Efficiency Monitor and Protocol Checker on the Controller Avalon Interface**.

To see the results of the data compiled by the Efficiency Monitor and Protocol Checker, use the External Memory Interface Toolkit.

 For information on the External Memory Interface Toolkit, refer to [UniPHY External Memory Interface Debug Toolkit](#), in section 2 of this volume. For information about the Avalon interface, refer to [Avalon Interface Specifications](#).

## Avalon CSR Slave and JTAG Memory Map

**Table 1-18** lists the memory map of registers inside the Efficiency Monitor and Protocol Checker; this information is only of interest if you want to communicate directly with the Efficiency Monitor and Protocol Checker without using the External Memory Interface Toolkit. This CSR map is not part of the UniPHY CSR map.

**Table 1-18. Avalon CSR Slave and JTAG Memory Map (Part 1 of 3)**

Address	Bit	Name	Default	Access	Description
0x01	31:0	Reserved	0	—	Used internally by EMIF Toolkit to identify Efficiency Monitor type.
0x02	31:0	Reserved	0	—	Used internally by EMIF Toolkit to identify Efficiency Monitor version.

**Table 1–18. Avalon CSR Slave and JTAG Memory Map (Part 2 of 3)**

<b>Address</b>	<b>Bit</b>	<b>Name</b>	<b>Default</b>	<b>Access</b>	<b>Description</b>
0x08	0	Efficiency Monitor reset	—	Write only	Write a 0 to reset.
	7:1	Reserved	—	—	Reserved for future use.
	8	Protocol Checker reset	—	Write only	Write a 0 to reset.
	15:9	Reserved	—	—	Reserved for future use.
	16	Start/stop Efficiency Monitor	—	Read/Write	Starting and stopping stastics gathering.
	23:17	Reserved	—	—	Reserved for future use.
	31:24	Efficiency Monitor status	—	Read Only	bit 0: Efficiency Monitor stopped bit 1: Waiting for start of pattern bit 2: Running bit 3: Counter saturation
0x10	15:0	Efficiency Monitor address width	—	Read Only	Address width of the Efficiency Monitor.
	31:16	Efficiency Monitor data width	—	Read Only	Data Width of the Efficiency Monitor.
0x11	15:0	Efficiency Monitor byte enable	—	Read Only	Byte enable width of the Efficiency Monitor.
	31:16	Efficiency Monitor burst count width	—	Read Only	Burst count width of the Efficiency Monitor.
0x14	31:0	Cycle counter	—	Read Only	Clock cycle counter for the Efficiency Monitor. Lists the number of clock cycles elapsed before the Efficiency Monitor stopped.
0x18	31:0	Transfer counter	—	Read Only	Counts any read or write data transfer cycle.
0x1C	31:0	Write counter	—	Read Only	Counts write requests, including those during bursts.
0x20	31:0	Read counter	—	Read Only	Counts read requests.
0x24	31:0	Readtotal counter	—	Read Only	Counts read requests (total burst requests).
0x28	31:0	NTC waitrequest counter	—	Read Only	Counts Non Transfer Cycles (NTC) due to slave wait request high.
0x2C	31:0	NTC noreaddatavalid counter	—	Read Only	Counts Non Transfer Cycles (NTC) due to slave not having read data.
0x30	31:0	NTC master write idle counter	—	Read Only	Counts Non Transfer Cycles (NTC) due to master not issuing command, or pause in write burst.
0x34	31:0	NTC master idle counter	—	Read Only	Counts Non Transfer Cycles (NTC) due to master not issuing command anytime.
0x40	31:0	Read latency min	—	Read Only	The lowest read latency value.
0x44	31:0	Read latency max	—	Read Only	The highest read latency value.
0x48	31:0	Read latency total [31:0]	—	Read Only	The lower 32 bits of the total read latency.

**Table 1–18. Avalon CSR Slave and JTAG Memory Map (Part 3 of 3)**

Address	Bit	Name	Default	Access	Description
0x49	31:0	Read latency total [63:32]	—	Read Only	The upper 32 bits of the total read latency.
0x50	7:0	Illegal command	—	Read Only	Bits used to indicate which illegal command has occurred. Each bit represents a unique error.
	31:8	Reserved	—		Reserved for future use.

## UniPHY Calibration Stages

This section describes the calibration stages performed by the DDR2, DDR3, and LPDDR2 SDRAM, QDR II and QDR II+ SRAM, and RLDRAM II Controllers with UniPHY, and the RLDRAM 3 PHY-only IP. This information is useful in debugging calibration failures. The section includes an overview of calibration, explanation of the calibration stages, and a list of generated calibration signals. The information in this section applies only to the Nios II-based sequencer used in the DDR2, DDR3, and LPDDR2 SDRAM Controllers with UniPHY versions 10.0 and later, and, optionally, in the QDR II and QDR II+ SRAM and RLDRAM II Controllers with UniPHY version 11.0 and later, and the RLDRAM 3 PHY-only IP. The information in this section applies to the Arria II GZ, Arria V, Arria V GZ, Cyclone V, Stratix III, Stratix IV, and Stratix V device families.

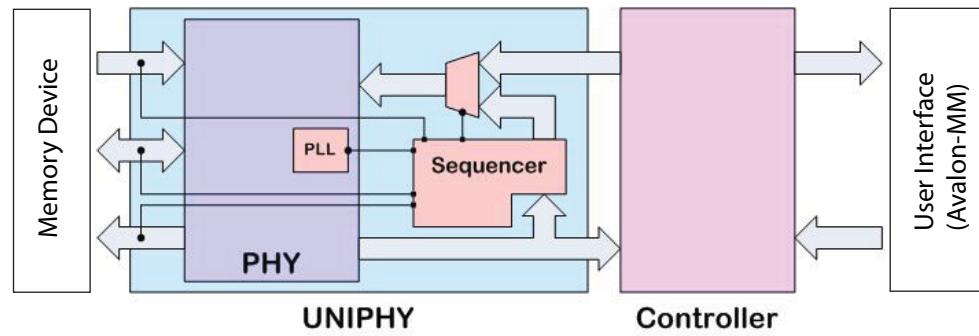
-  For QDR II and QDR II+ SRAM and RLDRAM II Controllers with UniPHY version 11.0 and later, you have the option to select either the RTL-based sequencer or the Nios II-based sequencer. Generally, choose the RTL-based sequencer when area is the major consideration, and choose the Nios II-based sequencer when performance is the major consideration.
-  For RLDRAM 3, write leveling is not performed. The sequencer does not attempt to optimize margin for the tCKDK timing requirement.

## Overview

Calibration configures the memory interface (PHY and I/Os) so that data can pass reliably to and from memory. The sequencer illustrated in [Figure 1–23](#) calibrates the PHY and the I/Os. To correctly transmit data between a memory device and the FPGA at high speed, the data must be center-aligned with the data clock.

Calibration also determines the delay settings needed to center-align the various data signals with respect to their clocks. I/O delay chains implement the required delays in accordance with the computed alignments. The Nios II-based sequencer performs two major tasks: FIFO buffer calibration and I/O calibration. FIFO buffer calibration adjusts FIFO lengths and I/O calibration adjusts any delay chain and phase settings to center-align data signals with respect to clock signals for both reads and writes. When the calibration process completes, the sequencer shuts off and passes control to the memory controller.

**Figure 1–23. Sequencer in Memory Interface Logic**



## Calibration Stages

The calibration process begins when the PHY reset signal deasserts and the PLL and DLL lock. The following stages of calibration take place:

1. Read calibration part one—DQS enable calibration (only for DDR2 and DDR3 SDRAM Controllers with UniPHY) and DQ/DQS centering
2. Write calibration part one—Leveling
3. Write calibration part two—DQ/DQS centering
4. Read calibration part two—Read latency minimization



For multirank calibration, the sequencer transmits every read and write command to each rank in sequence. Each read and write test is successful only if all ranks pass the test. The sequencer calibrates to the intersection of all ranks.

## Assumptions

The calibration process assumes the following conditions; if either of these conditions is not true, calibration likely fails in its early stages:

- The address and command paths must be functional; calibration does not tune the address and command paths. (The Quartus II software fully analyzes the timing for the address and command paths, and the slack report is accurate, assuming the correct board timing parameters.)
- At least one bit per group must work before running per-bit-deskew calibration. (This assumption requires that DQ-to-DQS skews be within the recommended 20 ps.)

## Memory Initialization

The memory is powered up according to protocol initialization specifications. All ranks power up simultaneously. Once powered, the device is ready to receive mode register load commands. This part of initialization occurs separately for each rank. The sequencer issues mode register set commands on a per-chip-select basis and initializes the memory to the user-specified settings.

## Stage 1: Read Calibration Part One—DQS Enable Calibration and DQ/DQS Centering

Read calibration occurs in two parts. Part one is DQS enable calibration with DQ/DQS centering, which happens during stage 1 of the overall calibration process; part two is read latency minimization, which happens during stage 4 of the overall calibration process.

The objectives of DQS enable calibration and DQ/DQS centering are as follows:

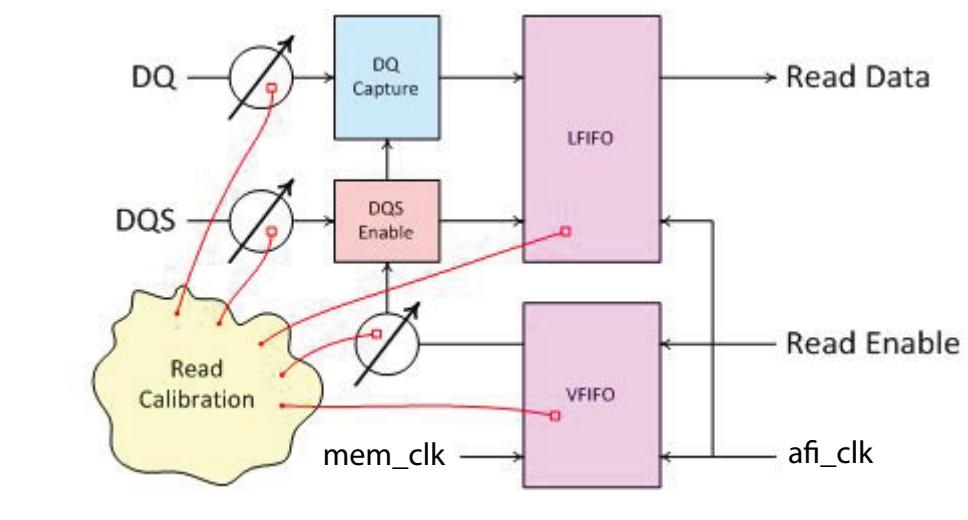
- To calculate when the read data is received after a read command is issued to setup the Data Valid Prediction FIFO (VFIFO) cycle
- To align the input data (DQ) with respect to the clock (DQS) to maximize the read margins (DDR2 and DDR3 only)

DQS enable calibration and DQ/DQS centering consists of the following actions:

- Guaranteed Write
- DQS Enable Calibration
- DQ/DQS Centering

Figure 1-24 illustrates the components in the read data path that the sequencer calibrates in this stage. (The round knobs in the figure represent configurable hardware over which the sequencer has control.)

**Figure 1-24. Read Data Path Calibration Model**

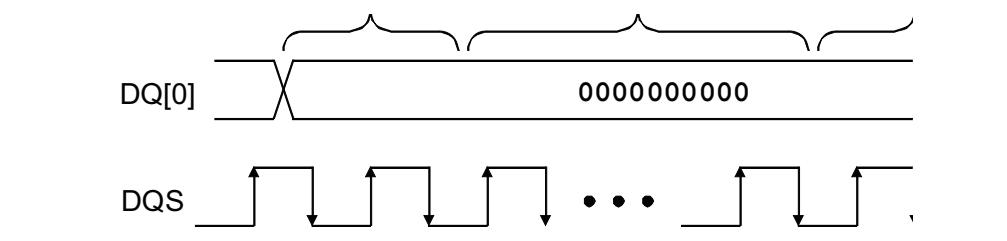


## Guaranteed Write

Since initially no communication can be reliably performed with the memory device, the sequencer uses a guaranteed write mechanism to write data into the memory device. (For the QDR II protocol, guaranteed write is not necessary, a simple write mechanism is sufficient.)

The guaranteed write is a write command issued with all data pins, all address and bank pins, and all command pins (except chip select) held constant. The sequencer begins toggling DQS well before the expected latch time at memory and continues to toggle DQS well after the expected latch time at memory. DQ-to-DQS relationship is not a factor at this stage because DQ is held constant. [Figure 1-25](#) illustrates a guaranteed write of zeros.

**Figure 1-25. Guaranteed Write of Zeros**



The guaranteed write consists of a series of back-to-back writes to alternating columns and banks. For example, for DQ[0] for the DDR3 protocol, the guaranteed write performs the following operations:

- Writes a full burst of zeros to bank 0, column 0
- Writes a full burst of zeros to bank 0, column 1
- Writes a full burst of ones to bank 3, column 0
- Writes a full burst of ones to bank 3, column 1

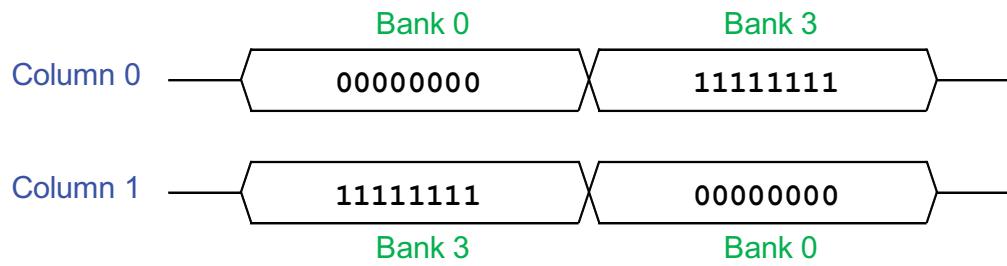
(Different protocols may use different combinations of banks and columns.)

The guaranteed write is followed by back-to-back read operations at alternating banks, effectively producing a stream of zeros followed by a stream of ones, or vice versa. The sequencer uses the zero-to-one and one-to-zero transitions in between the two bursts to identify a correct read operation, as shown in [Figure 1-26](#).

Although the approach described above for pin DQ[0] would work by writing the same pattern to all DQ pins, it is more effective and robust to write (and read) alternating ones and zeros to alternating DQ bits. The value of the DQ bit is still constant across the burst, and the back-to-back read mechanism works exactly as described above, except that odd DQ bits have ones instead of zeros, or vice versa.

The guaranteed write does not ensure a correct DQS-to-memory clock alignment at the memory device—DQS-to-memory clock alignment is performed later, in stage 2 of the calibration process. However, the process of guaranteed write followed by read calibration is repeated several times for different DQS-to-memory clock alignments, to ensure at least one correct alignment is found.

**Figure 1–26. Back to Back Reads on pin DQ[0]**



### DQS Enable Calibration

The full DQS enable calibration is applicable only for DDR2 and DDR3 protocols; QDR II and RLDRAM protocols use only the VFIFO-based cycle-level calibration, described below.

Delay and phase values used in this section are examples, for illustrative purposes. Your exact values may vary depending on device and configuration.

DQS enable calibration ensures reliable capture of the DQ signal without glitches on the DQS line. At this point LFIFO is set to its maximum value to guarantee a reliable read from read capture registers to the core. Read latency is minimized later.

DQS enable calibration controls the timing of the enable signal using 3 independent controls: a cycle-based control (the VFIFO), a phase control, and a delay control. The VFIFO selects the cycle by shifting the controller-generated read data enable signal, `rdata_en`, by a number of full-rate clock cycles. The phase is controlled using the DLL, while the delays are adjusted using a sequence of individual delay taps. The resolution of the phase and delay controls varies with family and configuration, but is approximately 45° for the phase, and between 10 and 50 picoseconds for the delays.

The sequencer finds the two edges of the DQS enable window by searching the space of cycles, phases, and delays (an exhaustive search can usually be avoided by initially assuming the window is at least one phase wide). During the search, to test the current settings, the sequencer issues back-to-back reads from column 0 of bank 0 and bank 3, and column 1 of bank 0 and bank 3, as shown in [Figure 1–26](#). Two full bursts are read and compared with the reference data for each phase and delay setting.

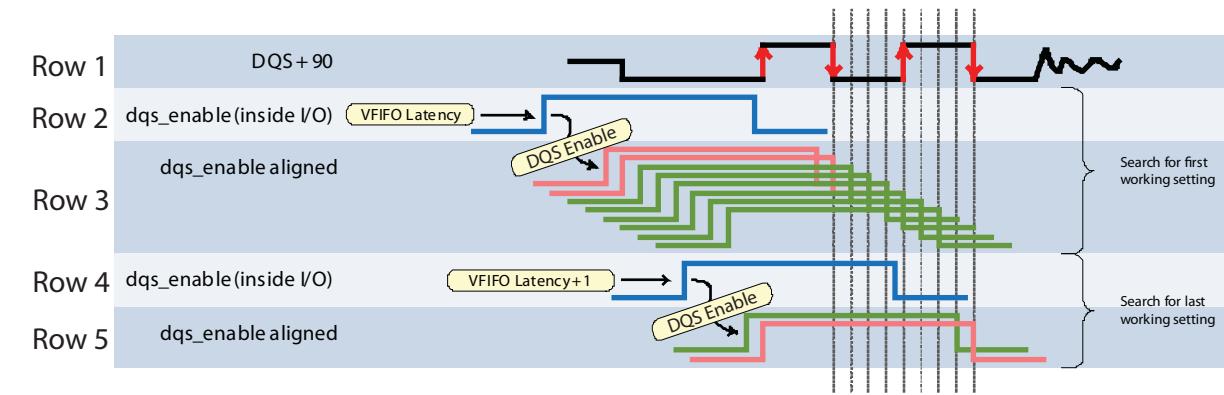
Once the sequencer identifies the two edges of the window, it center-aligns the falling edge of the DQS enable signal within the window. At this point, per-bit deskew has not yet been performed, therefore not all bits are expected to pass the read test; however, for read calibration to succeed, at least one bit per group must pass the read test.

**Figure 1–27** shows the DQS and DQS enable signal relationship. The goal of DQS enable calibration is to find settings that satisfy the following conditions:

- The DQS enable signal rises before the first rising edge of DQS.
- The DQS enable signal is at one after the second-last falling edge of DQS.
- The DQS enable signal falls before the last falling edge of DQS.

The ideal position for the falling edge of the DQS enable signal is centered between the second-last and last falling edges of DQS.

**Figure 1–27. DQS and DQS Enable Signal Relationships**



The following points describe each row of **Figure 1–27**:

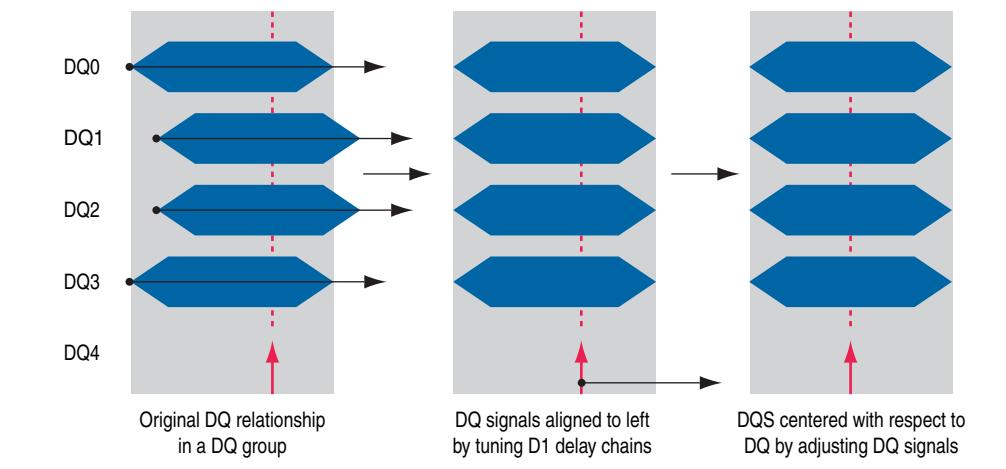
- Row 1 shows the DQS signal shifted by 90° to center-align it to the DQ data.
- Row 2 shows the raw DQS enable signal from the VFIFO.
- Row 3 shows the effect of sweeping DQS enable phases. The first two settings (shown in red) fail to properly gate the DQS signal because the enable signal turns off before the second-last falling edge of DQS. The next six settings (shown in green) gate the DQS signal successfully, with the DQS signal covering DQS from the first rising edge to the second-last falling edge.
- Row 4 shows the raw DQS enable signal from the VFIFO, increased by one clock cycle relative to Row 2.
- Row 5 shows the effect of sweeping DQS enable, beginning from the initial DQS enable of Row 4. The first setting (shown in green) successfully gates DQS, with the signal covering DQS from the first rising edge to the second-last falling edge. The second signal (shown in red), does not gate DQS successfully because the enable signal extends past the last falling edge of DQS. Any further adjustment would show the same failure.

## Centering DQ/DQS

The centering DQ/DQS stage attempts to align DQ and DQS signals on reads within a group. Each DQ signal within a DQS group might be skewed and consequently arrive at the FPGA at a different time. At this point, the sequencer sweeps each DQ signal in a DQ group to align them, by adjusting DQ input delay chains (D1).

Figure 1–28 illustrates a four DQ/DQS group per-bit-deskew and centering.

**Figure 1–28. Per-bit Deskew**



To align and center DQ and DQS, the sequencer finds the right edge of DQ signals with respect to DQS by sweeping DQ signals within a DQ group to the right until a failure occurs. In Figure 1–28, DQ0 and DQ3 fail after six taps to the right; DQ1 and DQ2 fail after 5 taps to the right. To align the DQ signals, DQ0 and DQ3 are shifted to the right by 1 tap.

To find the center of DVW, the DQS signal is shifted to the right until a failure occurs. In Figure 1–28, a failure occurs after 3 taps, meaning that there are 5 taps to the right edge and 3 taps to the left edge. To center-align DQ and DQS, the sequencer shifts the aligned DQ signal by 1 more tap to the right.

The sequencer does not adjust DQS directly; instead, the sequencer center-aligns DQS with respect to DQ by delaying the DQ signals.

## Stage 2: Write Calibration Part One

The objectives of the write calibration stage are to align DQS to the memory clock at each memory device, and to compensate for address, command, and memory clock skew at each memory device. This stage is important because the address, command, and clock signals for each memory component arrive at different times.

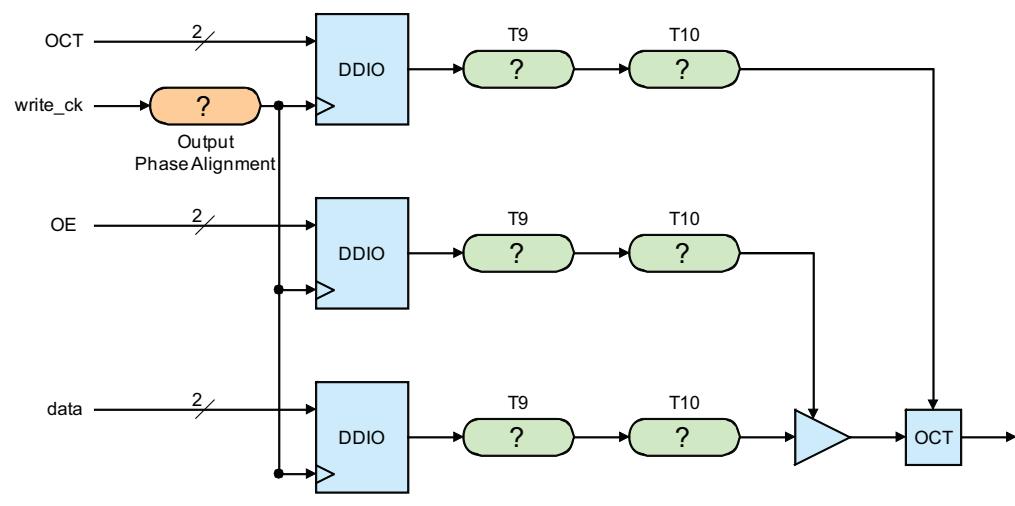
This stage applies only to DDR2, DDR3, LPDDR2, and RLDRAM II protocols; it does not apply to the QDR II and QDR II+ protocols.

Memory clock signals and DQ/DM and DQS signals have specific relationships mandated by the memory device. The PHY must ensure that these relationships are met by skewing DQ/DM and DQS signals. The relationships between DQ/DM and DQS and memory clock signals must meet the tDQSS, tDSS, and tDSH timing constraints.

The sequencer calibrates the write data path using a variety of random burst patterns to compensate for the jitter on the output data path. Simple write patterns are insufficient to ensure a reliable write operation because they might cause imprecise DQS-to-CK alignments, depending on the actual capture circuitry on a memory device. The write patterns in the write leveling stage have a burst length of 8, and are generated by a linear feedback shift register in the form of a pseudo-random binary sequence.

The write data path architecture is the same for DQ, DM, and DQS pins. [Figure 1-29](#) illustrates the write data path for a DQ signal. The phase coming out of the Output Phase Alignment block can be set to different values to center-align DQS with respect to DQ, and it is the same for data, OE, and OCT of a given output.

**Figure 1-29. Write Data Path**



In write leveling, the sequencer performs write operations with different delay and phase settings, followed by a read. The sequencer can implement any phase shift between 0° and 720° (depending on device and configuration). The sequencer uses the Output Phase Alignment for coarse delays and D5 and D6 for fine delays; D5 has 15 taps of 50 ps each, and D6 has 7 taps of 50 ps each. The DQS signal phase is held at +90° with respect to DQ signal phase (Stratix IV example).

- 👉 Coarse delays are called *phases*, and fine delays are called *delays*; phases are process, voltage, and temperature (PVT) compensated, delays are not (depending on family).
- 👉 Delay and phase values used in this section are examples, for illustrative purposes. Your exact values may vary depending on device and configuration.

The sequencer writes and reads back several burst-length-8 patterns. Because the sequencer has not performed per-bit deskew on the write data path, not all bits are expected to pass the write test. However, for write calibration to succeed, at least one bit per group must pass the write test. The test begins by shifting the DQ/DQS phase until the first write operation completes successfully. The DQ/DQS signals are then delayed to the left by D5 and D6 to find the left edge for that working phase. Then DQ/DQS phase continues the shift to find the last working phase. For the last working phase, DQ/DQS is delayed in 50 ps steps to find the right edge of the last working phase.

The sequencer sweeps through all possible phase and delay settings for each DQ group where the data read back is correct, to define a window within which the PHY can reliably perform write operations. The sequencer picks the closest value to the center of that window as the phase/delay setting for the write data path.

### **Stage 3: Write Calibration Part Two—DQ/DQS Centering**

The process of DQ/DQS centering in write calibration is similar to that performed in read calibration, except that write calibration is performed on the output path, using D5 and D6 delay chains.

### **Stage 4: Read Calibration Part Two—Read Latency Minimization**

At this stage of calibration the sequencer adjusts LFIFO latency to determine the minimum read latency that guarantees correct reads.

#### **Read Latency Tuning**

In general, DQ signals from different DQ groups may arrive at the FPGA in a staggered fashion. In a DIMM or multiple memory device system, the DQ/DQS signals from the first memory device arrive sooner, while the DQ/DQS signals from the last memory device arrive the latest at the FPGA.

LFIFO transfers data from the capture registers in IOE to the core and aligns read data to the AFI clock. Up to this point in the calibration process, the read latency has been a maximum value set initially by LFIFO; now, the sequencer progressively lowers the read latency until the data can no longer be transferred reliably. The sequencer then increases the latency by one cycle to return to a working value and adds an additional cycle of margin to assure reliable reads.

## **Calibration Signals**

Table 1-19 lists signals produced by the calibration process.

**Table 1-19. Calibration Signals**

Signal	Description
afi_cal_fail	Asserts high if calibration fails.
afi_cal_success	Asserts high if calibration is successful.

## Calibration Time

The time needed for calibration varies, depending on many factors including the interface width, the number of ranks, frequency, board layout, and difficulty of calibration. In general, designs using the Nios II-based sequencer will take longer to calibrate than designs using the RTL-based sequencer.

Table 1–20 lists approximate typical and maximum calibration times for various protocols.

**Table 1–20. Approximate Calibration Times**

Protocol	Typical	Maximum
DDR2, DDR3, LPDDR2, RLDRAM 3	50-250 ms	Can take several minutes if the interface is difficult to calibrate, or if calibration initially fails and exhausts multiple retries.
QDR II/II+, RLDRAM II (with Nios II-based sequencer)	50-100 ms	Can take several minutes if the interface is difficult to calibrate, or if calibration initially fails and exhausts multiple retries.
QDR II/II+, RLDRAM II (with RTL-based sequencer)	<5 ms	<5 ms

## Document Revision History

Table 1–21 lists the revision history for this document.

**Table 1–21. Document Revision History**

Date	Version	Changes
November 2012	3.1	<ul style="list-style-type: none"> <li>■ Moved <a href="#">Controller Register Map</a> to <a href="#">Functional Description—HPC II Controller</a> chapter.</li> <li>■ Updated Sequencer States information in <a href="#">Table 1–2</a>.</li> <li>■ Enhanced <a href="#">Using a Custom Controller</a> information.</li> <li>■ Enhanced <a href="#">Tracking Manager</a> information.</li> <li>■ Added <a href="#">Ping Pong PHY</a> information.</li> <li>■ Added RLDRAM 3 support.</li> <li>■ Added LRDIMM support.</li> <li>■ Added Arria V GZ support.</li> </ul>
June 2012	3.0	<ul style="list-style-type: none"> <li>■ Added <a href="#">Shadow Registers</a> section.</li> <li>■ Added LPDDR2 support.</li> <li>■ Added new AFI signals.</li> <li>■ Added <a href="#">Calibration Time</a> section.</li> <li>■ Added Feedback icon.</li> </ul>
November 2011	2.1	<ul style="list-style-type: none"> <li>■ Consolidated UniPHY information from <a href="#">11.0 DDR2 and DDR3 SDRAM Controller with UniPHY User Guide</a>, <a href="#">QDR II and QDR II+ SRAM Controller with UniPHY User Guide</a>, and <a href="#">RLDRAM II Controller with UniPHY IP User Guide</a>.</li> <li>■ Revised <a href="#">Reset and Clock Generation</a> and <a href="#">Dedicated Clock Networks</a> sections.</li> <li>■ Revised <a href="#">Figure 1–3</a> and <a href="#">Figure 1–5</a>.</li> <li>■ Added Tracking Manager to <a href="#">Sequencer</a> section.</li> <li>■ Revised <a href="#">Interfaces</a> section for DLL, PLL, and OCT sharing interfaces.</li> <li>■ Revised <a href="#">Using a Custom Controller</a> section.</li> <li>■ Added <a href="#">UniPHY Calibration Stages</a> section; reordered stages 3 and 4, removed stage 5.</li> </ul>



The ALTMEMPHY megafunction creates the datapath between the memory device and the memory controller, and user logic in various Altera devices. The ALTMEMPHY megafunction GUI helps you configure multiple variations of a memory interface. You can then connect the ALTMEMPHY megafunction variation with either a user-designed controller or with the Altera high-performance controller. In addition, the ALTMEMPHY megafunction and the Altera high-performance controller are available for half-rate DDR3 SDRAM interfaces.



If the ALTMEMPHY megafunction does not meet your requirements, you can also create your own memory interface datapath using the ALTDL and ALTDQ\_DQS megafunctions, available in the Quartus II software. However, you are then responsible for every aspect of the interface, including timing analysis and debugging.

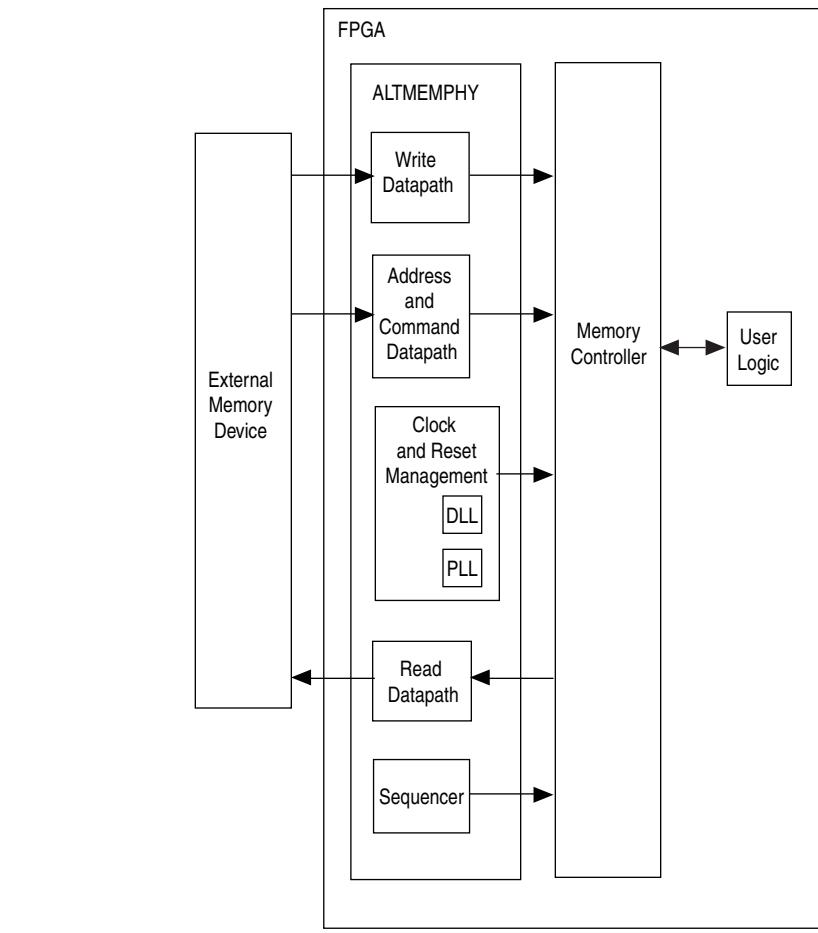
This chapter describes the DDR3 SDRAM ALTMEMPHY megafunction, which uses AFI as the interface between the PHY and the controller.



## Block Description

Figure 2–1 shows the major blocks of the ALTMEMPHY megafunction and how it interfaces with the external memory device and the controller. The ALTPLL megafunction is instantiated inside the ALTMEMPHY megafunction, so that you do not need to generate the clock to any of the ALTMEMPHY blocks.

**Figure 2–1. ALTMEMPHY Megafunction Interfacing with the Controller and the External Memory**



The ALTMEMPHY megafunction comprises the following blocks:

- Write datapath
- Address and command datapath
- Clock and reset management, including DLL and PLL
- Sequencer for calibration
- Read datapath

The major advantage of the ALTMEMPHY megafunction is that it supports an initial calibration sequence to remove process variations in both the Altera device and the memory device. In Arria series devices, the DDR3 SDRAM ALTMEMPHY calibration process centers the resynchronization clock phase into the middle of the captured data valid window to maximize the resynchronization setup and hold margin. During the user operation, the VT tracking mechanism eliminates the effects of VT variations on resynchronization timing margin.

## Calibration

The sequencer performs calibration to find the optimal clock phase for the memory interface.

For information about calibration, refer to [ALTMEMPHY Calibration Stages](#).

## Address and Command Datapath

This topic discusses the address and command datapath.

### Arria II GX Devices

The address and command datapath is responsible for taking the address and command outputs from the controller and converting them from half-rate clock to full-rate clock. Two types of addressing are possible:

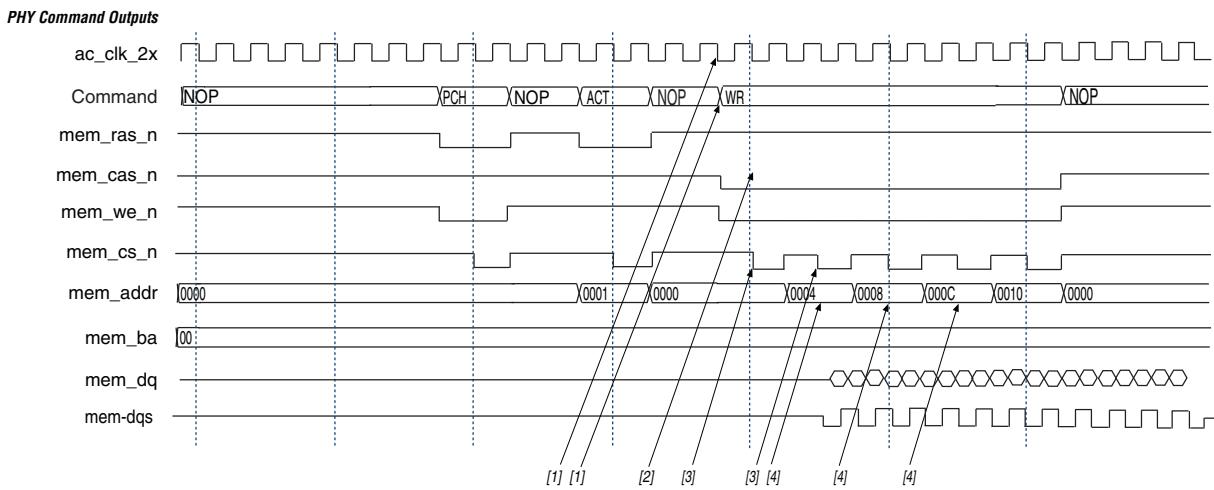
- 1T (full rate)—the duration of the address and command is a single memory clock cycle (`mem_clk_2x`, [Figure 2-2](#)). This applies to all address and command signals in full-rate designs or `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate designs.
- 2T (half rate)—the duration of the address and command is two memory clock cycles. For half-rate designs, the ALTMEMPHY megafunction supports only a burst size of four, which means the burst size on the local interface is always set to 1. The size of the data is  $4n$ -bits wide on the local side and is  $n$ -bits wide on the memory side. To transfer all the  $4n$ -bits at the double data rate, two memory-clock cycles are required. The new address and command can be issued to memory every two clock cycles. This scheme applies to all address and command signals, except for `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate mode.



Refer to [Table 2-1 on page 2-6](#) to see the frequency relationship of `mem_clk_2x` with the rest of the clocks.

Figure 2–2 shows a 1T chip select signal (`mem_cs_n`), which is active low, and disables the command in the memory device. All commands are masked when the chip-select signal is inactive. The `mem_cs_n` signal is considered part of the command code.

**Figure 2–2. Arria II GX Address and Command Datapath**



The command interface is made up of the signals `mem_ras_n`, `mem_cas_n`, `mem_we_n`, `mem_cs_n`, `mem_cke`, and `mem_odt`.

The waveform in Figure 2–2 shows a NOP command followed by five back-to-back write commands. The following sequence corresponds with the numbered items in Figure 2–2.

1. The commands are asserted either on the rising edge of `ac_clk_2x`. The `ac_clk_2x` is derived from either `mem_clk_2x` ( $0^\circ$ ), `write_clk_2x` ( $270^\circ$ ), or the inverted variations of those two clocks (for  $180^\circ$  and  $90^\circ$  phase shifts). This depends on the setting of the address and command clock in the ALTMEMPHY parameter editor. Refer to “Address and Command Datapath” on page 2–3 for illustrations of this clock in relation to the `mem_clk_2x` or `write_clk_2x` signals.
2. All address and command signals (except for `mem_cs_n`, `mem_cke`, and `mem_odt` signals) remain asserted on the bus for two clock cycles, allowing sufficient time for the signals to settle.
3. The `mem_cs_n`, `mem_cke`, and `mem_odt` signals are asserted during the second cycle of the address/command phase. By asserting the chip-select signal in alternative cycles, back-to-back read or write commands can be issued.
4. The address is incremented every other `ac_clk_2x` cycle.

The `ac_clk_2x` clock is derived from either `mem_clk_2x` (when you choose  $0^\circ$  or  $180^\circ$  phase shift) or `write_clk_2x` (when you choose  $90^\circ$  or  $270^\circ$  phase shift).

The address and command clock can be  $0$ ,  $90$ ,  $180$ , or  $270^\circ$  from the system clock.

## Clock and Reset Management

The clocking and reset block is responsible for clock generation, reset management, and phase shifting of clocks. It also has control of clock network types that route the clocks, which is handled in the `<variation_name>_alt_mem_phy_clk_reset` module in the `<variation_name>_alt_mem_phy.v/.vhdl` file.

### Clock Management

The clock management feature allows the ALTMEMPHY megafunction to work out the optimum phase during calibration, and to track voltage and temperature variation relies on phase shifting the clocks relative to each other.

-  Certain clocks require phase shifting during the ALTMEMPHY megafunction operation.

You can implement clock management circuitry using PLLs and DLLs.

The ALTMEMPHY MegaWizard Plug-In Manager automatically generates an ALTPPLL megafunction instance. The ALTPPLL megafunction generates the different clock frequencies and relevant phases used within the ALTMEMPHY megafunction.

The available device families have different PLL capabilities. The minimum PHY requirement is to have 16 phases of the highest frequency clock. The PLL uses **With No Compensation** option to minimize jitter. Changing the PLL compensation to a different operation mode may result in inaccurate timing results.

The input clock to the PLL does not have any other fan-out to the PHY, so you do not have to use a global clock resource for the path between the clock input pin to the PLL. You must use the PLL located in the same device quadrant or side as the memory interface and the corresponding clock input pin for that PLL, to ensure optimal performance and accurate timing results from the Quartus II software.

You must choose a PLL and PLL input clock pin that are located on the same side of the device as the memory interface to ensure minimal jitter. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset (by driving the `global_reset_n` signal low) and relock the PLL to ensure that the phase relationship between all PLL outputs is properly set.

-  If the design cascades PLLs, the source (upstream) PLL should have a low-bandwidth setting, and the destination (downstream) PLL should have a high-bandwidth setting. Adjacent PLLs cascading is recommended to reduce clock jitter.
-  For more information about the VCO frequency range and the available phase shifts, refer to the *Clock Networks and PLLs* chapter in the respective device family handbook.

Table 2–1 shows the clock outputs that Arria II GX devices use.

**Table 2–1. DDR3 SDRAM Clocking in Arria II GX Devices (Part 1 of 2)**

Clock Name <sup>(1)</sup>	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type		Notes
				All Quadrants	Any 3 Quadrants <sup>(2)</sup>	
phy_clk_1x and aux_half_rate_clk	C0	0°	Half-Rate	Global	Global	The only clocks parameterizable for the ALTMEMPHY megafunction. These clocks also feed into a divider circuit to provide the PLL scan_clk signal (for reconfiguration) that must be lower than 100 MHz.
mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Regional <sup>(3)</sup> Global <sup>(4)</sup>	This clock is for clocking DQS and as a reference clock for the memory devices.
mem_clk_1x	C2	0°	Half-Rate	Global	Regional	This clock is for clocking DQS and as a reference clock for the memory devices.
write_clk_2x	C3	-90°	Full-Rate	Global	Regional	This clock is for clocking the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the mem_clk_2x by 90°.
ac_clk_2x	C3	-90°	Full-Rate	Global	Regional	Address and command clock. The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift). Refer to <a href="#">“Address and Command Datapath” on page 2–3</a> for illustrations of the address and command clock relationship with the mem_clk_2x or write_clk_2x signals.
cs_n_clk_2x	C3	-90°	Full-Rate	Global	Global	Memory chip-select clock. The cs_n_clk_2x clock is derived from ac_clk_2x.
resync_clk_2x	C4	Calibrated	Full-Rate	Global	Regional	Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-coded DDIO groups.

**Table 2–1. DDR3 SDRAM Clocking in Arria II GX Devices (Part 2 of 2)**

Clock Name <sup>(1)</sup>	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type		Notes
				All Quadrants	Any 3 Quadrants <sup>(2)</sup>	
measure_clk_2x	C5	Calibrated	Full-Rate	Global	Regional	This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects.

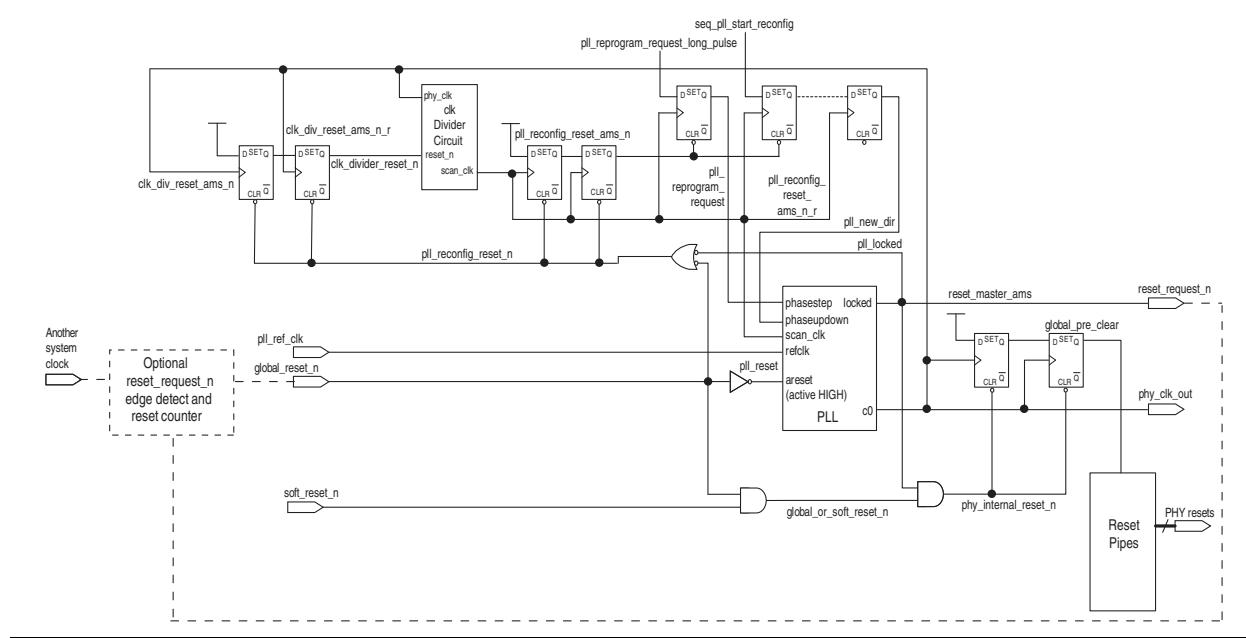
**Note to Table 2–1:**

- (1) The `_1x` clock represents a frequency that is half of the memory clock frequency; the `_2x` clock represents the memory clock frequency.
- (2) The default clock network type is Global, however you can specify a regional clock network to improve clock jitter if your design uses any three quadrants.
- (3) For `mem_clk2x`.
- (4) For `aux_full_rate_clk`.

## Reset Management

Figure 2–3 shows the main features of the reset management block for the DDR3 SDRAM PHY. You can use the `pll_ref_clk` input to feed the optional `reset_request_n` edge detect and reset counter module. However, this requires the `pll_ref_clk` signal to use a global clock network resource.

There is a unique reset metastability protection circuit for the clock divider circuit because the `phy_clk` domain reset metastability protection registers have fan-in from the `soft_reset_n` input so these registers cannot be used.

**Figure 2-3.** ALTMEMPHY Reset Management Block for Arria II GX Devices

## Read Datapath

This topic discusses the read datapath.

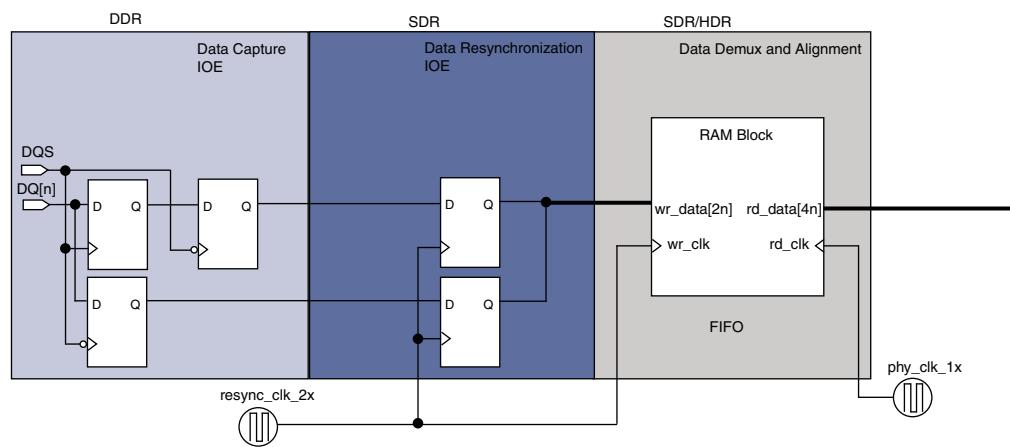
### Arria II GX Devices

The read datapath logic captures data sent by the memory device and subsequently aligns the data back to the system clock domain. The read datapath for DDR3 SDRAM consists of the following three main blocks:

- Data capture
- Data resynchronization
- Data demultiplexing and alignment

As the DQS/DQSn signal is not continuous, the PHY also has postamble protection logic to ensure that any glitches on the DQS input signals at the end of the read postamble time do not cause erroneous data to be captured as a result of postamble glitches.

Figure 2-4 shows the order of the functions performed by the read datapath and the frequency at which the read data is handled.

**Figure 2–4. DDR3 SDRAM Read Datapath in Arria II GX Devices**

### Data Capture and Resynchronization

The data capture and resynchronization registers for Arria II GX devices are implemented in the I/O element (IOE) to achieve maximum performance. Data capture and resynchronization is the process of capturing the read data (DQ) with the DQS/DQSn strobes and resynchronizing the captured data to an internal free-running full-rate clock supplied by the enhanced PLL. The resynchronization clock is an intermediate clock whose phase shift is determined during the calibration stage. The captured data (`rdata_p_captured` and `rdata_n_captured`) is synchronized to the resynchronization clock (`resync_clk_2x`), refer to [Figure 2–4](#). For Arria II GX devices, the ALTMEMPHY instances an ALTDQ\_DQS megafunction that instantiates the required IOEs for all the DQ and DQS pins.

### Data Demultiplexing

Data demultiplexing is the process of changing the SDR data into HDR data. Data demultiplexing is required to bring the frequency of the resynchronized data down to the frequency of the system clock, so that data from the external memory device can ultimately be brought into the FPGA controller clock domain. Before data capture, the data is DDR and  $n$ -bit wide. After data capture, the data is SDR and  $2n$ -bit wide. After data demuxing, the data is HDR of width  $4n$ -bits wide. The system clock frequency is half the frequency of the memory clock. Demultiplexing is achieved using a dual-port memory with a  $2n$ -bit wide write-port operating on the resynchronization clock (SDR) and a  $4n$ -bit wide read-port operating on the PHY clock (HDR). The basic principle of operation is that data is written to the memory at the SDR rate and read from the memory at the HDR rate while incrementing the read- and write-address pointers. As the SDR and HDR clocks are generated, the read and write pointers are continuously incremented by the same PLL, and the  $4n$ -bit wide read data follows the  $2n$ -bit wide write data with a constant latency

### Read Data Alignment

Data alignment is the process controlled by the sequencer to ensure the correct captured read data is present in the same half-rate clock cycle at the output of the read data DPRAM. Data alignment is implemented using memory blocks in the core of devices.

### Postamble Protection

A dedicated postamble register controls the gating of the shifted DQS signal that clocks the DQ input registers at the end of a read operation. Any glitches on the DQS input signals at the end of the read postamble time do not cause erroneous data to be captured as a result of postamble glitches. The postamble path is also calibrated to determine the correct clock cycle, clock phase shift, and delay chain settings.

## ALTMEMPHY Signals

This section describes the ALMEMPHY megafunction signals for DDR3 SDRAM variants.

Table 2–2 through Table 2–4 show the signals.



Signals with the prefix `mem_` connect the PHY with the memory device; ports with the prefix `ctl_` connect the PHY with the controller.

The signal lists include the following signal groups:

- I/O interface to the SDRAM devices
- Clocks and resets
- External DLL signals
- User-mode calibration OCT control
- Write data interface
- Read data interface
- Address and command interface
- Calibration control and status interface
- Debug interface

**Table 2–2. Interface to the DDR3 SDRAM Devices (Part 1 of 2) <sup>(1)</sup>**

Signal Name	Type	Width <sup>(2)</sup>	Description
<code>mem_addr</code>	Output	<code>MEM_IF_ROWADDR_WIDTH</code>	The memory row and column address bus.
<code>mem_ba</code>	Output	<code>MEM_IF_BANKADDR_WIDTH</code>	The memory bank address bus.
<code>mem_cas_n</code>	Output	1	The memory column address strobe.
<code>mem_cke</code>	Output	<code>MEM_IF_CS_WIDTH</code>	The memory clock enable.
<code>mem_clk</code>	Bidirectional	<code>MEM_IF_CLK_PAIR_COUNT</code>	The memory clock, positive edge clock. <sup>(3)</sup>
<code>mem_clk_n</code>	Bidirectional	<code>MEM_IF_CLK_PAIR_COUNT</code>	The memory clock, negative edge clock.
<code>mem_cs_n</code>	Output	<code>MEM_IF_CS_WIDTH</code>	The memory chip select signal.
<code>mem_dm</code>	Output	<code>MEM_IF_DM_WIDTH</code>	The optional memory DM bus.
<code>mem_dq</code>	Bidirectional	<code>MEM_IF_DWIDTH</code>	The memory bidirectional data bus.
<code>mem_dqs</code>	Bidirectional	<code>MEM_IF_DWIDTH/</code> <code>MEM_IF_DQ_PER_DQS</code>	The memory bidirectional data strobe bus.
<code>mem_dqs_n</code>	Bidirectional	<code>MEM_IF_DWIDTH/</code> <code>MEM_IF_DQ_PER_DQS</code>	The memory bidirectional data strobe bus.
<code>mem_odt</code>	Output	<code>MEM_IF_CS_WIDTH</code>	The memory on-die termination control signal.

**Table 2–2. Interface to the DDR3 SDRAM Devices (Part 2 of 2) <sup>(1)</sup>**

Signal Name	Type	Width <sup>(2)</sup>	Description
mem_ras_n	Output	1	The memory row address strobe.
mem_reset_n	Output	1	The memory reset signal. This signal is derived from the PHY's internal reset signal, which is generated by gating the global reset, soft reset, and the PLL locked signal.
mem_we_n	Output	1	The memory write enable signal.
mem_ac_parity <sup>(4)</sup>	Output	1	The address or command parity signal generated by the PHY and sent to the DIMM.
parity_error_n <sup>(4)</sup>	Output	1	The active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset.
mem_err_out_n <sup>(4)</sup>	Input	1	The signal sent from the DIMM to the PHY to indicate that a parity error has occurred for a particular cycle.

**Notes to Table 2–2:**

- (1) Connected to I/O pads.
- (2) Refer to Table 2–5 for parameter description.
- (3) Output is for memory device, and input path is fed back to ALTMEMPHY megafunction for VT tracking.
- (4) This signal is for Registered DIMMs only.

**Table 2–3. AFI Signals (Part 1 of 4)**

Signal Name	Type	Width <sup>(1)</sup>	Description
<b>Clocks and Resets</b>			
pll_ref_clk	Input	1	The reference clock input to the PHY PLL.
global_reset_n	Input	1	Active-low global reset for PLL and all logic in the PHY. A level set reset signal, which causes a complete reset of the whole system. The PLL may maintain some state information.
soft_reset_n	Input	1	Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. Causes a complete reset of PHY, but not the PLL used in the PHY.
reset_request_n	Output	1	Directly connected to the locked output of the PLL and is intended for optional use either by automated tools such as SOPC Builder or could be manually ANDed with any other system-level signals and combined with any edge detect logic as required and then fed back to the <code>global_reset_n</code> input.  Reset request output that indicates when the PLL outputs are not locked. Use this as a reset request input to any system-level reset controller you may have. This signal is always low while the PLL is locking (but not locked), and so any reset logic using it is advised to detect a reset request on a falling-edge rather than by level detection.

**Table 2–3. AFI Signals (Part 2 of 4)**

<b>Signal Name</b>	<b>Type</b>	<b>Width (1)</b>	<b>Description</b>
ctl_clk	Output	1	Half-rate clock supplied to controller and system logic. The same signal as the non-AFI phy_clk.
ctl_reset_n	Output	1	Reset output on ctl_clk clock domain.
<b>Other Signals</b>			
aux_half_rate_clk	Output	1	In half-rate designs, a copy of the phy_clk_1x signal that you can use in other parts of your design, same as phy_clk port.
aux_full_rate_clk	Output	1	In full-rate designs, a copy of the mem_clk_2x signal that you can use in other parts of your design.
aux_scan_clk	Output	1	Low frequency scan clock supplied primarily to clock any user logic that interfaces to the PLL and DLL reconfiguration interfaces.
aux_scan_clk_reset_n	Output	1	This reset output asynchronously asserts (drives low) when global_reset_n is asserted and de-assert (drives high) synchronous to aux_scan_clk when global_reset_n is de-asserted. It allows you to reset any external circuitry clocked by aux_scan_clk.
<b>Write Data Interface</b>			
ctl_dqs_burst	Input	$\text{MEM\_IF\_DQS\_WIDTH} \times \text{DWIDTH\_RATIO} / 2$	When asserted, mem_dqs is driven. The ctl_dqs_burst signal must be asserted before the ctl_wdata_valid signal and must be driven for the correct duration to generate a correctly timed mem_dqs signal.
ctl_wdata_valid	Input	$\text{MEM\_IF\_DQS\_WIDTH} \times \text{DWIDTH\_RATIO} / 2$	Write data valid. Generates ctl_wdata and ctl_dm output enables.
ctl_wdata	Input	$\text{MEM\_IF\_DWIDTH} \times \text{DWIDTH\_RATIO}$	Write data input from the controller to the PHY to generate mem_dq.
ctl_dm	Input	$\text{MEM\_IF\_DM\_WIDTH} \times \text{DWIDTH\_RATIO}$	DM input from the controller to the PHY.
ctl_wlat	Output	5	Required write latency between address/command and write data that is issued to ALTMEMPHY controller local interface. This signal is only valid when the ALTMEMPHY sequencer successfully completes calibration, and does not change at any point during normal operation. The legal range of values for this signal is 0 to 31; and the typical values are between 0 and ten, 0 mostly for low CAS latency DDR memory types.

**Table 2–3. AFI Signals (Part 3 of 4)**

Signal Name	Type	Width (1)	Description
<b>Read Data Interface</b>			
ctl_doing_rd	Input	MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2	Doing read input. Indicates that the DDR3 SDRAM controller is currently performing a read operation. The controller generates <code>ctl_doing_rd</code> to the ALTMEMPHY megafunction. The <code>ctl_doing_rd</code> signal is asserted for one <code>phy_clk</code> cycle for every read command it issues. If there are two read commands, <code>ctl_doing_rd</code> is asserted for two <code>phy_clk</code> cycles. The <code>ctl_doing_rd</code> signal also enables the capture registers and generates the <code>ctl_mem_rdata_valid</code> signal. The <code>ctl_doing_rd</code> signal should be issued at the same time the read command is sent to the ALTMEMPHY megafunction.
ctl_rdata	Output	DWIDTH_RATIO × MEM_IF_DWIDTH	Read data from the PHY to the controller.
ctl_rdata_valid	Output	DWIDTH_RATIO/2	Read data valid indicating valid read data on <code>ctl_rdata</code> . This signal is two-bits wide (as only half-rate or <code>DWIDTH_RATIO = 4</code> is supported) to allow controllers to issue reads and writes that are aligned to either the half-cycle of the half-rate clock.
ctl_rlat	Output	READ_LAT_WIDTH	Contains the number of clock cycles between the assertion of <code>ctl_doing_rd</code> and the return of valid read data ( <code>ctl_rdata</code> ). This signal is unused by the Altera high-performance controller.
<b>Address and Command Interface</b>			
ctl_addr	Input	MEM_IF_ROWADDR_WIDTH × DWIDTH_RATIO / 2	Row address from the controller.
ctl_ba	Input	MEM_IF_BANKADDR_WIDTH × DWIDTH_RATIO / 2	Bank address from the controller.
ctl_cke	Input	MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2	Clock enable from the controller.
ctl_cs_n	Input	MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2	Chip select from the controller.
ctl_odi	Input	MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2	On-die-termination control from the controller.
ctl_ras_n	Input	DWIDTH_RATIO / 2	Row address strobe signal from the controller.
ctl_we_n	Input	DWIDTH_RATIO / 2	Write enable.
ctl_cas_n	Input	DWIDTH_RATIO / 2	Column address strobe signal from the controller.
ctl_RST_n	Input	DWIDTH_RATIO / 2	Reset from the controller.
<b>Calibration Control and Status Interface</b>			
ctl_mem_clk_disable	Input	MEM_IF_CLK_PAIR_COUNT	When asserted, <code>mem_clk</code> and <code>mem_clk_n</code> are disabled.
ctl_cal_success	Output	1	A 1 indicates that calibration was successful.
ctl_cal_fail	Output	1	A 1 indicates that calibration has failed.

**Table 2–3. AFI Signals (Part 4 of 4)**

Signal Name	Type	Width <sup>(1)</sup>	Description
ctl_cal_req	Input	1	When asserted, a new calibration sequence is started. Currently not supported.
ctl_cal_byte_lane_sel_n	Input	MEM_IF_DQS_WIDTH × MEM_CS_WIDTH	Indicates which DQS groups should be calibrated. Not supported.

**Note to Table 2–2:**

(1) Refer to Table 2–5 for parameter descriptions.

**Table 2–4. Other Interface Signals (Part 1 of 2)**

Signal Name	Type	Width	Description
<b>External DLL Signals</b>			
dqs_delay_ctrl_export	Output	DQS_DELAY_CTL_WI_DTH	Allows sharing DLL in this ALTMEMPHY instance with another ALTMEMPHY instance. Connect the dqs_delay_ctrl_export port on the ALTMEMPHY instance with a DLL to the dqs_delay_ctrl_import port on the other ALTMEMPHY instance.
dqs_delay_ctrl_import	Input	DQS_DELAY_CTL_WI_DTH	Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the dqs_delay_ctrl_export port on the ALTMEMPHY instance with a DLL to the dqs_delay_ctrl_import port on the other ALTMEMPHY instance.
dqs_offset_delay_ctrl_width	Input	DQS_DELAY_CTL_WI_DTH	Connects to the DQS delay logic when dll_import_export is set to IMPORT. Only connect if you are using a DLL offset, which can otherwise be tied to zero. If you are using a DLL offset, connect this input to the offset_ctrl_out output of the dll_offset_ctrl block.
dll_reference_clk	Output	1	Reference clock to feed to an externally instantiated DLL. This clock is typically from one of the PHY PLL outputs.
<b>User-Mode Calibration OCT Control Signals</b>			
oct_ctl_rs_value	Input	14	OCT RS value port for use with ALT_OCT megafunction if you want to use OCT with user-mode calibration.
oct_ctl_rt_value	Input	14	OCT RT value port for use with ALT_OCT megafunction if you want to use OCT with user-mode calibration.
<b>Debug Interface Signals <sup>(1)</sup>, <sup>(2)</sup></b>			
dbg_clk	Input	1	Debug interface clock.
dbg_reset_n	Input	1	Debug interface reset.
dbg_addr	Input	DBG_A_WI_DTH	Address input.
dbg_wr	Input	1	Write request.
dbg_rd	Input	1	Read request.
dbg_cs	Input	1	Chip select.
dbg_wr_data	Input	32	Debug interface write data.
dbg_rd_data	Output	32	Debug interface read data.
dbg_waitrequest	Output	1	Wait signal.

**Table 2–4. Other Interface Signals (Part 2 of 2)**

Signal Name	Type	Width	Description
<b>Calibration Interface Signals—without leveling only</b>			
rsu_codvw_phase	Output	—	The sequencer sweeps the phase of a resynchronization clock across 360° or 720° of a memory clock cycle. Data reads from the DIMM are performed for each phase position, and a data valid window is located, which is the set of resynchronization clock phase positions where data is successfully read. The final resynchronization clock phase is set at the center of this range: the center of the data valid window or CODVW. This output is set to the current calculated value for the CODVW, and represents how many phase steps were performed by the PLL to offset the resynchronization clock from the memory clock.
rsu_codvw_size	Output	—	The final centre of data valid window size (rsu_codvw_size) is the number of phases where data was successfully read in the calculation of the resynchronization clock centre of data valid window phase (rsu_codvw_phase).
rsu_read_latency	Output	—	The rsu_read_latency output is then set to the read latency (in phy_clk cycles) using the rsu_codvw_phase resynchronization clock phase. If calibration is unsuccessful then this signal is undefined.
rsu_no_dvw_err	Output	—	If the sequencer sweeps the resynchronization clock across every phase and does not see any valid data at any phase position, then calibration fails and this output is set to 1.
rsu_grt_one_dvw_err	Output	—	If the sequencer sweeps the resynchronization clock across every phase and sees multiple data valid windows, this is indicative of unexpected read data (random bit errors) or an incorrectly configured PLL that must be resolved. Calibration has failed and this output is set to 1.

**Notes to Table 2–4:**

- (1) The debug interface uses the simple Avalon-MM interface protocol.
- (2) These ports exist in the Quartus II software, even though the debug interface is for Altera's use only.

Table 2–5 shows the parameters to which Table 2–2 through Table 2–4 refer.

**Table 2–5. Parameters (Part 1 of 2)**

Parameter Name	Description
DWIDTH_RATIO	The data width ratio from the local interface to the memory interface. DWIDTH_RATIO of 2 means full rate, while DWIDTH_RATIO of 4 means half rate.
LOCAL_IF_DWIDTH	The width of the local data bus must be quadrupled for half-rate and doubled for full-rate.
MEM_IF_DWIDTH	The data width at the memory interface. MEM_IF_DWIDTH can have values that are multiples of MEM_IF_DQ_PER_DQS.
MEM_IF_DQS_WIDTH	The number of DQS pins in the interface.
MEM_IF_ROWADDR_WIDTH	The row address width of the memory device.
MEM_IF_BANKADDR_WIDTH	The bank address width of the memory device.
MEM_IF_CS_WIDTH	The number of chip select pins in the interface. The sequencer only calibrates one chip select pin.
MEM_IF_DM_WIDTH	The number of mem_dm pins on the memory interface.

**Table 2–5. Parameters (Part 2 of 2)**

Parameter Name	Description
MEM_IF_DQ_PER_DQS	The number of mem_dq[] pins per mem_dqs pin.
MEM_IF_CLK_PAIR_COUNT	The number of mem_clk/mem_clk_n pairs in the interface.

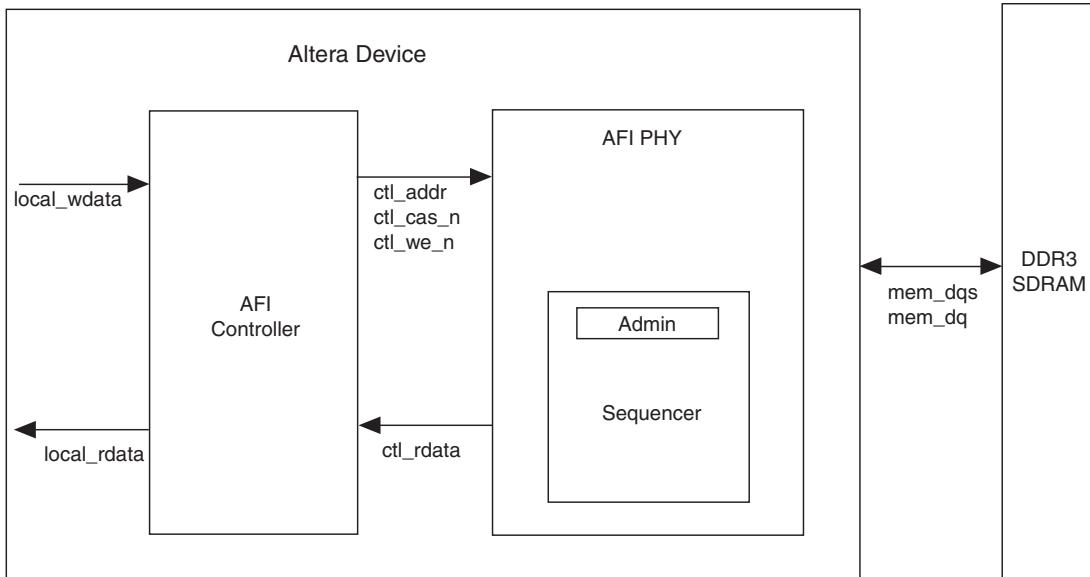
## PHY-to-Controller Interfaces

The following section describes the typical modules that are connected to the ALTMEMPHY variation and the port name prefixes each module uses. This section also describes using a custom controller. This section describes the AFI.

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controller. The AFI PHY includes an administration block that configures the memory for calibration and performs necessary mode registers accesses to configure the memory as required (these calibration processes are different). [Figure 2–5](#) shows an overview of the connections between the PHY, the controller, and the memory device.



Altera recommends that you use the AFI for new designs.

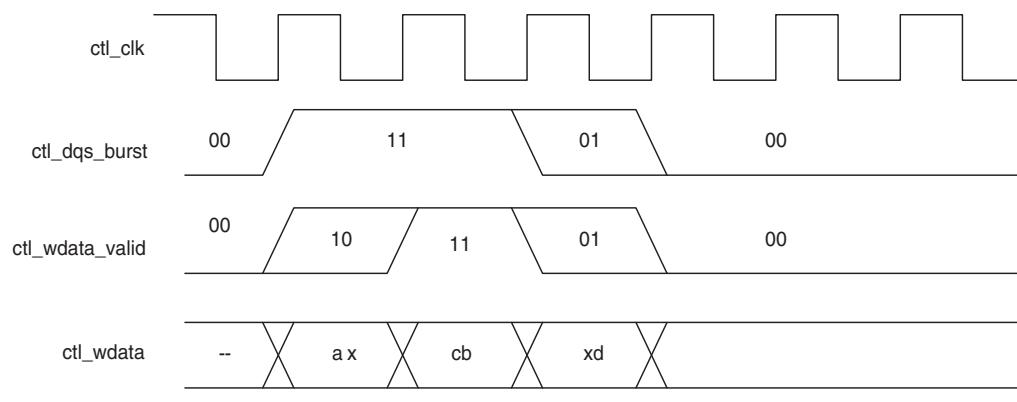
**Figure 2–5. AFI PHY Connections**

For half-rate designs, the address and command signals in the ALTMEMPHY megafunction are asserted for one mem\_clk cycle (1T addressing), such that there are two input bits per address and command pin in half-rate designs. If you require a more conservative 2T addressing, drive both input bits (of the address and command signal) identically in half-rate designs.

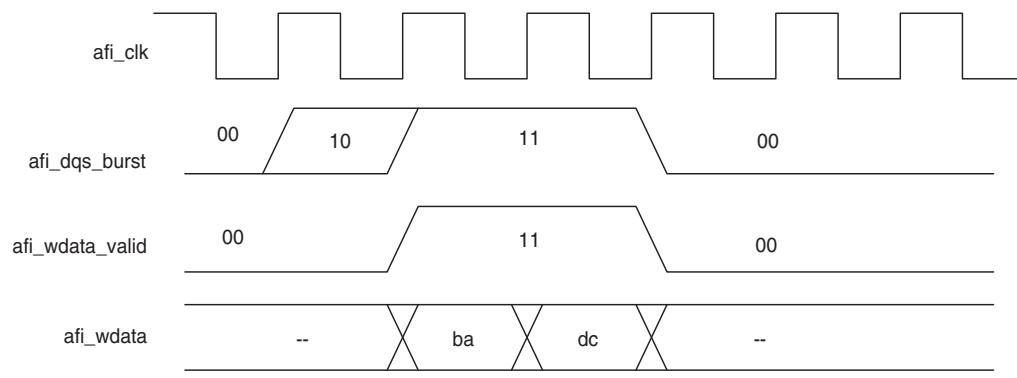
For DDR3 SDRAM with the AFI, the read and write control signals are on a per-DQS group basis. The controller can calibrate and use a subset of the available DDR3 SDRAM devices. For example, the controller can calibrate and use two devices out of a 64- or 72-bit DIMM for better debugging mechanism.

For half-rate designs, the AFI allows the controller to issue reads and writes that are aligned to either half-cycle of the half-rate phy\_clk, which means that the datapaths can support multiple data alignments—word-unaligned and word-aligned writes and reads. [Figure 2–6](#) and [Figure 2–7](#) display the half-rate write operation.

**Figure 2–6. Half-Rate Write with Word-Unaligned Data**



**Figure 2–7. Half-Rate Write with Word-Aligned Data**



After calibration process is complete, the sequencer sends the write latency in number of clock cycles to the controller.

[Figure 2–8](#) and [Figure 2–9](#) show word-aligned writes and reads. In the following read and write examples the data is written to and read from the same address. In each example, ctl\_rdata and ctl\_wdata are aligned with controller clock (ctl\_clk) cycles. All the data in the bit vector is valid at once. For comparison, refer [Figure 2–10](#) and [Figure 2–11](#) that show the word-unaligned writes and reads.



The `ctl_doing_rd` is represented as a half-rate signal when passed into the PHY. Therefore, the lower half of this bit vector represents one memory clock cycle and the upper half the next memory clock cycle. [Figure 2-11 on page 2-22](#) shows separated word-unaligned reads as an example of two `ctl_doing_rd` bits are different. Therefore, for each x16 device, at least two `ctl_doing_rd` bits need to be driven, and two `ctl_rdata_valid` bits need to be interpreted.

The AFI has the following conventions:

- With the AFI, high and low signals are combined in one signal, so for a single chip select (`ctl_cs_n`) interface, `ctl_cs_n[1:0]`, where location 0 appears on the memory bus on one `mem_clk` cycle and location 1 on the next `mem_clk` cycle.



This convention is maintained for all signals so for an 8 bit memory interface, the write data (`ctl_wdata`) signal is `ctl_wdata[31:0]`, where the first data on the DQ pins is `ctl_wdata[7:0]`, then `ctl_wdata[15:8]`, then `ctl_wdata[23:16]`, then `ctl_wdata[31:24]`.

- Word-aligned and word-unaligned reads and writes have the following definitions:
  - Word-aligned for the single chip select is active (low) in location 1 (\_1). `ctl_cs_n[1:0] = 01` when a write occurs. This alignment is the easiest alignment to design with.
  - Word-unaligned is the opposite, so `ctl_cs_n[1:0] = 10` when a read or write occurs and the other control and data signals are distributed across consecutive `ctl_clk` cycles.



The Altera high-performance controller uses word-aligned data only.



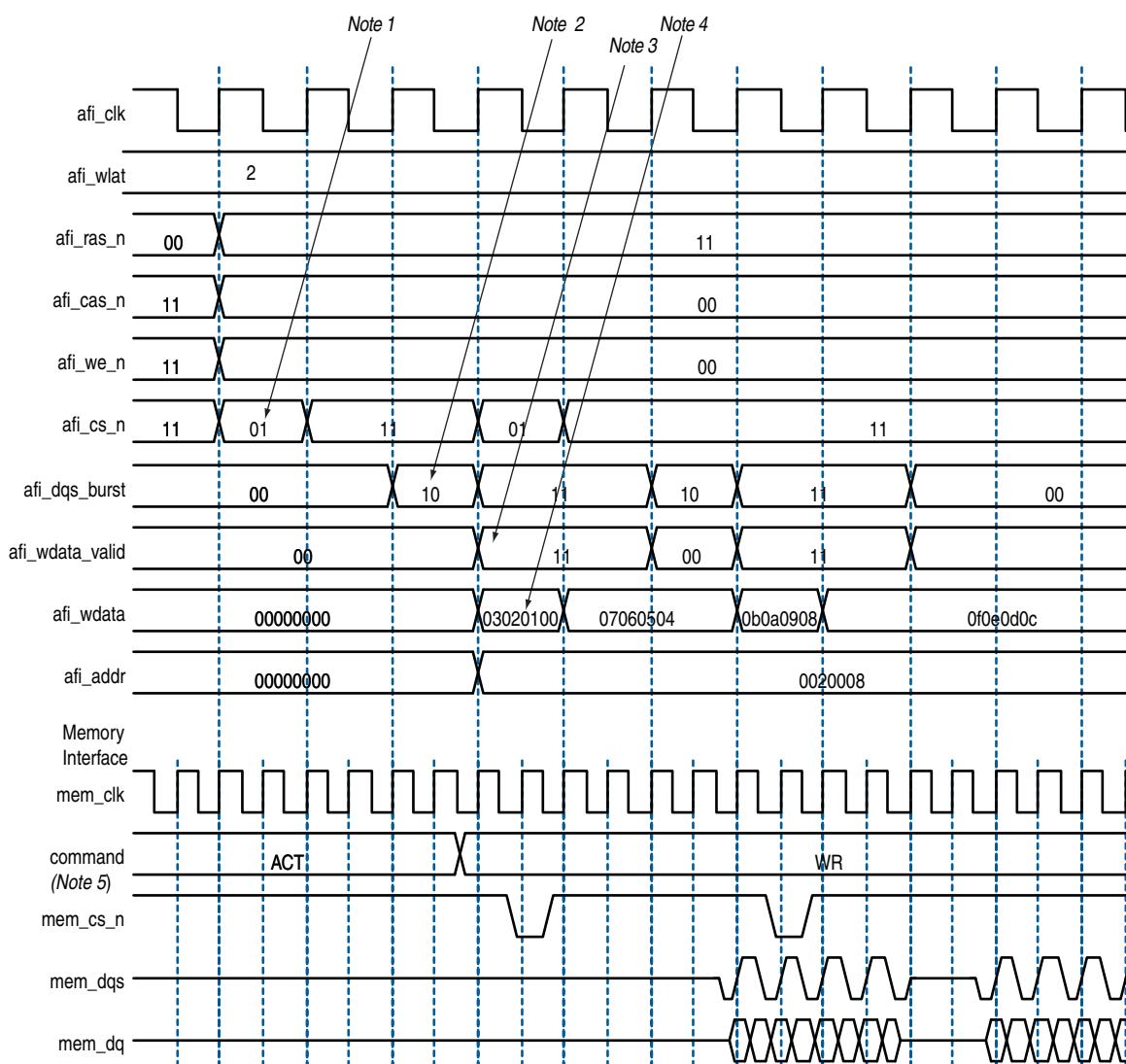
The timing analysis script does not support word-unaligned reads and writes.

- Spaced reads and writes have the following definitions:
  - Spaced writes—write commands separated by a gap of one controller clock (`ctl_clk`) cycle
  - Spaced reads—read commands separated by a gap of one controller clock (`ctl_clk`) cycle

[Figure 2-8](#) through [Figure 2-11](#) assume the following general points:

- The burst length is four. A DDR2 SDRAM is used—the interface timing is identical for DDR3 devices.
- An 8-bit interface with one chip select.
- The data for one controller clock (`ctl_clk`) cycle represents data for two memory clock (`mem_clk`) cycles (half-rate interface).

**Figure 2–8. Word-Aligned Writes**



**Notes to Figure 2–8:**

- (1) To show the even alignment of `ctl_cs_n`, expand the signal (this convention applies for all other signals).
- (2) The `ctl_dqs_burst` must go high one memory clock cycle before `ctl_wdata_valid`. Compare with the word-unaligned case.
- (3) The `ctl_wdata_valid` is asserted two `ctl_clk` cycles after chip select (`ctl_cs_n`) is asserted. The `ctl_wlat` indicates the required write latency in the system. The value is determined during calibration and is dependant upon the relative delays in the address and command path and the write datapath in both the PHY and the external DDR SDRAM subsystem. The controller must drive `ctl_cs_n` and then wait `ctl_wlat` (two in this example) `ctl_clks` before driving `ctl_wdata_valid`.
- (4) Observe the ordering of write data (`ctl_wdata`). Compare this to data on the `mem_dq` signal.
- (5) In all waveforms a command record is added that combines the memory pins `ras_n`, `cas_n` and `we_n` into the current command that is issued. This command is registered by the memory when chip select (`mem_cs_n`) is low. The important commands in the presented waveforms are WR = write, ACT = activate.

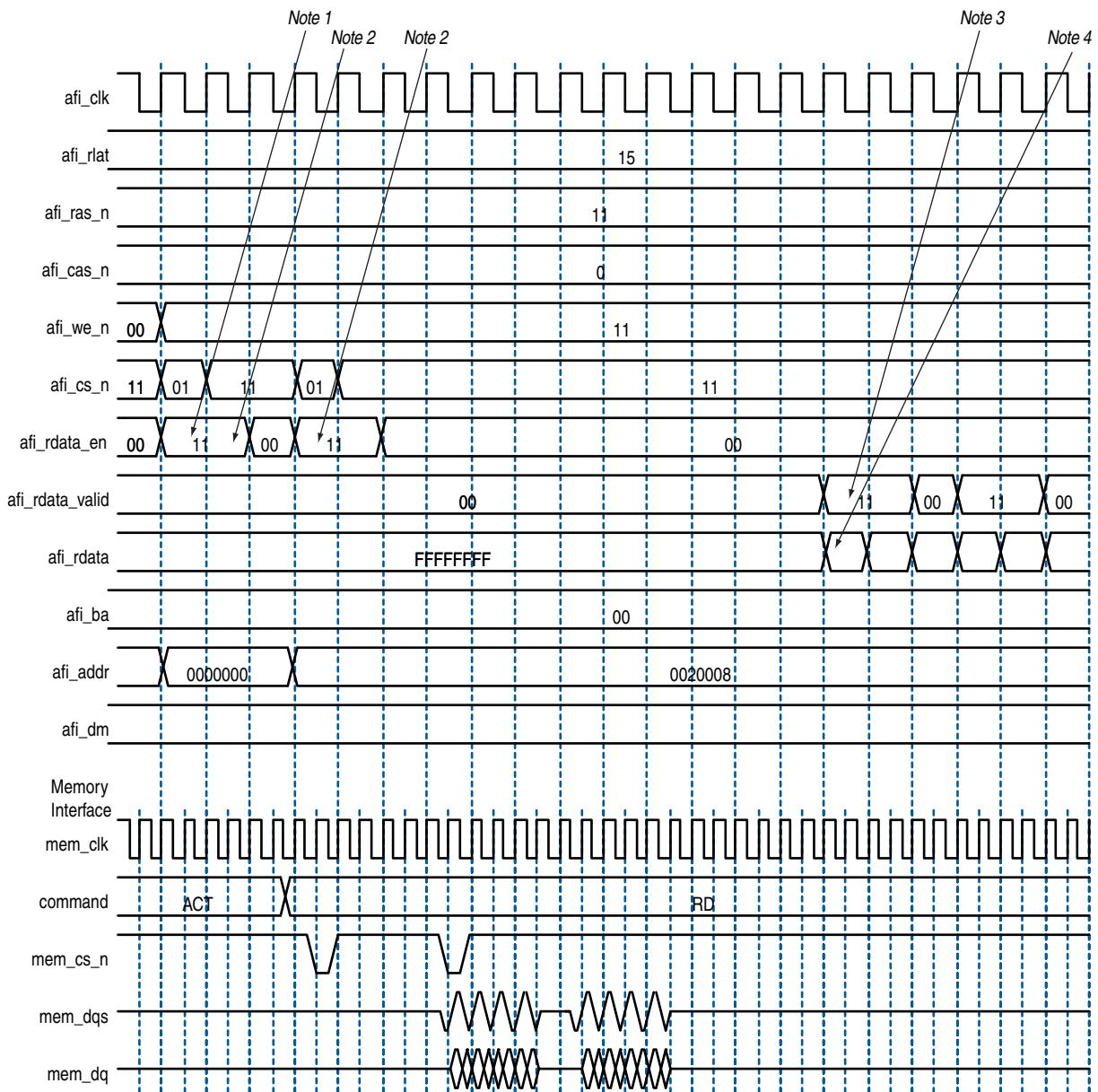
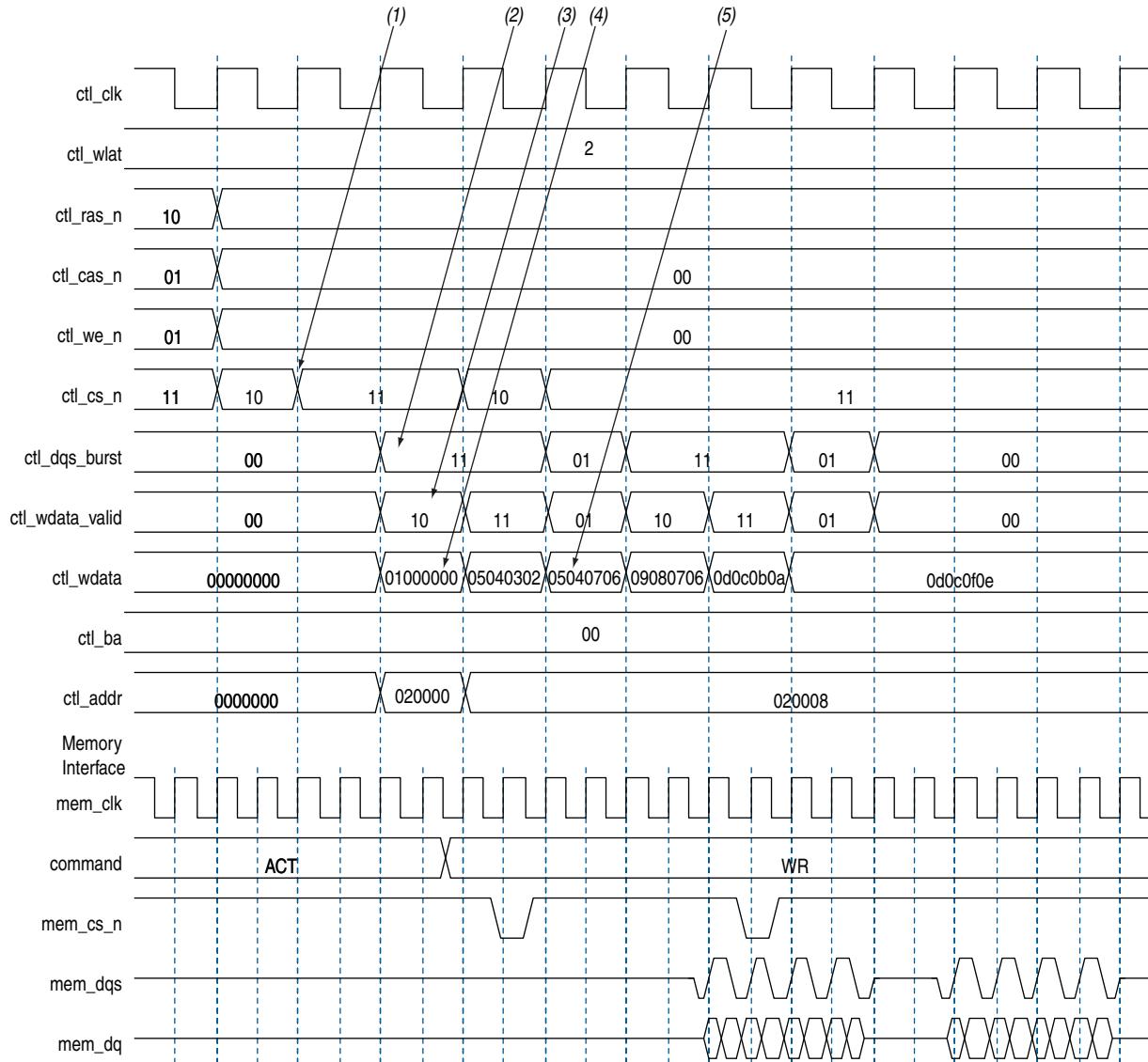
**Figure 2–9. Word-Aligned Reads**

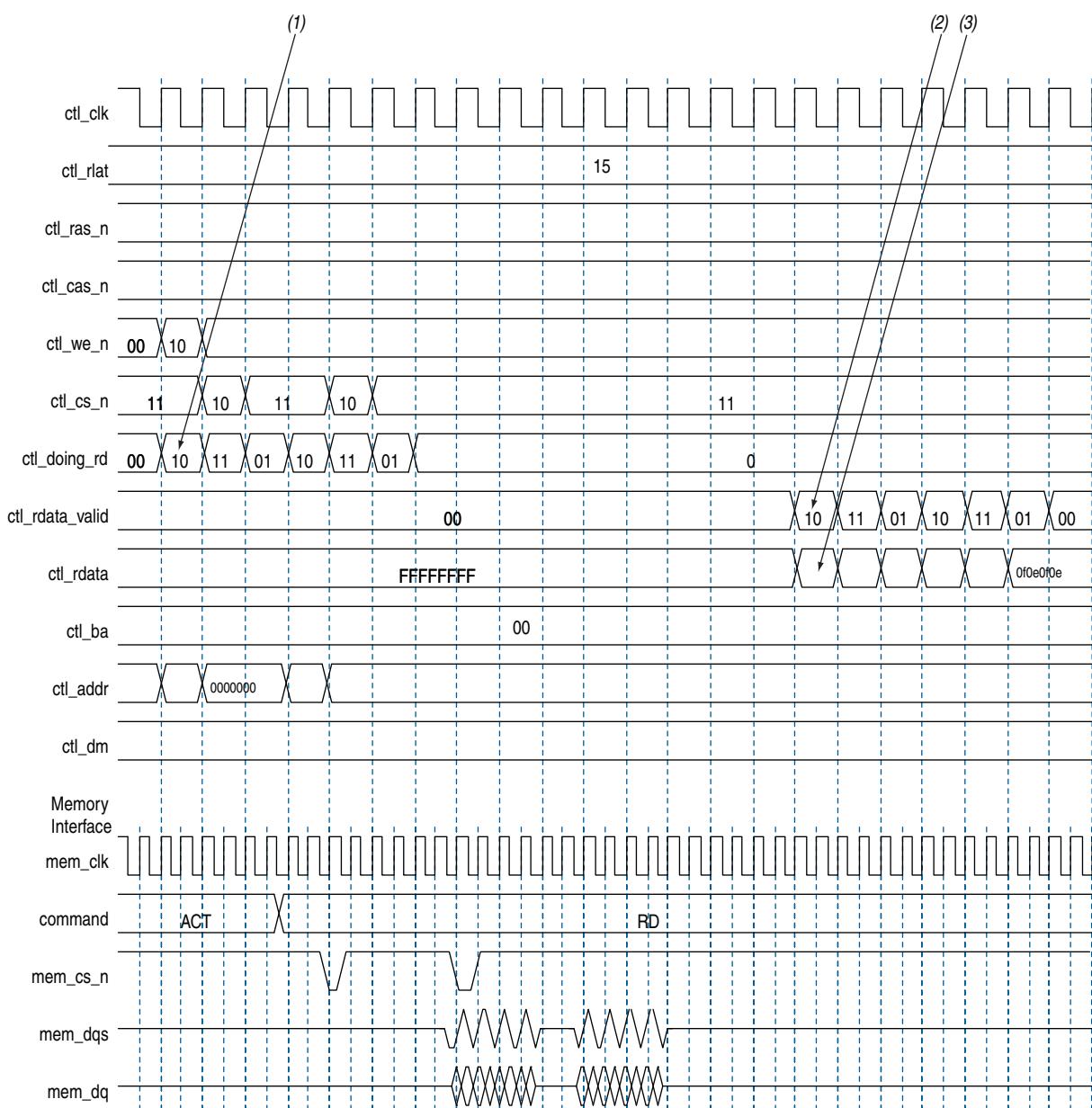
Figure 2–10 and Figure 2–11 show spaced word-unaligned writes and reads.

**Figure 2–10. Word-Unaligned Writes**



**Notes to Figure 2–10:**

- (1) Alternative word-unaligned chip select (`ctl_cs_n`).
- (2) As with word-aligned writes, `ctl_dqs_burst` is asserted one memory clock cycle before `ctl_wdata_valid`. You can see `ctl_dqs_burst` is 11 in the same cycle where `ctl_wdata_valid` is 10. The LSB of these two becomes the first value the signal takes in the `mem_clk` domain. You can see that `ctl_dqs_burst` has the necessary one `mem_clk` cycle lead on `ctl_wdata_valid`.
- (3) The latency between `ctl_cs_n` being asserted and `ctl_wdata_valid` going high is effectively `ctl_wlat` (in this example, two) controller clock (`ctl_clk`) cycles. This can be thought of in terms of relative memory clock (`mem_clk`) cycles, in which case the latency is four `mem_clk` cycles.
- (4) Only the upper half is valid (as the `ctl_wdata_valid` signal demonstrates, there is one `ctl_wdata_valid` bit to two 8-bit words). The write data bits go out on the bus in order, least significant byte first. So for a continuous burst of write data on the DQ pins, the most significant half of write data is used, which goes out on the bus last and is therefore contiguous with the following data. The converse is true for the end of the burst. Write data is spread across three controller clock (`ctl_clk`) cycles, but still only four memory clock (`mem_clk`) cycles. However, in relative memory clock cycles the latency is equivalent in the word-aligned and word-unaligned cases.
- (5) The 0504 here is residual from the previous clock cycle. In the same way that only the upper half of the write data is used for the first beat of the write, only the lower half of the write data is used in the last beat of the write. These upper bits can be driven to any value in this alignment.

**Figure 2–11. Word-Unaligned Reads****Notes to Figure 2–11:**

- (1) Similar to word-aligned reads, **ctl\_doing\_rd** is asserted one memory clock cycle before chip select (**ctl\_cs\_n**) is asserted, which for a word-unaligned read is in the previous controller clock (**ctl\_clk**) cycle. In this example the **ctl\_doing\_rd** signal is now spread over three controller clock (**ctl\_clk**) cycles, the high bits in the sequence '10','11','01','10','11','01' providing the required four memory clock cycles of assertion for **ctl\_doing\_rd** for the two 4-beat reads in the full-rate memory clock domain, '011110','011110'.
- (2) The return pattern of **ctl\_rdata\_valid** is a delayed version of **ctl\_doing\_rd**. Advertised read latency (**ctl\_rlat**) is the number of controller clock (**ctl\_clk**) cycles delay inserted between **ctl\_doing\_rd** and **ctl\_rdata\_valid**.
- (3) The read data (**ctl\_rdata**) is spread over three controller clock cycles and in the pointed to vector only the upper half of the **ctl\_rdata** bit vector is valid (denoted by **ctl\_rdata\_valid**).

## Using a Custom Controller

The ALTMEMPHY megafunction can be integrated with your own controller. This section describes the interface requirement and the handshake mechanism for efficient read and write transactions.

### Preliminary Steps

Perform the following steps to generate the ALTMEMPHY megafunction:

1. If you are creating a custom DDR3 SDRAM controller, generate the Altera high-performance controller targeting your chosen Altera and memory devices.
2. Compile and verify the timing. This step is optional.
3. If targeting a DDR3 SDRAM device, simulate the high-performance controller design so you can determine how to drive the PHY signals using your own controller.
4. Integrate the top-level ALTMEMPHY design with your controller. If you started with the high-performance controller, the PHY variation name is `<controller_name>_phy.v/.vhd`. Details about integrating your controller with Altera's ALTMEMPHY megafunction are described in the following sections.
5. Compile and simulate the whole interface to ensure that you are driving the PHY properly and that your commands are recognized by the memory device.

### Design Considerations

This section discuss the important considerations for implementing your own controller with the ALTMEMPHY megafunction. This section describes the design considerations for AFI variants.



Simulating the high-performance controller is useful if you do not know how to drive the PHY signals.

### Clocks and Resets

The ALTMEMPHY megafunction automatically generates a PLL instance, but you must still provide the reference clock input (`p11_ref_clk`) with a clock of the frequency that you specified in the MegaWizard Plug-In Manager. An active-low global reset input is also provided, which you can deassert asynchronously. The clock and reset management logic synchronizes this reset to the appropriate clock domains inside the ALTMEMPHY megafunction.

A clock output, half the memory clock frequency for a half-rate controller, is provided and all inputs and outputs of the ALTMEMPHY megafunction are synchronous to this clock. For AFIs, this signal is called `ctl_clk`.

There is also an active-low synchronous reset output signal provided, `ctl_reset_n`. This signal is synchronously de-asserted with respect to the `ctl_clk` or `phy_clk` clock domain and it can reset any additional user logic on that clock domain.

## Calibration Process Requirements

When the global `reset_n` is released the ALTMEMPHY handles the initialization and calibration sequence automatically. The sequencer calibrates memory interfaces by issuing reads to multiple ranks of DDR3 SDRAM (multiple chip select). Timing margins decrease as the number of ranks increases. It is impractical to supply one dedicated resynchronization clock for each rank of memory, as it consumes PLL resources for the relatively small benefit of improved timing margin. When calibration is complete `ctl_cal_success` goes high if successful; `ctl_cal_fail` goes high if calibration fails. Calibration can be repeated by the controller using the `soft_reset_n` signal, which when asserted puts the sequencer into a reset state and when released the calibration process begins again.

## Other Local Interface Requirements

The memory burst length for DDR3 SDRAM devices can be set at either four or eight; but when using the Altera high-performance controller, only burst length eight is supported. For a half-rate controller, the memory clock runs twice as fast as the clock provided to the local interface, so data buses on the local interface are four times as wide as the memory data bus.

## Address and Command Interfacing

Address and command signals are automatically sized for 1T operation, such that for full-rate designs there is one input bit per pin (for example, one `cs_n` input per chip select configured); for half-rate designs there are two. If you require a more conservative 2T address and command scheme, use a full-rate design and drive the address/command inputs for two clock cycles, or in a half-rate design drive both address/command bits for a given pin identically.



Although the PHY inherently supports 1T addressing, the high-performance controller supports only 2T addressing, so PHY timing analysis is performed assuming 2T address and command signals.

## Handshake Mechanism Between Read Commands and Read Data

When performing a read, a high-performance controller with the AFI asserts `ctl_doing_read` to indicate that a read command is requested and the byte lanes that it expects valid data to return on. ALTMEMPHY uses `ctl_doing_read` for the following actions:

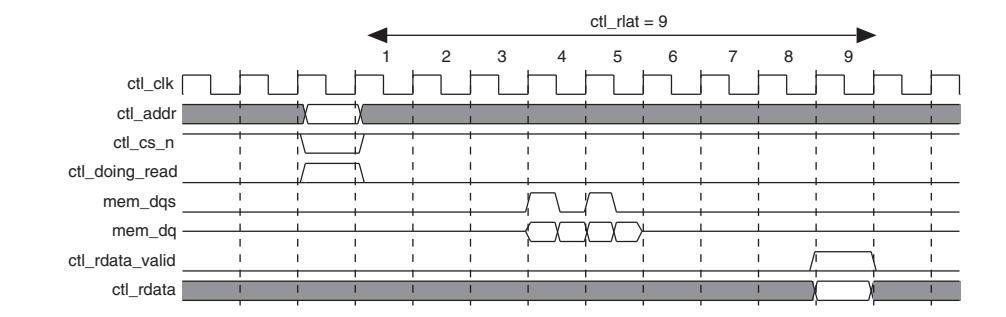
- Control of the postamble circuit
- Generation of `ctl_rdata_valid`
- Dynamic termination (Rt) control timing

The read latency, `ctl_rlat`, is advertised back to the controller. This signal indicates how long it takes in `ctl_clk` clock cycles from assertion of `ctl_doing_read` to valid read data returning on `ctl_rdata`. The `ctl_rlat` signal is only valid when calibration has successfully completed and never changes values during normal user mode operation.

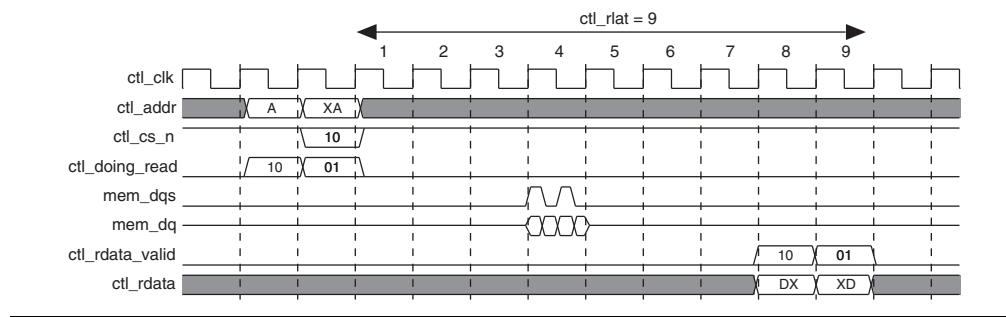
The ALTMEMPHY provides a signal, `ctl_rdata_valid`, to indicate that the data on read data bus is valid. The width of this signal varies between half-rate and full-rate designs to support the option to indicate that the read data is not word aligned.

[Figure 2–12](#) and [Figure 2–13](#) show these relationships.

**Figure 2–12. Address and Command and Read-Path Timing—Full-Rate Design**



**Figure 2–13. Second Read Alignment—Half-Rate Design**



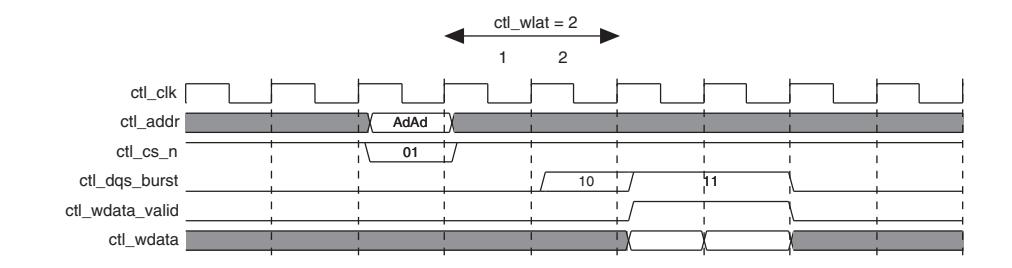
### Handshake Mechanism Between Write Commands and Write Data

In the AFI, the ALTMEMPHY output `ctl_wlat` gives the number of `ctl_clk` cycles between the write command that is issued `ctl_cs_n` asserted and `ctl_dqs_burst` asserted. The `ctl_wlat` signal considers the following actions to provide a single value in `ctl_clk` clock cycles:

- CAS write latency
- Additive latency
- Datapath latencies and relative phases
- Board layout
- Address and command path latency and 1T register setting, which is dynamically setup to take into account any leveling effects

The `ctl_wlat` signal is only valid when the calibration has been successfully completed by the ALTMEMPHY sequencer and does not change at any point during normal user mode operation. Figure 2-14 shows the operation of `ctl_wlat` port.

**Figure 2-14. Timing for `ctl_dqs_burst`, `ctl_wdata_valid`, Address, and Command—Half-Rate Design**



For a half-rate design `ctl_cs_n` is 2 bits, not 1. Also the `ctl_dqs_burst` and `ctl_wdata_valid` waveforms indicate a half-rate design. This write results in a burst of 8 at the DDR. Where `ctl_cs_n` is driven 2'b01, the LSB (1) is the first value driven out of `mem_cs_n`, and the MSB (0) follows on the next `mem_clk`. Similarly, for `ctl_dqs_burst`, the LSB is driven out of `mem_dqs` first (0), then a 1 follows on the next clock cycle. This sequence produces the continuous DQS pulse as required. Finally, the `ctl_addr` bus is twice `MEM_IF_ADDR_WIDTH` bits wide and so the address is concatenated to result in an address phase two `mem_clk` cycles wide.

### Partial Writes

As part of the DDR3 SDRAM memory specifications, you have the option for partial write operations by asserting the DM pins for part of the write signal.

For designs targeting the Arria II devices, deassert the `ctl_wdata_valid` signal during partial writes, when the write data is invalid, to save power by not driving the DQ outputs.

For designs targeting other devices, use only the DM pins if you require partial writes. Assert the `ctl_dqs_burst` and `ctl_wdata_valid` signals as for full write operations, so that the DQ and DQS pins are driven during partial writes.

## Controller Register Map

The controller register map allows you to control the memory controller settings.



For information on the controller register map, refer to “[Controller Register Map](#)” on page 5-29 of this volume.

## ALTMEMPHY Calibration Stages

In all configurations, the noncalibrated address, command and control interfaces must be correctly constrained and meet timing.

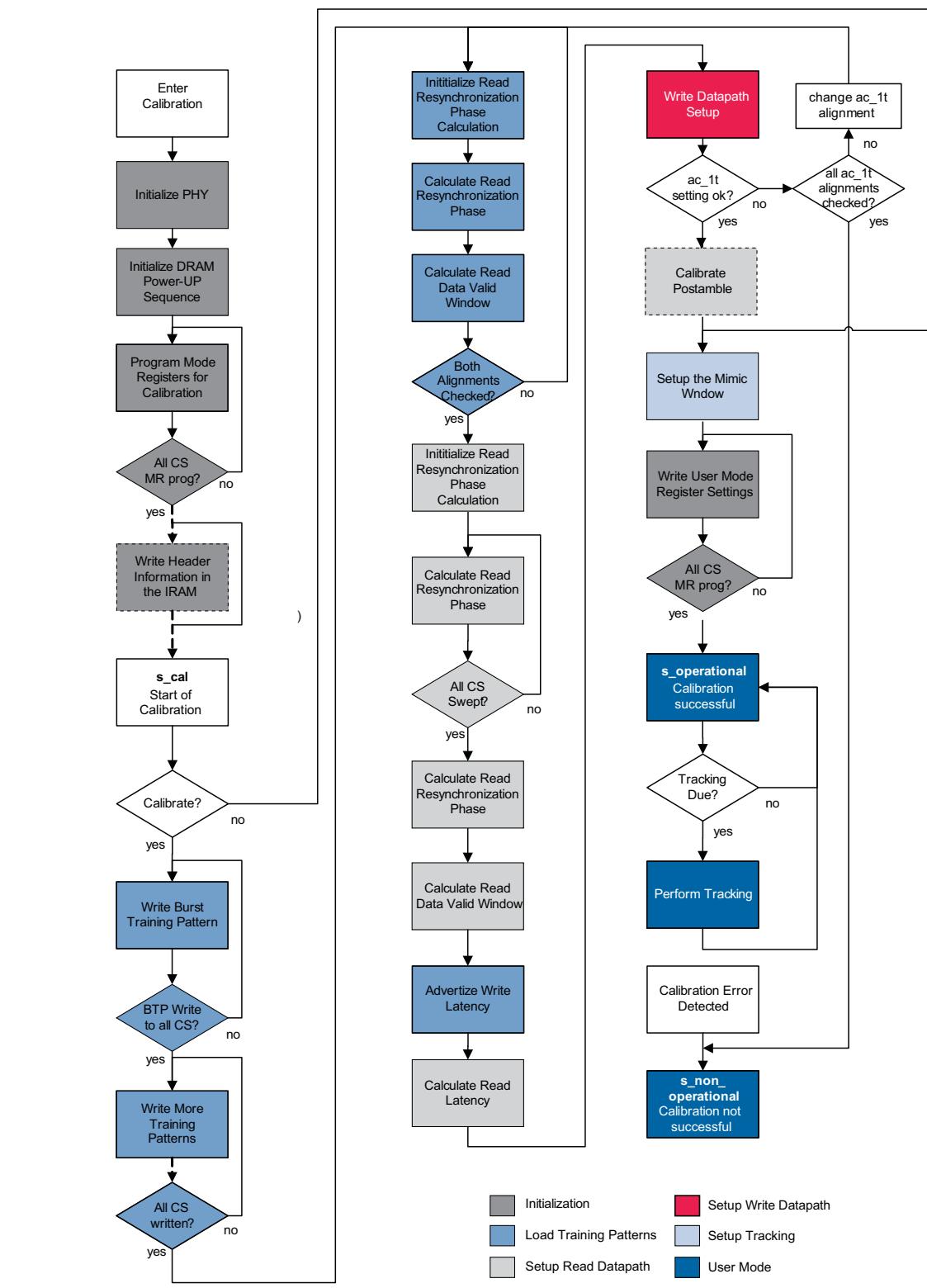
If calibration fails at a specific stage, use this chapter to understand what functionally happens at that stage, to assist with the debug.

The ALTMEMPHY IP performs the following calibration stages:

1. Enter Calibration (`s_reset`)
2. Initialize PHY (`s_phy_initialize`)
3. Initialize DRAM
4. Write Header Information in the internal RAM (`s_write_ihi`)
5. Load Training Patterns
6. Test More Pattern Writes
7. Calibrate Read Resynchronization Phase
8. Advertise Write Latency (`s_was`)
9. Calculate Read Latency (`s_adv_rlat`)
10. Output Write Latency (`s_adv_wlat`)
11. Calibrate Postamble (`s_poa`)
12. Set Up Address and Command Clock Cycle
13. Write User Mode Register Settings (`s_prep_customer_mr_setup`)
14. Voltage and Temperature Tracking

This chapter discusses these stages. [Figure 2–15 on page 2–28](#) shows a flow chart of the calibration stages for the ALTMEMPHY IP.

## **Figure 2–15. Calibration Stages**



## Enter Calibration (s\_reset)

Calibration starts when the ALTMEMPHY IP deasserts the PHY reset signal and the AFI signal `ctl_cal_req` is low.

## Initialize PHY (s\_phy\_initialize)

This stage holds off calibration until the DLL has locked, and (if debug toolkit is enabled) internal RAM contents are all reset to zero.

## Initialize DRAM

Initializing the DRAM has the following two stages:

- Initialize DRAM Power Up Sequence (s\_int\_dram)
- Program Mode Registers for Calibration (s\_prog\_mr)

### Initialize DRAM Power Up Sequence (s\_int\_dram)

This stage brings the SDRAM out of a reset state (from any previous state) through the initialization sequence specified in the JEDEC specification for each device type, up to but not including mode register set commands. At the end of this stage, the SDRAM is ready to receive mode register load commands, which must occur (on each rank) before refreshes can occur.



This calibration stage applies to all chip selects in parallel.

### Program Mode Registers for Calibration (s\_prog\_mr)

The ALTMEMPHY IP issues mode register set commands on a per chip select basis, which allows you great flexibility to issue different mode register settings to different chip selects. When all chip selects have mode registers programmed (the initialization of that chip select is complete), refreshes are enabled.

The following overrides apply to user settings:

- For DDR and DDR2 SDRAM:
  - DLL enable
  - Burst length 4
  - OCD calibration (DDR2 only)
- For DDR3 SDRAM:
  - DLL enable
  - Output buffer enable
  - Disable write leveling
  - Runtime burst length select
  - Test mode disabled
  - DLL reset

-  For DDR3 SDRAM this stage also includes a ZQ-cal long operation (refer to the JEDEC specification).

## Write Header Information in the internal RAM (`s_write_ihi`)

In this stage, the ALTMEMPHY IP loads the internal header information in the first eight locations in the internal RAM through the parameterization of the ALTMEMPHY IP. The debug toolkit uses this information to provide the current ALTMEMPHY IP parameterization and IP version numbers.

-  The ALTMEMPHY IP only executes this stage when you enable debug toolkit.
-  For information about the debug toolkit, refer to the [ALTMEMPHY External Memory Interface Debug Toolkit](#).

## Load Training Patterns

In this stage, the ALTMEMPHY IP writes training patterns to the memory to be read in later calibration stages. Because of the matched trace lengths to DDR SDRAM components, after memory initialization, you can assume write capture works.

You can divide the training pattern writes into the following two stages:

- Write Block Training Pattern (`s_write_btp`)
- Write More Training Patterns (`s_write_mtp`)

-  The ALTMEMPHY IP writes further training pattern in the calibration of the write datapath, refer to [Advertise Write Latency \(`s\_was`\)](#).

### Write Block Training Pattern (`s_write_btp`)

This stage applies to read data valid alignment (`s_rdv`), advertise read latency (`s_adv_rd_lat`) and postamble calibration (`s_poa`). For these calibration stages a pattern of all 1s and all 0s is sufficient to set up the PHY.

Writing of these two patterns is trivial and requires all DDIO outputs (high and low phases (bits)) to be held at either 1 or 0, for all 1 and all 0 patterns, respectively. To write these patterns, the ALTMEMPHY IP holds the DDIO outputs low (or high) and toggles DQS for a predetermined length of time and issues a single write command. The ALTMEMPHY IP tests a full range of memory write latencies.

To support DDR3 SDRAM discrete components (burst of eight reads), the ALTMEMPHY IP loads eight memory locations with 1s and eight with 0s.

The following memory locations contain the following patterns:

- Locations [7:0], all 0s
- Locations [15:8], all 1s

-  You need patterns of all 1s and all 0s for calibrating the read resynchronization phase.

## Write More Training Patterns (s\_write\_mtp)

This stage calculates the read resynchronization phase (s\_rrp\_sweep).

The pattern is 0x30F5 and comprises separately written patterns. The ALTMEMPHY IP requires this pattern to match the characterization behavior for nonDQS capture based schemes (for example, Cyclone III devices). All device families use the following pattern:

- All 0: 'b0000 to DDIO high and low bits held at 0
- All 1: 'b1111 to DDIO high and low bits held at 1
- Toggle: 'b0101 to DDIO high bits held at 0 and DDIO low bits held at 1
- Mixed: 'b0011 to DDIO high and low bits have to toggle

While you can ensure that all zeros, all ones, or toggle are written into a burst of memory locations (output DDIO bits are held at constant values), it is challenging to ensure that a pattern of 0011 is written into memory. The challenge occurs because the write latency is unknown at this time. For example, if this pattern is repeated on DQ pins (0011 0011 0011) and a single write command issued (as for the other patterns), it is not known whether the memory location contains the pattern 0011 or 1100.

The ALTMEMPHY IP provides a methodology to robustly write these patterns ([Test More Pattern Writes](#)). For this section two locations (X and Y) are populated with write data, one contains the pattern 0011; the other contains 1100.

The memory locations contain the following patterns:

- x30 alignment 0 to location (X): 23..16
- x30 alignment 1 to location (Y): 31..24
- xF5 to location: 39..32

The ALTMEMPHY IP writes these patterns in bursts of four beats, so in the pattern xF5, F is written separately to 5. The ALTMEMPHY IP writes patterns F and 0 as a part of the writing of the block training pattern ([Write Block Training Pattern \(s\\_write\\_btp\)](#)).

## Test More Pattern Writes

This stage comprises a number of calibration stages of the PHY, but the stage is described as one entity. This stage comprises:

- Initialize read resynchronization phase calculation (s\_rrp\_reset)
- Calculate read resynchronization phase (s\_rrp\_sweep)
- Calculate read data valid window (s\_read\_mtp)

The algorithm ensures the pattern 0011 is written to a known location in memory. The following assumptions and PHY settings apply to this stage:

- Assumptions:
  - Burst length of 4 writes
  - Write capture in the memory device works. Data can be safely written to memory. No write leveling is required. (Refer to JEDEC specification for more information on DDR3 SDRAM).
  - Writes are aligned on a clock cycle basis. You know which beats of DQ data to write on the low and high phases of DQS (ultimately DQ and DQS board delays are well matched).
- Settings:
  - Address and command 1T setting. You can add additional latency to the address and command path (specified as a maximum of one memory clock cycles ( $t$ )). This setting aligns write data to address and command signals relative to the controller clock domain, where address and command signals are issued in a given alignment. This setting is not required ( $0t$ ) for a full-rate PHY.
  - Read data 1T alignment. Additional delay of captured read data to align with read commands in the half rate controller clock domain. The interpretation of 1T is the same as for address and command, but applied to read data. This setting does not apply to a full-rate PHY.
  - Read resynchronization phase setting. This setting is the primary task of the training pattern to correctly set the resynchronization phase in the middle of data valid window for the read data (on DQ pins) to be captured.

From this algorithm, to determine PHY settings B and C, given that A is not set, follow these steps:

1. Try to write the pattern and indistinguishable variations of it to different memory locations. For example, write to different locations with the following two patterns on the DQ bus (timed to a local controller rate clock), refer to [Write More Training Patterns \(s\\_write\\_mtp\)](#):
  - a. 0011 0011 0011 (try to write 0011) in location X
  - b. 1100 1100 1100 (try to write 1100) in location Y
2. Perform two single-pin DQ pin and single-chip select read resynchronization phase calibrations using location X and location Y, as part of the larger training pattern (0x30F5). You do not know at this time which locations X and Y contain the pattern 0011. This stage iterates through the following stages:
  - Initialize read resynchronization phase calculation (s\_rrp\_reset)
  - Calculate read resynchronization phase (s\_rrp\_sweep)
  - Calculate read data valid window (s\_read\_mtp)
  - Calculate read data valid window (s\_read\_mtp) is a special case of calibrate read resynchronization phase (s\_rrp\_sweep), refer to [Calibrate Read Resynchronization Phase \(s\\_rrp\\_sweep\)](#). This stage reports the size of the returned window without setting up the PLL phase or producing an error if no window is observed.

3. The single pin read resynchronization calibration (using pattern X or Y), which results in the largest data valid window, contains the optimal pattern. The read resynchronization calibration with the largest window indicates the location (X or Y) that contains the correct alignment (0011). Read resynchronization phase calibration uses this alignment (X or Y), to perform the full resynchronization phase calibration across all pins and chip selects.



During calibration of the read resynchronization phase, the ALTMEMPHY IP captures the DQ pin using a free running clock, phase shifted through a given number of steps. You should try to match a training pattern against read data, for each phase shift, and a window of valid phases is composed.

In waveforms, you observe two resynchronization phase sweeps, over single pins, before the larger phase sweep. However in fast simulation mode, you observe two resynchronization phase sweeps when the duration of all three sweeps is equal. For half-rate interfaces, you may observe a total of six phase sweeps, where the entire calibration is repeated when the address and command 1T setting is toggled.

## Calibrate Read Resynchronization Phase

This stage encompasses the following calibration stages:

- Initialize Read Resynchronization Phase Calibration (`s_rrp_reset`)
- Calibrate Read Resynchronization Phase (`s_rrp_sweep`)
- Calculate Read Resynchronization Phase (`s_rrp_seek`)

This stage adjusts the phase of the resynchronization (or capture) clock to determine the optimal phase that gives the greatest margin. For DQS based capture schemes the resynchronization clock captures the outputs of DQS capture registers (DQS is the capture clock). In a non-DQS capture based scheme, the capture clock captures the input DQ pin data (the DQS signal is unused).

For all half-rate PHY interfaces, a  $720^\circ$  resynchronization or capture clock phase sweep is performed. For a half-rate PHY this sweep is effectively  $360^\circ$  of the half-rate clock, because resynchronization or capture clock is at the memory clock rate. A  $720^\circ$  sweep is required, so that read data can be presented to a controller aligned to one half-rate controller clock cycle.

For full-rate DQS based capture, because the DQ pins are captured using the DQS signal, in a  $360^\circ$  phase sweep, all resynchronization clock phases may pass. In this case the correct resynchronization phase to set cannot be determined. The correct phase is the one in the center of a valid window, where returned read data is correct. Thus a  $720^\circ$  phase sweep is performed. From  $360$  to  $720^\circ$ , a clock cycle of latency may be added to a  $0$  to  $360^\circ$  sweep. The returned read data is compared to a training pattern pseudo half-rate (at half the clock rate of the sequencer), so data can only be valid for  $360^\circ$  of the sweep. This method introduces edges to the data valid window such that a correct phase can be chosen.

For non-DQS capture in general, up to half ( $180^\circ$ ) of the swept capture clock phases can result in correct capture data, because the DDR to SDR conversion is performed by the capture clock, and thus high and low phases of DQ are captured in the incorrect alignment for half of the capture clock phases. Therefore, only  $360^\circ$  of capture clock need be swept for full-rate non-DQS capture based PHYs.

The pseudo half-rate case potentially adds one clock cycle of latency into the read datapath because of the 720° sweep. The ALTMEMPHY IP detects this occurrence and removes the clock cycle of latency in the calculate read resynchronization phase (`s_rrp_seek`).

### **Initialize Read Resynchronisation Phase Calibration (`s_rrp_reset`)**

This stage returns the PLL to a nominal zero phase shift.

### **Calibrate Read Resynchronization Phase (`s_rrp_sweep`)**

This stage performs a sweep through a parameterised 360° or 720° of resynchronization clock phase. The ALTMEMPHY IP optionally stores these results in the internal RAM.

The command has the following attributes:

- `Single_pin` to indicate just to sweep DQ pin 0 as for the use in testing the write more training patterns stage.
- `mtp_alignment` to say which location (X/Y) to use from the write more training patterns stage.

### **Calculate Read Resynchronization Phase (`s_rrp_seek`)**

This stage calculates the size and center (in phase steps) of the largest data valid window found during the calibrate read resynchronization phase and sets PLL phase to the center phase.

Calculate read data valid window (`s_read_mtp`) is a special case of this stage which reports no errors for an invalid window (a failure is expected in one case) and does not setup the PLL.

### **Calculate Read Data Valid Window (`s_rdv`)**

This stage sets the latency on a delayed version of the `doing_rd` signal, so that it is aligned with the `rdata_valid` signal for the read data it is incident with the read command for.

This stage has the following process:

1. Reads a continuous stream of 1s followed by one read of zeros. The sequencer only asserts `doing_rd` when read command for zeros is issued.
2. Checks for alignment of read data valid signal (delayed version of `doing_rd`) to the zeros (`rdata = 0, rdata_valid = 1`).
3. If not aligned, reduces latency between `doing_rd` and `rdata_valid` signal and loop.



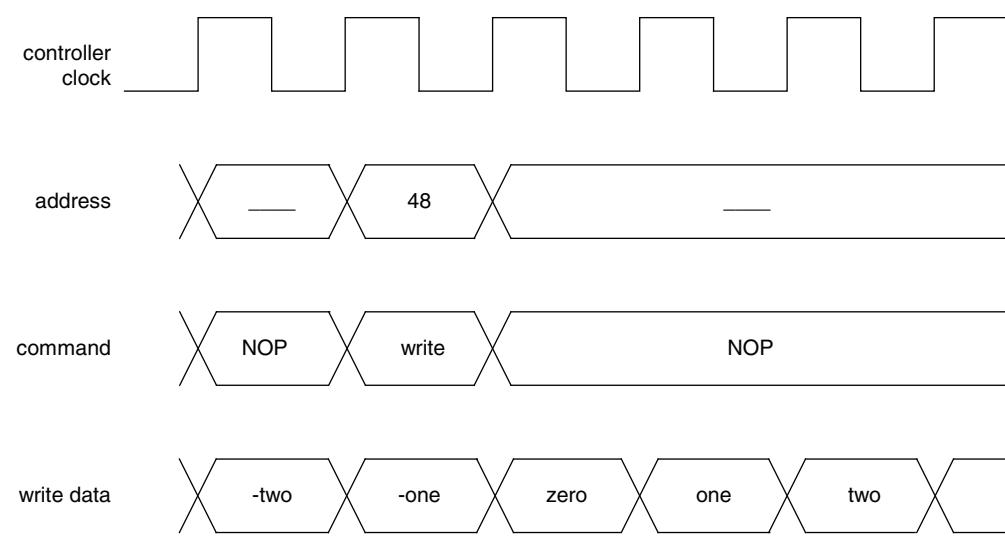
Read data valid latency is reset to a high value (before calibration) and then reduced until it matches the correct alignment.

## Advertise Write Latency (s\_was)

This stage writes a suitable pattern to the DRAM to calculate the write latency.

A write command is issued to a memory address 48 (Figure 2–16) while driving a count of frequency controller clock rate to the DQ pins (Figure 2–16), using the write datapath observed by the controller. For half-rate interfaces, each four beats of write data on DQ are identical. For full-rate interfaces, each two beats of write data are identical. In general, the write latency is written into addresses A... A + (n – 1), where n is two times the ratio of controller to memory clock rate and each n bits of write data must be identical. The pattern is written into memory locations 48 to 55.

**Figure 2–16. Description of Write Pattern**



## Calculate Read Latency (s\_adv\_rlat)

In addition to read data valid window calculation (s\_rdv), the advertised read latency is calculated in this stage in the following way:

- Issues a read command (with doing\_rd) and starts a counter at PHY clock rate
- When rdata\_valid returns, outputs the value of the counter to ctl\_rlat signal.

This signal is redundant, because a controller can use the rdata\_valid signal to determine when valid read data is returned.



The read data from the DRAM is not important here. The count is performed between the issue of doing\_rd and rdata\_valid returning.

## Output Write Latency (s\_adv\_wlat)

To calibrate a PHY write datapath to a minimum latency, a robust process is required to determine the write latency (WL) between a memory controller write command and the associated write data. Factors that can contribute to WL include memory CAS latency, arbitrary additive delays in the PHY, and board trace lengths. The presented approach extends from calibrating a PHY, where the controller operates at the memory clock frequency, to controller operation at half or a quarter of the memory clock frequency.

After a predetermined maximum read latency clock cycles have passed, the contents of the chosen memory address (0x48 in [Figure 2-16](#)) are read to recover the write latency. The first  $n$  beats of read data contain the write latency.

While this method recovers the write latency it can determine the address and command 1T setting in half rate mode.

The returned read data, in the controller clock domain, should be equal across the first four read data beats, as aligned to the controller clock domain. For this check to work an alternate read location must be read immediately before and after that containing the write latency.

If read data is not correctly aligned then the address and command 1T setting is toggled and calibration is rerun from write training patterns stage.

## Calibrate Postamble (s\_poa)

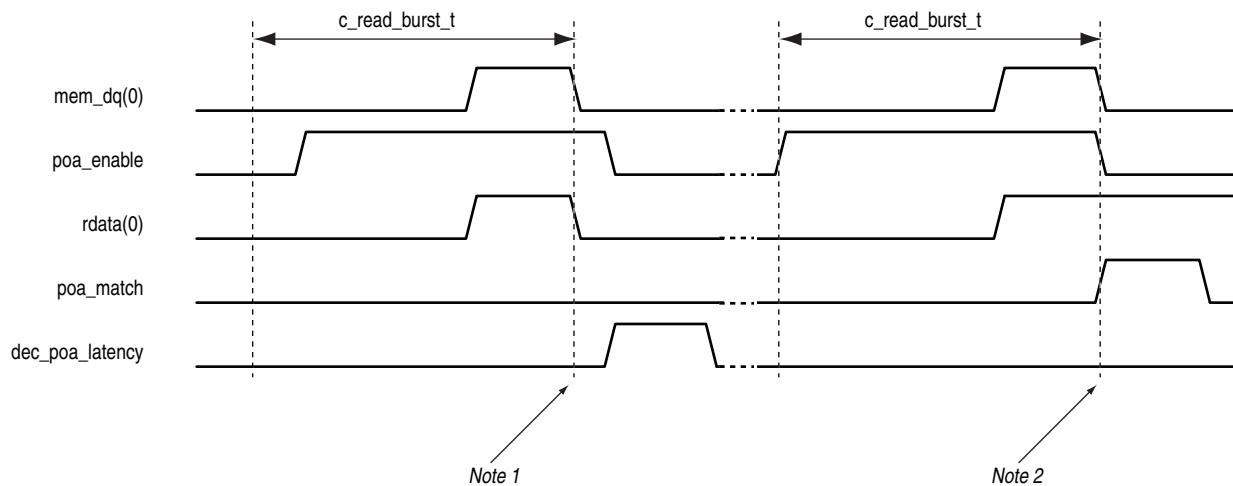
The ALTMEMPHY IP only implements this stage for DQS capture based PHYs (not used for Cyclone III and Cyclone IV devices).

For this stage, the PHY reads the pattern 0x30 from memory, so that the deassertion of the postamble protection signal (poa\_enable) can be aligned to the 1's in this pattern.

This stage sets the correct clock cycle for the postamble path. The aim of the postamble path is to eliminate false DQ data capture because of postamble glitches on the DQS signal, through an override on DQS. This stage ensures the correct clock cycle timing of the postamble enable (override) signal.

The delay on the postamble enable signal starts off too large. It is then iteratively reduced until postamble enable de-asserts in the clock cycle before the last falling edge on DQS. [Figure 2-17](#) shows the calibration timing diagram.

**Figure 2-17. Calibration Timing**



**Note to Figure 2-17:**

- (1) The `poa_enable` signal is late, and the zeros on `mem_dq` after here are captured.
- (2) The `poa_enable` signal is aligned. Zeros following here are not captured and `rdata` remains at 1.

## Set Up Address and Command Clock Cycle

For half-rate interfaces, this stage also optionally adds an additional memory clock cycle of delay from the address and command path. This stage aligns write data to memory commands given in the controller clock domain. You see this stage in the waveform as a rerun of calibration (from the writing of training patterns) to calibrate to the new setting.

This stage is detected in the advertise write latency stage (`s_adv_wlat`)

## Write User Mode Register Settings (`s_prep_customer_mr_setup`)

User mode register setting applies on a per chip select basis without the overrides in the program mode registers for calibration (`s_prog_mr`) stage.

## Voltage and Temperature Tracking

Voltage and temperature tracking is a background process that tracks the voltage and temperature variations to maintain the relationship between the resynchronization or capture clock and the data valid window that were achieved at calibration. When the data calibration phase completes, the sequencer issues the mimic calibration sequence every 128 ms (in user mode).

## Setup the Mimic Window (s\_tracking\_setup)

During initial calibration, the mimic path is sampled using the measure clock. The measure\_clk signal has a \_1x or \_2x suffix, depending whether the ALTMEMPHY IP is a full-rate or half-rate design. The sampled value is then stored by the sequencer. After a sample value is stored, the sequencer uses the PLL reconfiguration logic to change the phase of the measure clock by one voltage-controlled oscillator (VCO) phase tap. The sequencer then stores the sampled value for the new mimic path clock phase. This sequence continues until all mimic path clock phase steps are swept. After the sequencer stores all the mimic path sample values, it calculates the phase which corresponds to the center of the high period of the mimic path waveform. This reference mimic path sampling phase is used during the voltage and temperature tracking phase.

## Perform Tracking (s\_tracking)

In user mode, the sequencer periodically performs a tracking operation. At the end of the tracking calibration, the sequencer compares the most recent optimum tracking phase against the reference sampling phase. If the sampling phases do not match, the mimic path delays have changed because of voltage and temperature variations. When the sequencer detects that the mimic path reference and most recent sampling phases do not match, the sequencer uses the PLL reconfiguration logic to change the phase of the resynchronization clock by the VCO taps in the same direction. This procedure allows the tracking process to maintain the near-optimum capture clock phase setup during data tracking calibration as voltage and temperature vary over time. The relationship between the resynchronization or capture clock and the data valid window is maintained by measuring the mimic path variations because of the voltage and temperature variations and applying the same variation to the resynchronization clock.

## Document Revision History

Table 2–6 lists the revision history for this document.

**Table 2–6. Document Revision History**

Date	Version	Changes
November 2012	3.3	<ul style="list-style-type: none"> <li>■ Added <a href="#">Controller Register Map</a> information.</li> </ul>
June 2012	3.2	<ul style="list-style-type: none"> <li>■ Added Feedback icon</li> </ul>
November 2011	3.1	<ul style="list-style-type: none"> <li>■ Consolidated ALTMEMPHY FD information from 11.0 version <b>DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide</b> and <b>DDR3 SDRAM Controller with ALTMEMPHY IP User Guide</b>.</li> <li>■ Added <a href="#">ALTMEMPHY Calibration Stages</a> information.</li> </ul>

This chapter describes the hard (on-chip) memory interface components available in the Arria V and Cyclone V device families.

## Hard Memory Interface

The Arria V device family includes hard memory interface components supporting DDR2 and DDR3 SDRAM, LPDDR2, QDR II SRAM, and RLDRAM II memory protocols at speeds of up to 533 MHz. For the Quartus II software version 12.0 and later, the Cyclone V device family supports both hard and soft interface support.

## High-Level Feature Description

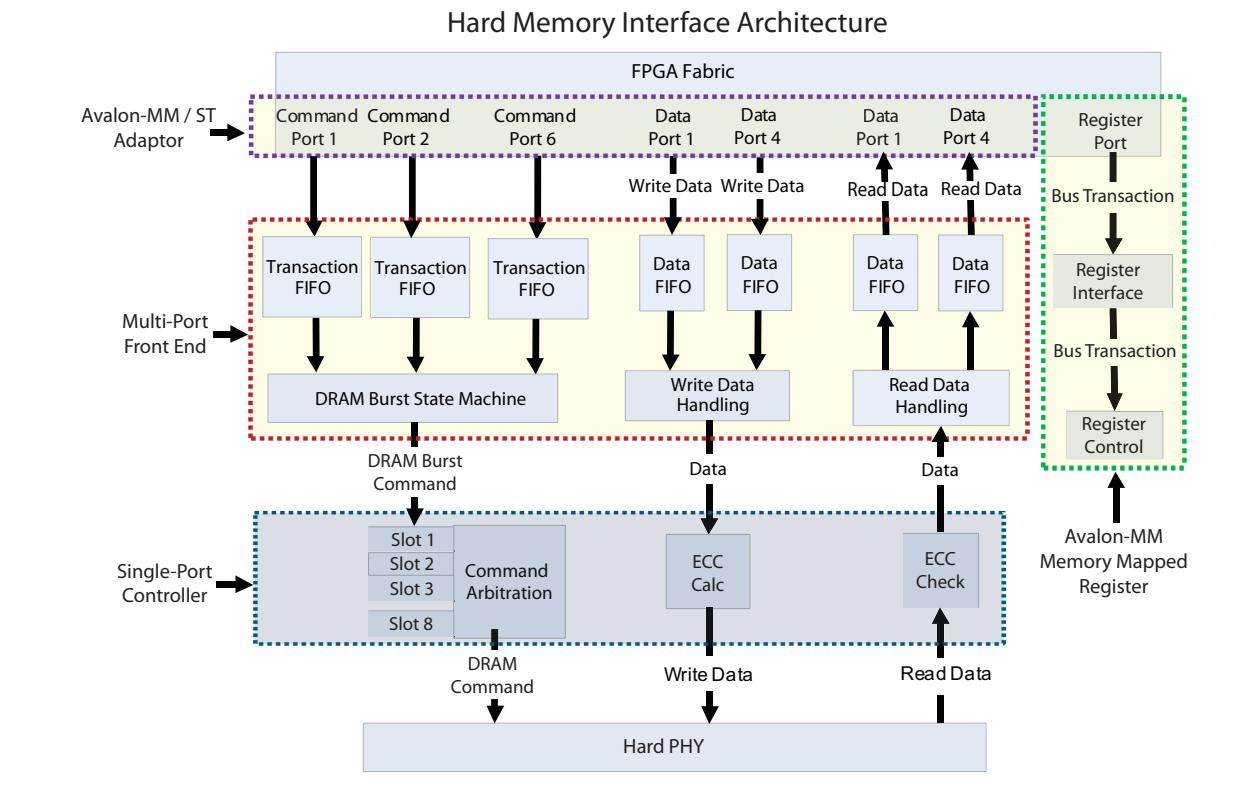
The hard memory interface consists of three main parts, as follows:

- The multi-port front end (MPFE), which allows multiple independent accesses to the hard memory controller.
- The hard memory controller, which initializes, refreshes, manages, and communicates with the external memory device.
- The hard PHY, which provides the physical layer interface to the external memory device.



Figure 3-1 shows the architecture of the Arria V hard memory interface.

**Figure 3-1. Hard Memory Interface Architecture**



## Multi-Port Front End (MPFE)

The multi-port front end and its associated fabric interface provide up to six command ports, four read-data ports and four write-data ports, through which user logic can access the hard memory controller. Each port can be configured as read only or write only, or read and write ports may be combined to form bidirectional data ports. Ports can be 32, 64, 128, or 256 data bits wide, depending on the number of ports used and the type (unidirectional or bidirectional) of the port.

## Fabric Interface

The fabric interface provides communication between the Avalon-ST-like internal protocol of the hard memory interface and the external Avalon-MM protocol. The fabric interface supports frequencies in the range of 10 MHz to one-half of the memory interface frequency. For example, for an interface running at 533 MHz, the maximum user logic frequency is 267 MHz. The MPFE handles the clock crossing between user logic and the hard memory interface.

Table 3–1 lists the types of ports available in the fabric interface.

**Table 3–1. Fabric Interface Port Types**

Port Type	Description
Fabric command ports	Accept both read and write commands.
Fabric 64-bit read data ports	Read and write data ports can be concatenated to form wider interfaces in power-of-two sizes (128-, 256-bit busses). Application which require only 32-bit interface can connect a 32-bit interface to the lower 32 bits of 64-bit port and configure that interface to be 32-bits wide. The remaining 32 nets to the hard memory controller are not usable when a port is configured to be 32-bits wide. 32-bit interfaces cannot be concatenated.
Fabric 64-bit write data ports	Provides access to address registers that control operation of the memory controller. Register values written across this interface can override values loaded during FPGA configuration.
Fabric register port	Paired with the fabric write data ports to provide return acknowledgement of write operations being committed. A read operation received after the write acknowledgement on any controller port for the same address will see the updated data.

## Operation Ordering

Requests arriving at a given port are executed in the order in which they are received.

Requests arriving at different ports have no guaranteed order of service, except when a first transaction has completed before the second arrives.

## Multi-port Scheduling

User-configurable priority and weight settings determine the absolute and relative scheduling policy for each port.

### Port Scheduling

Multi-port scheduling is governed by two considerations: the absolute priority of a request and the weighting of a port.

The evaluation of absolute priority ensures that ports carrying higher-priority traffic are served ahead of ports carrying lower-priority traffic. The scheduler recognizes eight priority levels, with higher values representing higher priorities. Priority is absolute; for example, any transaction with priority seven will always be scheduled before transactions of priority six or lower.

When ports carry traffic of the same absolute priority, relative priority is determined based on port weighting. Port weighting is a five-bit value, and is determined by a weighted round robin (WRR) algorithm.

The scheduler can alter priority if the latency target for a transaction is exceeded. The scheduler tracks latency on a per-port basis, and counts the cycles that a transaction is pending. Each port has a priority escalation register and a pending counter engagement register. If the number of cycles in the pending counter engagement register elapse without a pending transaction being served, that transaction's priority is escalated.

To ensure that high-priority traffic is served quickly and that long and short bursts are effectively interleaved on ports, bus transactions longer than a single DRAM burst are scheduled as a series of DRAM bursts, with each burst arbitrated separately.

The scheduler uses a form of deficit round robin (DRR) scheduling algorithm which corrects for past over-servicing or under-servicing of a port. Each port has an associated weight which is updated every cycle, with a user-configured weight added to it and the amount of traffic served subtracted from it. The port with the highest weighting is considered the most eligible.

To ensure that lower priority ports do not build up large running weights while higher priority ports monopolize bandwidth, the hard memory controller's DRR weights are updated only when a port matches the scheduled priority. Hence, if three ports have traffic, two being priority 7 and one being priority 4, the weights for both ports at priority 7 are updated but the port with priority 4 remains unchanged.

The scheduler can be configured to lock onto a given port for a specified number of transactions when the scheduler schedules traffic at that priority level. The number of transactions is configurable on a per-port basis. For ports with large numbers of sequential addresses, you can use this feature to allow efficient open page accesses without risk of the open page being pushed out by other transactions.

## DRAM Burst Scheduling

DRAM burst scheduling recognizes addresses that access the same column/row combination—also known as open page accesses. Such operations are always served in the order in which they are received in the single-port controller.

Selection of DRAM operations is a two-stage process; first, each pending transaction must wait for its timers to be eligible for execution, then the transaction arbitrates against other transactions that are also eligible for execution.

The following rules govern transaction arbitration:

- High priority operations take precedence over lower priority operations
- If multiple operations are in arbitration, read operations have precedence over write operations
- If multiple operations still exist, the oldest is served first

A high-priority transaction in the DRAM burst scheduler wins arbitration for that bank immediately if the bank is idle and the high-priority transaction's chip select/row/column address does not match an address already in the single-port controller. If the bank is not idle, other operations to that bank yield until the high-priority operation is finished. If the address matches another chip select/row/column, the high-priority transaction yeilds until the earlier transaction is completed.

You can force the DRAM burst scheduler to serve transactions in the order that they are received, by setting a bit in the register set.

## DRAM Power Saving Modes

The hard memory controller supports two DRAM power-saving modes: self-refresh, and fast/slow all-bank precharge powerdown exit.

Engagement of a DRAM power saving mode can occur due to inactivity, or in response to a user command.

The user command to enter power-down mode forces the DRAM burst-scheduling bank-management logic to close all banks and issue the power-down command. You can program the controller to power down when the DRAM burst-scheduling queue is empty for a specified number of cycles; the DRAM is reactivated when an active DRAM command is received.

## MPFE Signal Descriptions

Table 3–2 describes the signals for the multi-port front end.

**Table 3–2. MPFE Signals**

Signal	Direction	Description
avl_<signal_name>_#(1)	—	Local interface signals.
mp_cmd_clk_#_clk (1)	Input	Clock for the command FIFO buffer. Follow Avalon-MM master frequency. Maximum frequency is one-half of the interface frequency, and subject to timing closure.
mp_cmd_reset_n_#_reset_n (1)	Input	Reset signal for command FIFO buffer.
mp_rfifo_clk_#_clk (2)	Input	Clock for the read data FIFO buffer. Follow Avalon-MM master frequency. Maximum frequency is one-half of the interface frequency, and subject to timing closure.
mp_rfifo_reset_n_#_reset_n (2)	Input	Reset signal for read data FIFO buffer.
mp_wfifo_clk_#_clk (2)	Input	Clock for the write data FIFO buffer. Follow Avalon-MM master frequency. Maximum frequency is one-half of the interface frequency, and subject to timing closure.
mp_wfifo_reset_n_#_reset_n (2)	Input	Reset signal for write data FIFO buffer.
bonding_in_1/2/3	Input	Bonding interface input port. Connect second controller bonding output port to this port according to the port sequence.
bonding_out_1/2/3	Output	Bonding interface output port. Connect this port to the second controller bonding input port according to the port sequence.
<b>Note:</b>		
(1) # represents the number of the slave port. Values are 0—5.		
(2) # represents the number of the slave port. Values are 0—3.		

Every input interface (command, read data, and write data) has its own clock domain. Each command port can be connected to a different clock, but the read data and write data ports associated with a command port must connect to the same clock as that command port. Each input interface uses the same reset signal as its clock.

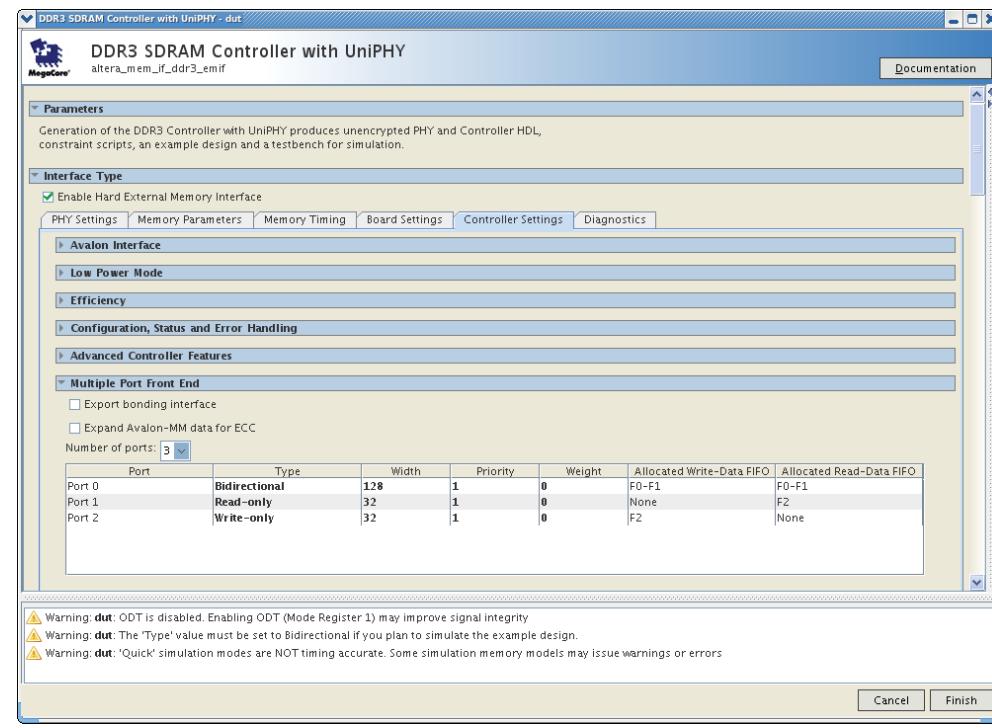
By default, the IP generates all clock signals regardless of the MPFE settings, but all unused ports and FIFO buffers are connected to ground.

The command ports can be used only in unidirectional configurations, with either 4 write and 2 read, 3 write and 3 read, or 2 write and 4 read scenarios. For bidirectional ports, the number of clocks is reduced from 6 to a maximum of 4.

For the scenario depicted in [Figure 3–2](#):

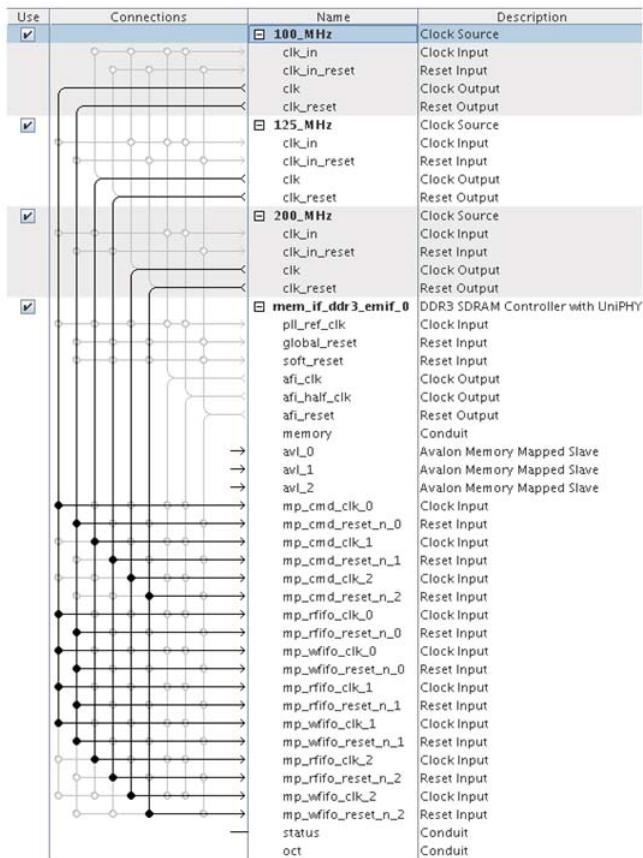
- command port 0 is associated with read and write data FIFO 0 and 1
- command port 1 is associated with read data FIFO 2
- command port 2 is associated with write data FIFO 2

**Figure 3–2. Sample MPFE Configuration**



Therefore, if port 0 (avl\_0) is clocked by a 100 MHz clock signal, mp\_cmd\_clk\_0, mp\_rfifo\_clk\_0, mp\_rfifo\_clk\_1, mp\_wfifo\_clk\_0, and mp\_wfifo\_clk\_1 must all be connected to the same 100 MHz clock, as illustrated in Figure 3–3.

**Figure 3–3. Sample Connection Mapping**



## Hard Memory Controller

The following sections describe the memory controller portion of the hard memory interface.



The hard memory controller is functionally similar to the High Performance Controller II (HPC II). For information on signals, refer to the [Functional Description—HPC II Controller](#) chapter.

## Clocking

The ports on the MPFE can be clocked at different frequencies, and synchronization is maintained by cross-domain clocking logic in the MPFE. Command ports can connect to different clocks, but the data ports associated with a given command port must be attached to the same clock as that command port. For example, a bidirectional command port that performs a 64-bit read/write function has its read port and write port connected to the same clock as the command port. Note that these clocks are separate from the EMIF core generated clocks.

## DRAM Interface

The DRAM interface is 40 bits wide, and can accommodate 8-bit, 16-bit, 16-bit plus ECC, 32-bit, or 32-bit plus ECC configurations. Any unused I/Os in the DRAM interface can be reused as user I/Os. The DRAM interface supports DDR2 and DDR3 memory protocols, and LPDDR2 for Cyclone V only. Fast and medium speed grade devices are supported to 533 MHz for Arria V and 400 MHz for Cyclone V.

## ECC

The hard controller supports both error-correcting code (ECC) calculated by the controller and by the user.

Controller ECC code employs standard Hamming logic which can detect and correct single-bit errors and detect double-bit errors. The controller ECC is available for 16-bit and 32-bit widths, each requiring an additional 8 bits of memory, resulting in an actual memory width of 24-bits and 40-bits, respectively.

In user ECC mode, all bits are treated as data bits, and are written to and read from memory. User ECC can implement nonstandard memory widths such as 24-bit or 40-bit, where ECC is not required.

### Controller ECC

Controller ECC provides the following features:

**Byte Writes**—The memory controller performs a read/modify/write operation to keep ECC valid when a subset of the bits of a word is being written. If an entire word is being written (but less than a full burst) and the DM pins are connected, no read is necessary and only that word is updated. If controller ECC is disabled, byte-writes have no performance impact.

**ECC Write Backs**—When a read operation detects a correctable error, the memory location is scheduled for a read/modify/write operation to correct the single-bit error.

**User ECC**—User ECC is 24-bits or 40-bits wide; with user ECC, the controller performs no ECC checking. The controller employs memory word addressing with byte enables, and can handle arbitrary memory widths. User ECC does not disable byte writes; hence, you must ensure that any byte writes do not result in corrupted ECC.

## Bonding of Memory Controllers

Bonding is a feature that allows data to be split between two memory controllers, providing the ability to service bandwidth streams similar to a single 64-bit controller. Bonding works by dividing data buses in proportion to the memory widths, and always sending a transaction to both controllers. When signals are returned, bonding ensures that both sets of signals are returned identically.

Bonding can be applied to asymmetric controllers, and allows controllers to have different memory clocks. Bonding does not attempt to synchronize the controllers. Bonding supports only one port. The Avalon port width can be varied from 64-bit to 256-bit; 32-bit port width is not supported.

The following signals require bonding circuitry:

**Read data return**—This bonding allows read data from the two controllers to return with effectively one ready signal to the bus master that initiated the bus transaction.

**Write ready**—For Avalon-MM, this is effectively bonding on the `waitrequest` signal.

**Write acknowledge**—Synchronization on returning the write completed signal.

For each of the above implementations, data is returned in order, hence the circuitry must match up for each valid cycle.

Bonded FIFO buffers must have identical FIFO numbers; that is, read FIFO 1 on controller 1 must be paired with Read FIFO 1 on controller 2.

## Data Return Bonding

Long loop times can lead to communications problems when using bonded controllers. The following effects are possible when using bonded controllers:

- If one memory controller completes its transaction and receives new data before the other controller, then the second controller can send data as soon as it arrives, and before the first controller acknowledges that the second controller has data.
- If the first controller has a single word in its FIFO buffer and the second controller receives single-word transactions, the second controller must determine whether the second word is a valid signal or not.

To accommodate the above effects, the hard controller maintains two counters for each bonded pair of FIFO buffers and implements logic that monitors those counters to ensure that the bonded controllers receive the same data on the same cycle, and that they send the data out on the same cycle.

## FIFO Ready

FIFO ready bonding is used for write command and write data buses. The implementation is similar to the data return bonding.

## Bonding Latency Impact

Bonding has no latency impact on ports that are not bonded.

## Bonding Controller Usage

Arria V and Cyclone V devices employ three shared bonding controllers to manage the read data return bonding, write acknowledge bonding, and command/write data ready bonding.

The three bonding controllers require three pairs of bonding I/Os, each based on a six port count; this means that a bonded hard memory controller requires 21 input signals and 21 output signals for its connection to the fabric, and another 21 input signals and 21 output signals to the paired hard memory controller.



The hard processor system (HPS) hard memory controller cannot be bonded with another hard memory controller on the FPGA portion of the device.

## Hard PHY

A physical layer interface (PHY) is embedded in the periphery of the Arria V device, and can run at the same high speed as the hard controller and hard sequencer. The hard PHY is located next to the hard controller. Differing device configurations have different numbers and sizes of hard controller and hard PHY pairs.

The hard PHY implements logic that connects the hard controller to the I/O ports. Because the hard controller and AFI interface support high frequencies, a portion of the sequencer is implemented as hard logic. The Nios II processor, the instruction/data RAM, and the Avalon fabric of the sequencer are implemented as core soft logic. The read/write manager and PHY manager components of the sequencer, which must operate at full rate, are implemented as hard logic in the hard PHY.

## Interconnections

The hard PHY resides on the device between the hard controller and the I/O register blocks. The hard PHY is instantiated or bypassed entirely, depending on the parameterization that you specify.

The hard PHY connects to the hard memory controller and the core, enabling the use of either the hard memory controller or a software-based controller. (You can have the hard controller and hard PHY, or the soft controller and soft PHY; however, the combination of soft controller with hard PHY is not supported.) The hard PHY also connects to the I/O register blocks and the DQS logic. The path between the hard PHY and the I/O register blocks can be bypassed, but not reconfigured—in other words, if you use the hard PHY datapath, the pins to which it connects are predefined and specified by the device pin table.

## Clock Domains

The hard PHY contains circuitry that uses the following clock domains:

**AFI clock domain (pll\_afi\_clk)**—The main full-rate clock signal that synchronizes most of the circuit logic.

**Avalon clock domain (pll\_avl\_clk)**—Synchronizes data on the internal Avalon bus, namely the Read/Write Manager, PHY Manager, and Data Manager data. The data is then transferred to the AFI clock domain. To ensure reliable data transfer between clock domains, the Avalon clock period must be an integer multiple of the AFI clock period, and the phases of the two clocks must be aligned.

**Address and Command clock domain (pll\_addr\_cmd\_clk)**—Synchronizes the global asynchronous reset signal, used by the I/Os in this clock domain.

## Hard Sequencer

The sequencer initializes the memory device and calibrates the I/Os, with the objective of maximizing timing margins and achieving the highest possible performance. When the hard memory controller is in use, a portion of the sequencer must run at full rate; for this reason, the Read/Write Manager, PHY Manager, and Data Manager are implemented as hard components within the hard PHY. The hard sequencer communicates with the soft-logic sequencer components (including the Nios II processor) via an Avalon bus.

## Hard Memory Interface Implementation Guidelines

The following sections provide guidelines for implementing a hard memory interface.

### MPFE Setup Guidelines

This section provides information on configuring the multi-port front end of the hard memory interface.

1. To enable the hard memory interface, turn on **Enable Hard External Memory Interface** in the **Interface Type** tab in the parameter editor.
2. To export bonding interface ports to the top level, turn on **Export bonding interface** in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor.

 The system exports three bonding-in ports and three bonding-out ports. You must generate two controllers and connect the bonding ports manually.

3. To expand the interface data width from a maximum of 32 bits to a maximum of 40 bits, turn on **Enable Avalon-MM data for ECC** in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor.

 The controller does not perform ECC checking when this option is turned on.

4. Select the required **Number of ports** for the multi-port front end in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor.

 The maximum number of ports is 6, depending on the port type and width. The maximum port width is 256 bits, which is the maximum data width of the read data FIFO and write data FIFO buffers.

5. The table in the **Multiple Port Front End** pulldown on the **Controller Settings** tab in the parameter editor lists the ports that are created. The columns in the table describe each port, as follows:

- **Port:** Indicates the port number.
- **Type:** Indicates whether the port is read only, write only, or bidirectional.
- **Width:** To achieve optimum MPFE throughput, Altera recommends setting the MPFE data port width according to the following calculation:  

$$2 \times (\text{frequency ratio of HMC to user logic}) \times (\text{interface data width})$$

For example, if the frequency of your user logic is one-half the frequency of the hard memory controller, you should set the port width to be 4x the interface data width. If the frequency ratio of the hard memory controller to user logic is a fractional value, you should use a larger value; for example, if the ratio is 1.5, you can use 2.
- **Priority:** The priority setting specifies the priority of the slave port, with higher values representing higher priority. The slave port with highest priority is served first.
- **Weight:** The weight setting has a range of values of 0–31, and specifies the relative priority of a slave port, with higher weight representing higher priority. The weight value can determine relative bandwidth allocations for slave ports with the same priority values.

For example, if two ports have the same priority value, and weight values of 4 and 6, respectively, the port with a weight of 4 will receive 40% of the bus bandwidth, while the port with a weight of 6 will receive 60% of the bus bandwidth—assuming 100% total available bus bandwidth.

## Soft Memory Interface to Hard Memory Interface Migration Guidelines

This section provides information on mapping your soft memory interface to a hard memory interface.

### Pin Connections

1. The hard and soft memory controllers have compatible pinouts. Assign interface pins to the hard memory interface according to the pin table.
2. Ensure that your soft memory interface pins can fit into the hard memory interface. The hard memory interface can support a maximum of a 40-bit interface with user ECC, or a maximum of 80-bits with same-side bonding. The soft memory interface does not support bonding.
3. Follow the recommended board layout guidelines for the hard memory interface.

### Software Interface Preparation

Observe the following points in preparing your soft memory interface for migration to a hard memory interface:

- You cannot use the hard PHY without also using the hard memory controller.
- The hard memory interface supports only full-rate controller mode.

- Ensure that the MPFE data port width is set according to the soft memory interface half-rate mode Avalon data width.
- The hard memory interface uses a different Avalon port signal naming convention than the software memory interface. Ensure that you change the `.avl_<signal_name>` signals in the soft memory interface to `.avl_<signal_name>_0` signals in the hard memory interface.
- The hard memory controller MPFE includes an additional three clocks and three reset ports (CMD port, RFIFO port, and WFIFO port) that do not exist with the soft memory controller. You should connect the user logic clock signal to the MPFE clock port, and the user logic reset signal to the MPFE reset port.
- In the soft memory interface, the half-rate `afi_clk` is a user logic clock. In the hard memory interface, `afi_clk` is a full-rate clock, because the core fabric may not be able to achieve full-rate speed. When you migrate your soft memory interface to a hard memory interface, you need to supply an additional slower rate clock. The maximum clock rate supported by core logic is one-half of the maximum interface frequency.

### **Latency**

Overall, you should expect to see slightly more latency when using the hard memory controller and multi-port front end, than when using the soft memory controller.

The hard memory controller typically exhibits lower latency than the soft memory controller; however, the multi-port front end does introduce additional latency cycles due to FIFO buffer stages used for synchronization. The MPFE cannot be bypassed, even if only one port is needed.

## **Bonding Interface Guidelines**

Bonding allows a single data stream to be split between two memory controllers, providing the ability to expand the interface data width similar to a single 64-bit controller. This section provides some guidelines for setting up the bonding interface.

1. Bonding interface ports are exported to the top level in your design. You should connect each `bonding_in_*` port in one hard memory controller to the corresponding `bonding_out_*` port in the other hard memory controller, and vice versa.

2. You should modify the Avalon signal connections to drive the bonding interface with a single user logic/master, as follows:
  - a. AND both avl\_ready signals from both hard memory controllers before the signals enter the user logic.
  - b. AND both avl\_rdata\_valid signals from both hard memory controllers before the signals enter the user logic. (The avl\_rdata\_valid signals should be identical for both hard memory controllers.)
  - c. Branch the following signals from the user logic to both hard memory controllers:
    - avl\_burstbegin
    - avl\_addr
    - avl\_read\_req
    - avl\_write\_req
    - avl\_size
  - d. Split the following signals according to each multi-port front end data port width:
    - avl\_rdata
    - avl\_wdata
    - avl\_be

## Document Revision History

Table 3-3 lists the revision history for this document.

**Table 3-3. Document Revision History**

Date	Version	Changes
November 2012	2.1	<ul style="list-style-type: none"> <li>■ Added <a href="#">Hard Memory Interface Implementation Guidelines</a>.</li> <li>■ Moved content of EMI-Related HPS Features in SoC Devices section to chapter 4. Functional Description—HPS Memory Controller.</li> </ul>
June 2012	2.0	<ul style="list-style-type: none"> <li>■ Added EMI-Related HPS Features in SoC Devices.</li> <li>■ Added LPDDR2 support.</li> <li>■ Added Feedback icon.</li> </ul>
November 2011	1.0	Initial release.

The hard processor system (HPS) SDRAM controller subsystem provides efficient access to external SDRAM for the ARM® Cortex™-A9 microprocessor unit (MPU) subsystem, the level 3 (L3) interconnect, and the FPGA fabric. The SDRAM controller provides an interface between the FPGA fabric and HPS. The interface accepts Advanced Microcontroller Bus Architecture (AMBA®) Advanced eXtensible Interface (AXI™) and Avalon® Memory-Mapped (Avalon-MM) transactions, converts those commands to the correct commands for the SDRAM, and manages the details of the SDRAM access.

### Features of the SDRAM Controller Subsystem

The SDRAM controller subsystem offers the following features:

- Support for double data rate 2 (DDR2), DDR3, and low-power DDR2 (LPDDR2) SDRAM
- User-configurable timing parameters
- Up to 4 Gb density parts
- Two chip selects
- Integrated error correction code (ECC), 24- and 40-bit widths
- User-configurable memory width of 8, 16, 16+ECC, 32, 32+ECC
- Command reordering (look-ahead bank management)
- Data reordering (out of order transactions)
- User-controllable bank policy on a per port basis for either closed page or conditional open page accesses
- User-configurable priority support with both absolute and relative priority scheduling
- Flexible FPGA fabric interface configuration with up to 6 ports and data widths up to 256 bits wide using Avalon-MM and AXI interfaces.
- Power management supporting self refresh, partial array self-refresh (PASR), power down, and LPDDR2 deep power down

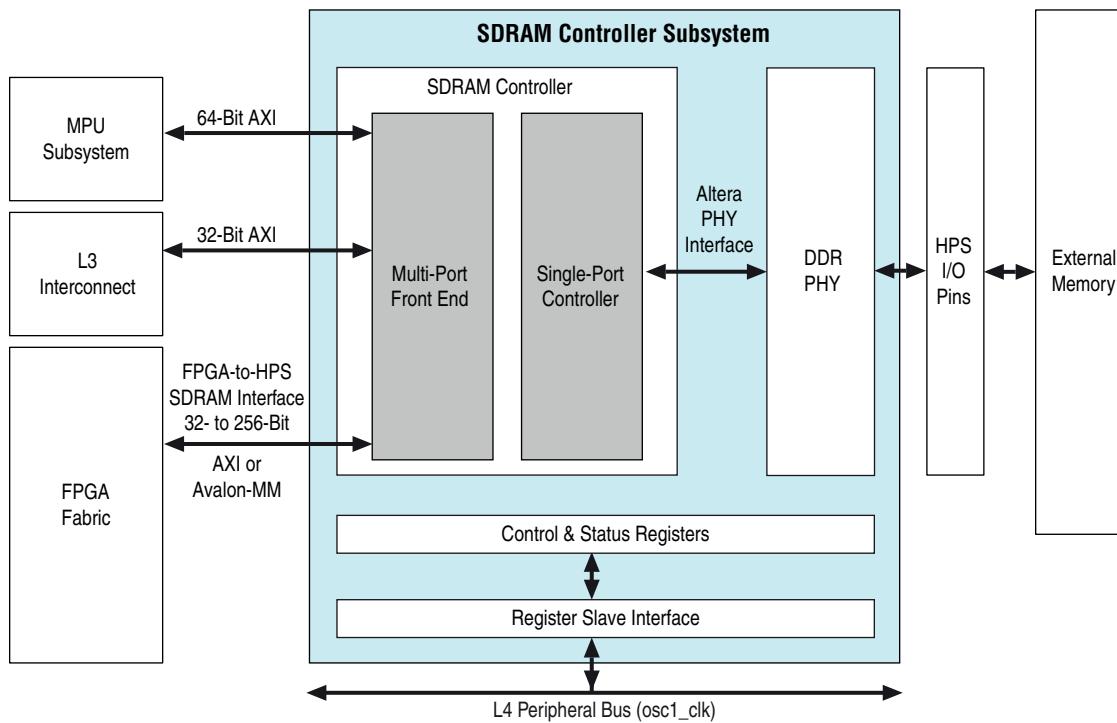


## SDRAM Controller Subsystem Block Diagram and System Integration

The SDRAM controller subsystem connects to the MPU subsystem, the main switch of the L3 interconnect, and the FPGA fabric. The memory interface consists of the SDRAM controller, the physical layer (PHY), control and status registers (CSRs), and their associated interfaces.

Figure 4–1 shows a high-level block diagram of the SDRAM controller subsystem.

**Figure 4–1. SDRAM Controller Subsystem High-Level Block Diagram**



### SDRAM Controller

The SDRAM controller provides high performance data access and run-time programmability. The controller reorders data to reduce row conflicts and bus turn-around time by grouping read and write transactions together, allowing for efficient traffic patterns and reduced latency.

The SDRAM controller consists of a multiport front end (MPFE) and a single-port controller. The MPFE provides multiple independent interfaces to the single-port controller. The single-port controller communicates with and manages each external memory device. For more information, refer to “[Memory Controller Architecture](#)” on page 4–4.

### DDR PHY

The DDR PHY provides a physical layer interface between the memory controller and memory devices, which performs read and write memory operations. The DDR PHY has dataflow components, control components, and calibration logic that handle the calibration for the SDRAM interface timing.

## SDRAM Controller Subsystem Interfaces

The following sections describe the SDRAM controller subsystem interfaces.

### MPU Subsystem Interface

The SDRAM controller is connected to the MPU subsystem with a dedicated 64-bit AXI interface, operating on the `mpu_12_ram_clk` clock domain.

### L3 Interconnect Interface

The SDRAM controller is connected to the L3 interconnect with a dedicated 32-bit AXI interface, operating on the `l3_main_clk` clock domain.

### CSR Interface

The CSR interface is connected to the level 4 (L4) bus and operates on the `l4_sp_clk` clock domain. The MPU subsystem uses the CSR interface to configure the controller and PHY, for example, setting the memory timing parameter values or placing the memory to a low power state. The CSR interface also provides access to the status registers in the controller and PHY.

### FPGA-to-HPS SDRAM Interface

The FPGA-to-HPS SDRAM interface provides masters implemented in the FPGA fabric access to the SDRAM controller subsystem in the HPS. The interface has three port types that are used to construct the following AXI or Avalon-MM interfaces:

- Command ports—issue read and write commands, and receive write acknowledge responses
- 64-bit read data ports—receive data returned from a memory read
- 64-bit write data ports—transmit write data

The FPGA-to-HPS SDRAM interface supports six command ports, allowing up to six Avalon-MM interfaces or three AXI interfaces. Each command port can be used to implement either a read or write command port for AXI, or be used as part of an Avalon-MM interface. The AXI and Avalon-MM interfaces can be configured to support 32-, 64-, 128-, and 256-bit data.

Table 4-1 lists the FPGA-to-HPS SDRAM controller interface ports connected to the FPGA.

**Table 4-1. FPGA-to-HPS SDRAM Controller Port Types**

Port Type	Number
Command	6
64-bit read data	4
64-bit write data	4

The FPGA-to-HPS SDRAM controller interface can be configured with the following characteristics:

- Avalon-MM interfaces and AXI interfaces can be mixed and matched as required by the fabric logic, within the bounds of the number of ports provided to the fabric.

- Each Avalon-MM or AXI interface of the FPGA-to-HPS SDRAM interface operates on an independent clock domain.
- The FPGA-to-HPS SDRAM interfaces are configured during FPGA configuration.

Table 4–2 shows the number of ports needed to configure different bus protocols, based on type and data width.

**Table 4–2. FPGA-to-HPS SDRAM Port Utilization**

Bus Protocol	Command	Read Data	Write Data
32- or 64-bit AXI	2 (1)	1	1
128-bit AXI	2 (1)	2 (2)	2 (2)
256-bit AXI	2 (1)	4 (2)	4 (2)
32- or 64-bit Avalon-MM	1	1	1
128-bit Avalon-MM	1	2	2
256-bit Avalon-MM	1	4	4
32- or 64-bit Avalon-MM write-only	1	0	1
128-bit Avalon-MM write-only	1	0	2
256-bit Avalon-MM write-only	1	0	4
32- or 64-bit Avalon-MM read-only	1	1	0
128-bit Avalon-MM read-only	1	2	0
256-bit Avalon-MM read-only	1	4	0

**Notes to Table 4–2:**

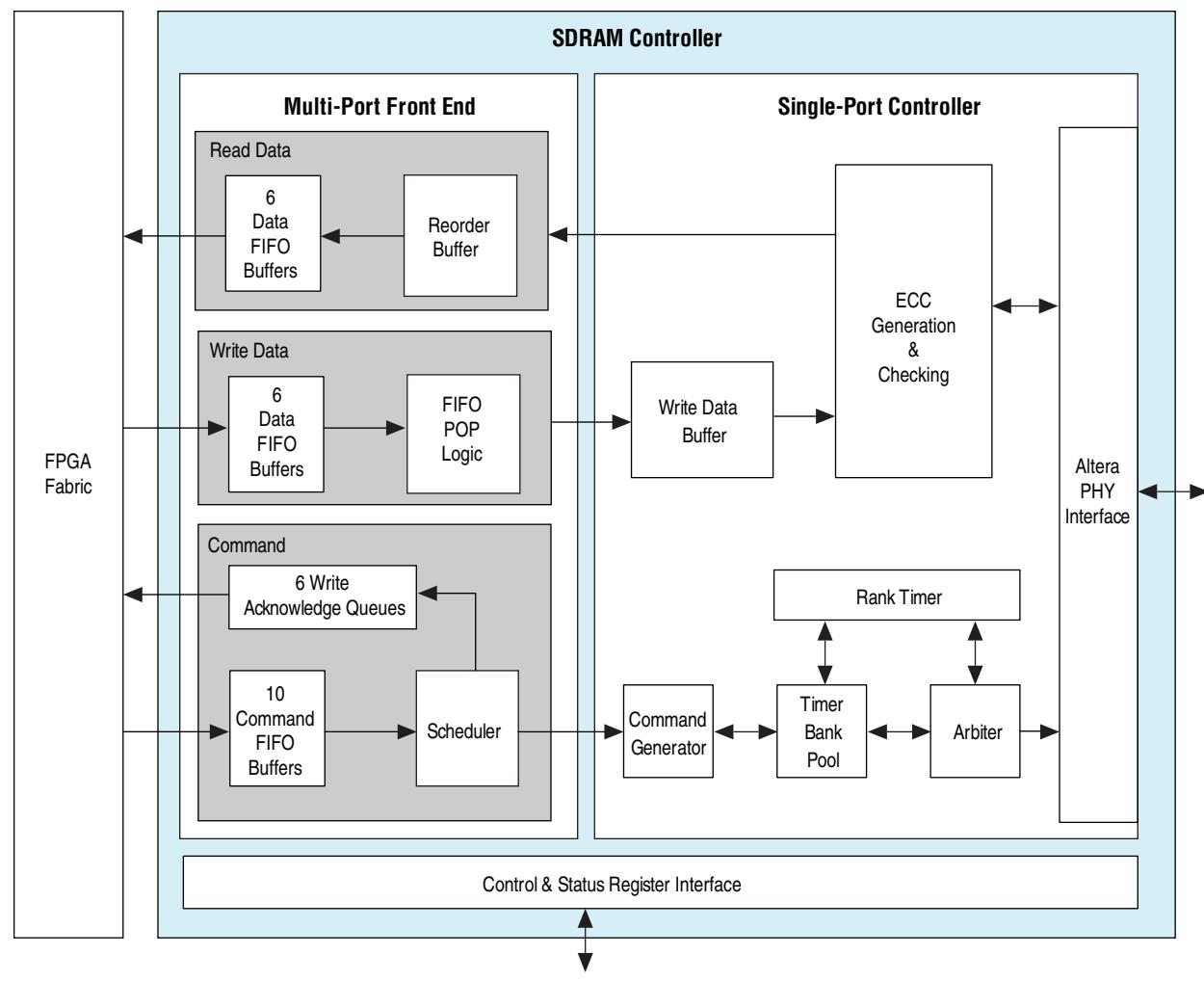
- (1) Because the AXI protocol allows simultaneous read and write commands to be issued, two SDRAM control ports are required to form an AXI interface.
- (2) Because the native size of the data ports is 64 bits, extra read and write ports are required to form an AXI interface.

## Memory Controller Architecture

The SDRAM controller consists of an MPFE, a single-port controller, and an interface to the CSRs.

Figure 4–2 shows a block diagram of the SDRAM controller portion of the SDRAM controller subsystem.

**Figure 4–2. SDRAM Controller Block Diagram**



## MPFE

The MPFE is responsible for scheduling pending transactions from the configured interfaces and sending the scheduled memory transactions to the single-port controller. The MPFE handles all functions related to individual ports.

The MPFE consists of the following three primary sub-blocks.

### Command Block

The command block accepts read and write transactions from the FPGA fabric and the HPS. When the command FIFO buffer is full, the command block applies backpressure by deasserting the ready signal. For each pending transaction, the command block calculates the next SDRAM burst needed to progress on that transaction. The command block schedules pending SDRAM burst commands based on the user-supplied configuration, available write data, and unallocated read data space.

## Write Data Block

The write data block transmits data to the single-port controller. The write data block maintains write data FIFO buffers and clock boundary crossing for the write data. The write data block informs the command block of the amount of pending write data for each transaction so that the command block can calculate eligibility for the next SDRAM write burst.

## Read Data Block

The read data block receives data from the single-port controller. Depending on the port state, the read data block either buffers the data in its internal buffer or passes the data straight to the clock boundary crossing FIFO buffer. The read data block reorders out-of-order data for Avalon-MM ports.

In order to prevent the read FIFO buffer from overflowing, the read data block informs the command block of the available buffer area so the command block can pace read transaction dispatch.

## Single-Port Controller

The single-port logic is responsible for following actions:

- Queuing the pending SDRAM bursts
- Choosing the most efficient burst to send next
- Keeping the SDRAM pipeline full
- Ensuring all SDRAM timing parameters are met

Transactions passed to the single-port logic for a single page in SDRAM are guaranteed to be executed in order, but transactions can be reordered between pages. Each SDRAM burst read or write is converted to the appropriate Altera PHY interface (AFI) command to open a bank on the correct row for the transaction (if required), execute the read or write command, and precharge the bank (if required).

The single-port logic implements command reordering (looking ahead at the command sequence to see which banks can be put into the correct state to allow a read or write command to be executed) and data reordering (allowing data transactions to be dispatched even if the data transactions are executed in an order different than they were received from the multiport logic).

## Command Generator

The command generator accepts commands from the MPFE and from the internal ECC logic, and provides those commands to the timer bank pool.

## Timer Bank Pool

The timer bank pool is a parallel queue that operates with the arbiter to enable data reordering. The timer bank pool tracks incoming requests, ensures that all timing requirements are met, and, on receiving write-data-ready notifications from the write data buffer, passes the requests to the arbiter.

## **Arbiter**

The arbiter determines the order in which requests are passed to the memory device. When the arbiter receives a single request, that request is passed immediately. When multiple requests are received, the arbiter uses arbitration rules to determine the order to pass requests to the memory device.

## **Rank Timer**

The rank timer performs the following functions:

- Maintains rank-specific timing information
- Ensures that only four activates occur within a specified timing window
- Manages the read-to-write and write-to-read bus turnaround time
- Manages the time-to-activate delay between different banks

## **Write Data Buffer**

The write data buffer receives write data from the MPFE and passes the data to the PHY, on approval of the write request.

## **ECC Block**

The ECC block consists of an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC block can correct single-bit errors and detect double-bit errors resulting from noise or other impairments during data transmission.

## **AFI Interface**

The AFI interface provides communication between the controller and the PHY.

## **CSR Interface**

The CSR interface is accessible from the L4 bus. The interface allows code executing in the HPS MPU and FPGA fabric to configure and monitor the SDRAM controller.

# **Functional Description of the SDRAM Controller Subsystem**

This section provides a functional description of the SDRAM controller subsystem.

## **MPFE Operational Behavior**

This section describes the operational behavior of the MPFE.

### **Operation Ordering**

Requests to the same SDRAM page arriving at a given port are executed in the order in which they are received. Requests arriving at different ports have no guaranteed order of service, except when a first transaction has completed before the second arrives.

Operation ordering is defined and enforced within a port, but not between ports. All transactions received on a single port for overlapping addresses execute in order. Transactions received on different ports have no guaranteed order unless the second transaction is presented after the first has completed.

Avalon-MM does not support write acknowledgement. When a port is configured to support Avalon-MM, you should read from the location that was previously written to ensure that the write operation has completed. When a port is configured to support AXI, the master accessing the port can safely issue a read operation to the same address as a write operation as soon as the write has been acknowledged. To keep write latency low, writes are acknowledged as soon as the transaction order is guaranteed—meaning that any operations received on any port to the same address as the write operation are executed after the write operation.

To ensure that the overall latency of traffic is as low as possible, the single port logic can return read data out of order to the multi-port logic which will reorder it when transactions return out of order. A large percentage of traffic reordering will be between ports and transactions only are ordered within a port. For traffic which is reordered between ports but not within a port, no reordering needs to be done. Eliminating unnecessary reordering reduces average latency.

## Multiport Scheduling

Multiport scheduling is governed by two factors, the absolute priority of a request and the weighting of a port.

The evaluation of absolute priority ensures that ports carrying higher-priority traffic are served ahead of ports carrying lower-priority traffic. The scheduler recognizes eight priority levels (0-7), with higher values representing higher priorities. For example, any transaction with priority seven is scheduled before transactions of priority six or lower.

When ports carry traffic of the same absolute priority, relative priority is determined based on port weighting. Port weighting is a five-bit value (0-31), and is determined by a deficit-weighted round robin (DWRR) algorithm, which corrects for past over-servicing or under-servicing of a port. Each port has an associated weight which is updated every cycle, with a user-configured weight added to it and the amount of traffic served subtracted from it. The port with the highest weighting is considered the most eligible.

To ensure that high-priority traffic is served quickly and that long and short bursts are effectively interleaved between ports, incoming transactions longer than a single SDRAM burst are scheduled as a series of SDRAM bursts, with each burst arbitrated separately.

To ensure that lower priority ports do not build up large running weights while higher priority ports monopolize bandwidth, the controller's DWRR weights are updated only when a port matches the scheduled priority. Therefore, if three ports are being accessed, two being priority seven and one being priority four, the weights for both ports at priority seven are updated but the port with priority four remains unchanged.

Multiport scheduling is performed between all of the ports connected to the FPGA fabric and internally in the HPS to determine which transaction is serviced next. Arbitration is performed on a SDRAM burst basis to ensure that a long transaction does not lock other transactions or cause latency to significantly increase for high-priority ports.

Arbitration supports both absolute and relative priority. Absolute priority is intended for applications where one master should always get priority above or below others. Relative priority is supported through a programmable weight field which controls scheduling between ports at the same priority.

The scheduler is work-conserving. Write operations can only be scheduled when enough data for the SDRAM burst has been received. Read operations can only be scheduled when sufficient internal memory is free and the port is not occupying too much of the read buffer.

The multiport scheduling configuration can be updated while traffic is flowing. Both priority and weight for a port can be updated without interrupting traffic on a port. Updates are used in scheduling decisions within 10 memory clock cycles of being updated, so priority can be updated frequently if needed.

### **Read Data Handling**

The MPFE contains a read buffer shared by all ports. If a port is capable of receiving returned data then the read buffer is bypassed. If the size of a read transaction is smaller than twice the memory interface width, the buffer RAM cannot be bypassed.

### **SDRAM Burst Scheduling**

SDRAM burst scheduling recognizes addresses that access the same row/bank combination, known as open page accesses. Operations to a page are served in the order in which they are received by the single-port controller.

Selection of SDRAM operations is a two-stage process. First, each pending transaction must wait for its timers to be eligible for execution. Next, the transaction arbitrates against other transactions that are also eligible for execution.

The following rules govern transaction arbitration:

- High-priority operations take precedence over lower-priority operations
- If multiple operations are in arbitration, read operations have precedence over write operations
- If multiple operations still exist, the oldest is served first

A high-priority transaction in the SDRAM burst scheduler wins arbitration for that bank immediately if the bank is idle and the high-priority transaction's chip select, row, or column fields of the address do not match an address already in the single-port controller. If the bank is not idle, other operations to that bank yield until the high-priority operation is finished. If the chip select, row, and column fields match an earlier transaction, the high-priority transaction yields until the earlier transaction is completed.

## Clocking

The FPGA fabric ports of the MPFE can be clocked at different frequencies. Synchronization is maintained by clock-domain crossing logic in the MPFE. Command ports can operate on different clock domains, but the data ports associated with a given command port must be attached to the same clock as that command port. For example, a command port paired with a read and write port to form an Avalon-MM interface must operate at the same clock frequency as the data ports associated with it.

## Single-Port Controller Operational Behavior

This section describes the operational behavior of the single-port controller.

### SDRAM Interface

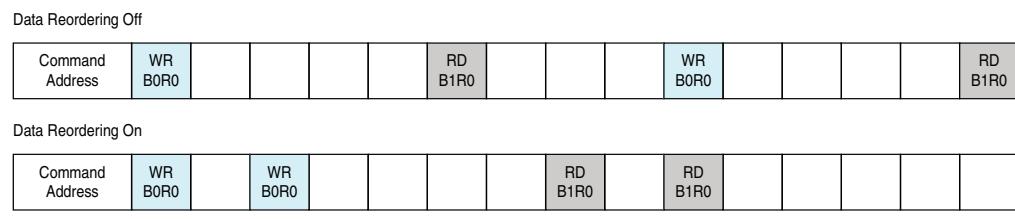
The SDRAM interface is up to 40 bits wide and can accommodate 8-bit, 16-bit, 16-bit plus ECC, 32-bit, or 32-bit plus ECC configurations. The SDRAM interface supports LPDDR2, DDR2, and DDR3 memory protocols.

### Command and Data Reordering

The heart of the SDRAM controller is a command and data reordering engine. Command reordering allows banks for future transactions to be opened before the current transaction finishes. Data reordering allows transactions to be serviced in a different order than they were received when that new order allows for improved utilization of the SDRAM bandwidth. Operations to the same bank and row are performed in order to ensure that operations which impact the same address preserve the data integrity.

**Figure 4–3** shows the relative timing for a write/read/write/read command sequence performed in order and then the same command sequence performed with data reordering. Data reordering allows the write and read operations to occur in bursts, without bus turnaround timing delay or bank reassignment.

**Figure 4–3. Data Reordering Effect**



The SDRAM controller schedules among all pending row and column commands every clock cycle.

### Bank Policy

The bank policy of the SDRAM controller allows users to request that a transaction's bank remain open after an operation has finished so that future accesses do not delay in activating the same bank and row combination. The controller supports only eight simultaneously-opened banks, so an open bank might get closed if the bank resource is needed for other operations.

Open bank resources are allocated dynamically as SDRAM burst transactions are scheduled. Bank allocation is requested automatically by the controller when an incoming transaction spans multiple SDRAM bursts or by the extended command interface. When a bank must be reallocated, the least-recently-used open bank is used as the replacement.

If the controller determines that the next pending command will cause the bank request to not be honored, the bank might be held open or closed depending on the pending operation. A request to close a bank with a pending operation in the timer bank pool to the same row address causes the bank to remain open. A request to leave a bank open with a pending command to the same bank but a different row address causes a precharge operation to occur.

### **Write Combining**

The SDRAM controller combines write operations from successive bursts on a port where the starting address of the second burst is one greater than the ending address of the first burst and the resulting burst length does not overflow the 11-bit burst-length counters. Write combining does not occur if the previous bus command has finished execution before the new command has been received.

### **Burst Length Support**

The controller supports burst lengths of 2, 4, 8, and 16, and data widths of 8, 16, and 32 bits for non-ECC operation, and widths of 24 and 40 operations with ECC enabled.

**Table 4-3** shows the type of SDRAM for each burst length.

**Table 4-3. SDRAM Burst Lengths**

<b>Burst Length</b>	<b>SDRAM</b>
4	LPDDR2, DDR2
8	DDR2, DDR3, LPDDR2
16	LPDDR2

### **Width Matching**

The SDRAM controller automatically performs data width conversion.

### **ECC**

The single-port controller supports memory ECC calculated by the controller. The controller ECC employs standard Hamming logic to detect and correct single-bit errors and detect double-bit errors. The controller ECC is available for 16-bit and 32-bit widths, each requiring an additional 8 bits of memory, resulting in an actual memory width of 24-bits and 40-bits, respectively.

Controller ECC provides the following features:

- **Byte writes**—The memory controller performs a read-modify-write operation to ensure that the ECC data remains valid when a subset of the bits of a word is being written. If an entire word is being written (but less than a full burst) and the DM pins are connected, no read is necessary and only that word is updated. If controller ECC is disabled, byte-writes have no performance impact.

- ECC write backs—When a read operation detects a correctable error, the memory location is scheduled for a read-modify-write operation to correct the single-bit error. ECC write backs are enabled and disabled through the `cfg_enable_ecc_code_overwrites` field in the `ctrlcfg` register.
- Notification of ECC errors—The memory controller provides interrupts for single-bit and double-bit errors. The status of interrupts and errors are recorded in status registers, as follows:
  - The `dramsts` register records interrupt status.
  - The `dramintr` register records interrupt masks.
  - The `sbeccount` register records the single-bit error count.
  - The `dbecount` register records the double-bit error count.
  - The `erraddr` register records the address of the most recent error.

### Byte Writes

Byte writes with ECC enabled are executed as a read-modify-write. Typical operations only use a single entry in the timer bank pool. Controller ECC enabled sub-word writes use two entries. The first operation is a read and the second operation is a write. These two operations are transferred to the timer bank pool with an address dependency so that the write cannot be performed until the read data has returned. This approach ensures that any subsequent operations to the same address (from the same port) are executed after the write operation, because they are ordered on the row list after the write operation.

If an entire word is being written (but less than a full burst), then no read is necessary and only that word is updated.

### ECC Write Backs

If the controller ECC is enabled and a read operation results in a correctable ECC error, the controller corrects the location in memory, if write backs are enabled. The correction results in scheduling a new read-modify-write. A new read is performed at the location to ensure that a write operation modifying the location is not overwritten. The actual ECC correction operation is performed as a read-modify-write operation.

### User Notification of ECC Errors

The following methods notify you of an ECC error:

For the MPU subsystem, an interrupt signal provides notification and the ECC error information is stored in the status registers.

 For more information, refer to the *Cortex-A9 Microprocessor Unit Subsystem* chapter in volume 3 of the *Arria V Device Handbook* or the *Cortex-A9 Microprocessor Unit SubSystem* chapter in volume 3 of the *Cyclone V Device Handbook*.

### Interleaving Options

The controller supports the following address-interleaving options:

- Noninterleaved
- Bank interleave without chip select interleave

- Bank interleave with chip select interleave

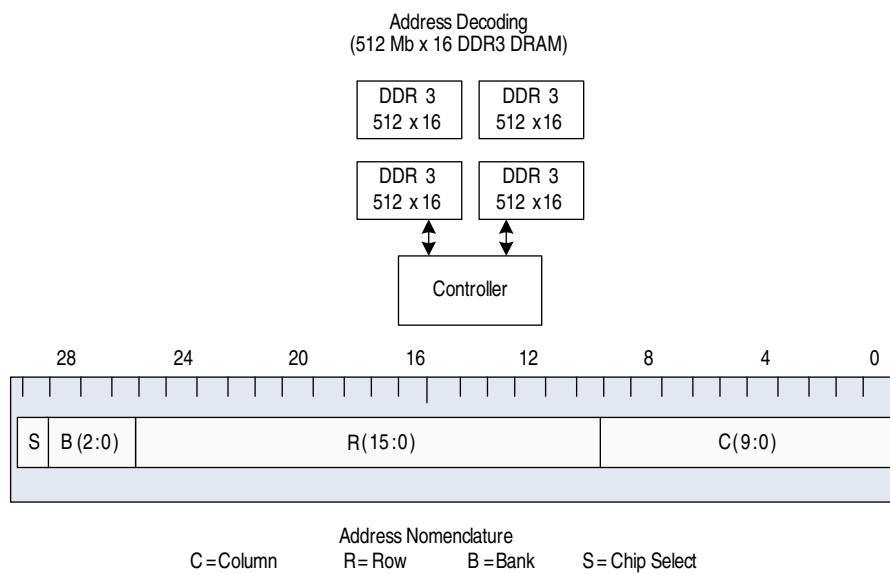
All of the interleaving examples use 512 megabits (Mb) x 16 DDR3 chips and are documented as byte addresses. For RAMs with smaller address fields, the order of the fields stays the same but the widths may change.

### Noninterleaved

RAM mapping is noninterleaved.

Figure 4–4 shows noninterleaved address decoding.

**Figure 4–4. Noninterleaved Address Decoding**

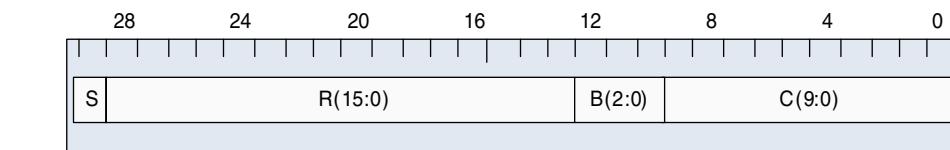


### Bank Interleave Without Chip Select Interleave

Bank interleave without chip select interleave swaps row and bank from the noninterleaved address mapping. This interleaving allows smaller data structures to spread across all banks in a chip.

Figure 4–5 shows bank interleave without chip select interleave address decoding.

**Figure 4–5. Bank Interleave Without Chip Select Interleave Address Decoding**

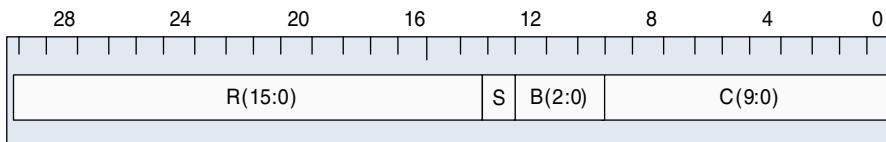


### Bank Interleave with Chip Select Interleave

Bank interleave with chip select interleave moves the row address to the top, followed by chip select, then bank, and finally column address. This interleaving allows smaller data structures to spread across multiple banks and chips (giving access to 16 total banks for multithreaded access to blocks of memory). Memory timing is degraded when switching between chips.

Figure 4–6 shows bank interleave with chip select interleave address decoding.

**Figure 4–6. Bank Interleave With Chip Select Interleave Address Decoding**



## AXI-Exclusive Support

The single-port controller supports AXI-exclusive operations. The controller implements a table shared across all masters, which can store up to 16 pending writes. Table entries are allocated on an exclusive read and table entries are deallocated on a successful write to the same address by any master.

Any exclusive write operation that is not present in the table returns an exclusive fail as acknowledgement to the operation. If the table is full when the exclusive read is performed, the table replaces a random entry.



When using AXI-exclusive operations, accessing the same location from Avalon-MM interfaces can result in unpredictable results.

## Memory Protection

The single-port controller has address protection to allow the software to configure basic protection of memory from all masters in the system. If the system has been designed exclusively with AXI masters, TrustZone® is supported. Ports that use Avalon-MM can be configured for port level protection.



For information about TrustZone®, refer to the ARM website ([www.arm.com](http://www.arm.com)).

Memory protection is based on physical addresses in memory. You can set rules to allow or disallow accesses to a range of memory, or to enable only secure accesses to a range of memory (or a combination of the two).

Secure and non-secure regions are specified by rules containing a starting address and ending address with 1 MB boundaries for both addresses. You can override the port defaults and allow or disallow all transactions.

The memory protection table, which is an internal table addressed through the CSR interface, contains rules to permit or deny memory access. You can configure up to a maximum of twenty rules to control memory access. **Table 4–4** lists the fields that you can specify for each rule.

**Table 4–4. Fields for Rules in Memory Protection Table (Part 1 of 2)**

Field	Width	Description
Valid	1	Set to 1 to activate the rule. Set to 0 to deactivate the rule.
Port Mask <sup>(1)</sup>	10	Specifies the set of ports to which the rule applies, with one bit representing each port, as follows: bits 0 to 5 correspond to FPGA fabric ports 0 to 5, bit 6 corresponds to AXI L3 switch read, bit 7 is the CPU read, bit 8 is L3 switch write, and bit 9 is the CPU write.

**Table 4–4. Fields for Rules in Memory Protection Table (Part 2 of 2)**

Field	Width	Description
TID_low <i>(1)</i>	12	Low transfer ID of the rules to which this rule applies. Incoming transactions match if they are greater than or equal to this value. Ports with smaller TIDs have the TID shifted to the lower bits and zero padded at the top.
TID_high <i>(1)</i>	12	High transfer ID of the rules to which this rule applies. Incoming transactions match if they are less than or equal to this value.
Address_low	12	Points to a 1MB block and is the lower address. Incoming addresses match if they are greater than or equal to this value.
Address_high	12	Upper limit of address. Incoming addresses match if they are less than or equal to this value.
Protection	2	A value of 00 indicates that the protection bit is not set; a value of 01 sets the protection bit. Systems that do not set AXI protection to a known value should program this for either protection value.
Fail/allow	1	Set this value to 1 to force the operation to fail or succeed.

**Note to Table 4–4:**

- (1) Although TID and Port Mask could be redundant, including both in the table allows possible compression of rules. If masters connected to a port do not have contiguous TIDs, a port-based rule might be more efficient than a TID-based rule, in terms of the number of rules needed.

A port has a default access status of either allow or fail, and rules with the opposite allow/fail value can override the default. The system evaluates each transaction against every rule in the memory protection table. A transaction received on a port which by default allows access, would fail only if a rule with the fail bit matches the transaction. Conversely, a port which by default prevents access, would allow access only if a rule allows that transaction to pass.

Exclusive transactions are security checked on the read operation only. A write operation can occur only if a valid read is marked in the internal exclusive table. Consequently, a master performing an exclusive read followed by a write, can write to memory only if the exclusive read was successful.

### Example of Configuration for TrustZone

For a TrustZone configuration, memory is divided into a range of memory accessible by secure masters and a range of memory accessible by nonsecure masters. The two memory address ranges may have a range of memory that overlaps.

This example implements the following memory configuration:

- 2 GB total RAM size
- 0—512 MB dedicated secure area
- 513—576 MB shared area
- 577—2048 MB dedicated nonsecure area

In this example, each port is configured by default to disallow all accesses. [Table 4–5](#) shows the two rules programmed into the memory protection table.

**Table 4–5. Rules in Memory Protection Table for Example Configuration**

Rule #	Port Mask	TID Low	TID High	Address Low	Address High	Prot	Fail/Allow
1	0'b111111111111	0	4095	0	576	b01	allow
2	0'b111111111111	0	4095	512	2047	b00	allow

The port mask value, TID Low, and TID High, apply to all ports and all transfers within those ports. Each access request is evaluated against the memory protection table, and will fail unless a rule matches allowing a transaction to complete successfully.

[Table 4–6](#) shows the result for a sample set of transactions.

**Table 4–6. Result for a Sample Set of Transactions**

Operation	Source	Address	Prot	Result	Comments
Read	CPU	4096	1	Allow	Matches rule 1.
Write	CPU	536, 870, 912 (512 MB)	1	Allow	Matches rule 1.
Write	L3 attached masters	605, 028, 350 (577 MB)	1	Fail	Does not match rule 1 (out of range of the address field), does not match rule 2 (protection bit incorrect).
Read	L3 attached masters	4096	0	Fail	Does not match rule 1 (prot value wrong), does not match rule 2 (not in address range).
Write	CPU	536, 870, 912 (512 MB)	0	Allow	Matches rule 2.
Write	L3 attached masters	605, 028, 350 (577 MB)	0	Allow	Matches rule 2.

If you did not want any overlap between the memory blocks, you could specify the address ranges in the two rules of [Table 4–5](#) to be mutually exclusive. Depending on your desired TrustZone configuration, you can add rules to the memory protection table to create multiple blocks of protected or unprotected space.

## SDRAM Power Management

The SDRAM controller subsystem supports the following power saving features in the SDRAM:

- Partial array self-refresh (PASR)
- Power down
- Deep power down for LPDDR2

Power-saving mode initiates either due to a user command or from inactivity.

Power-down mode is initiated by writing to the appropriate control register. It forces the SDRAM burst-scheduling bank-management logic to close all banks and issue the power down command. You can program the controller to enable power-down when the SDRAM burst-scheduling queue is empty for a specified number of clock cycles. The SDRAM automatically reactivates when an active SDRAM command is received.

Other power-down modes are performed only under user control.

## DDR PHY

The DDR PHY connects the memory controller and external memory devices in the speed critical command path.

The DDR PHY implements the following functions:

- Calibration—the DDR PHY supports the JEDEC-specified steps to synchronize the memory timing between the controller and the SDRAM chips. The calibration algorithm is implemented in software.
- Memory device initialization—the DDR PHY performs the mode register write operations to initialize the devices. The DDR PHY handles re-initialization after a deep power down.
- Single-data-rate to double-data-rate conversion.

## Clocks

All clocks are assumed to be asynchronous with respect to the `ddr_dqs_clk` memory clock. All transactions are synchronized to memory clock domain.

Table 4-7 shows the SDRAM controller subsystem clock domains.

**Table 4-7. SDRAM Controller Subsystem Clock Domains**

Clock Name	Description
<code>ddr_dq_clk</code>	Clock for PHY
<code>ddr_dqs_clk</code>	Clock for MPFE, single-port controller, CSR access, and PHY
<code>ddr_2x_dqs_clk</code>	Clock for PHY
<code>14_sp_clk</code>	Clock for CSR interface
<code>mpu_12_ram_clk</code>	Clock for MPU interface
<code>13_main_clk</code>	Clock for L3 interface
<code>f2h_sdram_clk[5:0]</code>	Six separate clocks used for the FPGA-to-HPS SDRAM ports to the FPGA fabric

In terms of clock relationships, the FPGA fabric connects the appropriate clocks to write data, read data, and command ports for the constructed ports.

-  For more information, refer to the *Clock Manager* chapter in volume 3 of the *Arria V Device Handbook* or the *Clock Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

## Resets

The SDRAM controller subsystem supports a full reset (cold reset) and a warm reset, which may or may not preserve the contents of memory. In order to preserve the memory contents, the reset manager can request that the single-port controller place the SDRAM in self-refresh mode prior to issuing the warm reset. If memory contents are preserved, the PHY and the memory timing logic is not reset, but the rest of the controller is reset.



For more information, refer to the *Reset Manager* chapter in volume 3 of the *Arria V Device Handbook* or the *Reset Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

## Initialization

The SDRAM controller subsystem has CSRs which control the operation of the controller including DRAM type, DRAM timing parameters and relative port priorities. It also has a small set of bits which depend on the FPGA fabric to configure ports between the memory controller and the FPGA fabric; these bits are set for you when you configure your implementation using the HPS GUI in Qsys.

The CSRs are configured using a dedicated slave interface, which provides accesses to registers. This region controls all SDRAM operation, MPFE scheduler configuration, and PHY calibration.

The FPGA fabric interface configuration is programmed into the FPGA fabric and the values of these register bits can be read by software. The ports can be configured without software developers needing to know how the FPGA-to-HPS SDRAM interface has been configured.

## Protocol Details

### Avalon-MM Bidirectional Port

The Avalon-MM bidirectional ports are standard Avalon-MM ports used to dispatch read and write operations. Each configured Avalon-MM bidirectional port consists of the signals listed in [Table 4-8](#).

**Table 4-8. Avalon-MM Bidirectional Port Signals (Part 1 of 2)**

Name	Bits	Direction	Function
clk	1	In	Clock for the Avalon-MM interface
read	1	In	Indicates read transaction
write	1	In	Indicates write transaction
address	32	In	Address of the transaction
readdata	32, 64, 128, or 256	Out	Read data return
readdatavalid	1	Out	Valid cycle flag for read data return
writedata	32, 64, 128, or 256	In	Write data for a transaction

**Table 4–8. Avalon-MM Bidirectional Port Signals (Part 2 of 2)**

Name	Bits	Direction	Function
byteenable	4 (32-bit data), 8(64-bit data), 16(128-bit data), 32(256-bit data)	In	Byte enables for each write byte
waitrequest	1	Out	Indicates need for additional cycles to complete a transaction
burstcount	11	In	Transaction burst length

The read and write interfaces are configured to the same size. The byte-enable size scales with the data bus size.

- For information about the Avalon-MM protocol, refer to the *Avalon Interface Specifications*.

### Avalon-MM Write Port

The Avalon-MM write ports are standard Avalon-MM ports used only to dispatch write operations. Each configured Avalon-MM write port consists of the signals listed in **Table 4–9**.

**Table 4–9. Avalon-MM Write Port Signals**

Name	Bits	Direction	Function
reset	1	In	Reset
clk	1	In	Clock
write	1	In	Indicates write transaction
address	32	In	Address of the transaction
writedata	32, 64, 128, or 256	In	Write data for a transaction
byteenable	4 (32-bit data), 8(64-bit data), 16(128-bit data), 32(256-bit data)	In	Byte enables for each write byte
waitrequest	1	Out	Indicates need for additional cycles to complete a transaction
burstcount	11	In	Transaction burst length

- For information about the Avalon-MM protocol, refer to the *Avalon Interface Specifications*.

### Avalon-MM Read Port

The Avalon-MM read ports are standard Avalon-MM ports used only to dispatch read operations. Each configured Avalon-MM read port consists of the signals listed in [Table 4-10](#).

**Table 4-10. Avalon-MM Read Port Signals**

Name	Bits	Direction	Function
reset	1	In	Reset
clk	1	In	Clock
read	1	In	Indicates read transaction
address	32	In	Address of the transaction
readdata	32, 64, 128, or 256	Out	Read data return
readdatavalid	1	Out	Flags valid cycles for read data return
waitrequest	1	Out	Indicates the need for additional cycles to complete a transaction. Needed for read operations when delay is needed to accept the read command.
burstcount	11	In	Transaction burst length

 For information about the Avalon-MM protocol, refer to the *Avalon Interface Specifications*.

### AXI Port

The AXI port uses an AXI-3 interface.

 For information about the AXI-3 interface, refer to the *AMBA Open Specifications* on the ARM website ([www.arm.com](http://www.arm.com)).

 For information about the AXI interface ports in the high-performance II controller (HPC II), refer to the *Functional Description—HPC II Controller* chapter, in the *External Memory Interface Handbook*.

Each configured AXI port consists of the signals listed in [Table 4-11](#). Each AXI interface signal is independent of the other interfaces for all signals, including clock and reset.

**Table 4-11. AXI Port Signals (Part 1 of 2)**

Name	Bits	Direction	Function
ARESETn	1	In	Reset
ACLK	1	In	Clock
<b>Write Address Channel Signals</b>			
AWID	4	In	Write identification tag
AWADDR	32	In	Write address
AWLEN	4	In	Write burst length
AWSIZE	3	In	Width of the transfer size
AWBURST	2	In	Burst type

**Table 4–11. AXI Port Signals (Part 2 of 2)**

Name	Bits	Direction	Function
AWREADY	1	Out	Indicates ready for a write command
AWVALID	1	In	Indicates valid write command.
<b>Write Data Channel Signals</b>			
WID	4	In	Write data transfer ID
WDATA	32, 64, 128 or 256	In	Write data
WSTRB	4, 8, 16, 32	In	Byte-based write data strobe. Each bit width corresponds to 8 bit wide transfer for 32-bit wide to 256-bit wide transfer.
WLAST	1	In	Last transfer in a burst
WVALID	1	In	Indicates write data+strobes are valid
WREADY	1	Out	Indicates ready for write data and strobes
<b>Write Response Channel Signals</b>			
BID	4	Out	Write response transfer ID
BRESP	2	Out	Write response status
BVALID	1	Out	Write response valid signal
BREADY	1	In	Write response ready signal
<b>Read Address Channel Signals</b>			
ARID	4	In	Read identification tag
ARADDR	32	In	Read address
ARLEN	4	In	Read burst length
ARSIZE	3	In	Width of the transfer size
ARBURST	2	In	Burst type
ARREADY	1	Out	Indicates ready for a read command
ARVALID	1	In	Indicates valid read command
<b>Read Data Channel Signals</b>			
RID	4	Out	Read data transfer ID
RDATA	32, 64, 128 or 256	Out	Read data
RRESP	2	Out	Read response status
RLAST	1	Out	Last transfer in a burst
RVALID	1	Out	Indicates read data is valid
RREADY	1	In	Read data channel ready signal

## SDRAM Controller Subsystem Programming Model

### Initialization

SDRAM controller configuration occurs through software programming of the configuration registers using the CSR interface. Initialization of the SDRAM controller has two separate regions with different controls.

## Timing Parameters

The SDRAM controller supports a complete set of timing parameters, configurable at run time.

## SDRAM Controller Address Map and Register Definitions

- To access address map and register definitions, open the file [hps.html](#).

To view the module description and base address, scroll to and click the link for the following module instance:

- sdr**

To then view the register and field descriptions, scroll to and click the register names. The register addresses are offsets relative to the base address of each module instance.

- The base addresses of all modules are also listed in the *Introduction to the Hard Processor System* chapter in volume 3 of the *Arria V Device Handbook* and the *Introduction to the Hard Processor System* chapter in volume 3 of the *Cyclone V Device Handbook*.

## Using EMI-Related HPS Features in SoC Devices

This section provides information on using HPS features in your external memory interface.

### Architecture

The configuration and initialization of the memory interface by the ARM processor is a significant difference compared to the FPGA memory interfaces, and results in several key differences in the way the HPS memory interface is defined and configured.

Boot-up configuration of the HPS memory interface is handled by the initial software boot code, not by the FPGA programmer, as is the case for the FPGA memory interfaces. The Quartus II software is involved in defining the configuration of I/O ports which is used by the boot-up code, as well as timing analysis of the memory interface. Therefore, the memory interface must be configured with the correct PHY-level timing information. Although configuration of the memory interface in Qsys is still necessary, it is limited to PHY- and board-level settings.

### Configuration

To configure the external memory interface components of the HPS, you open the HPS interface by selecting the Hard Processor System component in Qsys. Within the HPS interface, select the EMIF tab to open the EMIF parameter editor.

The EMIF parameter editor contains four additional tabs: PHY Settings, Memory Parameters, Memory Timing, and Board Settings. The parameters available on these tabs are similar to those available in the parameter editors for non-SoC device families.

There are significant differences between the EMIF parameter editor for the Hard Processor System and the parameter editors for non-SoC devices, as follows:

- Because the HPS memory controller is not configurable through the Quartus II software, the Controller and Diagnostic tabs, which exist for non-SoC devices, are not present in the EMIF parameter editor for the hard processor system.
- Unlike the protocol-specific parameter editors for non-SoC devices, the EMIF parameter editor for the Hard Processor System supports multiple protocols, therefore there is an SDRAM Protocol parameter, where you can specify your external memory interface protocol.

By default, the EMIF parameter editor assumes the DDR3 protocol, and other parameters are automatically populated with DDR3-appropriate values. If you select a protocol other than DDR3, you will have to change other associated parameter values appropriately.

- Unlike the memory interface clocks in the FPGA, the memory interface clocks for the HPS are initialized by the boot-up code using values provided by the configuration process. You may accept the values provided by UniPHY, or you may use your own PLL settings. If you choose to specify your own PLL settings, you must indicate that the clock frequency that UniPHY should use is the requested clock frequency, and not the achieved clock frequency calculated by UniPHY.



The HPS does not support EMIF synthesis generation, compilation, or timing analysis.

The HPS hard memory controller cannot be bonded with another hard memory controller on the FPGA portion of the device.

## Simulation

The HPS component provides a complete simulation model of the HPS memory interface controller and PHY, providing cycle-level accuracy, comparable to the simulation models for the FPGA memory interface.

The simulation model supports only the skip-cal simulation mode; quick-cal and full-cal are not supported. An example design is not provided, however you can create a test design by adding the traffic generator component to your design using Qsys. Also, the HPS simulation model does not use external memory pins to connect to the DDR memory model; instead, the memory model is incorporated directly into the HPS SDRAM interface simulation modules.

## Document Revision History

Table 4-12 lists the revision history for this document.

**Table 4-12. Document Revision History**

Date	Version	Changes
November 2012	1.0	<ul style="list-style-type: none"><li>■ Initial release.</li><li>■ Moved <a href="#">Using EMI-Related HPS Features in SoC Devices</a> from Hard Memory Interface chapter to this chapter.</li></ul>



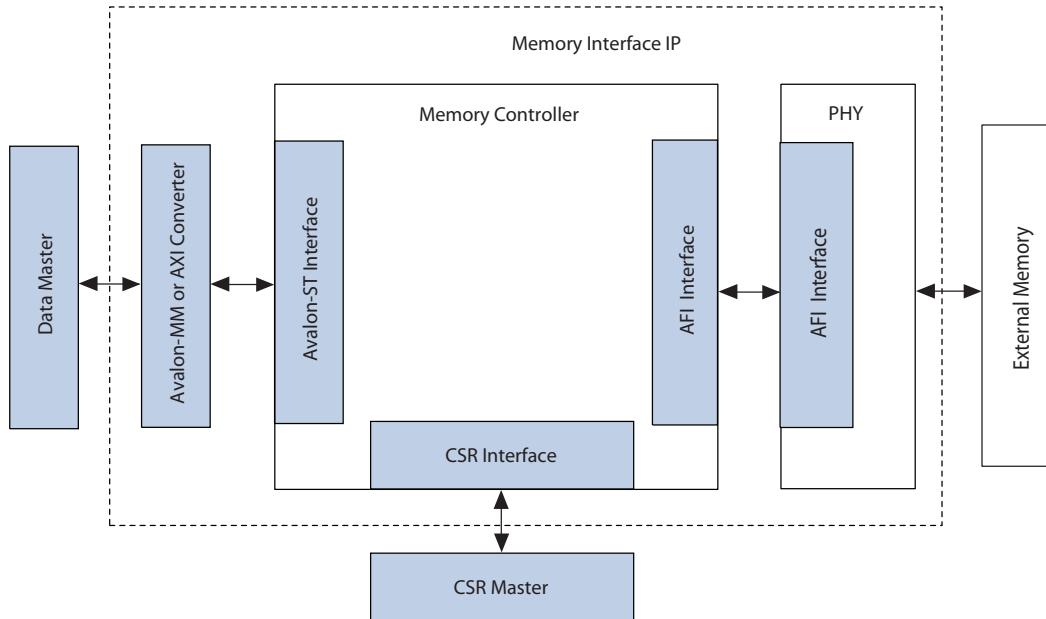
This chapter describes the High Performance Controller II (HPC II) with advanced features for designs generated in the Quartus II software version 11.0 and later. Designs created in earlier versions and regenerated in version 11.0 and later do not inherit the new advanced features; for information on HPC II without the version 11.0 and later advanced features, refer to the External Memory Interface Handbook for Quartus II version 10.1, available in the *External Memory Interfaces* section of the Altera Literature website.

The High Performance Controller II works with the UniPHY-based DDR2, DDR3, and LPDDR2 interfaces, and with the ALTMEMPHY-based DDR, DDR2, and DDR3 interfaces. The controller provides high memory bandwidth, high clock rate performance, and run-time programmability. The controller can reorder data to reduce row conflicts and bus turn-around time by grouping reads and writes together, allowing for efficient traffic patterns and reduced latency.

## Memory Controller Architecture

Figure 5–1 shows a high-level block diagram of the overall HPC II memory interface architecture.

**Figure 5–1. High-Level Diagram of Memory Interface Architecture**



©2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

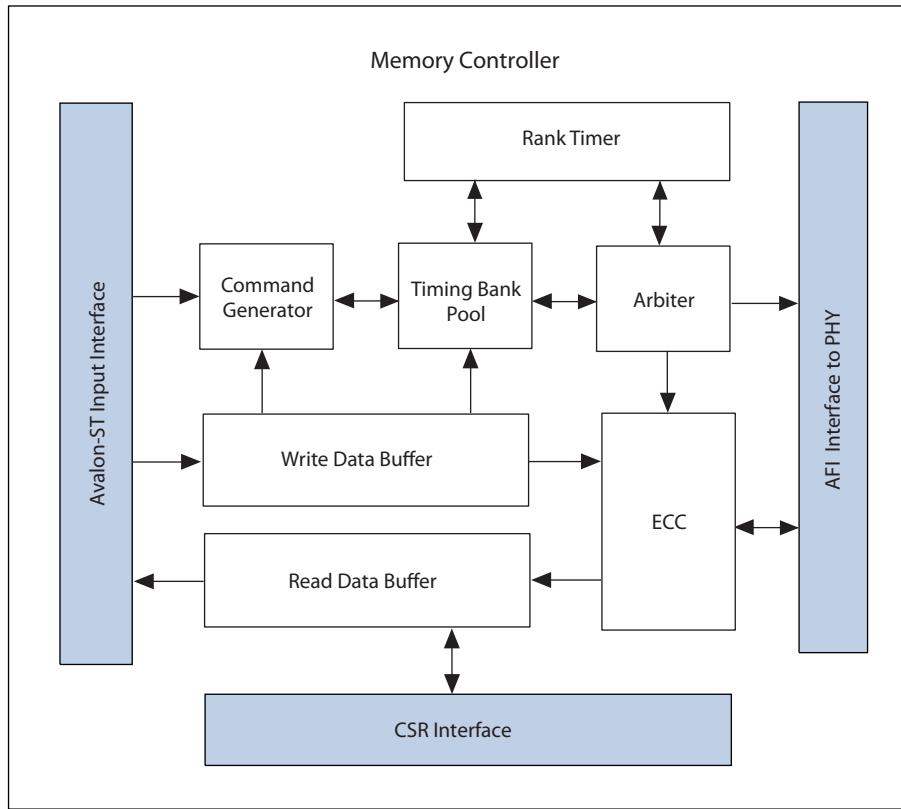


The memory interface consists of the memory controller logic block, the physical logic layer (PHY), and their associated interfaces.

The memory controller logic block uses an Avalon Streaming (Avalon-ST) interface as its native interface, and communicates with the PHY layer by the Altera PHY Interface (AFI).

**Figure 5–2** shows a block diagram of the memory controller architecture.

**Figure 5–2. Memory Controller Architecture Block Diagram**



The following sections describe the blocks in **Figure 5–2**.

## Avalon-ST Input Interface

The Avalon-ST interface serves as the entry point to the memory controller, and provides communication with the requesting data masters.

For information about the Avalon interface, refer to *Avalon Interface Specifications*.

## AXI to Avalon-ST Converter

The HPC II memory controller includes an AXI to Avalon-ST converter for communication with the AXI protocol. The AXI to Avalon-ST converter provides write address, write data, write response, read address, and read data channels on the AXI interface side, and command, write data, and read data channels on the Avalon-ST interface side.

## Handshaking

The AXI protocol employs a handshaking process similar to the Avalon-ST protocol, based on ready and valid signals.

## Command Channel Implementation

The AXI interface includes separate read and write channels, while the Avalon-ST interface has only one command channel. Arbitration of the read and write channels is based on these policies:

- Round robin
- Write priority—write channel has priority over read channel
- Read priority—read channel has priority over write channel

You can choose an arbitration policy by setting the `COMMAND_ARB_TYPE` parameter to one of `ROUND_ROBIN`, `WRITE_PRIORITY`, or `READ_PRIORITY`.

## Data Ordering

The AXI specification requires that write data IDs must arrive in the same order as write address IDs are received. Similarly, read data must be returned in the same order as its associated read address is received.

Consequently, the AXI to Avalon-ST converter does not support interleaving of write data; all data must arrive in the same order as its associated write address IDs. On the read side, the controller returns read data based on the read addresses received.

## Burst Types

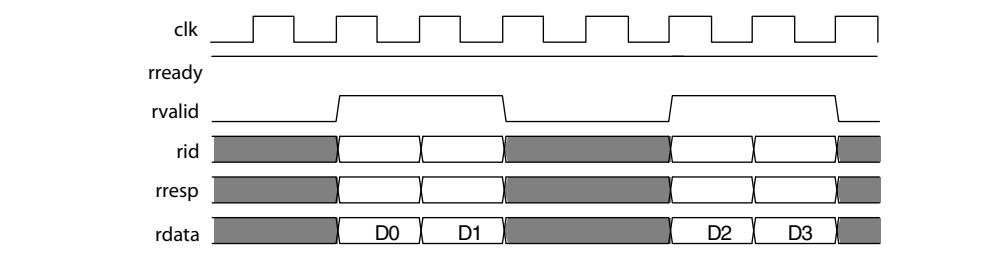
The AXI to Avalon-ST converter supports the following burst types:

- Incrementing burst—the address for each transfer is an increment of the previous transfer address; the increment value depends on the size of the transfer.
- Wrapping burst—similar to the incrementing burst, but wraps to the lower address when the burst boundary is reached. The starting address must be aligned to the size of the transfer. Burst length must be 2, 4, 8, or 16. The burst wrap boundary = burst size \* burst length.

## Backpressure Support

The write response and read data channels do not support data transfer with backpressure; consequently, you must assert the ready signal for the write response and read data channels to 1 as shown in Figure 5–3, to ensure acceptance of data at any time.

**Figure 5–3. Data Transfer Without Backpressure**



For information about data transfer with and without backpressure, refer to the *Avalon Interface Specifications*.

## Command Generator

The command generator accepts commands from the front-end Avalon-ST interface and from local ECC internal logic, and provides those commands to the timing bank pool.

## Timing Bank Pool

The timing bank pool is a parallel queue that works with the arbiter to enable data reordering. The timing bank pool tracks incoming requests, ensures that all timing requirements are met and, upon receiving write-data-ready notification from the write data buffer, passes the requests to the arbiter in an ordered and efficient manner.

## Arbiter

The arbiter determines the order in which requests are passed to the memory device. When the arbiter receives a single request, that request is passed immediately; however, when multiple requests are received, the arbiter uses arbitration rules to determine the order in which to pass requests to the memory device.

### Arbitration Rules

The arbiter uses the following arbitration rules:

- If only one master is issuing a request, grant that request immediately.
- If there are outstanding requests from two or more masters, the arbiter applies the following tests, in order:
  - a. Is there a read request? If so, the arbiter grants the read request ahead of any write requests.
  - b. If neither of the above conditions apply, the arbiter grants the oldest request first.

## Rank Timer

The rank timer maintains rank-specific timing information, and performs the following functions:

- Ensures that only four activates occur within a specified timing window.
- Manages the read-to-write and write-to-read bus turnaround time.
- Manages the time-to-activate delay between different banks.

## Read Data Buffer

The read data buffer receives data from the PHY and passes that data through the input interface to the master.

## Write Data Buffer

The write data buffer receives write data from the input interface and passes that data to the PHY, upon approval of the write request.

## ECC Block

The error-correcting code (ECC) block comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC block can remedy errors resulting from noise or other impairments during data transmission.

## AFI Interface

The AFI interface provides communication between the controller and the physical layer logic (PHY).

For more information about AFI signals, refer to [AFI 3.0 Specification](#).



Unaligned reads and writes on the AFI interface are not supported.

## CSR Interface

The CSR interface provides communication with your system's internal control status registers.

# Controller Features Descriptions

The following sections describe main features of the memory controller.

## Data Reordering

The controller implements data reordering to maximize efficiency for read and write commands. The controller can reorder read and write commands as necessary to mitigate bus turn-around time and reduce conflict between rows.

Inter-bank data reordering reorders commands going to different bank addresses. Commands going to the same bank address are not reordered. This reordering method implements simple hazard detection on the bank address level.

The controller implements logic to limit the length of time that a command can go unserved. This logic is known as starvation control. In starvation control, a counter is incremented for every command served. You can set a starvation limit, to ensure that a waiting command is served immediately, when the starvation counter reaches the specified limit.

## Pre-emptive Bank Management

Data reordering allows the controller to issue bank-management commands pre-emptively, based on the patterns of incoming commands; consequently, the desired page in memory can be already open when a command reaches the AFI interface.

## Quasi-1T and Quasi-2T

One controller clock cycle equals two memory clock cycles in a half-rate interface, and to four memory clock cycles in a quarter-rate interface. To fully utilize the command bandwidth, the controller can operate in Quasi-1T half-rate and Quasi-2T quarter-rate modes.

In Quasi-1T and Quasi-2T modes, the controller issues two commands on every controller clock cycle. The controller is constrained to issue a row command on the first clock phase and a column command on the second clock phase, or vice versa. Row commands include activate and precharge commands; column commands include read and write commands.

The controller operates in Quasi-1T in half-rate mode, and in Quasi-2T in quarter-rate mode; this operation is transparent and has no user settings.

## User Autoprecharge Commands

The autoprecharge read and autoprecharge write commands allow you to indicate to the memory device that this read or write command is the last access to the currently open row. The memory device automatically closes or autoprecharges the page it is currently accessing so that the next access to the same bank is quicker.

This command is useful for applications that require fast random accesses.

Since the HPC II controller can reorder transactions for best efficiency, when you assert the local\_autopch\_req signal, the controller evaluates the current command and buffered commands to determine the best autoprecharge operation.

## Half-Rate Bridge

Half-rate bridge support is available for ALTMEMPHY-based cores targeting device families other than Arria II GX. Half-rate bridge support is not available for UniPHY-based cores.

When using the half-rate bridge feature, you must ensure that the `local_size` data for each write command remains constant until the next write command is issued. In other words, the `local_size` bus should not be allowed to change unless the `burst_begin` signal is high.

Before using the half-rate bridge feature, you should perform the following steps:

1. Open the Synopsis Design Constraints file (`.sdc`) for the half-rate bridge and set `slow_clk` to the path of the clock connected to the half-rate bridge's slave interface.
2. Before running the constraint, ensure that the clock `$slow_clk` is already created or declared using the `derive_pll_clocks`, `create_clock`, or `create_generated_clock` function; otherwise, the constraint may be ignored.

## Address and Command Decoding Logic

When the main state machine issues a command to the memory, it asserts a set of internal signals. The address and command decoding logic turns these signals into AFI-specific commands and address. This block generates the following signals:

- Clock enable and reset signals: `afi_cke`, `afi_rst_n`
- Command and address signals: `afi_cs_n`, `afi_ba`, `afi_addr`, `afi_ras_n`, `afi_cas_n`, `afi_we_n`

## Low-Power Logic

There are two types of low-power logic: the user-controlled self-refresh logic and automatic power-down with programmable time-out logic.

### User-Controlled Self-Refresh

When you assert the `local_self_rfsh_req` signal, the controller completes any currently executing reads and writes, and then interrupts the command queue and immediately places the memory into self-refresh mode. When the controller places the memory into self-refresh mode, it responds by asserting an acknowledge signal, `local_self_rfsh_ack`. You can leave the memory in self-refresh mode for as long as you choose.

To bring the memory out of self-refresh mode, you must deassert the request signal, and the controller responds by deasserting the acknowledge signal when the memory is no longer in self-refresh mode.



If a user-controlled refresh request and a system-generated refresh request occur at the same time, the user-controlled refresh takes priority; the system-generated refresh is processed only after the user-controlled refresh request is completed.

## Automatic Power-Down with Programmable Time-Out

The controller automatically places the memory in power-down mode to save power if the requested number of idle controller clock cycles is observed in the controller.

The **Auto Power Down Cycles** parameter on the **Controller Settings** tab allows you to specify a range between 1 to 65,535 idle controller clock cycles. The counter for the programmable time-out starts when there are no user read or write requests in the command queue. Once the controller places the memory in power-down mode, it responds by asserting the acknowledge signal, `local_power_down_ack`.

## ODT Generation Logic

The on-die termination (ODT) generation logic generates the necessary ODT signals for the controller, based on the scheme that Altera recommends.

### DDR2 SDRAM

**Table 5-1** lists which ODT signal is enabled for single-slot single chip-select per DIMM.



There is no ODT for reads.

**Table 5-1. ODT—DDR2 SDRAM Single Slot Single Chip-select Per DIMM (Write)**

Write On	ODT Enabled
<code>mem_cs [0]</code>	<code>mem_odt [0]</code>

**Table 5-2** lists which ODT signal is enabled for single-slot dual chip-select per DIMM.



There is no ODT for reads.

**Table 5-2. ODT—DDR2 SDRAM Single Slot Dual Chip-select Per DIMM (Write)**

Write On	ODT Enabled
<code>mem_cs [0]</code>	<code>mem_odt [0]</code>
<code>mem_cs [1]</code>	<code>mem_odt [1]</code>

**Table 5-3** lists which ODT signal is enabled for dual-slot single chip-select per DIMM.

**Table 5-3. ODT—DDR2 SDRAM Dual Slot Single Chip-select Per DIMM (Write)**

Write On	ODT Enabled
<code>mem_cs [0]</code>	<code>mem_odt [1]</code>
<code>mem_cs [1]</code>	<code>mem_odt [0]</code>

**Table 5-4** lists which ODT signal is enabled for dual-slot dual chip-select per DIMM.

**Table 5-4. ODT—DDR2 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

Write On	ODT Enabled
<code>mem_cs [0]</code>	<code>mem_odt [2]</code>
<code>mem_cs [1]</code>	<code>mem_odt [3]</code>

**Table 5–4. ODT—DDR2 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

Write On	ODT Enabled
mem_cs [2]	mem_odt [0]
mem_cs [3]	mem_odt [1]

### DDR3 SDRAM

**Table 5–5** lists which ODT signal is enabled for single-slot single chip-select per DIMM.



There is no ODT for reads.

**Table 5–5. ODT—DDR3 SDRAM Single Slot Single Chip-select Per DIMM (Write)**

Write On	ODT Enabled
mem_cs [0]	mem_odt [0]

**Table 5–6** lists which ODT signal is enabled for single-slot dual chip-select per DIMM.



There is no ODT for reads.

**Table 5–6. ODT—DDR3 SDRAM Single Slot Dual Chip-select Per DIMM (Write)**

Write On	ODT Enabled
mem_cs [0]	mem_odt [0]
mem_cs [1]	mem_odt [1]

**Table 5–7** lists which ODT signal is enabled for dual-slot single chip-select per DIMM.

**Table 5–7. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Write)**

Write On	ODT Enabled
mem_cs [0]	mem_odt [0] and mem_odt [1]
mem_cs [1]	mem_odt [0] and mem_odt [1]

**Table 5–8** lists which ODT signal is enabled for dual-slot single chip-select per DIMM.

**Table 5–8. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Read)**

Read On	ODT Enabled
mem_cs [0]	mem_odt [1]
mem_cs [1]	mem_odt [0]

**Table 5–9** lists which ODT signal is enabled for dual-slot dual chip-select per DIMM.

**Table 5–9. ODT—DDR3 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

Write On	ODT Enabled
mem_cs [0]	mem_odt [0] and mem_odt [2]
mem_cs [1]	mem_odt [1] and mem_odt [3]

**Table 5–9. ODT—DDR3 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)**

Write On	ODT Enabled
mem_cs [2]	mem_odt [0] and mem_odt [2]
mem_cs [3]	mem_odt [1] and mem_odt [3]

Table 5–10 lists which ODT signal is enabled for dual-slot dual chip-select per DIMM.

**Table 5–10. ODT—DDR3 SDRAM Dual Slot Dual Rank Per DIMM (Read)**

Read On	ODT Enabled
mem_cs [0]	mem_odt [2]
mem_cs [1]	mem_odt [3]
mem_cs [2]	mem_odt [0]
mem_cs [3]	mem_odt [1]

## Burst Merging

The burst merging feature is available to improve throughput for sequential addresses, when the Timing Bank Pool receives requests faster than they can be processed.

For designs created in a version of the high-performance controller II earlier than 11.0, burst merging is turned off by default. To turn on burst merging, perform these steps:

1. In a text editor, open the `<variation_name>.v` file for your design.
2. Search for the `ENABLE_BURST_MERGE` parameter in the `.v` file.
3. Change the `ENABLE_BURST_MERGE` value from 0 to 1.

## ECC

The ECC logic comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC logic is available in widths of 16, 24, 40, and 72 bits.



For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, ECC logic is limited to widths of 24 and 40.

- The ECC logic has the following features:
- Has Hamming code ECC logic that encodes every 64, 32, 16, or 8 bits of data into 72, 40, 24, or 16 bits of codeword.
- Has a latency increase of one clock for both writes and reads.
- For a 128-bit interface, ECC is generated as one 64-bit data path with 8-bits of ECC path, plus a second 64-bit data path with 8-bits of ECC path.
- Detects and corrects all single-bit errors.
- Detects all double-bit errors.
- Counts the number of single-bit and double-bit errors.

- Accepts partial writes, which trigger a read-modify-write cycle, for memory devices with DM pins.
- Can inject single-bit and double-bit errors to trigger ECC correction for testing and debugging purposes.
- Generates an interrupt signal when an error occurs.



When using ECC, you must initialize memory before writing to it.

When a single-bit or double-bit error occurs, the ECC logic triggers the `ecc_interrupt` signal to inform you that an ECC error has occurred. When a single-bit error occurs, the ECC logic reads the error address, and writes back the corrected data. When a double-bit error occurs, the ECC logic does not do any error correction but it asserts the `avl_rdata_error` signal to indicate that the data is incorrect. The `avl_rdata_error` signal follows the same timing as the `avl_rdata_valid` signal.

Enabling autocorrection allows the ECC logic to delay all controller pending activities until the correction completes. You can disable autocorrection and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, follow these steps:

1. When an interrupt occurs, read out the `SBE_ERROR` register. When a single-bit error occurs, the `SBE_ERROR` register is equal to one.
2. Read out the `ERR_ADDR` register.
3. Correct the single-bit error by issuing a dummy write to the memory address stored in the `ERR_ADDR` register. A dummy write is a write request with the `local_be` signal zero, that triggers a partial write which is effectively a read-modify-write event. The partial write corrects the data at that address and writes it back.

## Partial Writes

The ECC logic supports partial writes. Along with the address, data, and burst signals, the Avalon-MM interface also supports a signal vector, `local_be`, that is responsible for byte-enable. Every bit of this signal vector represents a byte on the data-bus. Thus, a logic low on any of these bits instructs the controller not to write to that particular byte, resulting in a partial write. The ECC code is calculated on all bytes of the data-bus. If any bytes are changed, the IP core must recalculate the ECC code and write the new code back to the memory.

For partial writes, the ECC logic performs the following steps:

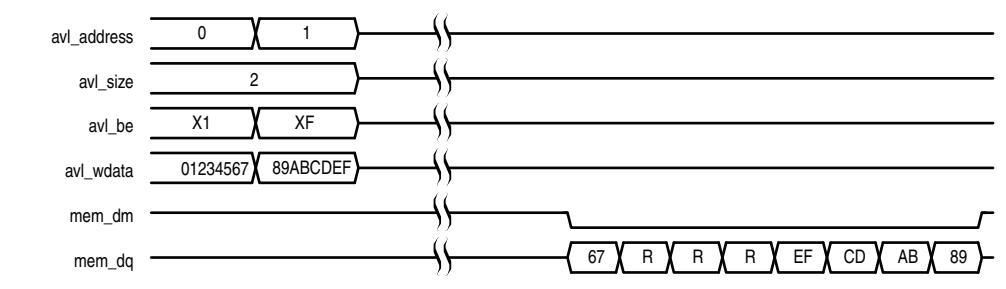
1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a write to write back the updated data and the new ECC code.

The following corner cases can occur:

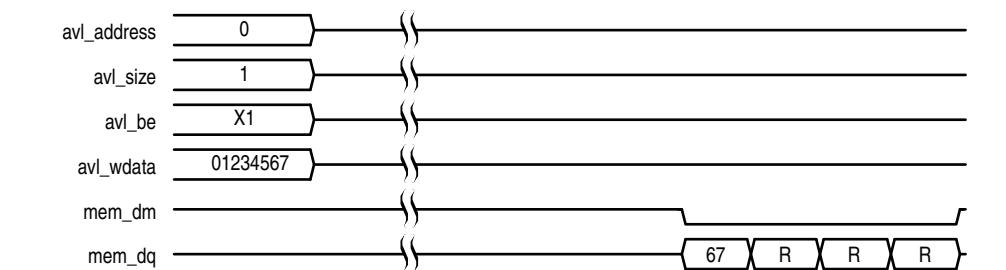
- A single-bit error during the read phase of the read-modify-write process. In this case, the IP core corrects the single-bit error first, increments the single-bit error counter and then performs a partial write to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the IP core increments the double-bit error counter and issues an interrupt. The IP core writes a new write word to the location of the error. The ECC status register keeps track of the error information.

[Figure 5–4](#) and [Figure 5–5](#) show partial write operations for the controller, for full and half rate configurations, respectively.

**Figure 5–4. Partial Write for the Controller—Full Rate**



**Figure 5–5. Partial Write for the Controller—Half Rate**

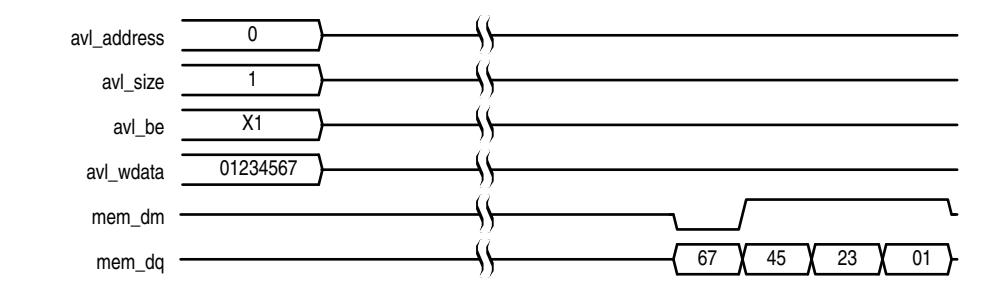


## Partial Bursts

DIMMs that do not have the DM pins do not support partial bursts. You must write a minimum (or multiples) of memory-burst-length-equivalent words to the memory at the same time.

[Figure 5–6](#) shows a partial burst operation for the controller.

**Figure 5–6. Partial Burst for Controller**



## External Interfaces

This section discusses the interfaces between the controller and other external memory interface components.

### Clock and Reset Interface

The clock and reset interface is part of the AFI interface.

The controller can have up to two clock domains, which are synchronous to each other. The controller operates with a single clock domain when there is no integrated half-rate bridge, and with two-clock domains when there is an integrated half-rate bridge. The clocks are provided by UniPHY.

The main controller clock is `afi_clk`, and the optional half-rate controller clock is `afi_half_clk`. The main and half-rate clocks must be synchronous and have a 2:1 frequency ratio. The optional quarter-rate controller clock is `afi_quarter_clk`, which must also be synchronous and have a 4:1 frequency ratio.

### Avalon-ST Data Slave Interface

The Avalon-ST data slave interface consists of the following Avalon-ST channels, which together form a single data slave:

- The command channel, which serves as command and address for both read and write operations.
- The write data channel, which carries write data.
- The read data channel, which carries read data.



For information about the Avalon interface, refer to *Avalon Interface Specifications*.

### AXI Data Slave Interface

The AXI data interface consists of the following channels, which communicate with the Avalon-ST interface through the AXI to Avalon-ST converter:

- The write address channel, which carries address information for write operations.
- The write data channel, which carries write data.
- The write response channel, which carries write response data.
- The read address channel, which carries address information for read operations.
- The read data channel, which carries read data.

### Enabling the AXI Interface

This section provides guidance for enabling the AXI interface.

1. To enable the AXI interface, first open in an editor the file appropriate for the required flow, as indicated below:
  - For synthesis flow: <working\_dir>/<variation\_name>/<variation\_name>\_c0.v
  - For simulation flow: <working\_dir>/<variation\_name>\_sim/<variation\_name>/<variation\_name>\_c0.v
  - Example design fileset for synthesis:  
 <working\_dir>/<variation\_name>\_example\_design/example\_project/<variation\_name>\_example/submodules/<variation\_name>\_example\_if0\_c0.v
  - Example design fileset for simulation:  
 <working\_dir>/<variation\_name>\_example\_design/simulation/verilog/submodules/<variation\_name>\_example\_sim\_e0\_if0\_c0.v
2. Locate and remove the **alt\_mem\_ddrx\_mm\_st\_converter** instantiation from the .v file opened in the preceding step.
3. Instantiate the **alt\_mem\_ddrx\_axi\_st\_converter** module into the open .v file. Refer to the following code fragment as a guide:

```

module ? # ( parameter
  // AXI parameters
  AXI_ID_WIDTH      = <replace parameter value>,
  AXI_ADDR_WIDTH    = <replace parameter value>,
  AXI_LEN_WIDTH     = <replace parameter value>,
  AXI_SIZE_WIDTH    = <replace parameter value>,
  AXI_BURST_WIDTH   = <replace parameter value>,
  AXI_LOCK_WIDTH    = <replace parameter value>,
  AXI_CACHE_WIDTH   = <replace parameter value>,
  AXI_PROT_WIDTH    = <replace parameter value>,
  AXI_DATA_WIDTH    = <replace parameter value>,
  AXI_RESP_WIDTH    = <replace parameter value>
)
(
// Existing ports
...
// AXI Interface ports
// Write address channel
input wire [AXI_ID_WIDTH - 1 : 0] awid,
input wire [AXI_ADDR_WIDTH - 1 : 0] awaddr,
input wire [AXI_LEN_WIDTH - 1 : 0] awlen,
input wire [AXI_SIZE_WIDTH - 1 : 0] awsize,
input wire [AXI_BURST_WIDTH - 1 : 0] awburst,
input wire [AXI_LOCK_WIDTH - 1 : 0] awlock,
input wire [AXI_CACHE_WIDTH - 1 : 0] awcache,
input wire [AXI_PROT_WIDTH - 1 : 0] awprot,
input wire                               awvalid,
output wire                             awready,
// Write data channel
input wire [AXI_ID_WIDTH - 1 : 0] wid,
input wire [AXI_DATA_WIDTH - 1 : 0] wdata,
input wire [AXI_DATA_WIDTH / 8 - 1 : 0] wstrb,

```

```

        input wire                               wlast,
        input wire                               wvalid,
        output wire                             wready,

        // Write response channel
        output wire [AXI_ID_WIDTH - 1 : 0] bid,
        output wire [AXI_RESP_WIDTH - 1 : 0] bresp,
        output wire                               bvalid,
        input  wire                               bready,

        // Read address channel
        input  wire [AXI_ID_WIDTH - 1 : 0] arid,
        input  wire [AXI_ADDR_WIDTH - 1 : 0] araddr,
        input  wire [AXI_LEN_WIDTH - 1 : 0] arlen,
        input  wire [AXI_SIZE_WIDTH - 1 : 0] arsize,
        input  wire [AXI_BURST_WIDTH - 1 : 0] arburst,
        input  wire [AXI_LOCK_WIDTH - 1 : 0] arlock,
        input  wire [AXI_CACHE_WIDTH - 1 : 0] arcache,
        input  wire [AXI_PROT_WIDTH - 1 : 0] arprot,
        input  wire                               arvalid,
        output wire                             arready,

        // Read data channel
        output wire [AXI_ID_WIDTH - 1 : 0] rid,
        output wire [AXI_DATA_WIDTH - 1 : 0] rdata,
        output wire [AXI_RESP_WIDTH - 1 : 0] rresp,
        output wire                               rlast,
        output wire                               rvalid,
        input  wire                               rready
    );

    // Existing wire, register declaration and instantiation
    ...
    // AXI interface instantiation
    alt_mem_ddrx_axi_st_converter #
    (
        .AXI_ID_WIDTH      (AXI_ID_WIDTH      ),
        .AXI_ADDR_WIDTH    (AXI_ADDR_WIDTH    ),
        .AXI_LEN_WIDTH     (AXI_LEN_WIDTH     ),
        .AXI_SIZE_WIDTH    (AXI_SIZE_WIDTH    ),
        .AXI_BURST_WIDTH   (AXI_BURST_WIDTH   ),
        .AXI_LOCK_WIDTH    (AXI_LOCK_WIDTH    ),
        .AXI_CACHE_WIDTH   (AXI_CACHE_WIDTH   ),
        .AXI_PROT_WIDTH    (AXI_PROT_WIDTH    ),
        .AXI_DATA_WIDTH    (AXI_DATA_WIDTH    ),
        .AXI_RESP_WIDTH    (AXI_RESP_WIDTH    ),
        .ST_ADDR_WIDTH     (ST_ADDR_WIDTH     ),
        .ST_SIZE_WIDTH     (ST_SIZE_WIDTH     ),
        .ST_ID_WIDTH       (ST_ID_WIDTH       ),
        .ST_DATA_WIDTH     (ST_DATA_WIDTH     ),
        .COMMAND_ARB_TYPE (COMMAND_ARB_TYPE)
    )

```

```

a0
(
    .ctl_clk           (afi_clk),
    .ctl_reset_n      (afi_reset_n),
    .awid             (awid),
    .awaddr            (awaddr),
    .awlen             (awlen),
    .awszie            (awszie),
    .awburst            (awburst),
    .awlock            (awlock),
    .awcache            (awcache),
    .awprot            (awprot),
    .awvalid            (awvalid),
    .awready            (awready),
    .wid               (wid),
    .wdata              (wdata),
    .wstrb             (wstrb),
    .wlast              (wlast),
    .wvalid             (wvalid),
    .wready             (wready),
    .bid               (bid),
    .bresp              (bresp),
    .bvalid             (bvalid),
    .bready             (bready),
    .arid               (arid),
    .araddr             (araddr),
    .arlen              (arlen),
    .arsize             (arsize),
    .arburst            (arburst),
    .arlock             (arlock),
    .arcache            (arcache),
    .arprot             (arprot),
    .arvalid            (arvalid),
    .arready            (arready),
    .rid                (rid),
    .rdata              (rdata),
    .rresp              (rresp),
    .rlast              (rlast),
    .rvalid             (rvalid),
    .rready              (rready),
    .itf_cmd_ready     (ng0_native_st_itf_cmd_ready),
    .itf_cmd_valid     (a0_native_st_itf_cmd_valid),
    .itf_cmd            (a0_native_st_itf_cmd),
    .itf_cmd_address   (a0_native_st_itf_cmd_address),
    .itf_cmd_burstlen  (a0_native_st_itf_cmd_burstlen),
    .itf_cmd_id         (a0_native_st_itf_cmd_id),
    .itf_cmd_priority  (a0_native_st_itf_cmd_priority),
    .itf_cmd_autoprecharge (a0_native_st_itf_cmd_autopercharge),
    .itf_cmd_multicast (a0_native_st_itf_cmd_multicast),
    .itf_wr_data_ready (ng0_native_st_itf_wr_data_ready),
    .itf_wr_data_valid (a0_native_st_itf_wr_data_valid),
    .itf_wr_data        (a0_native_st_itf_wr_data),
)

```

```

    .itf_wr_data_byte_en      (a0_native_st_itf_wr_data_byte_en),
    .itf_wr_data_begin        (a0_native_st_itf_wr_data_begin),
    .itf_wr_data_last         (a0_native_st_itf_wr_data_last),
    .itf_wr_data_id           (a0_native_st_itf_wr_data_id),
    .itf_rd_data_ready        (a0_native_st_itf_rd_data_ready),
    .itf_rd_data_valid        (ng0_native_st_itf_rd_data_valid),
    .itf_rd_data              (ng0_native_st_itf_rd_data),
    .itf_rd_data_error        (ng0_native_st_itf_rd_data_error),
    .itf_rd_data_begin         (ng0_native_st_itf_rd_data_begin),
    .itf_rd_data_last          (ng0_native_st_itf_rd_data_last),
    .itf_rd_data_id            (ng0_native_st_itf_rd_data_id)
);

```

4. Set the required parameters for the AXI interface. [Table 5–11](#) lists the available parameters.
5. Export the AXI interface to the top-level wrapper, making it accessible to the AXI master.
6. To add the AXI interface to the Quartus II project:
  - a. On the Assignments > Settings menu in the Quartus II software, open the File tab.
  - b. Add the `alt_mem_ddrx_axi_st_converter.v` file to the project.

**Table 5–11. AXI Interface Parameters (Part 1 of 2)**

Parameter Name	Description / Value
AXI_ID_WIDTH	Width of the AXI ID bus. Default value is 4.
AXI_ADDR_WIDTH	Width of the AXI address bus. Must be set according to the Avalon interface address and data bus width as shown below:  $\text{AXI\_ADDR\_WIDTH} = \text{LOCAL\_ADDR\_WIDTH} + \log_2(\text{LOCAL\_DATA\_WIDTH}/8)$ $\text{LOCAL\_ADDR\_WIDTH}$ is the memory controller Avalon interface address width. $\text{LOCAL\_DATA\_WIDTH}$ is the memory controller Avalon data interface width.
AXI_LEN_WIDTH	Width of the AXI length bus. Default value is 8.
AXI_SIZE_WIDTH	Should be set to $\text{LOCAL\_SIZE\_WIDTH} - 1$ , where $\text{LOCAL\_SIZE\_WIDTH}$ is the memory controller Avalon interface burst size width
AXI_BURST_WIDTH	Width of the AXI burst bus. Default value is 2.
AXI_LOCK_WIDTH	Width of the AXI lock bus. Default value is 2.
AXI_CACHE_WIDTH	Width of the AXI cache bus. Default value is 4.
AXI_PROT_WIDTH	Width of the AXI protection bus. Default value is 3.

**Table 5–11. AXI Interface Parameters (Part 2 of 2)**

Parameter Name	Description / Value
AXI_DATA_WIDTH	Width of the AXI data bus. Should be set to match the Avalon interface data bus width.  AXI_DATA_WIDTH = LOCAL_DATA_WIDTH, where LOCAL_DATA_WIDTH is the memory controller Avalon interface input data width.
AXI_RESP_WIDTH	Width of the AXI response bus. Default value is 2.
ST_ADDR_WIDTH	Width of the Avalon interface address. Must be set to match the Avalon interface address bus width.  ST_ADDR_WIDTH = LOCAL_ADDR_WIDTH, where LOCAL_ADDR_WIDTH is the memory controller Avalon interface address width.
ST_SIZE_WIDTH	Width of the Avalon interface burst size.  ST_SIZE_WIDTH = AXI_LEN_WIDTH + 1
ST_ID_WIDTH	Width of the Avalon interface ID. Default value is 4.  ST_ID_WIDTH = AXI_ID_WIDTH
ST_DATA_WIDTH	Width of the Avalon interface data.  ST_DATA_WIDTH = AXI_DATA_WIDTH.
COMMAND_ARB_TYPE	Specifies the AXI command arbitration type, as shown:  ROUND_ROBIN: arbitrates between read and write address channel in round robin fashion. Default option.  WRITE_PRIORITY: write address channel has priority if both channels send request simultaneously.  READ_PRIORITY: read address channel has priority if both channels send request simultaneously.
REGISTERED	Setting this parameter to 1 adds an extra register stage in the AXI interface and incurs one extra clock cycle of latency. Default value is 1.

Table 5–12 lists the AXI interface ports.

**Table 5–12. AXI Interface Ports (Part 1 of 3)**

Name	Direction	Description
awid	Input	AXI write address channel ID bus.
awaddr	Input	AXI write address channel address bus.
awlen	Input	AXI write address channel length bus.
awsize	Input	AXI write address channel size bus.
awburst	Input	AXI write address channel burst bus.  (Interface supports only INCR and WRAP burst types.)

**Table 5–12. AXI Interface Ports (Part 2 of 3)**

Name	Direction	Description
awlock	Input	AXI write address channel lock bus. (Interface does not support this feature.)
awcache	Input	AXI write address channel cache bus. (Interface does not support this feature.)
awprot	Input	AXI write address channel protection bus. (Interface does not support this feature.)
awvalid	Input	AXI write address channel valid signal.
awready	Output	AXI write address channel ready signal.
wid	Input	AXI write address channel ID bus.
wdata	Input	AXI write address channel data bus.
wstrb	Input	AXI write data channel strobe bus.
wlast	Input	AXI write data channel last burst signal.
wvalid	Input	AXI write data channel valid signal.
wready	Output	AXI write data channel ready signal.
bid	Output	AXI write response channel ID bus.
bresp	Output	AXI write response channel response bus.  Response encoding information: 'b00 - OKAY 'b01 - Reserved 'b10 - Reserved 'b11 - Reserved
bvalid	Output	AXI write response channel valid signal.
bready	Input	AXI write response channel ready signal.  Must be set to 1. Interface does not support back pressure for write response channel.
arid	Input	AXI read address channel ID bus.
araddr	Input	AXI read address channel address bus.
arlen	Input	AXI read address channel length bus.
arsize	Input	AXI read address channel size bus.
arburst	Input	AXI read address channel burst bus. (Interface supports only INCR and WRAP burst types.)
arlock	Input	AXI read address channel lock bus. (Interface does not support this feature.)
arcache	Input	AXI read address channel cache bus. (Interface does not support this feature.)

**Table 5–12. AXI Interface Ports (Part 3 of 3)**

Name	Direction	Description
arprot	Input	AXI read address channel protection bus. (Interface does not support this feature.)
arvalid	Input	AXI read address channel valid signal.
arready	Output	AXI read address channel ready signal.
rid	Output	AXI read data channel ID bus.
rdata	Output	AXI read data channel data bus.
rresp	Output	AXI read data channel response bus. Response encoding information: 'b00 - OKAY 'b01 - Reserved 'b10 - Data error 'b11 - Reserved
rlast	Output	AXI read data channel last burst signal.
rvalid	Output	AXI read data channel valid signal.
rready	Input	AXI read data channel ready signal. Must be set to 1. Interface does not support back pressure for write response channel.



For information about the AXI specification, refer to the ARM website, at [www.arm.com](http://www.arm.com).

## Controller-PHY Interface

The interface between the controller and the PHY is part of the AFI interface.

The controller assumes that the PHY performs all necessary calibration processes without any interaction with the controller.

For more information about AFI signals, refer to [AFI 3.0 Specification](#).

## Memory Side-Band Signals

This section describes supported side-band signals.

### Self-Refresh (Low Power) Interface

The optional low power self-refresh interface consists of a request signal and an acknowledgement signal, which you can use to instruct the controller to place the memory device into self-refresh mode. This interface is clocked by afi\_clk.

When you assert the request signal, the controller places the memory device into self-refresh mode and asserts the acknowledge signal. To bring the memory device out of self-refresh mode, you deassert the request signal; the controller then deasserts the acknowledge signal when the memory device is no longer in self-refresh mode.



For multi-rank designs using the HPC II memory controller, a self-refresh and a user-refresh cannot be made to the same memory chip simultaneously.

Also, the self-refresh ack signal indicates that at least one device has entered self-refresh, but does not necessarily mean that all devices have entered self-refresh.

### User-Controlled Refresh Interface

The optional user-controlled refresh interface consists of a request signal, a chip select signal, and an acknowledgement signal. This interface provides increased control over worst-case read latency and enables you to issue refresh bursts during idle periods. This interface is clocked by `afi_clk`.

When you assert a refresh request signal to instruct the controller to perform a refresh operation, that request takes priority over any outstanding read or write requests that might be in the command queue. In addition to the request signal, you must also choose the chip to be refreshed by asserting the refresh chip select signal along with the request signal. If you do not assert the chip select signal with the request signal, unexpected behavior may result.

The controller attempts to perform a refresh as long as the refresh request signal is asserted; if you require only one refresh, you should deassert the refresh request signal after the acknowledgement signal is received. If you maintain the request signal high after the acknowledgement is sent, it would indicate that further refresh is required. You should deassert the request signal after the required number of acknowledgement/refresh is received from the controller. You can issue up to a maximum of nine consecutive refresh commands.



For multi-rank designs using the HPC II memory controller, a self-refresh and a user-refresh cannot be made to the same memory chip simultaneously.

### Configuration and Status Register (CSR) Interface

The controller has a configuration and status register (CSR) interface that allows you to configure timing parameters, address widths, and the behavior of the controller. The CSR interface is a 32-bit Avalon-MM slave of fixed address width; if you do not need this feature, you can disable it to save area.

This interface is clocked by `csr_clk`, which is the same as `afi_clk`, and is always synchronous relative to the main data slave interface.

Table 5–13 lists the controller’s external interfaces.

**Table 5–13. Summary of Controller External Interfaces (Part 1 of 2)**

Interface Name	Display Name	Type	Description
<b>Clock and Reset Interface</b>			
Clock and Reset Interface	Clock and Reset Interface	AFI <sup>(1)</sup>	Clock and reset generated by UniPHY to the controller.
<b>Avalon-ST Data Slave Interface</b>			
Command Channel	Avalon-ST Data Slave Interface	Avalon-ST <sup>(2)</sup>	Address and command channel for read and write, single command single data (SCSD).

**Table 5–13. Summary of Controller External Interfaces (Part 2 of 2)**

Interface Name	Display Name	Type	Description
Write Data Channel	Avalon-ST Data Slave Interface	Avalon-ST <sup>(2)</sup>	Write Data Channel, single command multiple data (SCMD).
Read Data Channel	Avalon-ST Data Slave Interface	Avalon-ST <sup>(2)</sup>	Read data channel, SCMD with read data error response.
<b>Controller-PHY Interface</b>			
AFI 3.0	AFI Interface	AFI <sup>(1)</sup>	Interface between controller and PHY.
<b>Memory Side-Band Signals</b>			
Self Refresh (Low Power) Interface	Self Refresh (Low Power) Interface	Avalon Control & Status Interface <sup>(2)</sup>	SDRAM-specific signals to place memory into low-power mode.
User-Controller Refresh Interface	User-Controller Refresh Interface	Avalon Control & Status Interface <sup>(2)</sup>	SDRAM-specific signals to request memory refresh.
<b>Configuration and Status Register (CSR) Interface</b>			
CSR	Configuration and Status Register Interface	Avalon-MM <sup>(2)</sup>	Enables on-the-fly configuration of memory timing parameters, address widths, and controller behaviour.

**Notes:**

- (1) For information about AFI signals, refer to [AFI 3.0 Specification](#).  
 (2) For information about Avalon signals, refer to [Avalon Interface Specifications](#).

## Top-Level Signals Description

**Table 5–14** lists the clock and reset signals.



The suffix \_n denotes active low signals.

**Table 5–14. Clock and Reset Signals (Part 1 of 2)**

Name	Direction	Description
global_reset_n	Input	The asynchronous reset input to the controller. The IP core derives all other reset signals from resynchronized versions of this signal. This signal holds the PHY, including the PLL, in reset while low.
pll_ref_clk	Input	The reference clock input to PLL.
phy_clk	Output	The system clock that the PHY provides to the user. All user inputs to and outputs from the controller must be synchronous to this clock.
reset_phy_clk_n	Output	The reset signal that the PHY provides to the user. The IP core asserts reset_phy_clk_n asynchronously and deasserts synchronously to phy_clk clock domain.
aux_full_rate_clk	Output	An alternative clock that the PHY provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate designs, this clock is twice the frequency of the phy_clk and you can use it whenever you require a 2x clock. In full-rate designs, the same PLL output as the phy_clk signal drives this clock.

**Table 5–14. Clock and Reset Signals (Part 2 of 2)**

Name	Direction	Description
aux_half_rate_clk	Output	An alternative clock that the PHY provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate designs, this clock is half the frequency of the phy_clk and you can use it, for example to clock the user side of a half-rate bridge. In half-rate designs, or if the <b>Enable Half Rate Bridge</b> option is turned on. The same PLL output that drives the phy_clk signal drives this clock.
dll_reference_clk	Output	Reference clock to feed to an externally instantiated DLL.
reset_request_n	Output	Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using Altera advises you detect a reset request on a falling edge rather than by level detection.
soft_reset_n	Input	Edge detect reset input for SOPC Builder or for control by other system reset logic. Assert to cause a complete reset to the PHY, but not to the PLL that the PHY uses.
seriesterminationcontrol	Input (for OCT slave)	Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode.
	Output (for OCT master)	Unconnected PHY signal, available for sharing with another PHY.
parallelterminationcontrol	Input (for OCT slave)	Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode.
	Output (for OCT master)	Unconnected PHY signal, available for sharing with another PHY.
oct_rdn	Input (for OCT master)	Must connect to calibration resistor tied to GND on the appropriate RDN pin on the device. (Refer to appropriate device handbook.)
oct_rup	Input (for OCT master)	Must connect to calibration resistor tied to $V_{ccio}$ on the appropriate RUP pin on the device. (See appropriate device handbook.)
dqs_delay_ctrl_import	Input	Allows the use of DLL in another PHY instance in this PHY instance. Connect the export port on the PHY instance with a DLL to the import port on the other PHY instance.
csr_clk <sup>(1)</sup>	Output	Clock for the configuration and status register (CSR) interface, which is the same as afi_clk and is always synchronous relative to the main data slave interface.

**Note for Table 5–14:**

(1) Applies only to the hard memory controller with multiport front end available in Arria V and Cyclone V devices.

Table 5–15 lists the controller local interface signals.

**Table 5–15. Local Interface Signals (Part 1 of 4)**

Signal Name	Direction	Description
avl_addr[] <sup>(1)</sup>	Input	<p>Memory address at which the burst should start.</p> <p>By default, the IP core maps local address to the bank interleaving scheme. You can change the ordering via the <b>Local-to-Memory Address Mapping</b> option in the <b>Controller Settings</b> page.</p> <p>This signal needs to remain stable only during the first transaction of a burst. The <code>constantBurstBehavior</code> property is always false for UniPHY controllers.</p> <p>The IP core sizes the width of this bus according to the following equations:</p> <ul style="list-style-type: none"> <li>■ Full rate controllers</li> </ul> <p>For one chip select: <math>\text{width} = \text{row bits} + \text{bank bits} + \text{column bits} - 1</math></p> <p>For multiple chip selects: <math>\text{width} = \text{chip bits}^* + \text{row bits} + \text{bank bits} + \text{column bits} - 1</math></p> <p>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 24 bits wide. To map <code>local_address</code> to bank, row and column address:</p> <p style="margin-left: 20px;"><code>avl_addr</code> is 24 bits wide</p> <p style="margin-left: 20px;"><code>avl_addr[23:11]</code> = row address [12:0]</p> <p style="margin-left: 20px;"><code>avl_addr[10:9]</code> = bank address [1:0]</p> <p style="margin-left: 20px;"><code>avl_addr[8:0]</code> = column address [9:1]</p> <p>The IP core ignores the least significant bit (LSB) of the column address (multiples of two) on the memory side, because the local data width is twice that of the memory data bus width.</p> <ul style="list-style-type: none"> <li>■ Half rate controllers</li> </ul> <p>For one chip select: <math>\text{width} = \text{row bits} + \text{bank bits} + \text{column bits} - 2</math></p> <p>For multiple chip selects: <math>\text{width} = \text{chip bits}^* + \text{row bits} + \text{bank bits} + \text{column bits} - 2</math></p> <p>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 23 bits wide. To map <code>local_address</code> to bank, row and column address:</p> <p style="margin-left: 20px;"><code>avl_addr</code> is 23 bits wide</p> <p style="margin-left: 20px;"><code>avl_addr[22:10]</code> = row address [12:0]</p> <p style="margin-left: 20px;"><code>avl_addr[9:8]</code> = bank address [1:0]</p> <p style="margin-left: 20px;"><code>avl_addr[7:0]</code> = column address [9:2]</p> <p>The IP core ignores two LSBs of the column address (multiples of four) on the memory side, because the local data width is four times that of the memory data bus width.</p> <ul style="list-style-type: none"> <li>■ Quarter rate controllers</li> </ul> <p>For one chip select: <math>\text{width} = \text{row bits} + \text{bank bits} + \text{column bits} - 3</math></p> <p>For multiple chip selects: <math>\text{width} = \text{chip bits}^* + \text{row bits} + \text{bank bits} + \text{column bits} - 3</math></p> <p>If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, the local address is 22 bits wide.</p> <p>(* <i>chip bits</i> is a derived value indicating the number of address bits necessary to uniquely address every memory rank in the system; this value is not user configurable.)</p>

**Table 5–15. Local Interface Signals (Part 2 of 4)**

<b>Signal Name</b>	<b>Direction</b>	<b>Description</b>
avl_be[] <sup>(2)</sup>	Input	<p>Byte enable signal, which you use to mask off individual bytes during writes. <code>avl_be</code> is active high; <code>mem_dm</code> is active low.</p> <p>To map <code>avl_wdata</code> and <code>avl_be</code> to <code>mem_dq</code> and <code>mem_dm</code>, consider a full-rate design with 32-bit <code>avl_wdata</code> and 16-bit <code>mem_dq</code>.</p> <pre>avl_wdata = &lt; 22334455 &gt;&lt; 667788AA &gt;&lt; BBCCDDEE &gt; avl_be    = &lt; 1100    &gt;&lt; 0110    &gt;&lt; 1010    &gt; These values map to: Mem_dq = &lt;4455&gt;&lt;2233&gt;&lt;88AA&gt;&lt;6677&gt;&lt;DDEE&gt;&lt;BBCC&gt; Mem_dm = &lt;1 1 &gt;&lt;0 0 &gt;&lt;0 1 &gt;&lt;1 0 &gt;&lt;0 1 &gt;&lt;0 1 &gt;</pre>
avl_burstbegin <sup>(3)</sup>	Input	<p>The Avalon burst begin strobe, which indicates the beginning of an Avalon burst. Unlike all other Avalon-MM signals, the burst begin signal does not stay asserted if <code>avl_ready</code> is deasserted.</p> <p>For write transactions, assert this signal at the beginning of each burst transfer and keep this signal high for one cycle per burst transfer, even if the slave deasserts <code>avl_ready</code>. The IP core samples this signal at the rising edge of <code>phy_clk</code> when <code>avl_write_req</code> is asserted. After the slave deasserts the <code>avl_ready</code> signal, the master keeps all the write request signals asserted until <code>avl_ready</code> signal becomes high again.</p> <p>For read transactions, assert this signal for one clock cycle when read request is asserted and <code>avl_addr</code> from which the data should be read is given to the memory. After the slave deasserts <code>avl_ready</code> (<code>waitrequest_n</code> in Avalon interface), the master keeps all the read request signals asserted until <code>avl_ready</code> becomes high again.</p>
avl_read_req <sup>(4)</sup>	Input	Read request signal. You cannot assert read request and write request signals at the same time. The controller must deassert <code>reset_phy_clk_n</code> before you can assert <code>local_autopch_req</code> .
local_refresh_req	Input	User-controlled refresh request. If <b>Enable User Auto-Refresh Controls</b> option is turned on, <code>local_refresh_req</code> becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including grouping together multiple refresh commands. Refresh requests take priority over read and write requests, unless the IP core is already processing the requests.
local_refresh_chip	Input	<p>Controls which chip to issue the user refresh to. The IP core uses this active high signal with <code>local_refresh_req</code>. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip.</p> <p>For example: If <code>local_refresh_chip</code> signal is assigned with a value of <code>4'b0101</code>, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.</p>
avl_size[] <sup>(5)</sup>	Input	<p>Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The IP core supports Avalon burst lengths from 1 to 64. The IP core derives the width of this signal based on the burst count that you specify in the <b>Local Maximum Burst Count</b> option. With the derived width, you specify a value ranging from 1 to the local maximum burst count specified.</p> <p>This signal needs to remain stable only during the first transaction of a burst. The <code>constantBurstBehavior</code> property is always false for UniPHY controllers.</p>

**Table 5–15. Local Interface Signals (Part 3 of 4)**

<b>Signal Name</b>	<b>Direction</b>	<b>Description</b>
avl_wdata[] <sup>(6)</sup>	Input	Write data bus. The width of avl_wdata is twice that of the memory data bus for a full-rate controller, four times the memory data bus for a half-rate controller, and eight times the memory data bus for a quarter-rate controller. If <b>Generate power-of-2 data bus widths for Qsys and SOPC Builder</b> is enabled, the width is rounded down to the nearest power of 2.
avl_write_req <sup>(7)</sup>	Input	Write request signal. You cannot assert read request and write request signal at the same time. The controller must deassert reset_phy_clk_n before you can assert avl_write_req.
local_autopch_req <sup>(8)</sup>	Input	<p>User control of autoprecharge. If you turn on <b>Enable Auto-Precache Control</b>, the local_autopch_req signal becomes available and you can request the controller to issue an autoprecharge write or autoprecharge read command.</p> <p>These commands cause the memory to issue a precharge command to the current bank at the appropriate time without an explicit precharge command from the controller. This feature is particularly useful if you know the current read or write is the last one you intend to issue to the currently open row. The next time you need to use that bank, the access could be quicker as the controller does not need to precharge the bank before activating the row you wish to access.</p> <p>Upon receipt of the local_autopch_req signal, the controller evaluates the pending commands in the command buffer and determines the most efficient autoprecharge operation to perform, reordering commands if necessary.</p> <p>The controller must deassert reset_phy_clk_n before you can assert local_autopch_req.</p>
local_self_rfsh_chip	Input	<p>Controls which chip to issue the user refresh to. The IP core uses this active high signal with local_self_rfsh_req. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip.</p> <p>For example: If local_self_rfsh_chip signal is assigned with a value of 4'b0101, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.</p>
local_self_rfsh_req	Input	User control of the self-refresh feature. If you turn on <b>Enable Self-Refresh Controls</b> , you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting local_self_rfsh_ack. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting local_self_rfsh_req and the controller responds by deasserting local_self_rfsh_ack when it has successfully brought the memory out of the self-refresh state.
local_init_done	Output	When the memory initialization, training, and calibration are complete, the PHY sequencer asserts ctrl_usr_mode_rdy to the memory controller, which then asserts this signal to indicate that the memory interface is ready for use.
avl_rdata[] <sup>(9)</sup>	Output	Read data bus. The width of avl_rdata is twice that of the memory data bus for a full rate controller; four times the memory data bus for a half rate controller. If <b>Generate power-of-2 data bus widths for Qsys and SOPC Builder</b> is enabled, the width is rounded down to the nearest power of 2.

**Table 5–15. Local Interface Signals (Part 4 of 4)**

Signal Name	Direction	Description
avl_rdata_error <sup>(10)</sup>	Output	Asserted if the current read data has an error. This signal is only available if you turn on <b>Enable Error Detection and Correction Logic</b> . The controller asserts this signal with the <code>avl_rdata_valid</code> signal. If the controller encounters double-bit errors, no correction is made and the controller asserts this signal.
avl_rdata_valid <sup>(11)</sup>	Output	Read data valid signal. The <code>avl_rdata_valid</code> signal indicates that valid data is present on the read data bus.
avl_ready <sup>(12)</sup>	Output	The <code>avl_ready</code> signal indicates that the controller is ready to accept request signals. If controller asserts the <code>avl_ready</code> signal in the clock cycle that it asserts a read or write request, the controller accepts that request. The controller deasserts the <code>avl_ready</code> signal to indicate that it cannot accept any more requests. The controller can buffer eight read or write requests, after which the <code>avl_ready</code> signal goes low.
local_refresh_ack	Output	Refresh request acknowledge, which the controller asserts for one clock cycle every time it issues a refresh. Even if you do not turn on <b>Enable User Auto-Refresh Controls</b> , <code>local_refresh_ack</code> still indicates to the local interface that the controller has just issued a refresh command.
local_self_rfsh_ack	Output	Self refresh request acknowledge signal. The controller asserts and deasserts this signal in response to the <code>local_self_rfsh_req</code> signal.
local_power_down_ack	Output	Auto power-down acknowledge signal. The controller asserts this signal for one clock cycle every time auto power-down is issued.
ecc_interrupt <sup>(13)</sup>	Output	Interrupt signal from the ECC logic. The controller asserts this signal when the ECC feature is turned on, and the controller detects an error.

**Notes for Table 5–15:**

- (1) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_addr` becomes a per port value, `avl_addr_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (2) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_be` becomes a per port value, `avl_be_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (3) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_burstbegin` becomes a per port value, `avl_burstbegin_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (4) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_read_req` becomes a per port value, `avl_read_req_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (5) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_size` becomes a per port value, `avl_size_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (6) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_wdata` becomes a per port value, `avl_wdata_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (7) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_write_req` becomes a per port value, `avl_write_req_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (8) This signal is not applicable to the hard memory controller.
- (9) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_rdata` becomes a per port value, `avl_rdata_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (10) This signal is not applicable to the hard memory controller.
- (11) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_rdata_valid` becomes a per port value, `avl_rdata_valid_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (12) For the hard memory controller with multiport front end available in Arria V and Cyclone V devices, `avl_ready` becomes a per port value, `avl_ready_#`, where # is a numeral from 0–5, based on the number of ports selected in the Controller tab.
- (13) This signal is not applicable to the hard memory controller.

Table 5–16 lists the controller interface signals.

**Table 5–16. Interface Signals**

Signal Name	Direction	Description
mem_dq []	Bidirectional	Memory data bus. This bus is half the width of the local read and write data busses.
mem_dqs []	Bidirectional	Memory data strobe signal, which writes data into the memory device and captures read data into the Altera device.
mem_dqs_n []	Bidirectional	Inverted memory data strobe signal, which with the mem_dqs signal improves signal integrity.
mem_ck	Output	Clock for the memory device.
mem_ck_n	Output	Inverted clock for the memory device.
mem_addr []	Output	Memory address bus.
mem_ac_parity <sup>(1)</sup>	Output	Address or command parity signal generated by the PHY and sent to the DIMM. DDR3 SDRAM only.
mem_ba []	Output	Memory bank address bus.
mem_cas_n	Output	Memory column address strobe signal.
mem_cke []	Output	Memory clock enable signals.
mem_cs_n []	Output	Memory chip select signals.
mem_dm []	Output	Memory data mask signal, which masks individual bytes during writes.
mem_odt	Output	Memory on-die termination control signal.
mem_ras_n	Output	Memory row address strobe signal.
mem_we_n	Output	Memory write enable signal.
parity_error_n <sup>(1)</sup>	Output	Active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset. DDR3 SDRAM only
mem_err_out_n <sup>(1)</sup>	Input	Signal sent from the DIMM to the PHY to indicate that a parity error has occurred for a particular cycle. DDR3 SDRAM only.

**Notes to Table 5–16:**

- (1) This signal is for registered DIMMs only.

Table 5–17 lists the CSR interface signals.

**Table 5–17. CSR Interface Signals (Part 1 of 2)**

Signal Name	Direction	Description
csr_addr []	Input	Register map address. The width of csr_addr is 16 bits.
csr_be []	Input	Byte-enable signal, which you use to mask off individual bytes during writes. csr_be is active high.
csr_clk <sup>(1)</sup>	Output	Clock for the configuration and status register (CSR) interface, which is the same as afi_clk and is always synchronous relative to the main data slave interface.
csr_wdata []	Input	Write data bus. The width of csr_wdata is 32 bits.
csr_write_req	Input	Write request signal. You cannot assert csr_write_req and csr_read_req signals at the same time.
csr_read_req	Input	Read request signal. You cannot assert csr_read_req and csr_write_req signals at the same time.

**Table 5–17. CSR Interface Signals (Part 2 of 2)**

Signal Name	Direction	Description
csr_rdata[]	Output	Read data bus. The width of <code>csr_rdata</code> is 32 bits.
csr_rdata_valid	Output	Read data valid signal. The <code>csr_rdata_valid</code> signal indicates that valid data is present on the read data bus.
csr_waitrequest	Output	The <code>csr_waitrequest</code> signal indicates that the HPC II is busy and not ready to accept request signals. If the <code>csr_waitrequest</code> signal goes high in the clock cycle when a read or write request is asserted, that request is not accepted. If the <code>csr_waitrequest</code> signal goes low, the HPC II is then ready to accept more requests.

**Note for Table 5–17:**

- (1) Applies only to the hard memory controller with multiport front end available in Arria V and Cyclone V devices.

## Controller Register Map

The controller register map allows you to control the memory controller settings.

**Table 5–18** lists the register map for the controller.

**Table 5–18. Controller Register Map (Part 1 of 5)**

Address	Bit	Name	Default	Access	Description
0x100	0	Reserved.	0	—	Reserved for future use.
	1	Reserved.	0	—	Reserved for future use.
	2	Reserved.	0	—	Reserved for future use.
	7:3	Reserved.	0	—	Reserved for future use.
	13:8	Reserved.	0	—	Reserved for future use.
	30:14	Reserved.	0	—	Reserved for future use.

**Table 5–18. Controller Register Map (Part 2 of 5)**

<b>Address</b>	<b>Bit</b>	<b>Name</b>	<b>Default</b>	<b>Access</b>	<b>Description</b>
0x110	15:0	AUTO_PD_CYCLES	0x0	Read write	The number of idle clock cycles after which the controller should place the memory into power-down mode. The controller is considered to be idle if there are no commands in the command queue. Setting this register to 0 disables the auto power-down mode. The default value of this register depends on the values set during the generation of the design.
	16	Reserved.	0	—	Reserved for future use.
	17	Reserved.	0	—	Reserved for future use.
	18	Reserved.	0	—	Reserved for future use.
	19	Reserved.	0	—	Reserved for future use.
	21:20	ADDR_ORDER	00	Read write	00 - Chip, row, bank, column. 01 - Chip, bank, row, column. 10 - reserved for future use. 11 - Reserved for future use.
	22	Reserved.	0	—	Reserved for future use.
	24:23	Reserved.	0	—	Reserved for future use.
	30:24	Reserved	0	—	Reserved for future use.
0x120	7:0	Column address width	—	Read write	The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12.
	15:8	Row address width	—	Read write	The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16.
	19:16	Bank address width	—	Read write	The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3.
	23:20	Chip select address width	—	Read write	The number of chip select address bits for the memory devices in your memory interface. The range of legal values is 0-2. If there is only one single chip select in the memory interface, set this bit to 0.
	31:24	Reserved.	0	—	Reserved for future use.
0x121	31:0	Data width representation (word)	—	Read only	The number of DQS bits in the memory interface. This bit can be used to derive the width of the memory interface by multiplying this value by the number of DQ pins per DQS pin (typically 8).
0x122	7:0	Chip select representation	—	Read only	The number of chip select in binary representation. For example, a design with 2 chip selects has the value of 00000011.
	31:8	Reserved.	0	—	Reserved for future use.

**Table 5–18. Controller Register Map (Part 3 of 5)**

Address	Bit	Name	Default	Access	Description
0x123	3:0	$t_{RCD}$	—	Read write	The activate to read or write a timing parameter. The range of legal values is 2-11 cycles.
	7:4	$t_{RRD}$	—	Read write	The activate to activate a timing parameter. The range of legal values is 2-8 cycles.
	11:8	$t_{RP}$	—	Read write	The precharge to activate a timing parameter. The range of legal values is 2-11 cycles.
	15:12	$t_{MRD}$	—	Read write	The mode register load time parameter. This value is not used by the controller, as the controller derives the correct value from the memory type setting.
	23:16	$t_{RAS}$	—	Read write	The activate to precharge a timing parameter. The range of legal values is 4-29 cycles.
	31:24	$t_{RC}$	—	Read write	The activate to activate a timing parameter. The range of legal values is 8-40 cycles.
0x124	3:0	$t_{WTR}$	—	Read write	The write to read a timing parameter. The range of legal values is 1-10 cycles.
	7:4	$t_{RTP}$	—	Read write	The read to precharge a timing parameter. The range of legal values is 2-8 cycles.
	15:8	$t_{FAW}$	—	Read write	The four-activate window timing parameter. The range of legal values is 6-32 cycles.
	31:16	Reserved.	0	—	Reserved for future use.
0x125	15:0	$t_{REFI}$	—	Read write	The refresh interval timing parameter. The range of legal values is 780-6240 cycles.
	23:16	$t_{RFC}$	—	Read write	The refresh cycle timing parameter. The range of legal values is 12-88 cycles.
	31:24	Reserved.	0	—	Reserved for future use.
0x126	3:0	Reserved.	0	—	Reserved for future use.
	7:4	Reserved.	0	—	Reserved for future use.
	11:8	Reserved.	0	—	Reserved for future use.
	15:12	Reserved.	0	—	Reserved for future use.
	23:16	Burst Length	—	Read write	Value must match memory burst length.
	31:24	Reserved.	0	—	Reserved for future use.

**Table 5–18. Controller Register Map (Part 4 of 5)**

<b>Address</b>	<b>Bit</b>	<b>Name</b>	<b>Default</b>	<b>Access</b>	<b>Description</b>
0x130	0	ENABLE_ECC	1	Read write	When this bit equals 1, it enables the generation and checking of ECC. This bit is only active if ECC was enabled during IP parameterization.
	1	ENABLE_AUTO_CORR	—	Read write	When this bit equals 1, it enables auto-correction when a single-bit error is detected.
	2	GEN_SBE	0	Read write	When this bit equals 1, it enables the deliberate insertion of single-bit errors, bit 0, in the data written to memory. This bit is used only for testing purposes.
	3	GEN_DBE	0	Read write	When this bit equals 1, it enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data written to memory. This bit is used only for testing purposes.
	4	ENABLE_INTR	1	Read write	When this bit equals 1, it enables the interrupt output.
	5	MASK_SBE_INTR	0	Read write	When this bit equals 1, it masks the single-bit error interrupt.
	6	MASK_DBE_INTR	0	Read write	When this bit equals 1, it masks the double-bit error interrupt
	7	CLEAR	0	Read write	When this bit equals 1, writing to this self-clearing bit clears the interrupt signal, and the error status and error address registers.
	8	MASK_CORDROP_INTR	0	Read write	When this bit equals 1, the dropped autocorrection error interrupt is dropped.
	9	Reserved.	0	—	Reserved for future use.
0x131	0	SBE_ERROR	0	Read only	Set to 1 when any single-bit errors occur.
	1	DBE_ERROR	0	Read only	Set to 1 when any double-bit errors occur.
	2	CORDROP_ERROR	0	Read only	Value is set to 1 when any controller-scheduled autocorrections are dropped.
	7:3	Reserved.	0	—	Reserved for future use.
	15:8	SBE_COUNT	0	Read only	Reports the number of single-bit errors that have occurred since the status register counters were last cleared.
	23:16	DBE_COUNT	0	Read only	Reports the number of double-bit errors that have occurred since the status register counters were last cleared.
	31:24	CORDROP_COUNT	0	Read only	Reports the number of controller-scheduled autocorrections dropped since the status register counters were last cleared.
	0x132	31:0	ERR_ADDR	0	Read only

**Table 5–18. Controller Register Map (Part 5 of 5)**

Address	Bit	Name	Default	Access	Description
0x133	31:0	CORDROP_ADDR	0	Read only	The address of the most recent autocorrection that was dropped. This is a memory burst-aligned local address.
0x134	0	REORDER_DATA	—	Read write	
	15:1	Reserved.	0	—	Reserved for future use.
	23:16	STARVE_LIMIT	0	Read write	Number of commands that can be served before a starved command.
	31:24	Reserved.	0	—	Reserved for future use.

## Sequence of Operations

This section explains how the various blocks pass information in common situations.

### Write Command

When a requesting master issues a write command together with write data, the following events occur:

- The input interface accepts the write command and the write data.
- The input interface passes the write command to the command generator and the write data to the write data buffer.
- The command generator processes the command and sends it to the timing bank pool.
- Once all timing requirements are met and a write-data-ready notification has been received from the write data buffer, the timing bank pool sends the command to the arbiter.
- When rank timing requirements are met, the arbiter grants the command request from the timing bank pool and passes the write command to the AFI interface.
- The AFI interface receives the write command from the arbiter and requests the corresponding write data from the write data buffer.
- The PHY receives the write command and the write data, through the AFI interface.

### Read Command

When a requesting master issues a read command, the following events occur:

- The input interface accepts the read command.
- The input interface passes the read command to the command generator.
- The command generator processes the command and sends it to the timing bank pool.
- Once all timing requirements are met, the timing bank pool sends the command to the arbiter.

- When rank timing requirements are met, the arbiter grants the command request from the timing bank pool and passes the read command to the AFI interface.
- The AFI interface receives the read command from the arbiter and passes the command to the PHY.
- The PHY receives the read command through the AFI interface, and returns read data through the AFI interface.
- The AFI interface passes the read data from the PHY to the read data buffer.
- The read data buffer sends the read data to the master through the input interface.

## Read-Modify-Write Command

A read-modify-write command can occur when enabling ECC for partial write, and for ECC correction commands. When a read-modify-write command is issued, the following events occur:

- The command generator issues a read command to the timing bank pool.
- The timing bank pool and arbiter passes the read command to the PHY through the AFI interface.
- The PHY receives the read command, reads data from the memory device, and returns the read data through the AFI interface.
- The read data received from the PHY passes to the ECC block.
- The read data is processed by the write data buffer.
- When the write data buffer issues a read-modify-write data ready notification to the command generator, the command generator issues a write command to the timing bank pool. The arbiter can then issue the write request to the PHY through the AFI interface.
- When the PHY receives the write request, it passes the data to the memory device.

## Document Revision History

Table 5–19 lists the revision history for this document.

**Table 5–19. Document Revision History**

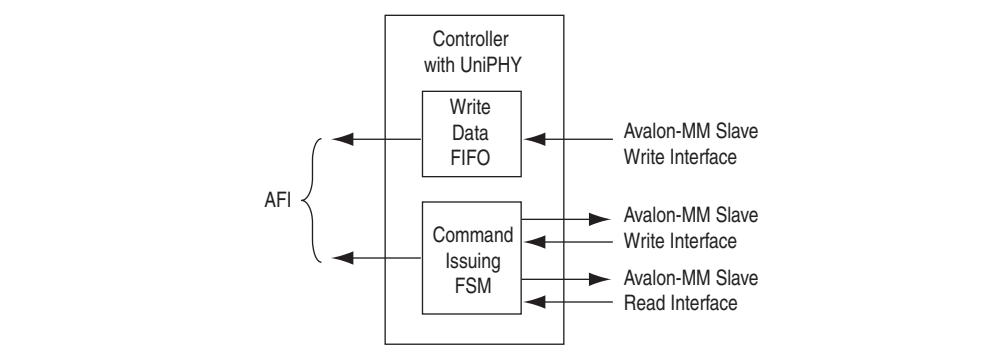
Date	Version	Changes
November 2012	2.1	<ul style="list-style-type: none"> <li>■ Added <a href="#">Controller Register Map</a> information.</li> <li>■ Added <a href="#">Burst Merging</a> information.</li> <li>■ Updated <a href="#">User-Controlled Refresh Interface</a> information.</li> <li>■ Changed chapter number from 4 to 5.</li> </ul>
June 2012	2.0	<ul style="list-style-type: none"> <li>■ Added LPDDR2 support.</li> <li>■ Added Feedback icon.</li> </ul>
November 2011	1.1	<ul style="list-style-type: none"> <li>■ Revised <a href="#">Figure 5–1</a>.</li> <li>■ Added <a href="#">AXI to Avalon-ST Converter</a> information.</li> <li>■ Added <a href="#">AXI Data Slave Interface</a> information.</li> <li>■ Added <a href="#">Half-Rate Bridge</a> information.</li> </ul>

The QDR II and QDR II+ controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configurations. QDR II and QDR II+ SRAM has unidirectional data buses, therefore read and write operations are highly independent of each other and each has its own interface and state machine.

### Block Description

This topic describes the blocks in the IP. [Figure 6–1](#) shows a block diagram of the QDR II and QDR II+ SRAM controller architecture.

**Figure 6–1. QDR II and QDR II+ SRAM Controller Architecture Block Diagram**



### Avalon-MM Slave Read and Write Interfaces

The read and write blocks accept read and write requests, respectively, from the Avalon-MM interface . Each block has a simple state machine that represents the state of the command and address registers, which stores the command and address when a request arrives.

The read data passes through without the controller registering it, as the PHY takes care of read latency. The write data goes through a pipeline stage to delay for a fixed number of cycles as specified by the write latency. In the full-rate burst length of four controller, the write data is also multiplexed into a burst of 2, which is then multiplexed again in the PHY to become a burst of 4 in DDR.

The user interface to the controller has separate read and write Avalon-MM interfaces because reads and writes are independent of each other in the memory device. The separate channels give efficient use of available bandwidth.

## Command Issuing FSM

The command-issuing full-state machine (FSM) has two states: INIT and INIT\_COMPLETE. In the INIT\_COMPLETE state, commands are issued immediately as requests arrive using combinational logic and do not require state transitions.

### AFI

In the full-rate burst length of two configuration, the controller can issue both read and write commands in the same clock cycle. In the memory device, both commands are clocked on the positive edge, but the read address is clocked on the positive edge, while the write address is clocked on the negative edge. Care must be taken on how these signals are ordered in the AFI.

For the half-rate burst length of four configuration the controller also issues both read and write commands, but the AFI width is doubled to fill two memory clocks per controller clock. As the controller only issues one write command and one read command per controller clock, the AFI read and write signals corresponding to the other memory cycle are tied to no operation (NOP).

For information on the AFI, refer to [AFI 3.0 Specification](#).

## Avalon-MM and Memory Data Width

**Table 6–1** lists the data width ratio between the memory interface and the Avalon-MM interface. The half-rate controller does not support burst-of-2 devices because it under-uses the available memory bandwidth. Regardless of full or half-rate decision and the device burst length, the Avalon-MM interface must supply all the data for the entire memory burst in a single clock cycle. Therefore the Avalon-MM data width of the full-rate controller with burst-of-4 devices is four times the memory data width. For width-expanded configurations, the data width is further multiplied by the expansion factor (not shown in table 5-1 and 5-2).

**Table 6–1. Data Width Ratio**

Memory Burst Length	Half-Rate Designs	Full-Rate Designs
QDR II 2-word burst	No Support	2:1
QDR II and II+ 4-word burst		4:1

## Signal Description

This topic discusses the signals for each interface.

For information on the AFI signals, refer to [AFI 3.0 Specification](#).

## Avalon-MM Slave Read Interface

Table 6–2 lists the signals of the controller’s Avalon-MM slave read interface.

**Table 6–2. Avalon-MM Slave Read Signals**

Signal	Width	Direction	Avalon-MM Signal Type
avl_r_ready	1	Out	waitrequest_n
avl_r_read_req	1	In	read
avl_r_addr	$\leq 20$	In	address
avl_r_rdata_valid	1	Out	readdatavalid
avl_r_rdata	16, 18, 36, 72, 144	Out	readdata
avl_r_size	$\log_2(\text{MAX\_BURST\_SIZE}) + 1$	In	burstcount

-  The data width of the Avalon-MM interface is restricted to powers of two when using SOPC Builder or Qsys. Non-power-of-two data widths are supported when using the MegaWizard Plug-In Manager.
-  For more information about the Avalon interface, refer to *Avalon Interface Specifications*.

## Avalon-MM Slave Write Interface

Table 6–3 lists the signals of the controller’s Avalon-MM slave write interface.

**Table 6–3. Avalon-MM Slave Write Signals**

Signal	Width	Direction	Avalon-MM Signal Type
avl_w_ready	1	Out	waitrequest_n
avl_w_write_req	1	In	write
avl_w_addr	$\leq 20$	In	address
avl_w_wdata	18, 36, 72, 144	In	writedata
avl_w_be	2,4,8,16	In	byteenable
avl_w_size	$\log_2(\text{MAX\_BURST\_SIZE}) + 1$	In	burstcount

-  For more information about the Avalon interface, refer to *Avalon Interface Specifications*.

## Document Revision History

Table 6-4 lists the revision history for this document.

**Table 6-4. Document Revision History**

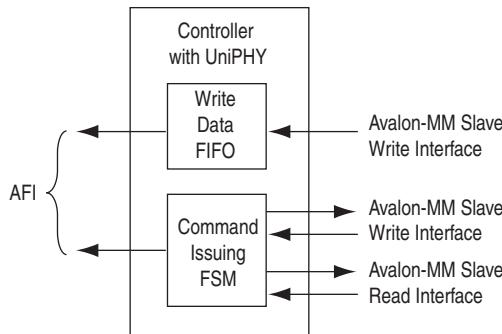
Date	Version	Changes
November 2012	3.3	Changed chapter number from 5 to 6.
June 2012	3.2	Added Feedback icon.
November 2011	3.1	Harvested Controller chapter from 11.0 <b>QDR II and QDR II+ SRAM Controller with UniPHY User Guide</b> .

The RLDARAM II controller translates memory requests from the Avalon Memory-Mapped (Avalon-MM) interface to AFI, while satisfying timing requirements imposed by the memory configurations.

### Block Description

This topic describes the blocks in the IP. Figure 7-1 shows a block diagram of the RLDARAM II controller architecture.

**Figure 7-1. RLDARAM II Controller Architecture Block Diagram**



### Avalon-MM Slave Interface

This Avalon-MM slave interface accepts read and write requests. A simple state machine represents the state of the command and address registers, which stores the command and address when a request arrives.

The Avalon-MM slave interface decomposes the Avalon-MM address to the memory bank, column, and row addresses. The IP automatically maps the bank address to the LSB of the Avalon address vector.

The Avalon-MM slave interface includes a burst adaptor, which has the following two parts:

- The first part is a read and write request combiner that groups requests to sequential addresses into the native memory burst. Given that the second request arrives within the read and write latency window of the first request, the controller can combine and satisfy both requests with a single memory transaction.
- The second part is the burst divider in the front end of the Avalon-MM interface, which breaks long Avalon bursts into individual requests of sequential addresses, which then pass to the controller state machine.

©2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



## Write Data FIFO Buffer

The write data FIFO buffer accepts write data from the Avalon-MM interface. The AFI controls the subsequent consumption of the FIFO buffer write data.

## Command Issuing FSM

The command issuing finite-state machine (FSM) has three states. The controller is in the INIT state when the PHY initializes the memory. Upon receiving the afi\_cal\_success signal, the state transitions to INIT\_COMPLETE. If the calibration fails, afi\_cal\_fail is asserted and the state transitions to INIT\_FAIL. The PHY receives commands only in the INIT\_COMPLETE state.

When a refresh request arrives at the state machine at the same time as a read or write request, the refresh request takes precedence. The read or write request waits until there are no more refresh requests, and is issued immediately if timing requirements are met.

## Refresh Timer

With automatic refresh, the refresh timer periodically issues refresh requests to the command issuing FSM. The refresh interval can be set at generation.

## Timer Module

The timer module contains one DQ timer and eight bank timers (one per bank). The DQ timer tracks how often read and write requests can be issued, to avoid bus contention. The bank timers track the cycle time ( $t_{RC}$ ).

The 8-bit wide output bus of the bank timer indicates to the command issuing FSM whether each bank can be issued a read, write, or refresh command.

## AFI

For information on the AFI, refer to [AFI 3.0 Specification](#).

## User-Controlled Features

The following features are available on the **General Settings** tab of the parameter editor. These features are disabled by default.

## Error Detection Parity

The error detection parity protection feature creates a simple parity encoder block which processes all read and write data. The error detection feature asserts an error signal if it detects any corrupted data during the read process. For every 8 bits of write data, a parity bit is generated and concatenated to the data before it is written to the memory. During the subsequent read operation, the parity bit is checked against the data bits to ensure data integrity.

Enabling the error detection parity protection feature reduces the local data width by one. For example, a nine-bit memory interface will present eight bits of data to the controller interface.

You can enable error detection parity protection in the **Controller Settings** section of the **General Settings** tab of the parameter editor.

## User-Controlled Refresh

The user-controlled refresh feature allows you to take control of the refresh process that the controller normally performs automatically. You can control when refresh requests occur, and, if there are multiple memory devices, you control which bank receives the refresh signal. When you enable this feature, you disable auto-refresh, and assume responsibility for maintaining the necessary average periodic refresh rate.

You can enable user-controlled refresh in the **Controller Settings** section of the **General Settings** tab of the parameter editor.

## Avalon-MM and Memory Data Width

**Table 7–1** lists the data width ratio between the memory interface and the Avalon-MM interface. The half-rate controller does not support burst-of-2 devices because it under-uses the available memory bandwidth.

**Table 7–1. Data Width Ratio**

Memory Burst Length	Half-Rate Designs	Full-Rate Designs
2-word	No Support	2:1
4-word	4:1	
8-word		

## Signal Description

This topic discusses the signals for each interface.

For information on the AFI signals, refer to [AFI 3.0 Specification](#).

## Avalon-MM Slave Interface

**Table 7–2** lists the signals of the controller’s Avalon-MM slave interface.

**Table 7–2. Avalon-MM Slave Signals**

Signal	Width	Direction	Avalon-MM Signal Type	Description
avl_size	1 to 11	In	burstcount	—
avl_ready	1	Out	waitrequest_n	—
avl_read_req	1	In	read	—
avl_write_req	1	In	write	—
avl_addr	25	In	address	—
avl_rdata_valid	1	Out	readdatavalid	—

**Table 7–2. Avalon-MM Slave Signals**

<b>Signal</b>	<b>Width</b>	<b>Direction</b>	<b>Avalon-MM Signal Type</b>	<b>Description</b>
avl_rdata	18, 36, 72, 144	Out	readdata	—
avl_wdata	18, 36, 72, 144	In	writedata	—



The data width of the Avalon-MM interface is restricted to powers of two when using SOPC Builder or Qsys. Non-power-of-two data widths are supported when using the MegaWizard Plug-In Manager.

## Document Revision History

Table 7–3 lists the revision history for this document.

**Table 7–3. Document Revision History**

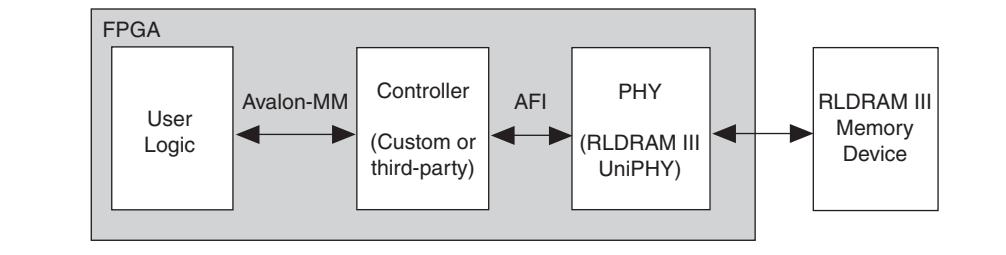
<b>Date</b>	<b>Version</b>	<b>Changes</b>
November 2012	3.3	Changed chapter number from 6 to 7.
June 2012	3.2	Added Feedback icon.
November 2011	3.1	Harvested Controller chapter from 11.0 <b>RLDRAM II Controller with UniPHY IP User Guide</b> .

The RLDRAm 3 PHY-only IP works with a customer-supplied memory controller to translate memory requests from user logic to RLDRAm 3 memory devices, while satisfying timing requirements imposed by the memory configurations.

### Block Description

The RLDRAm 3 UniPHY-based IP is a PHY-only offering which you can use with a third-party controller or a controller that you develop yourself. [Figure 8–1](#) shows a block diagram of the RLDRAm 3 system architecture.

**Figure 8–1. RLDRAm 3 System Architecture**



### Features

The RLDRAm 3 UniPHY-based IP supports features available from major RLDRAm 3 device vendors at speeds of up to 800 MHz. The following list summarizes key features of the RLDRAm 3 UniPHY-based IP:

- support for Arria V GZ and Stratix V devices
- standard AFI interface between the PHY and the memory controller
- quarter-rate and half-rate AFI interface
- maximum frequency of 533 MHz for half-rate operation and 800 MHz for quarter-rate operation
- burst length of 2, 4, or 8
- x18 and x36 memory organization
- common I/O device support
- nonmultiplexed addressing
- multibank write and refresh protocol (programmable through mode register)
- optional use of data mask pins

©2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

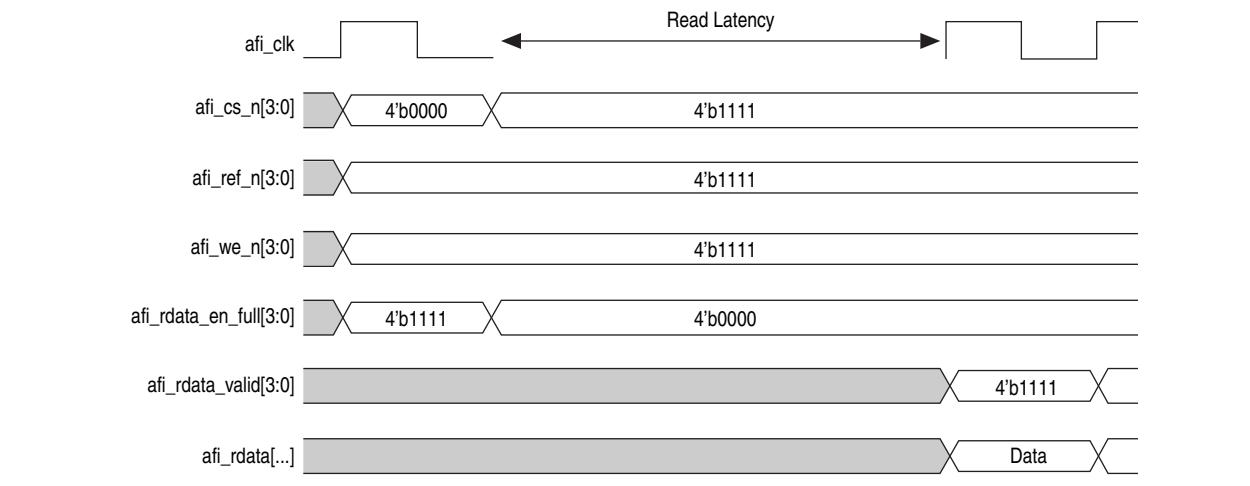


## RLDRAM III AFI Protocol

The RLDRAm 3 UniPHY-based IP communicates with the memory controller using an AFI interface that follows the AFI 3.0 specification. To maximize bus utilization efficiency, the RLDRAm 3 UniPHY-based IP can issue multiple memory read/write operations within a single AFI cycle.

**Figure 8–2** illustrates AFI bus activity when a quarter-rate controller issues four consecutive burst-length 2 read requests.

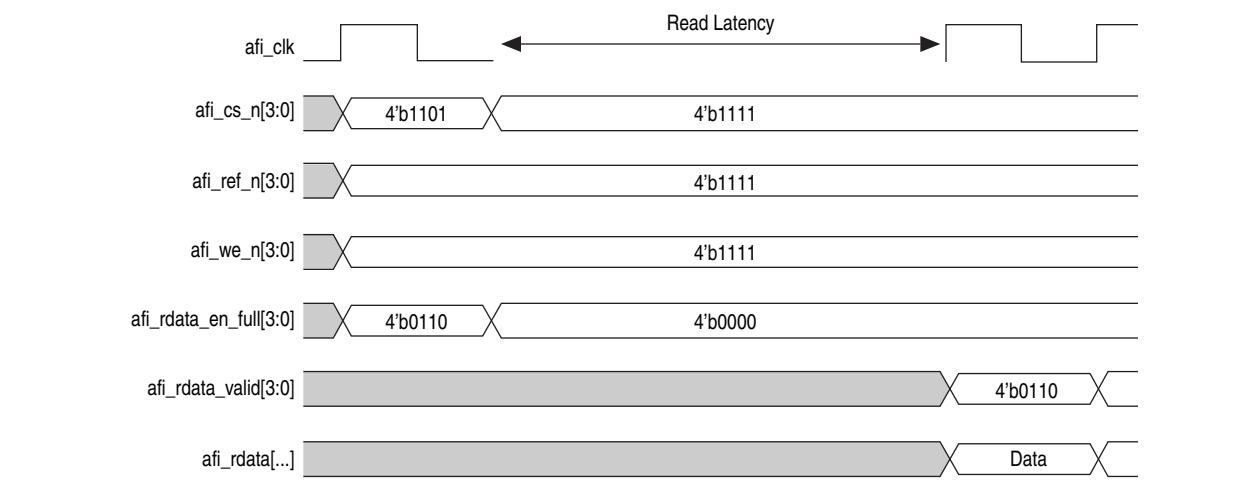
**Figure 8–2. AFI Bus Activity for Quarter-Rate Controller Issuing Four Burst-Length 2 Read Requests**



The controller does not have to begin a read or write command using channel 0 of the AFI bus. The flexibility afforded by being able to begin a command on any bus channel can facilitate command scheduling in the memory controller.

**Figure 8–3** illustrates the AFI bus activity when a quarter-rate controller issues a single burst-length 4 read command to the memory on channel 1 of the AFI bus.

**Figure 8–3. AFI Bus Activity for Quarter-Rate Controller Issuing One Burst-Length 4 Read Request**



For information on the AFI, refer to [AFI 3.0 Specification](#).

## **Document Revision History**

Table 8–1 lists the revision history for this document.

**Table 8–1. Document Revision History**

<b>Date</b>	<b>Version</b>	<b>Changes</b>
November 2012	1.0	Initial release.



This chapter describes the example designs and the traffic generator.

Two independent example designs are created during generation with the MegaWizard Plug-In Manager. These example designs illustrate how to instantiate and connect the memory interface for both synthesis and simulation flows.

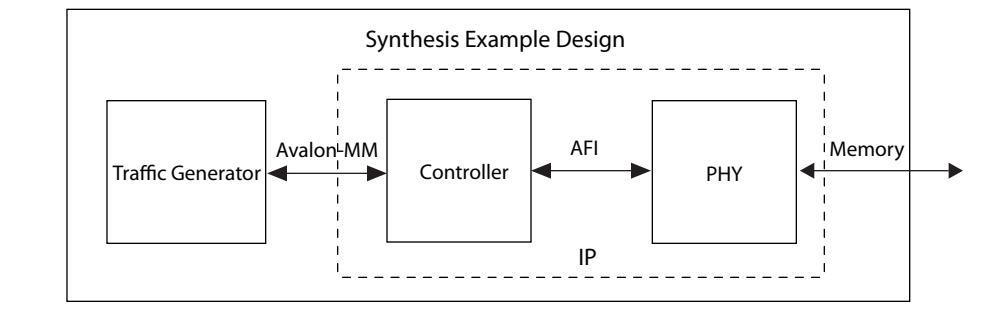
The two example designs are completely independent, and contain independent RTL files and other project files; they should be compiled or simulated separately, and the files should not be mixed. Nonetheless, the designs are related, as the simulation example design builds upon the design of the synthesis example design.

### Synthesis Example Design

The synthesis example design contains the following major blocks, as shown in [Figure 9–1](#):

- A traffic generator, which is a synthesizable Avalon-MM example driver that implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The traffic generator also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the UniPHY memory interface, which includes a memory controller that moderates between the Avalon-MM interface and the AFI interface, and the UniPHY, which serves as an interface between the memory controller and external memory devices to perform read and write operations.

**Figure 9–1. Synthesis Example Design**

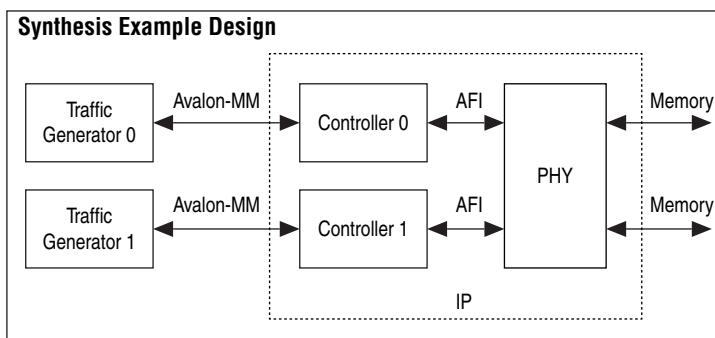


©2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



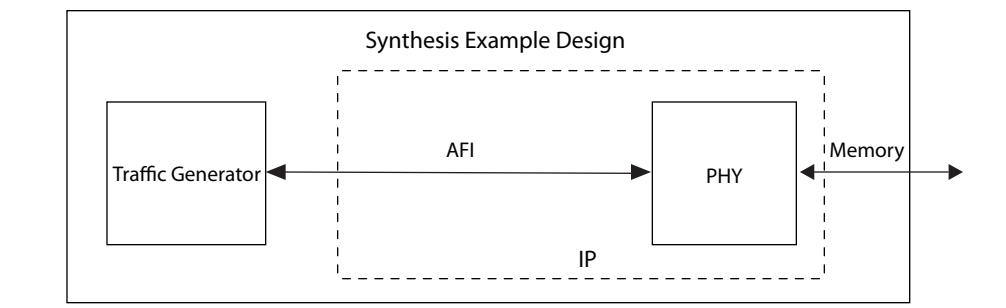
If you are using the Ping Pong PHY feature, the synthesis example design includes two traffic generators issuing commands to two independent memory devices through two independent controllers and a common PHY, as shown in [Figure 9–2](#).

**Figure 9–2. Synthesis Example Design for Ping Pong PHY**



If you are using RLDRAM 3, the traffic generator in the synthesis example design communicates directly with the PHY using AFI, as shown in [Figure 9–3](#).

**Figure 9–3. Synthesis Example Design for RLDRAM 3 Interfaces**



You can obtain the synthesis example design by generating your IP core using the MegaWizard Plug-In Manager flow. The files related to the synthesis example design reside at `<variation_name>_example_design/example_project`. The synthesis example design includes a Quartus II project file (`<variation_name>_example_design/example_project/<variation_name>_example.qpf`). The Quartus II project file can be compiled in the Quartus II software, and can be run on hardware.

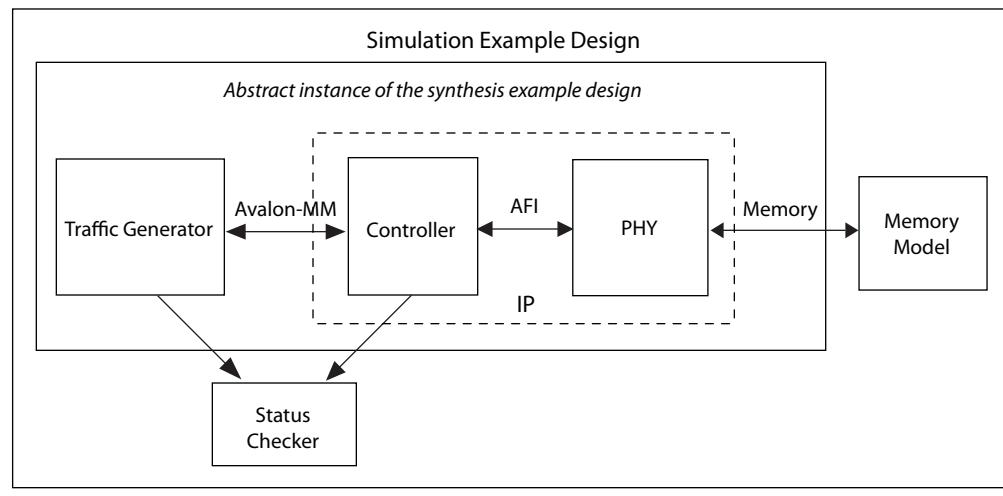
- ☞ If one or more of the **PLL Sharing Mode**, **DLL Sharing Mode**, or **OCT Sharing Mode** parameters are set to any value other than **No Sharing**, the synthesis example design will contain two traffic generator/memory interface instances. The two traffic generator/memory interface instances are related only by shared PLL/DLL/OCT connections as defined by the parameter settings. The traffic generator/memory interface instances demonstrate how you can make such connections in your own designs.

## Simulation Example Design

The simulation example design contains the following major blocks, as shown in [Figure 9–4](#):

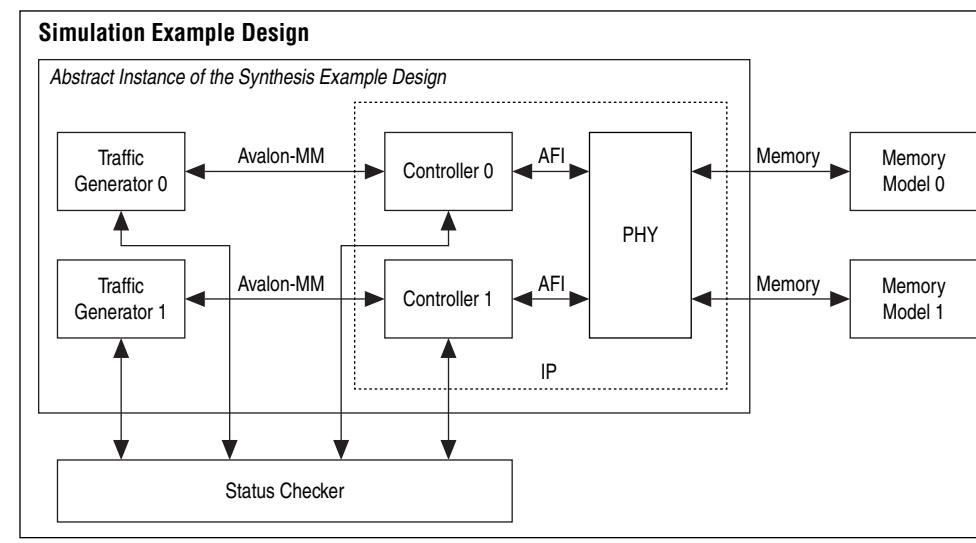
- An instance of the synthesis example design. As described in the previous section, the synthesis example design contains a traffic generator and an instance of the UniPHY memory interface. These blocks default to abstract simulation models where appropriate for rapid simulation.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Frequently, memory vendors provide simulation models for specific memory components that you can download from their websites.
- A status checker, which monitors the status signals from the UniPHY IP and the traffic generator, to signal an overall pass or fail condition.

**Figure 9–4. Simulation Example Design**



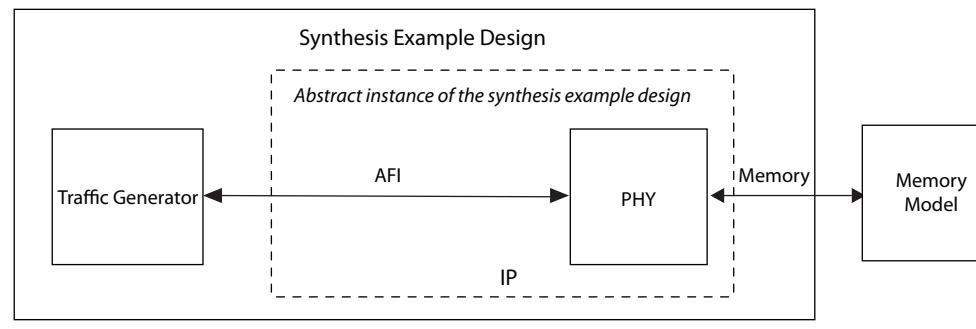
If you are using the Ping Pong PHY feature, the simulation example design includes two traffic generators issuing commands to two independent memory devices through two independent controllers and a common PHY, as shown in [Figure 9-5](#).

**Figure 9-5. Simulation Example Design for Ping Pong PHY**



If you are using RLDRAM 3, the traffic generator in the simulation example design communicates directly with the PHY using AFI, as shown in [Figure 9-6](#).

**Figure 9-6. Simulation Example Design for RLDRAM 3 Interface**



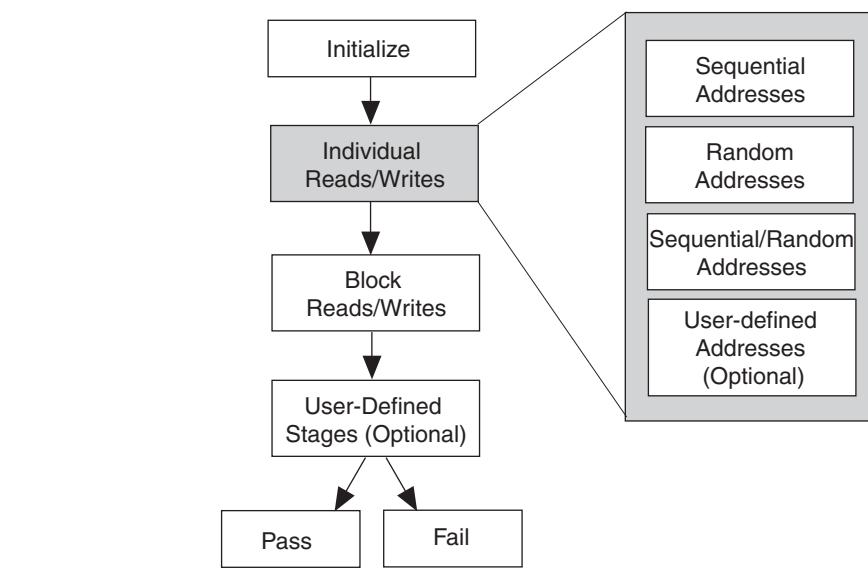
You can obtain the simulation example design by generating your IP core with the MegaWizard Plug-In Manager. The files related to the simulation example design reside at `<variation_name>_example_design/simulation`. After obtaining the files generated by the MegaWizard Plug-In Manager, you must still generate the simulation example design RTL for your desired HDL language. The file `<variation_name>_example_design/simulation/README.txt` contains details about how to generate the IP and to run the simulation in ModelSim-AE/SE.

## Traffic Generator and BIST Engine

The traffic generator and built-in self test (BIST) engine for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. The traffic generator creates read and write traffic, stores the expected read responses internally, and compares the expected responses to the read responses as they arrive. If all reads report their expected response, the pass signal is asserted; however, if any read responds with unexpected data a fail signal occurs.

Each operation generated by the traffic generator is a single write or block of writes followed by a single read or block of reads to the same addresses, which allows the driver to precisely determine the data that should be expected when the read data is returned by the memory interface. The traffic generator comprises a traffic generation block, the Avalon-MM interface and a read comparison block. The traffic generation block generates addresses and write data, which are then sent out over the Avalon-MM interface. The read comparison block compares the read data received from the Avalon-MM interface to the write data from the traffic generator. If at any time the data received is not the expected data, the read comparison block records the failure, finishes reading all the data, and then signals that there is a failure and the traffic generator enters a fail state. If all patterns have been generated and compared successfully, the traffic generator enters a pass state.

**Figure 9–7. Example Driver Operations**



Within the traffic generator, there are the following main states:

- Generation of individual read and writes
- Generation of block read and writes
- The pass state
- The fail state

Within each of the generation states there are the following substates:

- Sequential address generation
- Random address generation
- Mixed sequential and random address generation

For each of the states and substates, the order and number of operations generated for each substate is parameterizable—you can decide how many of each address pattern to generate, or can disable certain patterns entirely if you want. The sequential and random interleave substate takes in additions to the number of operations to generate. An additional parameter specifies the ratio of sequential to random addresses to generate randomly.

## Read and Write Generation

The traffic generator block can generate individual or block reads and writes.

### Individual Read and Write Generation

During the traffic generator's individual read and write generation state, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions is chosen according to the specific substate. The width of the Avalon-MM interface is a global parameter for the driver, but each substate can have a parameterizable range of burst lengths for each operation.

### Block Read and Write Generation

During the traffic generator's block read and write generation state, the traffic generator block generates a parameterizable number of write operations followed by the same number of read operations. The specific addresses generated for the blocks are chosen by the specific substates. The burst length of each block operation can be parameterized by a range of acceptable burst lengths.

## Address and Burst Length Generation

The traffic generator block can perform sequential or random addressing.

### Sequential Addressing

The sequential addressing substate defines a traffic pattern where addresses are chosen in sequential order starting from a user definable address. The number of operations in this substate is parameterizable.

### Random Addressing

The random addressing substate defines a traffic pattern where addresses are chosen randomly over a parameterizable range. The number of operations in this substate is parameterizable.

## Sequential and Random Interleaved Addressing

The sequential and random interleaved addressing substate defines a traffic pattern where addresses are chosen to be either sequential or random based on a parameterizable ratio. The acceptable address range is parameterizable as is the number of operations to perform in this substate.

## Traffic Generator Signals

Table 9–1 lists the signals used by the traffic generator.

**Table 9–1. Traffic Generator Signals**

Signal	Width	Signal Type
clk		
reset_n		
avl_ready		avl_ready
avl_write_req		avl_write_req
avl_read_req		avl_read_req
avl_addr	24	avl_addr
avl_size	3	avl_size
avl_wdata	72	avl_wdata
avl_rdata	72	avl_rdata
avl_rdata_valid		avl_rdata_valid
pnf_per_bit		pnf_per_bit
pnf_per_bit_persist		pnf_per_bit_persist
pass		pass
fail		fail
test_complete		test_complete

 For information about the Avalon signals and the Avalon interface, refer to [Avalon Interface Specifications](#).

## Traffic Generator Add-Ons

Some optional components that can be useful for verifying aspects of the controller and PHY operation are generated in conjunction with certain user-specified options. These add-on components are self-contained, and are not part of the controller or PHY, nor the traffic generator.

### User Refresh Generator

The user refresh generator sends refresh requests to the memory controller when user refresh is enabled. The memory controller returns an acknowledgement signal and then issues the refresh command to the memory device.

The user refresh generator is created when you turn on **Enable User Refresh** on the **Controller Settings** tab of the parameter editor.

## Traffic Generator Timeout Counter

The traffic generator timeout counter uses the Avalon interface clock.

When a test fails due to driver failure or timeout, the fail signal is asserted. When a test has failed, the traffic generator must be reset with the reset\_n signal.

## Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys

The traffic generator can be used in Qsys as a stand-alone component for use within a larger system. This section explains how to instantiate and configure the example driver, and includes tips on configuring a UniPHY memory interface which can apply in many general situations.



If you are using RLDRAM 3, you cannot use the example driver because the RLDRAM 3 IP is PHY-only, and does not include a controller.

### Creating the Qsys System

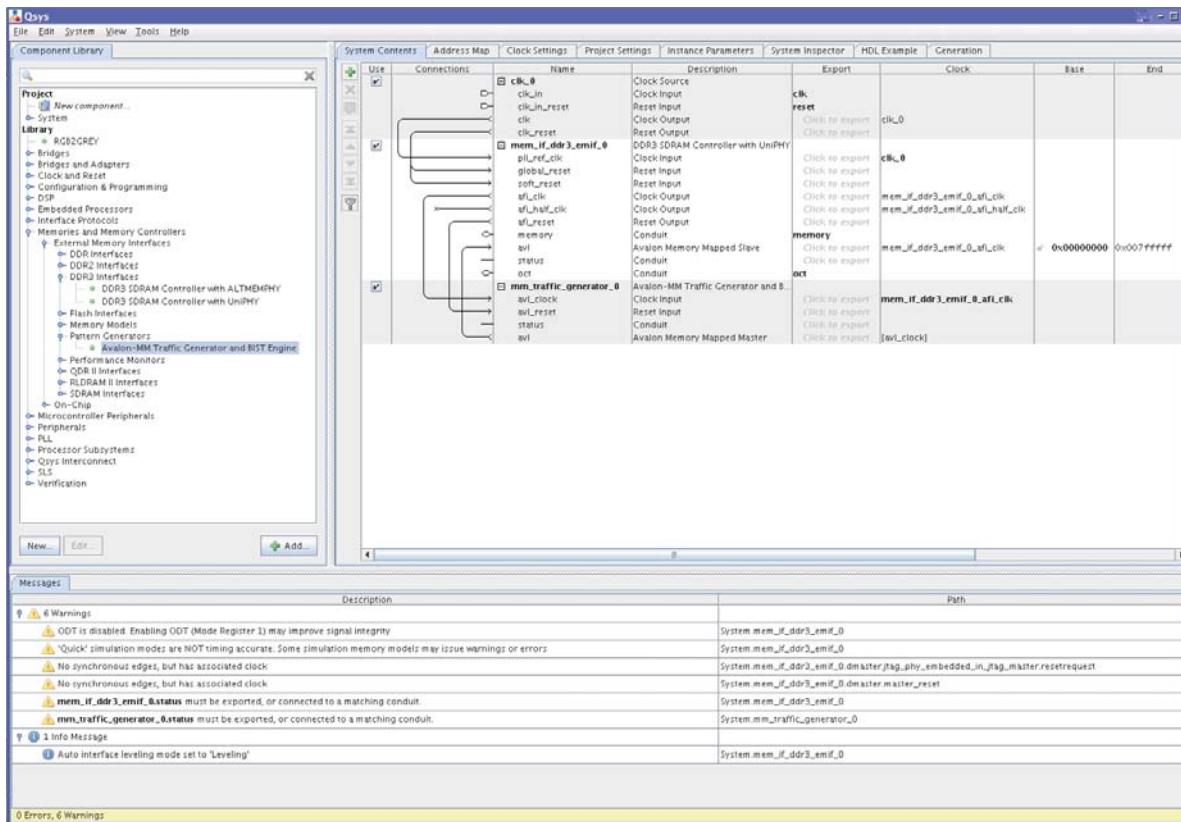
To create the system in Qsys, perform the following steps:

1. Start Qsys.
2. On the **Project Settings** tab, select the required device from the **Device Family** list.
3. In the **Component Library**, choose a UniPHY memory interface to instantiate. For example, under **Library > Memories and Memory Controllers > External Memory Interfaces**, select **DDR3 SDRAM Controller with UniPHY**.
4. Configure the parameters for your instantiation of the memory interface.
5. In the Component Library, find the example driver and instantiate it in the system. For example, under **Library > Memories and Memory Controllers > Pattern Generators**, select **Avalon-MM Traffic Generator and BIST Engine**.
6. Configure the parameters for your instantiation of the example driver.



The Avalon specification stipulates that Avalon-MM master interfaces issue byte addresses, while Avalon-MM slave interfaces accept word addresses. The default for the Avalon-MM Traffic Generator and BIST Engine is to issue word addresses. When using Qsys or SOPC Builder, you must enable the **Generate per byte address** setting in the traffic generator.

7. Connect the interfaces as illustrated in [Figure 9–8](#). At this point, you can generate synthesis RTL, Verilog or VHDL simulation RTL, or a simulation testbench system.

**Figure 9–8. Qsys System**

## Notes on Configuring UniPHY IP in Qsys

This section includes notes and tips on configuring the UniPHY IP in Qsys.

- The address ranges shown for the Avalon-MM slave interface on the UniPHY component should be interpreted as byte addresses that an Avalon-MM master would address, despite the fact that this range is modified by configuring the word addresses width of the Avalon-MM slave interface on the UniPHY controller.
- The `afi_clk` clock source is the associated clock to the Avalon-MM slave interface on the memory controller. This is the ideal clock source to use for all IP components connected on the same Avalon network. Using another clock would cause Qsys to automatically instantiate clock-crossing logic, potentially degrading performance.
- The `afi_clk` clock rate is determined by the **Rate on Avalon-MM interface** setting on the **UniPHY PHY Settings** tab. The `afi_half_clk` clock interface has a rate which is further halved. For example, if **Rate on Avalon-MM interface** is set to **Half**, the `afi_clk` rate is half of the memory clock frequency, and the `afi_half_clk` is one quarter of the memory clock frequency.

- The `global_reset` input interface can be used to reset the UniPHY memory interface and the PLL contained therein. The `soft_reset` input interface can be used to reset the UniPHY memory interface but allow the PLL to remain locked. You can use the `soft_reset` input to reset the memory but to maintain the AFI clock output to other components in the system.
- Do not connect a reset request from a system component (such as a Nios II processor) to the UniPHY `global_reset_n` port. Doing so would reset the UniPHY PLL, which would propagate as a reset condition on `afi_reset` back to the requester; the resulting reset loop could freeze the system.
- Qsys generates an interconnect fabric for each Avalon network. The interconnect fabric is capable of burst and width adaptation. If your UniPHY memory controller is configured with an Avalon interface data width which is wider than an Avalon-MM master interface connected to it, you must enable the byte enable signal on the Avalon-MM slave interface, by checking the **Enable Avalon-MM byte-enable signal** checkbox on the **Controller Settings** tab in the parameter editor.
- If you have a point-to-point connection from an Avalon-MM master to the Avalon-MM slave interface on the memory controller, and if the Avalon data width and burst length settings match, then the Avalon interface data widths may be multiples of either a power of two or nine. Otherwise, you must enable **Generate power-of-2 data bus widths for Qsys or SOPC Builder** on the **Controller Settings** tab of the parameter editor.

## Document Revision History

Table 9–2 lists the revision history for this document.

**Table 9–2. Document Revision History**

Date	Version	Changes
November 2012	1.3	<ul style="list-style-type: none"> <li>■ Added block diagrams of simulation and synthesis example designs for RLDRAM 3 and Ping Pong PHY.</li> <li>■ Changed chapter number from 7 to 9.</li> </ul>
June 2012	1.2	<ul style="list-style-type: none"> <li>■ Added Feedback icon.</li> </ul>
November 2011	1.1	<ul style="list-style-type: none"> <li>■ Added <b>Synthesis Example Design</b> and <b>Simulation Example Design</b> sections.</li> <li>■ Added <b>Creating and Connecting the UniPHY Memory Interface and the Traffic Generator in Qsys</b>.</li> <li>■ Revised Example Driver section as <b>Traffic Generator and BIST Engine</b>.</li> </ul>

This section provides reference information about the UniPHY-based external memory interface IP.

This section includes the following chapters:

- [Chapter 10, Introduction to UniPHY IP](#)
- [Chapter 11, Latency for UniPHY IP](#)
- [Chapter 12, Timing Diagrams for UniPHY IP](#)
- [Chapter 13, UniPHY External Memory Interface Debug Toolkit](#)
- [Chapter 14, Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers](#)



For information about the revision history for chapters in this section, refer to “Document Revision History” in each individual chapter.



The Altera<sup>®</sup> DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY, QDR II and QDR II+ SRAM controllers with UniPHY, RLDRAM II controller with UniPHY, and RLDRAM 3 PHY-only IP provide low latency, high-performance, feature-rich interfaces to industry-standard memory devices. The DDR2, QDR II and QDR II+, and RLDRAM II controllers with UniPHY offer full-rate and half-rate interfaces, while the DDR3 controller with UniPHY and the RLDRAM 3 PHY-only IP offer half-rate and quarter-rate interfaces, and the LPDDR2 controller with UniPHY offers a half-rate interface.

The UniPHY IP is an interface between a memory controller and memory devices and performs read and write operations to the memory. The UniPHY IP creates the datapath between the memory device and the memory controller and user logic in various Altera devices.

The MegaWizard™ interface generates an example top-level project, consisting of an example driver, and your controller custom variation. The controller instantiates an instance of the UniPHY datapath.

The example top-level project is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass, fail, and test-complete signals.



For device families not supported by the UniPHY-based designs, use the Altera ALTMEMPHY-based High Performance SDRAM Controller IP core.

If the UniPHY datapath does not match your requirements, you can create your own memory interface datapath using the ALTDLL, ALTDQ\_DQS, ALTDQ\_DQS2, ALTDQ, or ALTDQS megafunctions, available in the Quartus<sup>®</sup> II software, but you are then responsible for all aspects of the design including timing analysis and design constraints.

## Release Information

**Table 10–1** provides information about this release of the DDR2 and DDR3 SDRAM, QDR II and QDR II+ SRAM, and RLDRAM II controllers with UniPHY, and the RLDRAM 3 PHY-only IP.

**Table 10–1. Release Information**

Item	Protocol			
	DDR2, DDR3, LPDDR2	QDR II	RLDRAM II	RLDRAM 3
Version	12.1	12.1	12.1	12.1
Release Date	November 2012	November 2012	November 2012	November 2012
Ordering Code	IP-DDR2/UNI IP-DDR3/UNI IP-SDRAM/LPDDR2	IP-QDRII/UNI	IP-RDII/UNI	—

Altera verifies that the current version of the Quartus II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

## Device Family Support

**Table 10–2** defines the device support levels for Altera IP cores.

**Table 10–2. Altera IP Core Device Support Levels**

FPGA Device Families	HardCopy Device Families
<b>Preliminary support</b> —The IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.	<b>HardCopy Companion</b> —The IP core is verified with preliminary timing models for the HardCopy companion device. The IP core meets all functional requirements, but might still be undergoing timing analysis for the HardCopy device family. It can be used in production designs with caution.
<b>Final support</b> —The IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.	<b>HardCopy Compilation</b> —The IP core is verified with final timing models for the HardCopy device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

**Table 10–3** shows the level of support offered by each of the UniPHY-based external memory interface protocols for Altera device families.

**Table 10–3. Device Family Support (Part 1 of 2)**

Device Family	Support Level					
	DDR2	DDR3	LPDDR2	QDR II	RLDRAM II	RLDRAM 3
Arria® II GX	No support	No support	No support	Final	No support	No support
Arria II GZ	Final	Final	No support	Final	Final	No support
Arria V	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.					

**Table 10–3. Device Family Support (Part 2 of 2)**

Device Family	Support Level					
	DDR2	DDR3	LPDDR2	QDR II	RDRAM II	RDRAM 3
Arria V GZ	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		No support	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		
Cyclone V	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.			No support		
HardCopy® III	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		No support	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		No support
HardCopy IV	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		No support	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		No support
Stratix® III	Final	Final (Only $V_{CC} = 1.1V$ supported)	No support	Final	Final (Only $V_{CC} = 1.1V$ supported)	No support
Stratix IV	Final	Final	No support	Final	Final	No support
Stratix V	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		No support	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.		
Other device families	No support	No support	No support	No support	No support	No support

 For information about features and supported clock rates for external memory interfaces, refer to the [External Memory Specification Estimator](#).

## Features

Table 10–4 summarizes key feature support for Altera’s UniPHY-based external memory interfaces.

**Table 10–4. Feature Support (Part 1 of 2)**

Key Feature	Protocol					
	DDR2	DDR3	LPDDR2	QDR II	RDRAM II	RDRAM 3
High-performance controller II (HPC II)	✓	✓	✓	—	—	—
Half-rate core logic and user interface	✓	✓	✓	✓	✓	✓
Full-rate core logic and user interface	✓	—	—	✓	✓	—
Quarter-rate core logic and user interface	—	✓ <sup>(1)</sup>	—	—	—	✓
Dynamically generated Nios II-based sequencer	✓	✓	✓	✓	✓	✓
Choice of RTL-based or dynamically generated Nios® II-based sequencer	—	—	—	✓ <sup>(2) (3)</sup>	✓ 	—
Available Efficiency Monitor and Protocol Checker	✓	✓	✓	—	✓	✓
DDR3L support	—	✓ <sup>(4)</sup>	—	—	—	—
UDIMM and RDIMM in any form factor	✓	✓ <sup>(4) (5)</sup>	—	—	—	—

**Table 10-4. Feature Support (Part 2 of 2)**

Key Feature	Protocol					
	DDR2	DDR3	LPDDR2	QDR II	RLDRAM II	RLDRAM 3
Multiple components in a single-rank UDIMM or RDIMM layout	✓	✓	—	—	—	—
LRDIMM	—	✓	—	—	—	—
Burst length (half-rate)	8	—	8 or 16	4	4 or 8	2, 4, or 8
Burst length (full-rate)	4	—	—	2 or 4	2, 4, or 8	—
Burst length (quarter-rate)	—	8	—	—	—	2, 4, or 8
Burst length of 8 and burst chop of 4 (on the fly)	—	✓	—	—	—	—
With leveling	✓ (240 MHz and above) (10)	✓ (9) (10)	—	—	—	✓
Without leveling	✓ (below 240 MHz)	—	✓	—	—	—
Maximum data width	144 bits (6)	144 bits (6)	32 bits	72 bits	72 bits	72 bits
Reduced controller latency	—	—	—	✓ (2) (7)	✓ (2) (7)	—
Read latency	—	—	—	1.5 (QDR II) 2 or 2.5 (QDR II+)	—	—
ODT (in memory device)	—	✓	—	✓ (QDR II + only)	✓	✓
x36 emulation mode	—	—	—	✓ (8) (11)	—	—

**Notes:**

- (1) For Arria V GZ and Stratix V devices only.
- (2) Not available in Arria II GX devices.
- (3) Nios II-based sequencer not available for full-rate interfaces.
- (4) For DDR3, the DIMM form is not supported in Arria II GX, Arria II GZ, Arria V, or Cyclone V devices.
- (5) Arria II GZ uses leveling logic for discrete devices in DDR3 interfaces to achieve high speeds, but that leveling cannot be used to implement the DIMM form in DDR3 interfaces.
- (6) For any interface with data width above 72 bits, you must use Quartus II software timing analysis of your complete design to determine the maximum clock rate.
- (7) The maximum achievable clock rate when reduced controller latency is selected must be attained through Quatrus II software timing analysis of your complete design.
- (8) Emulation mode allows emulation of a larger memory-width interface using multiple smaller memory-width interfaces. For example, an x36 QDR II or QDR II+ interface can be emulated using two x18 interfaces.
- (9) The leveling delay on the board between first and last DDR3 SDRAM component laid out as a DIMM must be less than 0.69  $t_{CK}$ .
- (10) Leveling is not available for Arria V or Cyclone V devices.
- (11) x36 emulation mode is not supported in Arria V, Arria V GZ, Cyclone V, or Stratix V devices.

## Unsupported Features

Table 10–4 summarizes key feature support. Device, protocol, and architecture support is summarized in the [Protocol Support Matrix](#) in chapter 1 of volume 1 of this handbook.

## System Requirements

The DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY, QDR II and QDR II+ SRAM controllers with UniPHY, RLDRAM II controller with UniPHY, and RLDRAM 3 PHY-only IP are part of the MegaCore IP Library, which Altera distributes with the Quartus II software.



For system requirements and installation instructions, refer to [Altera Software Installation and Licensing](#).

## MegaCore Verification

Altera has carried out extensive random, directed tests with functional test coverage using industry-standard models to ensure the functionality of the external memory controllers with UniPHY. Altera's functional verification of the external memory controllers with UniPHY use modified Denali models, with certain assertions disabled.

## Resource Utilization

This section lists resource utilization information for the external memory controllers with UniPHY for supported device families.

### DDR2, DDR3, and LPDDR2 SDRAM Controllers with UniPHY

Table 10–5 shows typical resource usage of the DDR2, DDR3, and LPDDR2 SDRAM controllers with UniPHY in the current version of Quartus II software for Arria V devices.

**Table 10–5. Resource Utilization in Arria V Devices (Part 1 of 2)**

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	M10K Blocks	Memory (Bits)	Hard Memory Controller
<b>Controller</b>						
DDR2 (Half rate)	8	2286	1404	4	6560	0
	64	2304	1379	17	51360	0
DDR2 (Fullrate)	32	0	0	0	0	1
	8	2355	1412	4	6560	0
DDR3 (Half rate)	64	2372	1440	17	51360	0
	32	0	0	0	0	1

**Table 10–5. Resource Utilization in Arria V Devices (Part 2 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational ALUTS</b>	<b>Logic Registers</b>	<b>M10K Blocks</b>	<b>Memory (Bits)</b>	<b>Hard Memory Controller</b>
LPDDR2 (Half rate)	8	2230	1617	4	6560	0
	32	2239	1600	10	25760	0
<b>PHY</b>						
DDR2 (Half rate)	8	1652	2015	34	141312	0
	64	1819	2089	34	174080	0
DDR2 (Fullrate)	32	1222	1415	34	157696	1
DDR3 (Half rate)	8	1653	1977	34	141312	0
	64	1822	2090	34	174080	0
DDR3 (Full rate)	32	1220	1428	34	157696	0
LPDDR2 (Half rate)	8	2998	3187	35	150016	0
	32	3289	3306	35	174592	0
<b>Total</b>						
DDR2 (Half rate)	8	4555	3959	39	148384	0
	64	4991	4002	52	225952	0
DDR2 (Fullrate)	32	1776	1890	35	158208	1
DDR3 (Half rate)	8	4640	3934	39	148384	0
	64	5078	4072	52	225952	0
DDR3 (Full rate)	32	1774	1917	35	158208	1
LPDDR2 (Half rate)	8	5228	4804	39	156576	0
	32	5528	4906	45	200352	0

Table 10–6 shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus II software for Arria II GZ devices.

**Table 10–6. Resource Utilization in Arria II GZ Devices (Part 1 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational ALUTS</b>	<b>Logic Registers</b>	<b>Mem ALUTs</b>	<b>M9K Blocks</b>	<b>M144K Blocks</b>	<b>Memory (Bits)</b>
<b>Controller</b>							
DDR2 (Half rate)	8	1,781	1,092	10	2	0	4,352
	16	1,784	1,092	10	4	0	8,704
	64	1,818	1,108	10	15	0	34,560
	72	1,872	1,092	10	17	0	39,168

**Table 10–6. Resource Utilization in Arria II GZ Devices (Part 2 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational ALUTS</b>	<b>Logic Registers</b>	<b>Mem ALUTs</b>	<b>M9K Blocks</b>	<b>M144K Blocks</b>	<b>Memory (Bits)</b>
DDR2 (Full rate)	8	1,851	1,124	10	2	0	2,176
	16	1,847	1,124	10	2	0	4,352
	64	1,848	1,124	10	8	0	17,408
	72	1,852	1,124	10	9	0	19,574
DDR3 (Half rate)	8	1,869	1,115	10	2	0	4,352
	16	1,868	1,115	10	4	0	8,704
	64	1,882	1,131	10	15	0	34,560
	72	1,888	1,115	10	17	0	39,168
<b>PHY</b>							
DDR2 (Half rate)	8	2,560	2,042	183	22	0	157,696
	16	2,730	2,262	183	22	0	157,696
	64	3,606	3,581	183	22	0	157,696
	72	3,743	3,796	183	22	0	157,696
DDR2 (Full rate)	8	2,494	1,934	169	22	0	157,696
	16	2,652	2,149	169	22	0	157,696
	64	3,519	3,428	169	22	0	157,696
	72	3,646	3,642	169	22	0	157,696
DDR3 (Half rate)	8	2,555	2,032	187	22	0	157,696
	16	3,731	2,251	187	22	0	157,696
	64	3,607	3,572	187	22	0	157,696
	72	3,749	3,788	187	22	0	157,696
<b>Total</b>							
DDR2 (Half rate)	8	4,341	3,134	193	24	0	4,374
	16	4,514	3,354	193	26	0	166,400
	64	5,424	4,689	193	37	0	192,256
	72	5,615	4,888	193	39	0	196,864
DDR2 (Full rate)	8	4,345	3,058	179	24	0	159,872
	16	4,499	3,273	179	24	0	162,048
	64	5,367	4,552	179	30	0	175,104
	72	5,498	4,766	179	31	0	177,280
DDR3 (Half rate)	8	4,424	3,147	197	24	0	162,048
	16	5,599	3,366	197	26	0	166,400
	64	5,489	4,703	197	37	0	192,256
	72	5,637	4,903	197	39	0	196,864

Table 10–7 shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus II software for Stratix III devices.

**Table 10–7. Resource Utilization in Stratix III Devices (Part 1 of 2)**

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
<b>Controller</b>							
DDR2 (Half rate)	8	1,807	1,058	0	4	0	4,464
	16	1,809	1,058	0	6	0	8,816
	64	1,810	1,272	10	14	0	32,256
	72	1,842	1,090	10	17	0	39,168
DDR2 (Full rate)	8	1,856	1,093	0	4	0	2,288
	16	1,855	1,092	0	4	0	4,464
	64	1,841	1,092	0	10	0	17,520
	72	1,834	1,092	0	11	0	19,696
DDR3 (Half rate)	8	1,861	1,083	0	4	0	4,464
	16	1,863	1,083	0	6	0	8,816
	64	1,878	1,295	10	14	0	32,256
	72	1,895	1,115	10	17	0	39,168
<b>PHY</b>							
DDR2 (Half rate)	8	2,591	2,100	218	6	1	157,696
	16	2,762	2,320	218	6	1	157,696
	64	3,672	3,658	242	6	1	157,696
	72	3,814	3,877	242	6	1	157,696
DDR2 (Full rate)	8	2,510	1,986	200	6	1	157,696
	16	2,666	2,200	200	6	1	157,696
	64	3,571	3,504	224	6	1	157,696
	72	3,731	3,715	224	6	1	157,696
DDR3 (Half rate)	8	2,591	2,094	224	6	1	157,696
	16	2,765	2,314	224	6	1	157,696
	64	3,680	3,653	248	6	1	157,696
	72	3,819	3,871	248	6	1	157,696
<b>Total</b>							
DDR2 (Half rate)	8	4,398	3,158	218	10	1	162,160
	16	4,571	3,378	218	12	1	166,512
	64	5,482	4,930	252	20	1	189,952
	72	5,656	4,967	252	23	1	196,864
DDR2 (Full rate)	8	4,366	3,079	200	10	1	159,984
	16	4,521	3,292	200	10	1	162,160
	64	5,412	4,596	224	16	1	175,216
	72	5,565	4,807	224	17	1	177,392

**Table 10–7. Resource Utilization in Stratix III Devices (Part 2 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational ALUTS</b>	<b>Logic Registers</b>	<b>Mem ALUTs</b>	<b>M9K Blocks</b>	<b>M144K Blocks</b>	<b>Memory (Bits)</b>
DDR3 (Half rate)	8	4,452	3,177	224	10	1	162,160
	16	4,628	3,397	224	12	1	166,512
	64	5,558	4,948	258	20	1	189,952
	72	5,714	4,986	258	23	1	196,864

Table 10–8 shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus II software for Stratix IV devices.

**Table 10–8. Resource Utilization in Stratix IV Devices (Part 1 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational ALUTS</b>	<b>Logic Registers</b>	<b>Mem ALUTs</b>	<b>M9K Blocks</b>	<b>M144K Blocks</b>	<b>Memory (Bits)</b>
<b>Controller</b>							
DDR2 (Half rate)	8	1,785	1,090	10	2	0	4,352
	16	1,785	1,090	10	4	0	8,704
	64	1,796	1,106	10	15	0	34,560
	72	1,798	1,090	10	17	0	39,168
DDR2 (Full rate)	8	1,843	1,124	10	2	0	2,176
	16	1,845	1,124	10	2	0	4,352
	64	1,832	1,124	10	8	0	17,408
	72	1,834	1,124	10	9	0	19,584
DDR3 (Half rate)	8	1,862	1,115	10	2	0	4,352
	16	1,874	1,115	10	4	0	8,704
	64	1,880	1,131	10	15	0	34,560
	72	1,886	1,115	10	17	0	39,168
<b>PHY</b>							
DDR2 (Half rate)	8	2,558	2,041	183	6	1	157,696
	16	2,728	2,262	183	6	1	157,696
	64	3,606	3,581	183	6	1	157,696
	72	3,748	3,800	183	6	1	157,696
DDR2 (Full rate)	8	2,492	1,934	169	6	1	157,696
	16	2,652	2,148	169	6	1	157,696
	64	3,522	3,428	169	6	1	157,696
	72	3,646	3,641	169	6	1	157,696
DDR3 (Half rate)	8	2,575	2,031	187	6	1	157,696
	16	2,732	2,251	187	6	1	157,696
	64	3,602	3,568	187	6	1	157,696
	72	3,750	3,791	187	6	1	157,696
<b>Total</b>							

**Table 10–8. Resource Utilization in Stratix IV Devices (Part 2 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational ALUTs</b>	<b>Logic Registers</b>	<b>Mem ALUTs</b>	<b>M9K Blocks</b>	<b>M144K Blocks</b>	<b>Memory (Bits)</b>
DDR2 (Half rate)	8	4,343	3,131	193	8	1	162,048
	16	4,513	3,352	193	10	1	166,400
	64	5,402	4,687	193	21	1	192,256
	72	5,546	4,890	193	23	1	196,864
DDR2 (Full rate)	8	4,335	3,058	179	8	1	159,872
	16	4,497	3,272	179	8	1	162,048
	64	5,354	4,552	179	14	1	175,104
	72	5,480	4,765	179	15	1	177,280
DDR3 (Half rate)	8	4,437	3,146	197	8	1	162,048
	16	4,606	3,366	197	10	1	166,400
	64	5,482	4,699	197	21	1	192,256
	72	5,636	4,906	197	23	1	196,864

Table 10–9 shows typical resource usage of the DDR2 and DDR3 SDRAM controllers with UniPHY in the current version of Quartus II software for Arria V GZ and Stratix V devices.

**Table 10–9. Resource Utilization in Arria V GZ and Stratix V Devices (Part 1 of 2)**

<b>Protocol</b>	<b>Memory Width (Bits)</b>	<b>Combinational LCs</b>	<b>Logic Registers</b>	<b>M20K Blocks</b>	<b>Memory (Bits)</b>
<b>Controller</b>					
DDR2 (Half rate)	8	1,787	1,064	2	4,352
	16	1,794	1,064	4	8,704
	64	1,830	1,070	14	34,304
	72	1,828	1,076	15	38,400
DDR2 (Full rate)	8	2,099	1,290	2	2,176
	16	2,099	1,290	2	4,352
	64	2,126	1,296	7	16,896
	72	2,117	1,296	8	19,456
DDR3 (Quarter rate)	8	2,101	1,370	4	8,704
	16	2,123	1,440	7	16,896
	64	2,236	1,885	28	69,632
	72	2,102	1,870	30	74,880
DDR3 (Half rate)	8	1,849	1,104	2	4,352
	16	1,851	1,104	4	8,704
	64	1,853	1,112	14	34,304
	72	1,889	1,116	15	38,400

**Table 10–9. Resource Utilization in Arria V GZ and Stratix V Devices (Part 2 of 2)**

Protocol	Memory Width (Bits)	Combinational LCs	Logic Registers	M20K Blocks	Memory (Bits)
<b>PHY</b>					
DDR2 (Half rate)	8	2,567	1,757	13	157,696
	16	2,688	1,809	13	157,696
	64	3,273	2,115	13	157,696
	72	3,377	2,166	13	157,696
DDR2 (Full rate)	8	2,491	1,695	13	157,696
	16	2,578	1,759	13	157,696
	64	3,062	2,137	13	157,696
	72	3,114	2,200	13	157,696
DDR3 (Quarter rate)	8	2,209	2,918	18	149,504
	16	2,355	3,327	18	157,696
	64	3,358	5,228	18	182,272
	72	4,016	6,318	18	198,656
DDR3 (Half rate)	8	2,573	1,791	13	157,696
	16	2,691	1,843	13	157,696
	64	3,284	2,149	13	157,696
	72	3,378	2,200	13	157,696
<b>Total</b>					
DDR2 (Half rate)	8	4,354	2,821	15	162,048
	16	4,482	2,873	17	166,400
	64	5,103	3,185	27	192,000
	72	5,205	3,242	28	196,096
DDR2 (Full rate)	8	4,590	2,985	15	159,872
	16	4,677	3,049	15	162,048
	64	5,188	3,433	20	174,592
	72	5,231	3,496	21	177,152
DDR3 (Quarter rate)	8	4,897	4,844	23	158,720
	16	5,065	5,318	26	175,104
	64	6,183	7,669	47	252,416
	72	6,705	8,744	49	274,048
DDR3 (Half rate)	8	4,422	2,895	15	162,048
	16	4,542	2,947	17	166,400
	64	5,137	3,261	27	192,000
	72	5,267	3,316	28	196,096

## QDR II and QDR II+ SRAM Controllers with UniPHY

**Table 10–10** shows typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus II software for Arria V devices.

<sup>8</sup> **Table 10–10. Resource Utilization in Arria V Devices**

PHY Rate	Memory Width (Bits)	Combinational ALUTs	Logic Registers	M10K Blocks	Memory (Bits)	Hard Memory Controller
<b>Controller</b>						
Half	9	98	120	0	0	0
	18	96	156	0	0	0
	36	94	224	0	0	0
<b>PHY</b>						
Half	9	234	257	0	0	0
	18	328	370	0	0	0
	36	522	579	0	0	0
<b>Total</b>						
Half	9	416	377	0	0	0
	18	542	526	0	0	0
	36	804	803	0	0	0

**Table 10–11** shows typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus II software for Arria II GX devices.

**Table 10–11. Resource Utilization in Arria II GX Devices**

PHY Rate	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Memory (Bits)	M9K Blocks
Half	9	620	701	0	0
	18	921	1122	0	0
	36	1534	1964	0	0
Full	9	584	708	0	0
	18	850	1126	0	0
	36	1387	1962	0	0

Table 10–12 shows typical resource usage of the QDR II and QDR II+ SRAM controllers with UniPHY in the current version of Quartus II software for Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V devices.

**Table 10–12. Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices**

PHY Rate	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Memory (Bits)	M9K Blocks
Half	9	602	641	0	0
	18	883	1002	0	0
	36	1457	1724	0	0
Full	9	586	708	0	0
	18	851	1126	0	0
	36	1392	1962	0	0

## RLDRAM II Controller with UniPHY

Table 10–13 shows typical resource usage of the RLDARAM II controller with UniPHY in the current version of Quartus II software for Arria V devices.

**Table 10–13. Resource Utilization in Arria V Devices**

PHY Rate	Memory Width (Bits)	Combinational ALUTs	Logic Registers	M10K Blocks	Memory (Bits)	Hard Memory Controller
<b>Controller</b>						
Half	9	353	303	1	288	0
	18	350	324	2	576	0
	36	350	402	4	1152	0
<b>PHY</b>						
Half	9	295	474	0	0	0
	18	428	719	0	0	0
	36	681	1229	0	0	0
<b>Total</b>						
Half	9	705	777	1	288	0
	18	871	1043	2	576	0
	36	1198	1631	4	1152	0

Table 10–14 shows typical resource usage of the RLDRAM II controller with UniPHY in the current version of Quartus II software for Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V devices.

**Table 10–14. Resource Utilization in Arria II GZ, Arria V GZ, Stratix III, Stratix IV, and Stratix V Devices** [\(1\)](#)

PHY Rate	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Memory (Bits)	M9K Blocks
Half	9	829	763	288	1
	18	1145	1147	576	2
	36	1713	1861	1152	4
Full	9	892	839	288	1
	18	1182	1197	576	1
	36	1678	1874	1152	2

**Note to Table 10–14:**

- (1) Half-rate designs use the same amount of memory as full-rate designs, but the data is organized in a different way (half the width, double the depth) and the design may need more M9K resources.

## Document Revision History

Table 10–15 lists the revision history for this document.

**Table 10–15. Document Revision History**

Date	Version	Changes
November 2012	2.1	<ul style="list-style-type: none"> <li>■ Added RLDRAM 3 support</li> <li>■ Added LRDIMM support</li> <li>■ Added Arria V GZ support</li> <li>■ Changed chapter number from 8 to 10.</li> </ul>
June 2012	2.0	<ul style="list-style-type: none"> <li>■ Added LPDDR2 support.</li> <li>■ Moved <a href="#">Protocol Support Matrix</a> to Volume 1.</li> <li>■ Added Feedback icon.</li> </ul>
November 2011	1.1	<ul style="list-style-type: none"> <li>■ Combined <a href="#">Release Information</a>, <a href="#">Device Family Support</a>, <a href="#">Features</a> list, and <a href="#">Unsupported Features</a> list for DDR2, DDR3, QDR II, and RLDRAM II.</li> <li>■ Added Protocol Support Matrix.</li> <li>■ Combined <a href="#">Resource Utilization</a> information for DDR2, DDR3, QDR II, and RLDRAM II. Updated data for 11.1.</li> </ul>

Altera defines read and write latencies in terms of memory clock cycles. There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.

Latency of the memory interface depends on its configuration and traffic patterns, therefore you should simulate your system to determine precise latency values. The numbers presented in this chapter are typical values meant only as guidelines.

Latency found in simulation may differ from latency found on the board, because functional simulation does not consider board trace delays and differences in process, voltage, and temperature. For a given design on a given board, the latency found may differ by one clock cycle (for full-rate designs), or two clock cycles (for quarter-rate or half-rate designs) upon resetting the board. The same design can yield different latencies on different boards.



For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.



## DDR2, DDR3, and LPDDR2

Table 11–1 shows the DDR2 SDRAM latency in full-rate memory clock cycles.

**Table 11–1. DDR2 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) <sup>(1)</sup> <sup>(2)</sup>**

Latency in Full-Rate Memory Clock Cycles							
Rate	Controller Address & Command	PHY Address & Command	Memory Maximum Read	PHY Read Return	Controller Read Return	Round Trip	Round Trip Without Memory
Half	10	EWL: 3	3–7	6	4	EWL: 26–30	EWL: 23
		OWL: 4				OWL: 27–31	OWL: 24
Full	5	0	3–7	4	10	22–26	19

**Notes:**

- (1) EWL = Even write latency
- (2) OWL = Odd write latency

Table 11–2 shows the DDR3 SDRAM latency in full-rate memory clock cycles.

**Table 11–2. DDR3 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) <sup>(1)</sup> <sup>(2)</sup> <sup>(3)</sup> <sup>(4)</sup>**

Latency in Full-Rate Memory Clock Cycles							
Rate	Controller Address & Command	PHY Address & Command	Memory Maximum Read	PHY Read Return	Controller Read Return	Round Trip	Round Trip Without Memory
Quarter	20	EWER: 8	5–11	EWER: 16	8	EWER: 57–63	EWER: 52
		EWOR: 8		EWOR: 17		EWOR: 58–64	EWOR: 53
		OWER: 11		OWER: 17		OWER: 61–67	OWER: 56
		OWOR: 11		OWOR: 14		OWOR: 58–64	OWOR: 53
Half	10	EWER: 3	5–11	EWER: 7	4	EWER: 29–35	EWER: 24
		EWOR: 3		EWOR: 6		EWOR: 28–34	EWOR: 23
		OWER: 4		OWER: 6		OWER: 29–35	OWER: 24
		OWOR: 4		OWOR: 7		OWOR: 30–36	OWOR: 25
Full	5	0	5–11	4	10	24–30	19

**Notes:**

- (1) EWER = Even write latency and even read latency
- (2) EWOR = Even write latency and odd read latency
- (3) OWER = Odd write latency and even read latency
- (4) OWOR = Odd write latency and odd read latency

Table 11–3 shows the LPDDR2 SDRAM latency in full-rate memory clock cycles.

**Table 11–3. LPDDR2 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles)** [\(1\)](#) [\(2\)](#) [\(3\)](#) [\(4\)](#)

Latency in Full-Rate Memory Clock Cycles							
Rate	Controller Address & Command	PHY Address & Command	Memory Maximum Read	PHY Read Return	Controller Read Return	Round Trip	Round Trip Without Memory
Half	10	EWER: 3	5–11	EWER: 7	4	EWER: 29–35	EWER: 24
		EWOR: 3		EWOR: 6		EWOR: 28–34	EWOR: 23
		OWER: 4		OWER: 6		OWER: 29–35	OWER: 24
		OWOR: 4		OWOR: 7		OWOR: 30–36	OWOR: 25
Full	5	0	5–11	4	10	24–30	19

**Notes:**

- (1) EWER = Even write latency and even read latency
- (2) EWOR = Even write latency and odd read latency
- (3) OWER = Odd write latency and even read latency
- (4) OWOR = Odd write latency and odd read latency

## QDR II and QDR II+

Table 11–4 shows the latency in full-rate memory clock cycles.

**Table 11–4. QDR II Latency (In Full-Rate Memory Clock Cycles)** [\(1\)](#)

Latency in Full-Rate Memory Clock Cycles							
Rate	Controller Address & Command	PHY Address & Command	Memory Maximum Read	PHY Read Return	Controller Read Return	Round Trip	Round Trip Without Memory
Half	2	1	1.5, 2.0, 2.5	RL 1.5: 5.5	0	RL 1.5: 10	RL 1.5: 8.5
				RL 2.0: 5.0		RL 2.0: 10	RL 2.0: 8
				RL 2.5 :4.5		RL 2.5: 10	RL 2.5: 7.5
Full	1	1	1.5, 2.0, 2.5	RL 1.5: 4.5	0	RL 1.5: 8	RL 1.5: 6.5
				RL 2.0: 4.0		RL 2.0: 8	RL 2.0: 6.0
				RL 2.5 :4.5		RL 2.5: 9	RL 2.5: 6.5

**Note:**

- (1) RL = Read latency

## RLDRAM II

Table 11–5 shows the latency in full-rate memory clock cycles.

**Table 11–5. RLDRAm II Latency (In Full-Rate Memory Clock Cycles) (1)(2)**

Latency in Full-Rate Memory Clock Cycles							
Rate	Controller Address & Command	PHY Address & Command	Memory Maximum Read	PHY Read Return	Controller Read Return	Round Trip	Round Trip Without Memory
Half	4	EWL: 1	3–8	EWL: 4	0	EWL: 12–17	EWL: 9
		OWL: 2		OWL: 4		OWL: 13–18	OWL: 10
Full	2	1	3–8	4	0	10–15	7

**Notes:**

- (1) EWL = Even write latency
- (2) OWL = Odd write latency

## RLDRAM 3

Table 11–6 shows the latency in full-rate memory clock cycles.

**Table 11–6. RLDRAm 3 Latency (In Full-Rate Memory Clock Cycles)**

Latency in Full-Rate Memory Clock Cycles						
Rate	PHY Address & Command	Memory Maximum Read	PHY Read Return	Controller Read Return	Round Trip	Round Trip Without Memory
Quarter	7	3–16	18	0	28–41	25
Half	4	3–16	6	0	13–26	10

## Variable Controller Latency

The variable controller latency feature allows you to take advantage of lower latency for variations designed to run at lower frequency. When deciding whether to vary the controller latency from the default value of 1, be aware of the following considerations:

- Reduced latency can help achieve a reduction in resource usage and clock cycles in the controller, but might result in lower  $f_{MAX}$ .
- Increased latency can help achieve greater  $f_{MAX}$ , but might consume more clock cycles in the controller and result in increased resource usage.

If you select a latency value that is inappropriate for the target frequency, the system displays a warning message in the text area at the bottom of the parameter editor.

You can change the controller latency by altering the value of the **Controller Latency** setting in the **Controller Settings** section of the **General Settings** tab of the QDR II and QDR II+ SRAM controller with UniPHY parameter editor.

## Document Revision History

Table 11-7 lists the revision history for this document.

**Table 11-7. Document Revision History**

Date	Version	Changes
November 2012	2.1	<ul style="list-style-type: none"><li>■ Added latency information for RLDRAM 3.</li><li>■ Changed chapter number from 9 to 11.</li></ul>
June 2012	2.0	<ul style="list-style-type: none"><li>■ Added latency information for LPDDR2.</li><li>■ Added Feedback icon.</li></ul>
November 2011	1.0	<ul style="list-style-type: none"><li>■ Consolidated latency information from <b>11.0 DDR2 and DDR3 SDRAM Controller with UniPHY User Guide</b>, <b>QDR II and QDR II+ SRAM Controller with UniPHY User Guide</b>, and <b>RLDRAM II Controller with UniPHY IP User Guide</b>.</li><li>■ Updated data for 11.1.</li></ul>



This chapter contains timing diagrams for the UniPHY-based external memory interface IP.

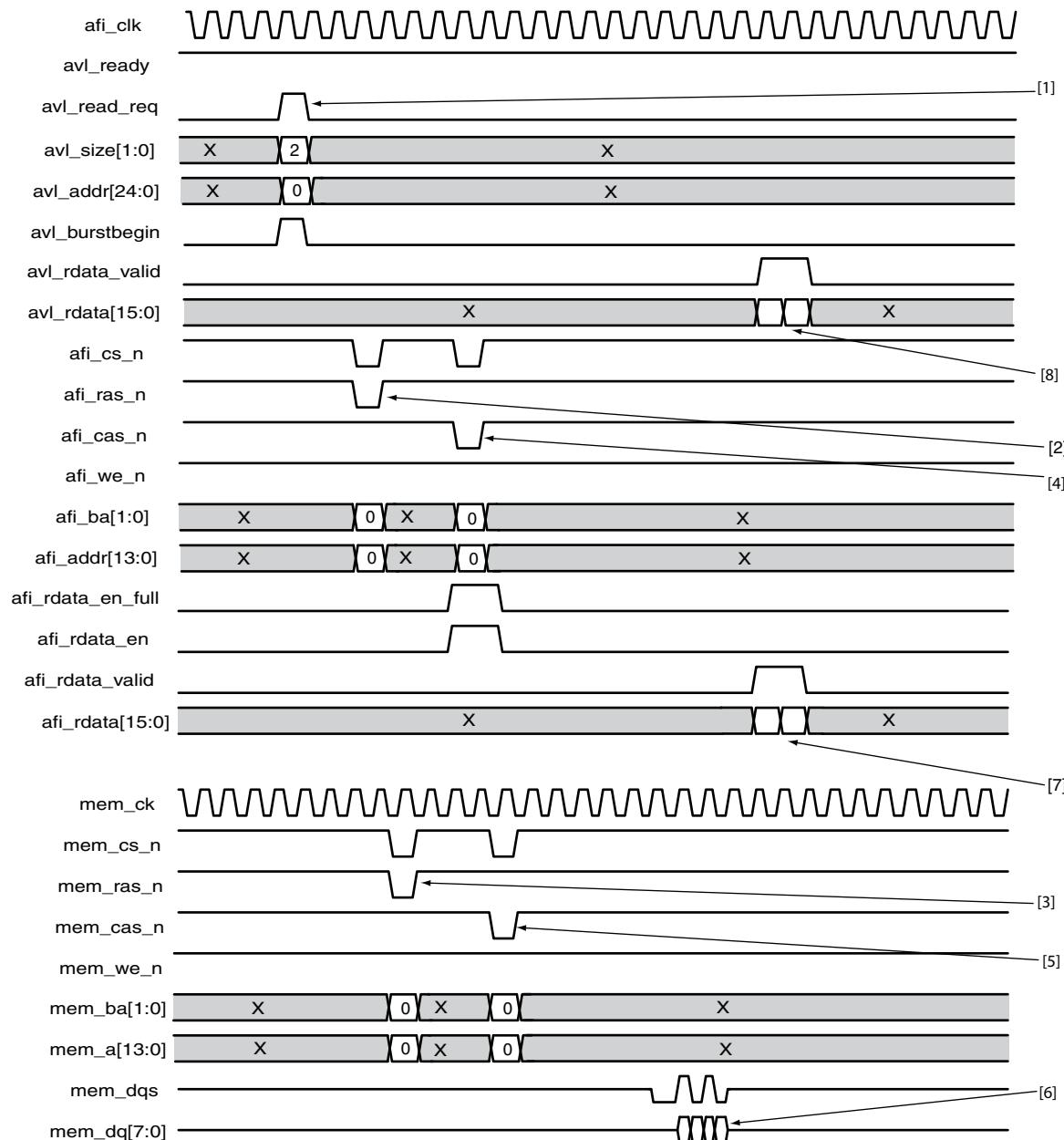
## DDR2 and DDR3 Timing Diagrams

This section contains timing diagrams for DDR2 and DDR3 protocols.

**Figure 12–1 through Figure 12–8** present the following timing diagrams, based on a Stratix III device:

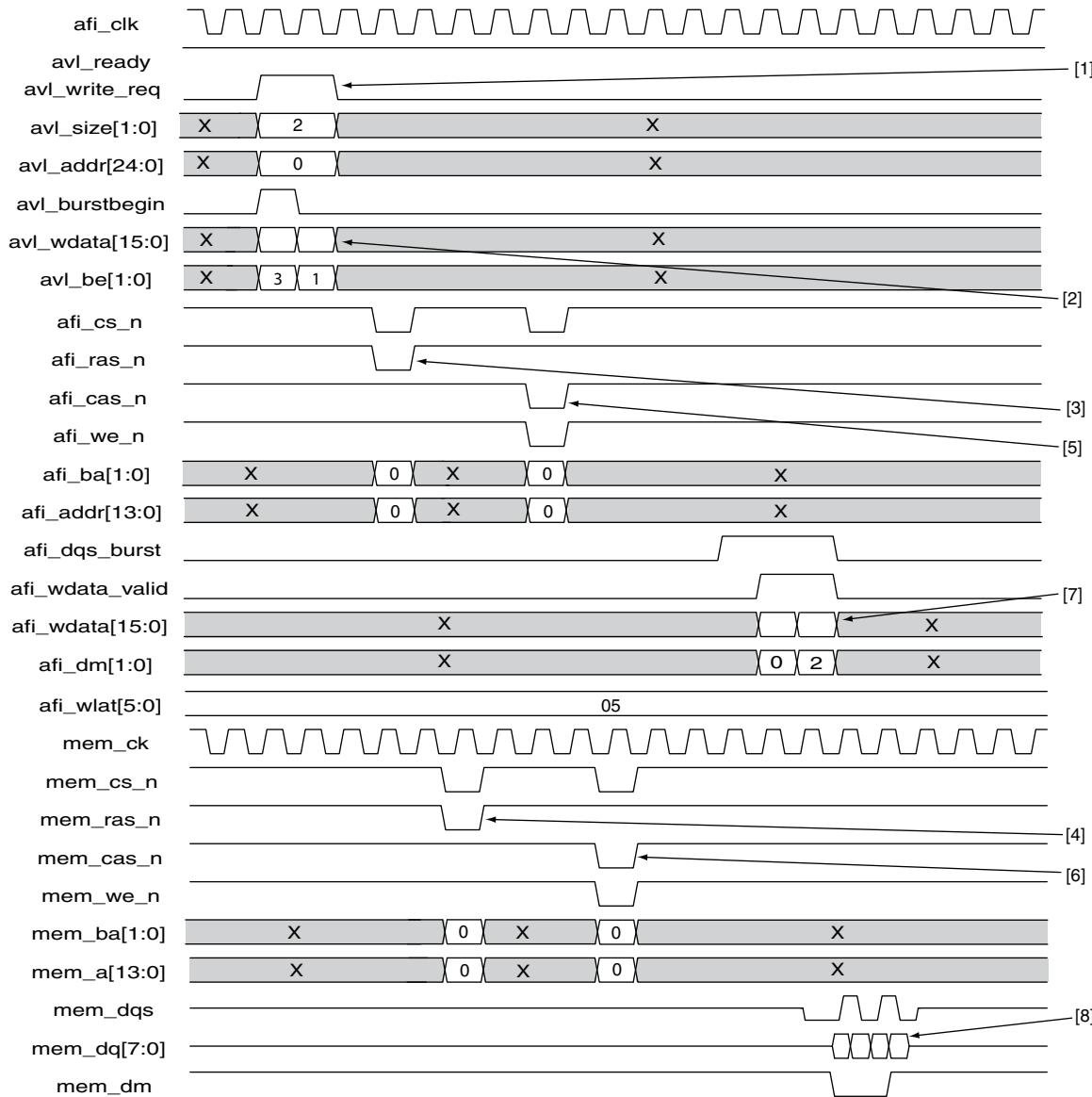
- Full-Rate DDR2 SDRAM Read
- Full-Rate DDR2 SDRAM Write
- Half-Rate DDR2 SDRAM Read
- Half-Rate DDR2 SDRAM Write
- Half-Rate DDR3 SDRAM Read
- Half-Rate DDR3 SDRAM Writes
- Quarter-Rate DDR3 SDRAM Reads
- Quarter-Rate DDR3 SDRAM Writes



**Figure 12-1. Full-Rate DDR2 SDRAM Read****Notes for Figure 12-1**

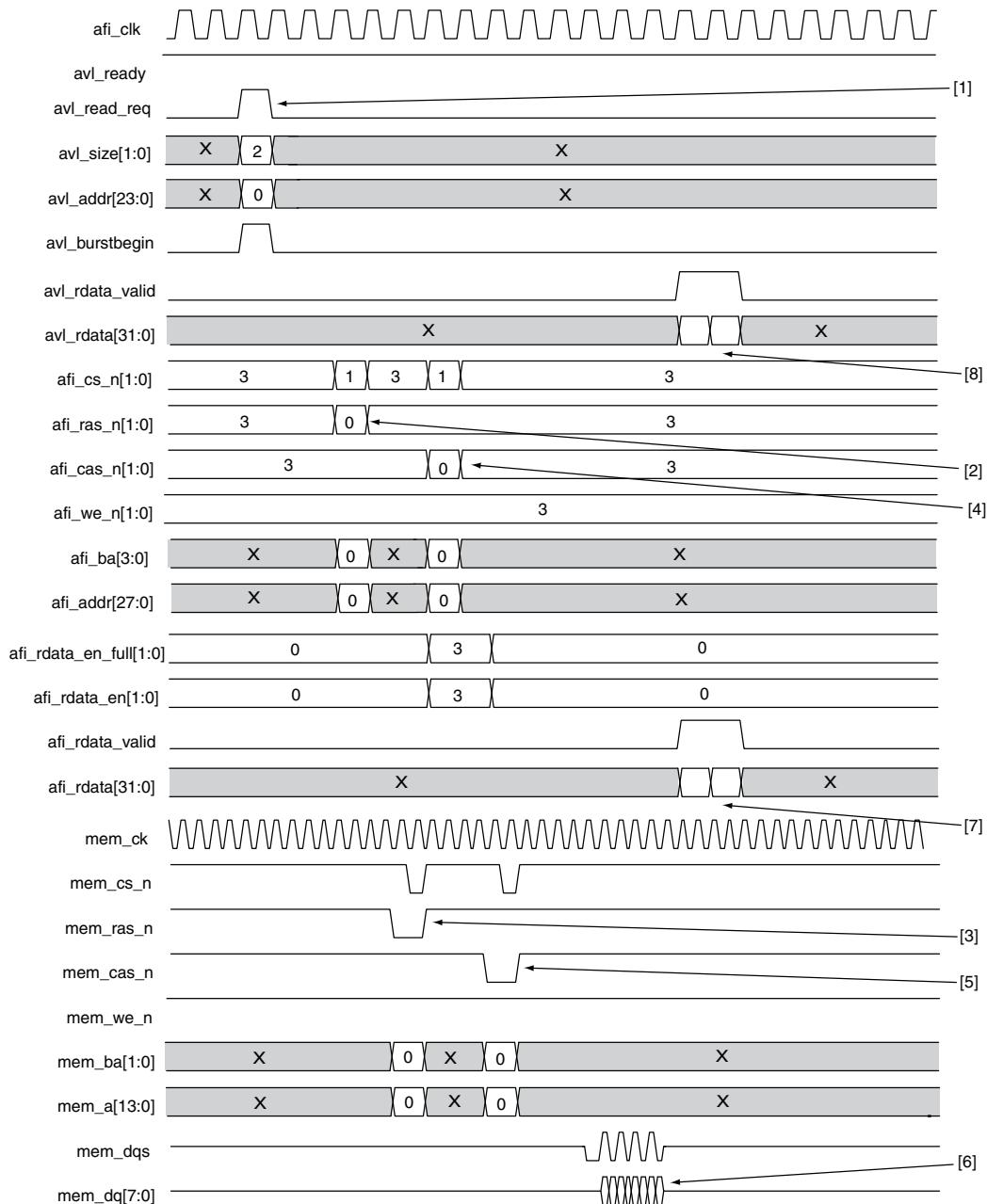
- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.
- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory.
- (7) Controller receives read data from PHY.
- (8) User logic receives read data from controller.

**Figure 12-2. Full-Rate DDR2 SDRAM Write**



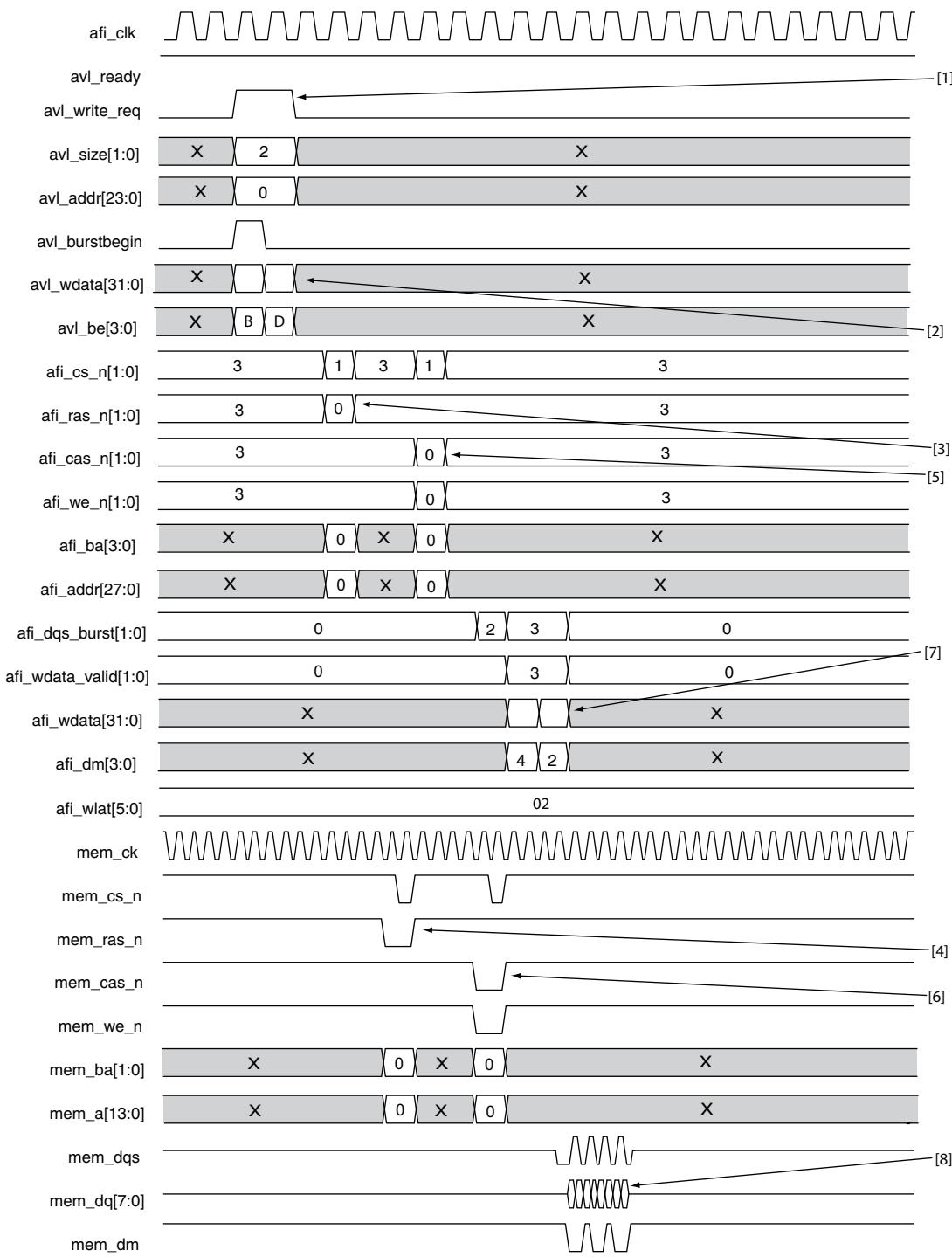
**Notes for Figure 12-2:**

- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues activate command to PHY.
- (4) PHY issues activate command to memory.
- (5) Controller issues write command to PHY.
- (6) PHY issues write command to memory.
- (7) Controller sends write data to PHY.
- (8) PHY sends write data to memory.

**Figure 12–3. Half-Rate DDR2 SDRAM Read****Notes for Figure 12–3:**

- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.
- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory.
- (7) Controller receives read data from PHY.
- (8) User logic receives read data from controller.

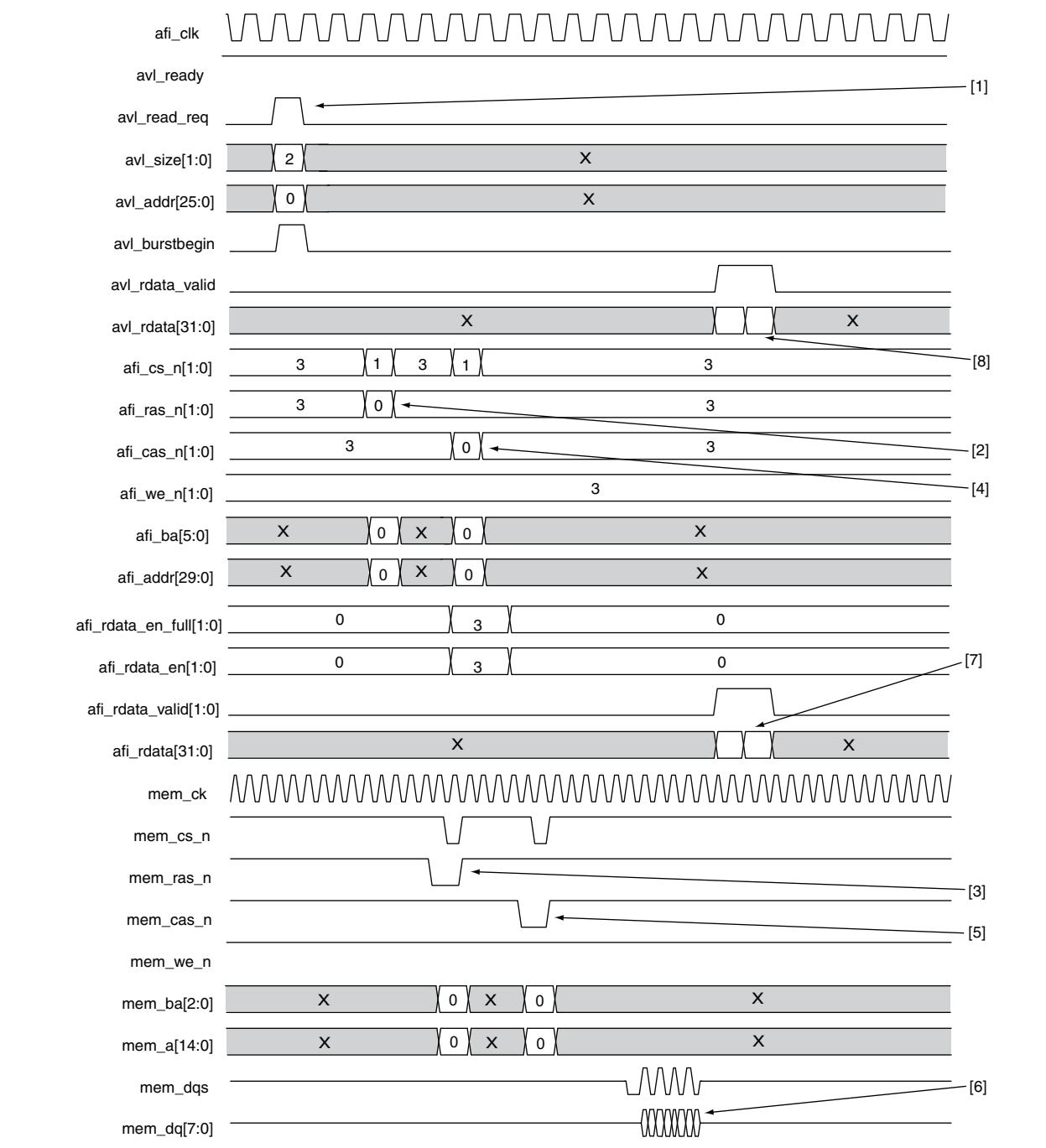
**Figure 12–4. Half-Rate DDR2 SDRAM Write**



**Notes for Figure 12–4:**

- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues activate command to PHY.
- (4) PHY issues activate command to memory.

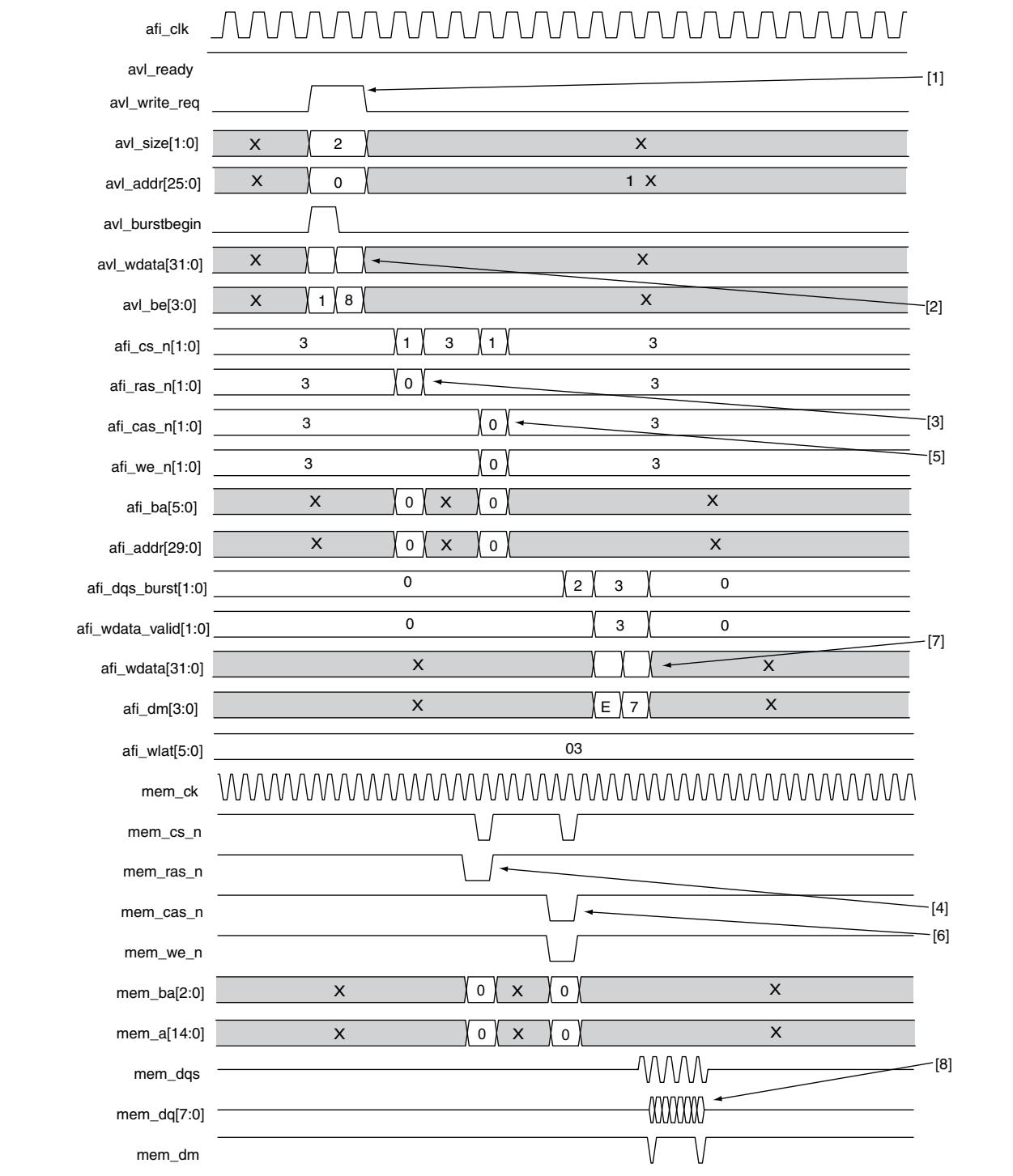
- (5) Controller issues write command to PHY.
- (6) PHY issues write command to memory.
- (7) Controller sends write data to PHY.
- (8) PHY sends write data to memory.

**Figure 12-5. Half-Rate DDR3 SDRAM Read****Notes for Figure 12-5:**

- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.

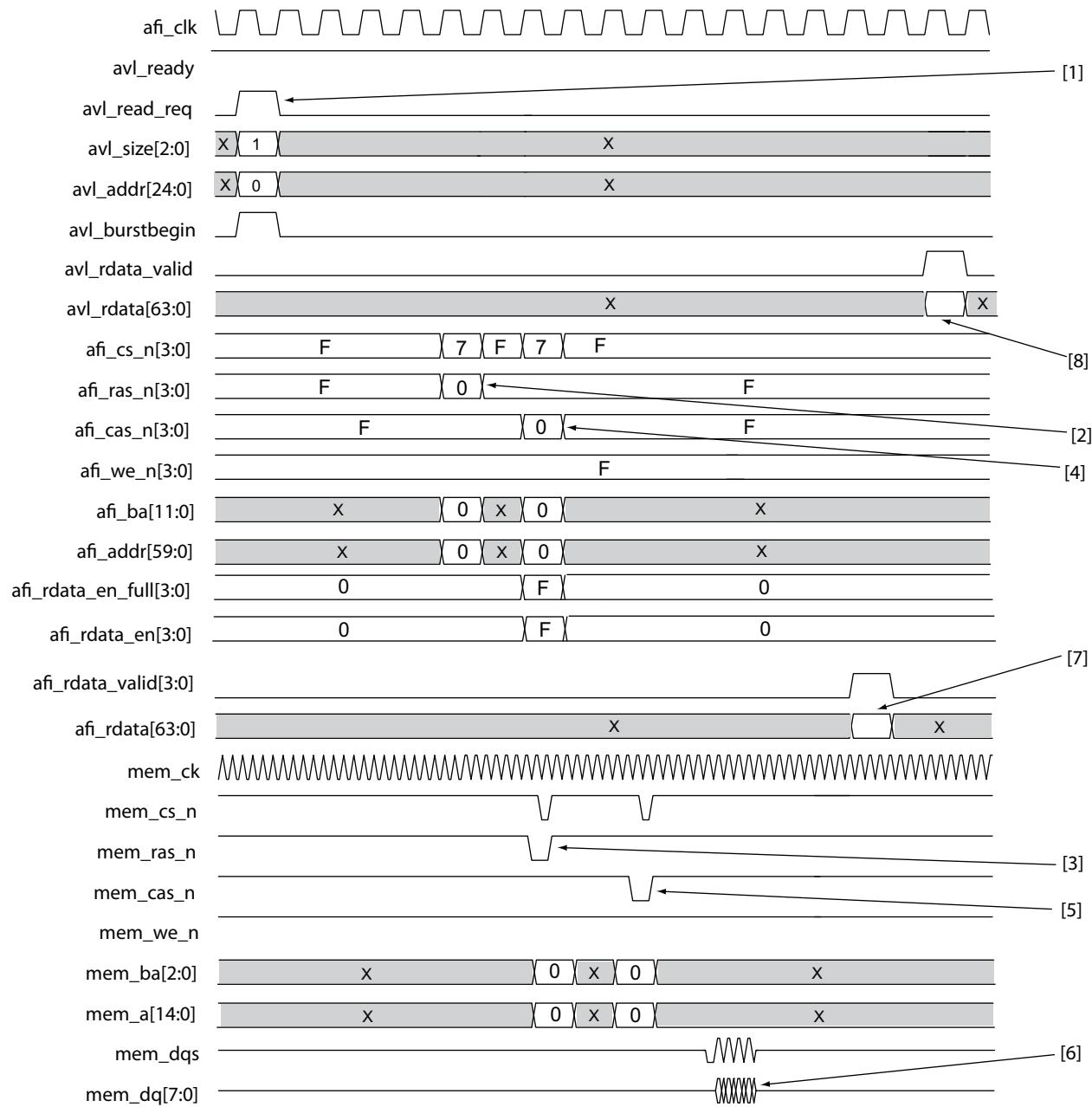
- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory.
- (7) Controller receives read data from PHY.
- (8) User logic receives read data from controller.

**Figure 12-6. Half-Rate DDR3 SDRAM Writes**



**Notes for Figure 12-6:**

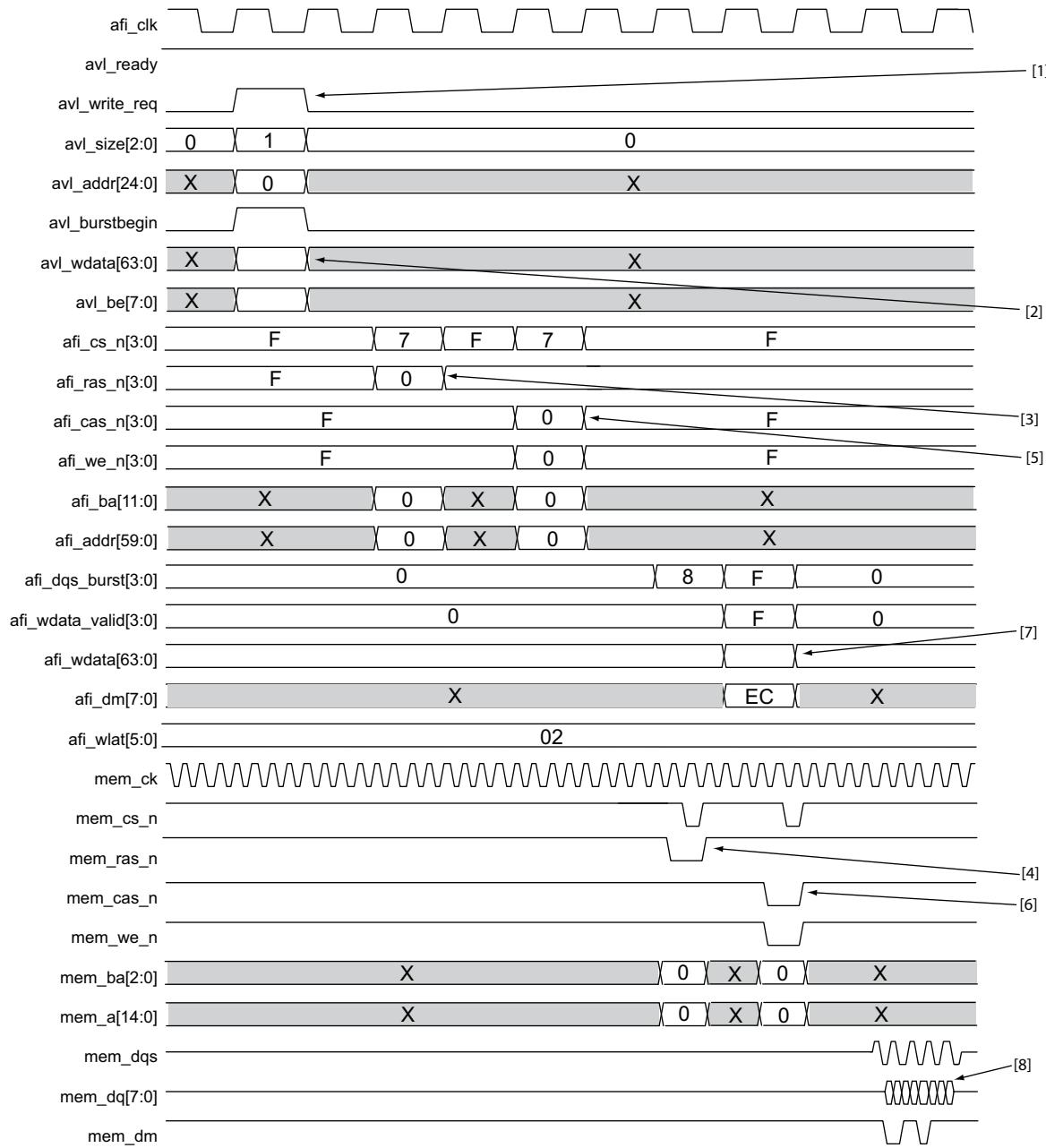
- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues activate command to PHY.
- (4) PHY issues activate command to memory.
- (5) Controller issues write command to PHY.
- (6) PHY issues write command to memory.
- (7) Controller sends write data to PHY.
- (8) PHY sends write data to memory.

**Figure 12-7. Quarter-Rate DDR3 SDRAM Reads**

**Notes for Figure 12-7:**

- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.
- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory
- (7) Controller receives read data from PHY
- (8) User logic receives read data from controller.

**Figure 12-8. Quarter-Rate DDR3 SDRAM Writes**



**Notes for Figure 12–8:**

- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues activate command to PHY
- (4) PHY issues activate command to memory.
- (5) Controller issues write command to PHY
- (6) PHY issues write command to memory
- (7) Controller sends write data to PHY
- (8) PHY sends write data to memory.

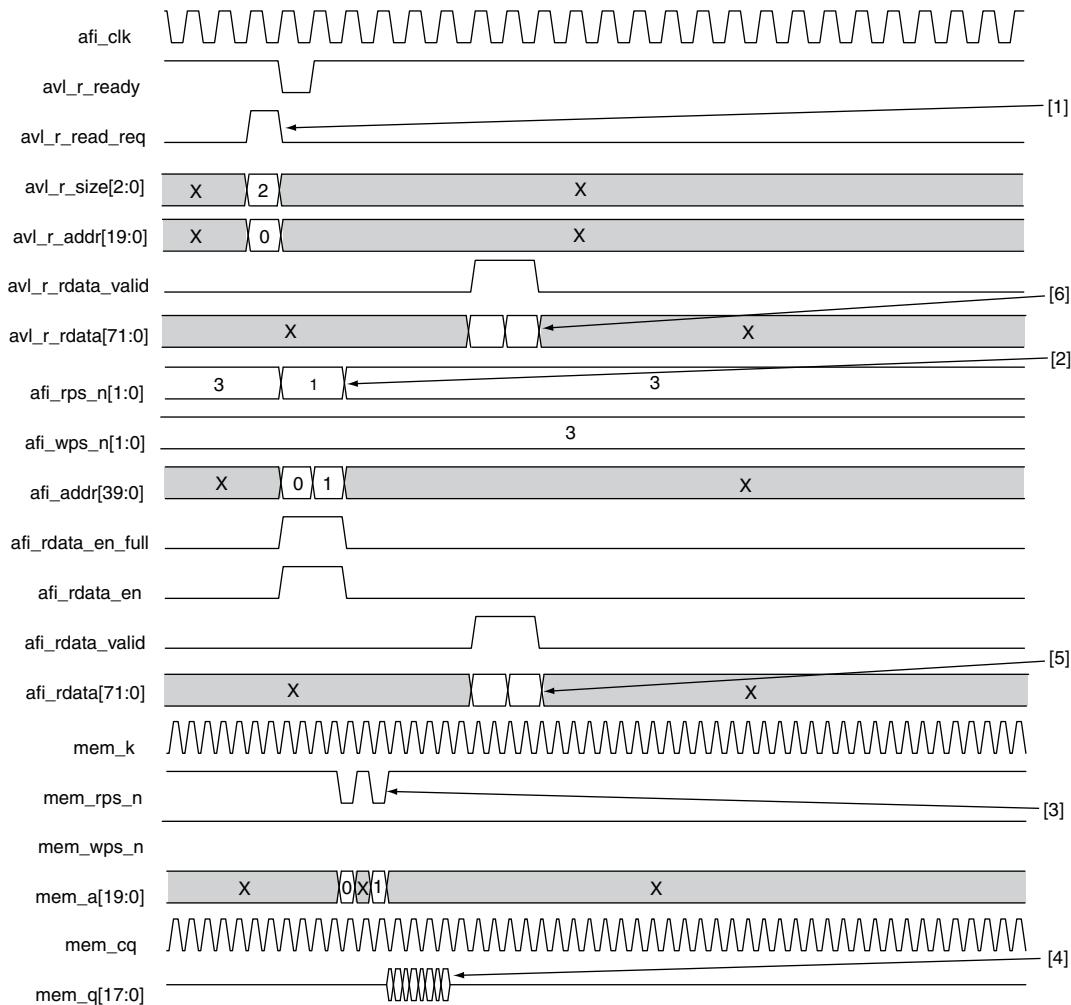
## **QDR II and QDR II+ Timing Diagrams**

This section contains timing diagrams for QDR II and QDR II+ protocols.

Figure 12–9 through Figure 12–12 present the following timing diagrams, based on a Stratix III device:

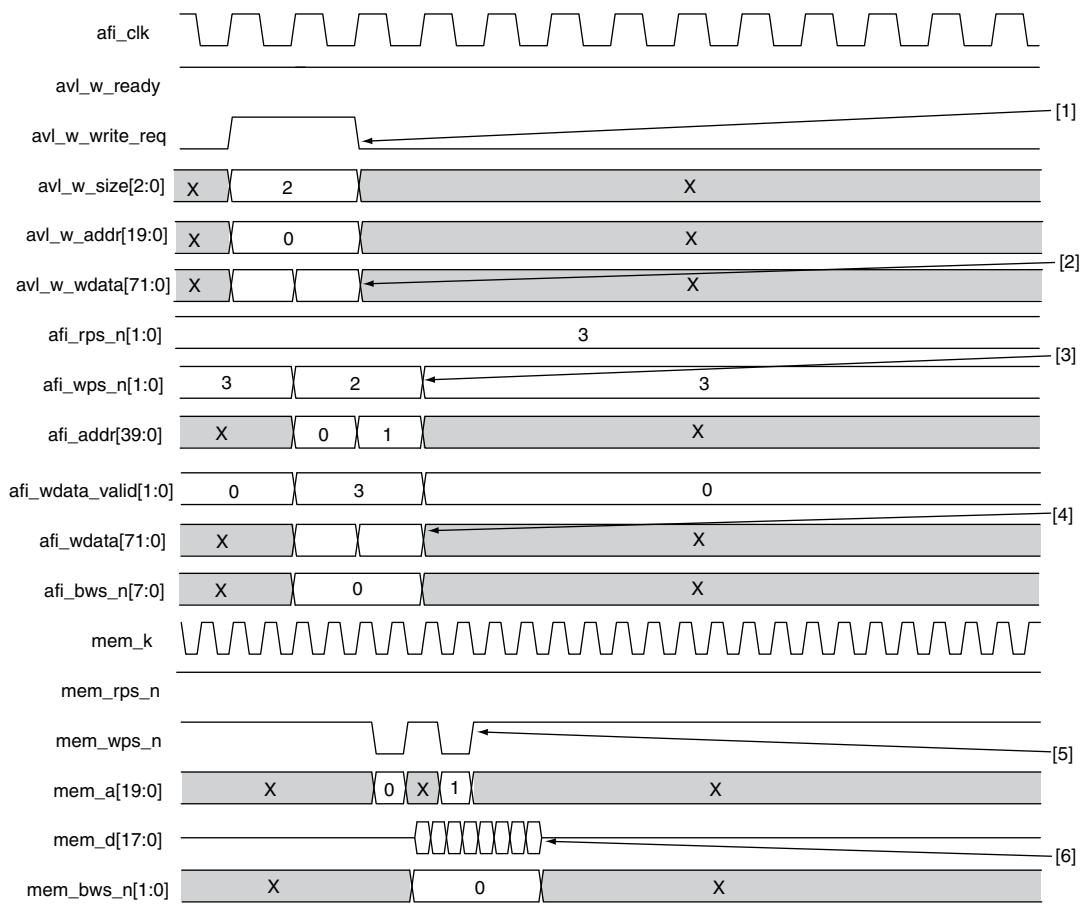
- Half-Rate QDR II and QDR II+ SRAM Read
- Half-Rate QDR II and QDR II+ SRAM Write
- Full-Rate QDR II and QDR II+ SRAM Read
- Full-Rate QDR II and QDR II+ SRAM Write

**Figure 12–9. Half-Rate QDR II and QDR II+ SRAM Read**



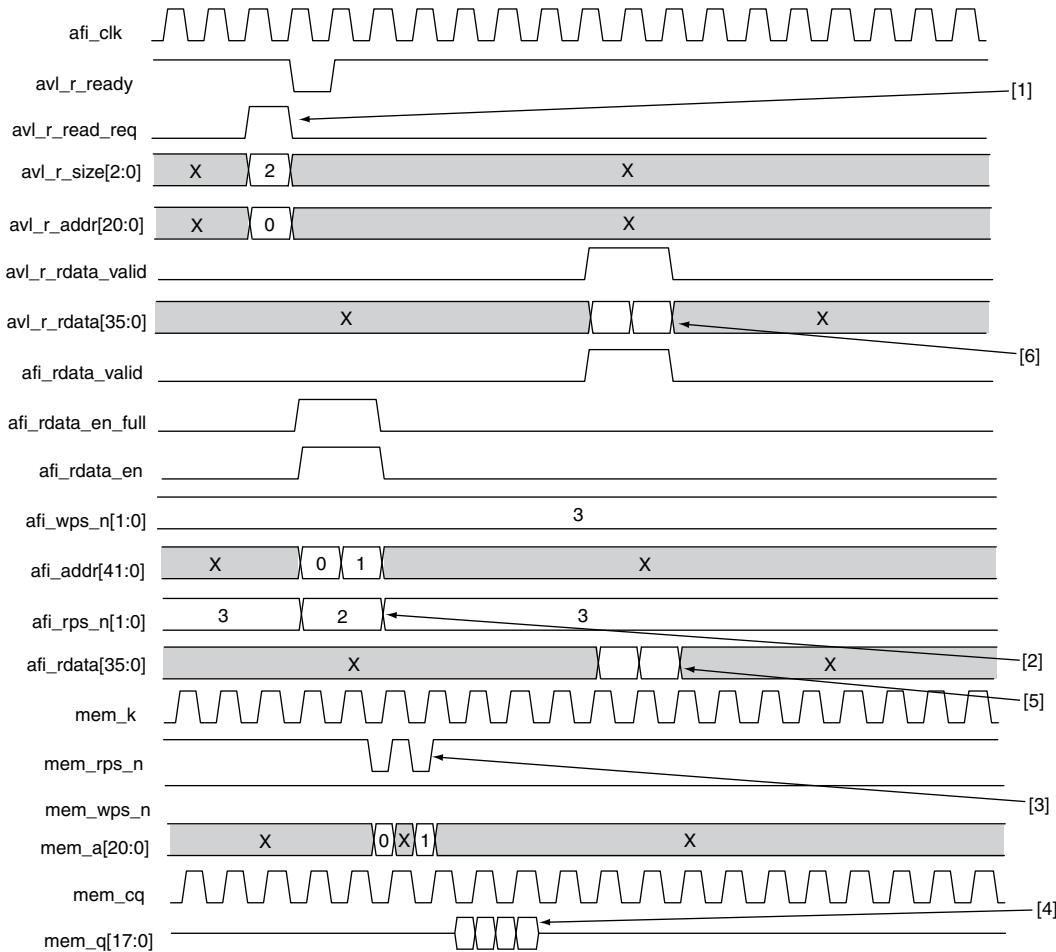
**Notes for Figure 12–9:**

- (1) Controller receives read command.
- (2) Controller issues two read commands to PHY.
- (3) PHY issues two read commands to memory.
- (4) PHY receives read data from memory.
- (5) Controller receives read data from PHY.
- (6) User logic receives read data from controller.

**Figure 12–10. Half-Rate QDR II and QDR II+ SRAM Write****Notes for Figure 12–10:**

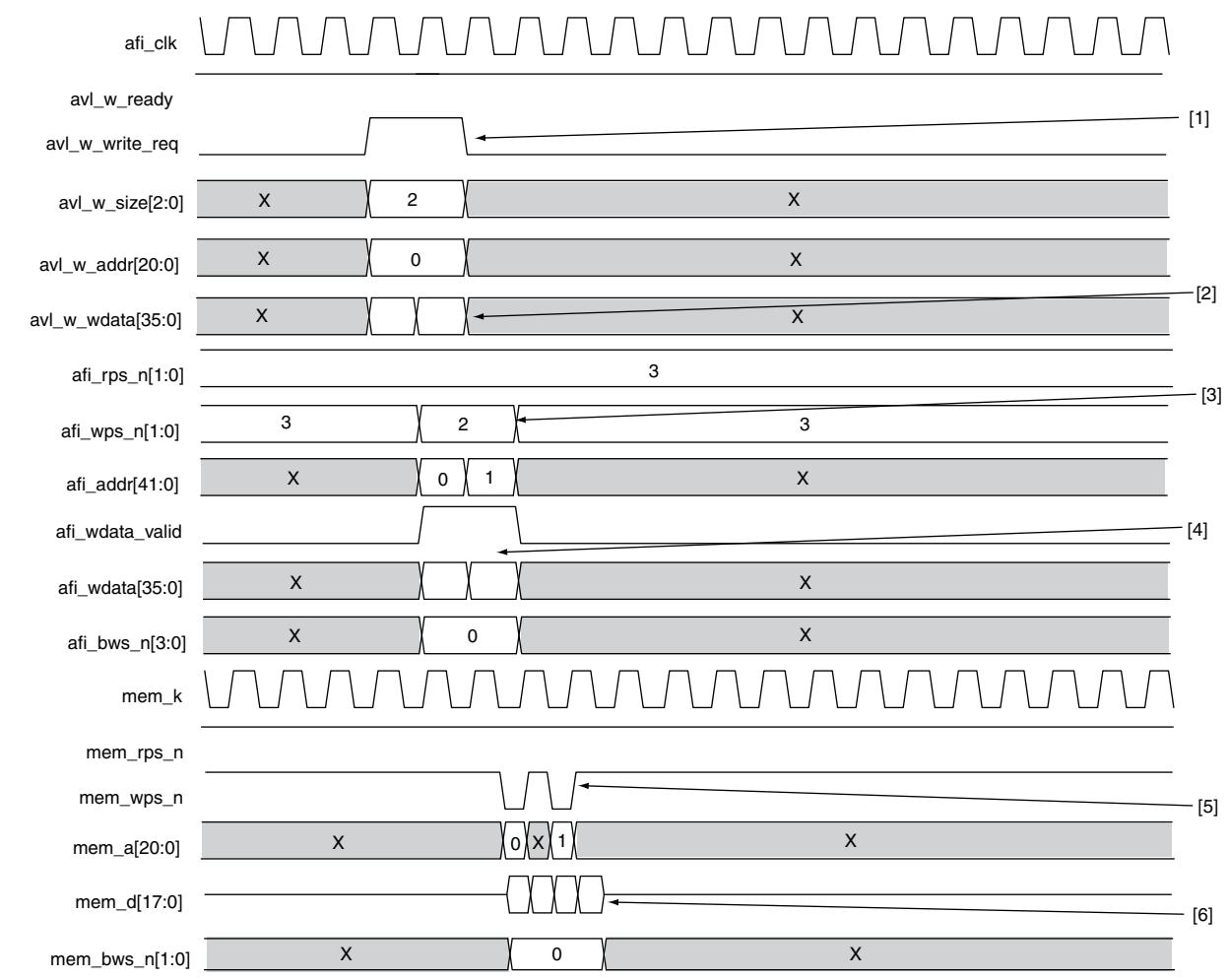
- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues two write commands to PHY.
- (4) Controller sends write data to PHY.
- (5) PHY issues two write commands to memory.
- (6) PHY sends write data to memory.

**Figure 12-11. Full-Rate QDR II and QDR II+ SRAM Read**



**Notes for Figure 12-11:**

- (1) Controller receives read command.
- (2) Controller issues two read commands to PHY.
- (3) PHY issues two read commands to memory.
- (4) PHY receives read data from memory.
- (5) Controller receives read data from PHY.
- (6) User logic receives read data from controller.

**Figure 12-12. Full-Rate QDR II and QDR II+ SRAM Write****Notes for Figure 12-12:**

- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues two write commands to PHY.
- (4) Controller sends write data to PHY.
- (5) PHY issues two write commands to memory.
- (6) PHY sends write data to memory.

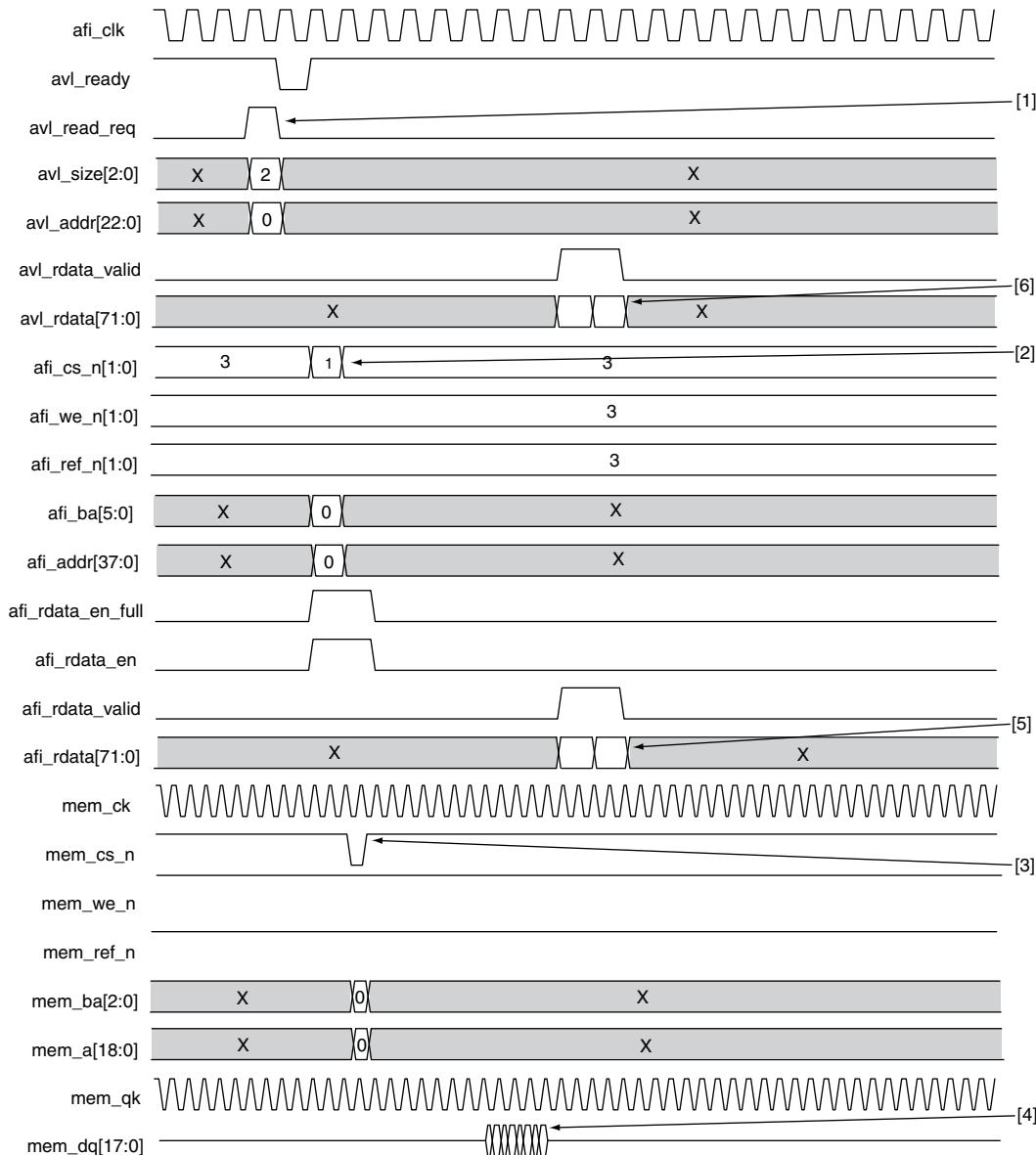
## RLDRAM II Timing Diagrams

This section contains timing diagrams for RLDRAm protocols.

**Figure 12–13 through Figure 12–16** present the following timing diagrams, based on a Stratix III device:

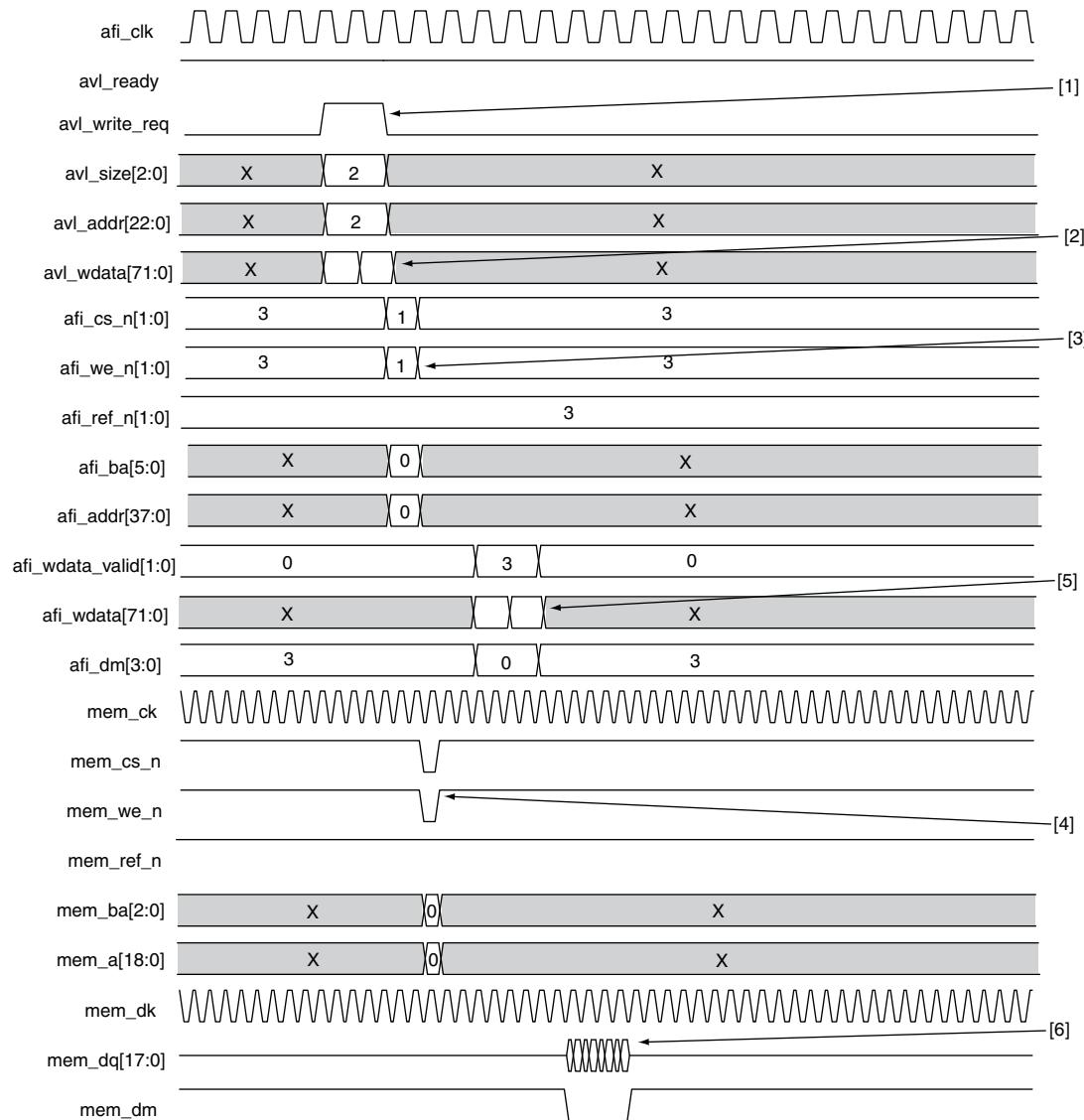
- Half-Rate RLDRAm II Read
- Half-Rate RLDRAm II Write
- Full-Rate RLDRAm II Read
- Full-Rate RLDRAm II Write

**Figure 12–13. Half-Rate RLDRAm II Read**



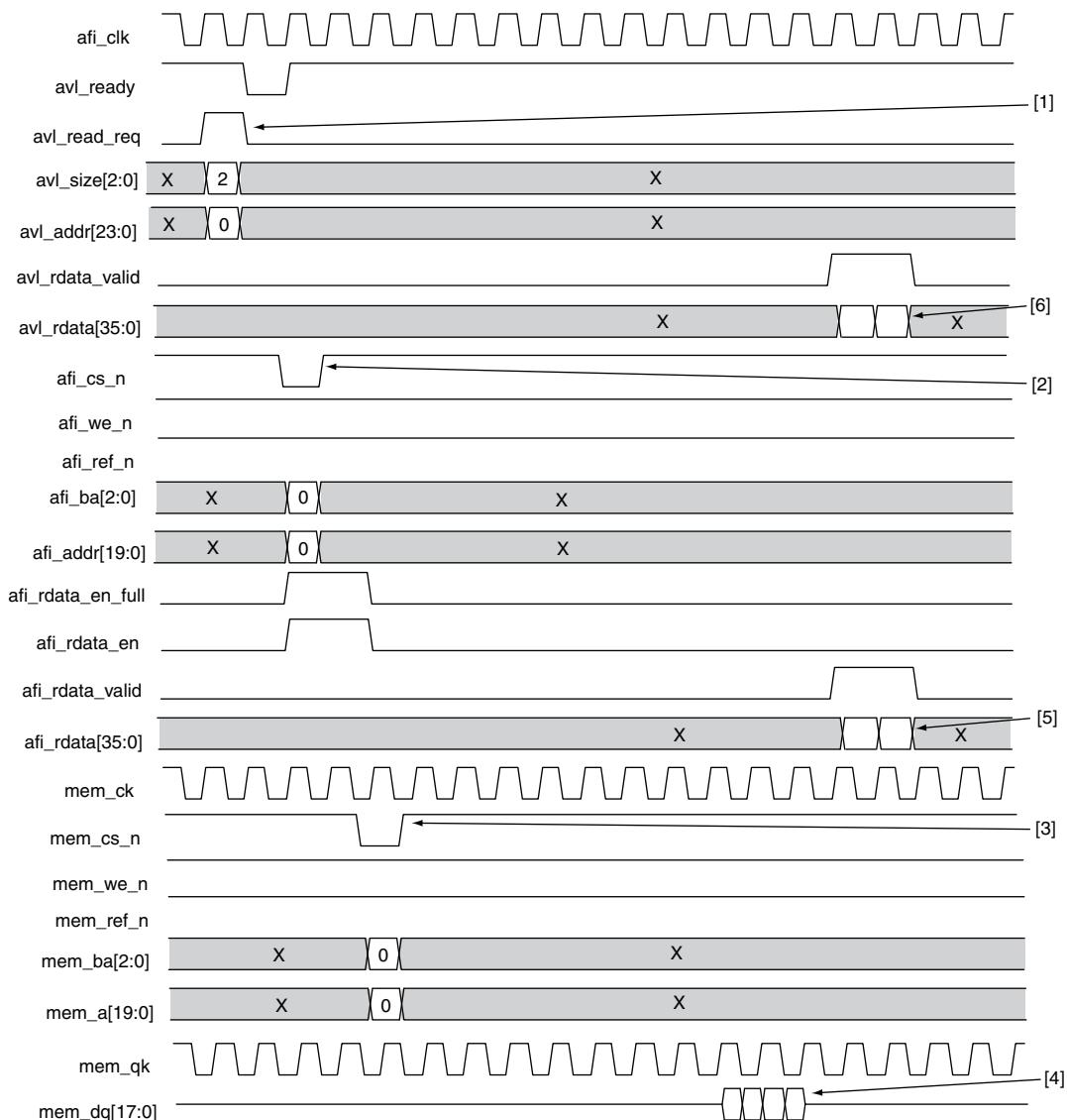
**Notes for Figure 12–13:**

- (1) Controller receives read command.
- (2) Controller issues read command to PHY.
- (3) PHY issues read command to memory.
- (4) PHY receives read data from memory.
- (5) Controller receives read data from PHY.
- (6) User logic receives read data from controller.

**Figure 12–14. Half-Rate RLDRAM II Write****Notes for Figure 12–14:**

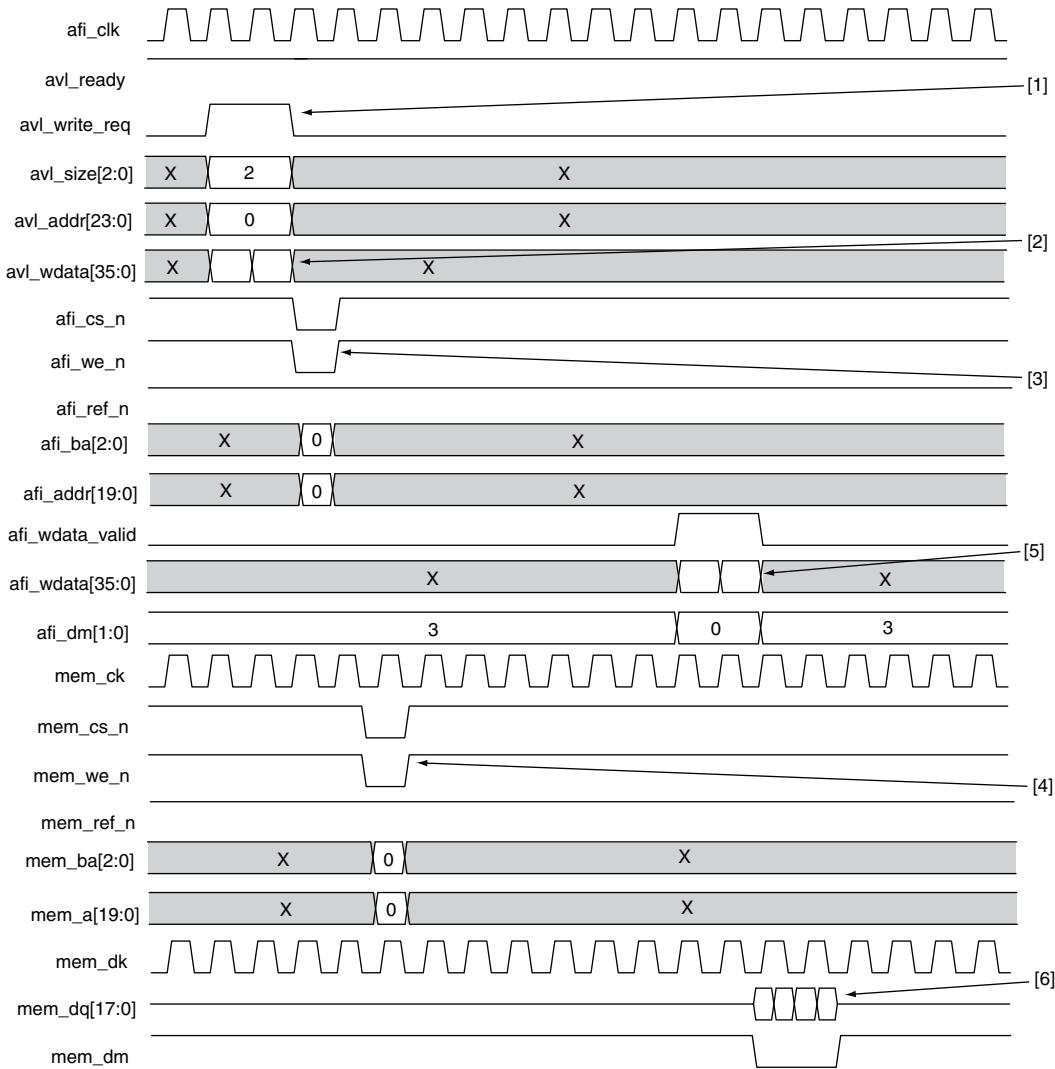
- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues write command to PHY.
- (4) PHY issues write command to memory.
- (5) Controller sends write data to PHY.
- (6) PHY sends write data to memory.

**Figure 12–15. Full-Rate RLDIMM II Read**



**Notes for Figure 12–15:**

- (1) Controller receives read command.
- (2) Controller issues read command to PHY.
- (3) PHY issues read command to memory.
- (4) PHY receives read data from memory.
- (5) Controller receives read data from PHY.
- (6) User logic receives read data from controller.

**Figure 12–16. Full-Rate RLDIMM II Write****Notes for Figure 12–16:**

- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues write command to PHY.
- (4) PHY issues write command to memory.
- (5) Controller sends write data to PHY.
- (6) PHY sends write data to memory.

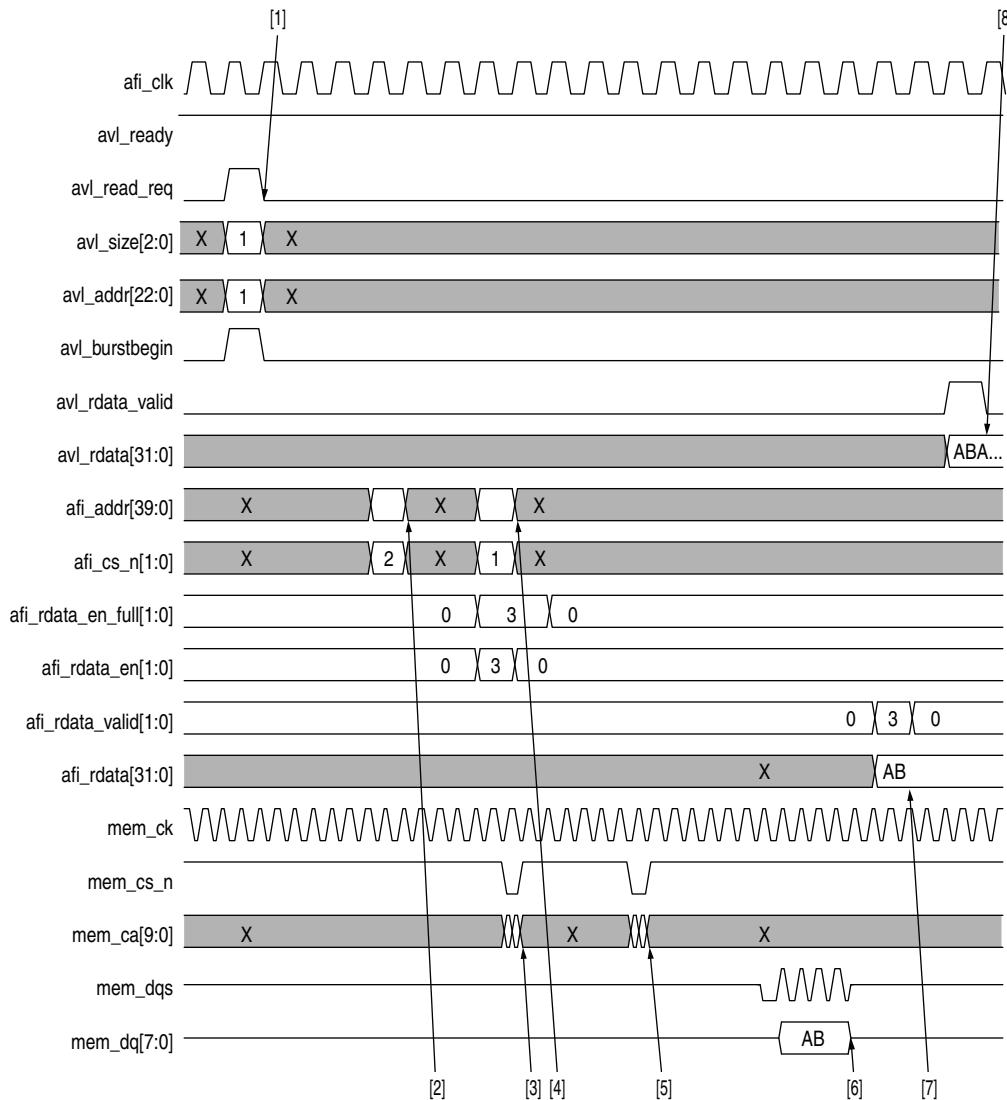
## LPDDR2 Timing Diagrams

This section contains timing diagrams for LPDDR2 protocols.

**Figure 12–17** through **Figure 12–20** present the following timing diagrams:

- Half-Rate LPDDR2 Read
- Half-Rate LPDDR2 Write
- Full-Rate LPDDR2 Read
- Full-Rate LPDDR2 Write

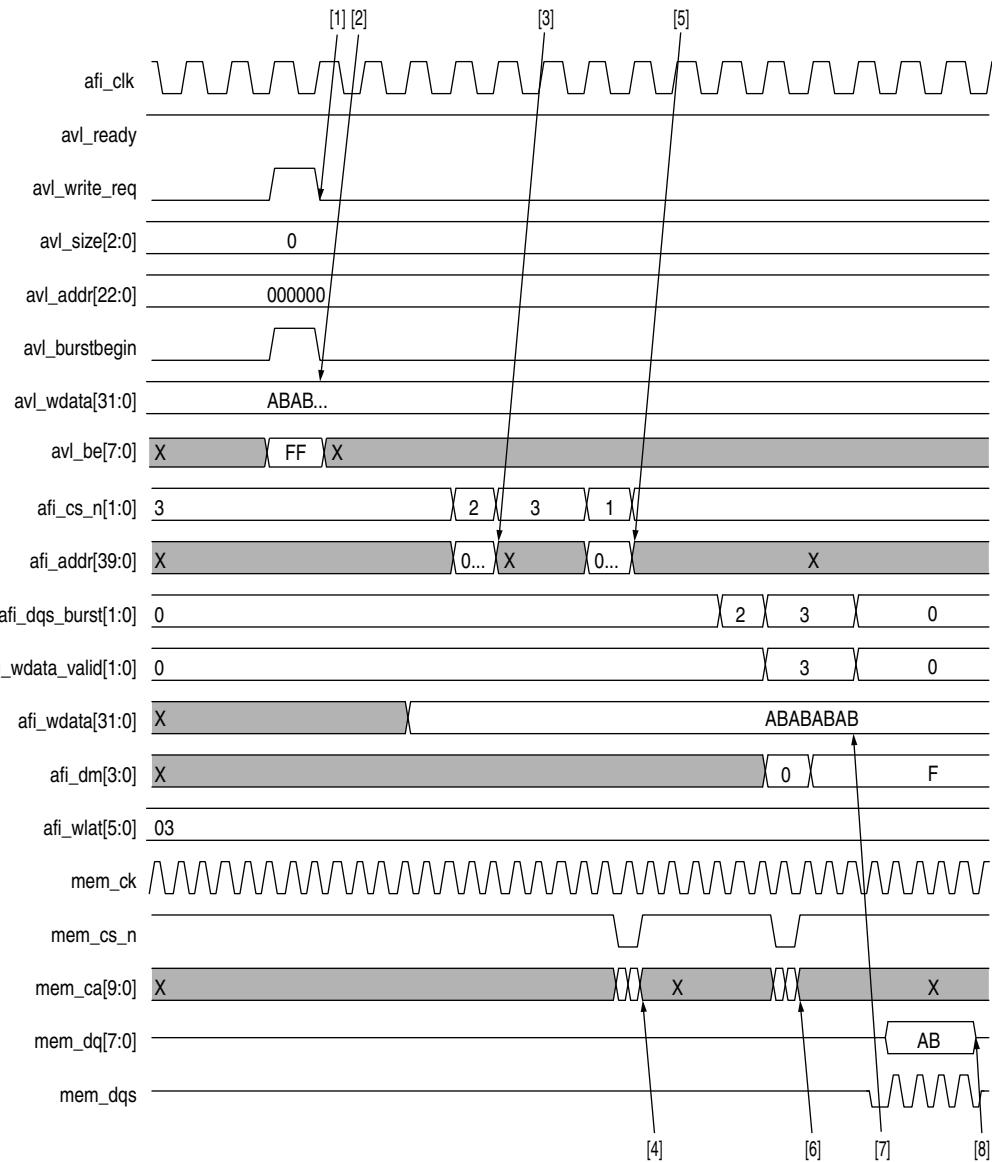
**Figure 12–17. Half-Rate LPDDR2 Read**



### Notes for **Figure 12–17**:

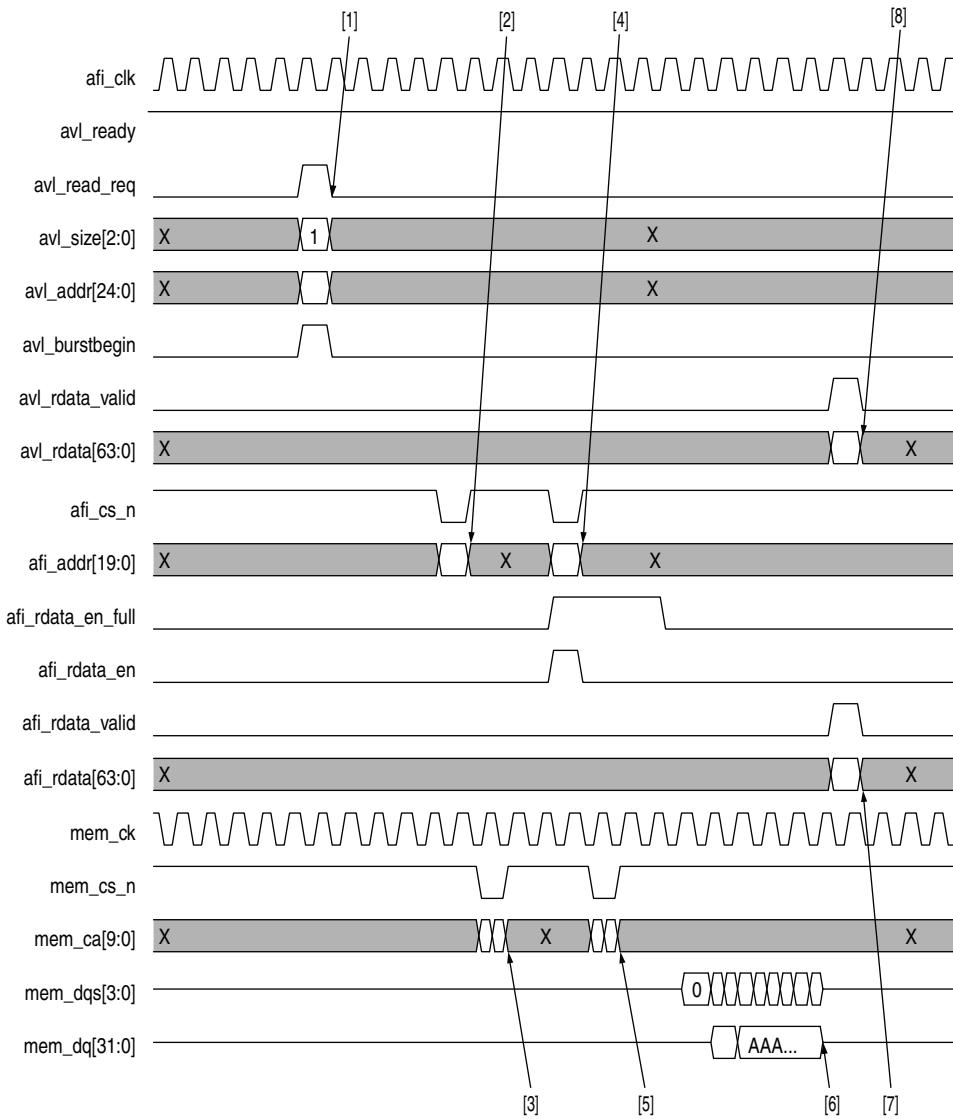
- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.

- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory.
- (7) Controller receives read data from PHY.
- (8) User logic receives read data from controller.

**Figure 12-18. Half-Rate LPDDR2 Write****Notes for Figure 12-18:**

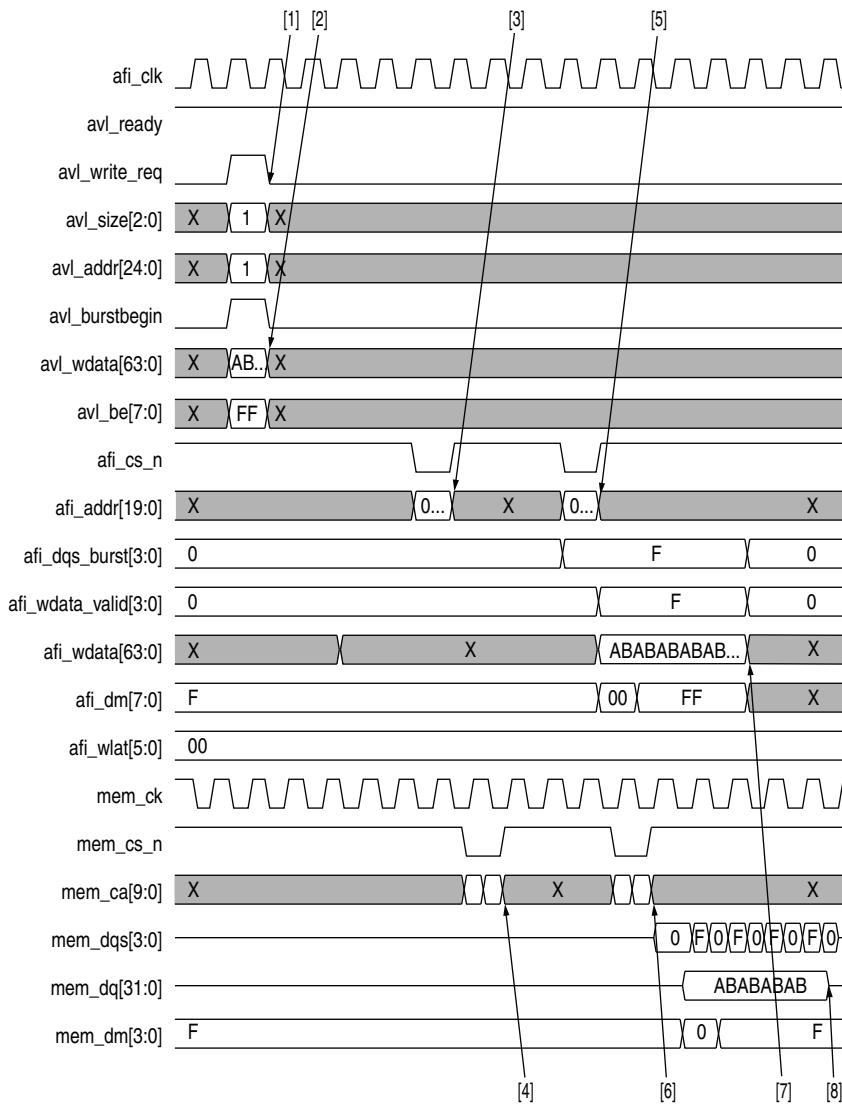
- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues activate command to PHY.
- (4) PHY issues activate command to memory.
- (5) Controller issues write command to PHY.
- (6) PHY issues write command to memory.
- (7) Controller sends write data to PHY.
- (8) PHY sends write data to memory.

**Figure 12-19. Full-Rate LPDDR2 Read**



**Notes for Figure 12-19:**

- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.
- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory.
- (7) Controller receives read data from PHY.
- (8) User logic receives read data from controller.

**Figure 12–20. Full-Rate LPDDR2 Write****Notes for Figure 12–20:**

- (1) Controller receives write command.
- (2) Controller receives write data.
- (3) Controller issues activate command to PHY.
- (4) PHY issues activate command to memory.
- (5) Controller issues write command to PHY.
- (6) PHY issues write command to memory.
- (7) Controller sends write data to PHY.
- (8) PHY sends write data to memory.

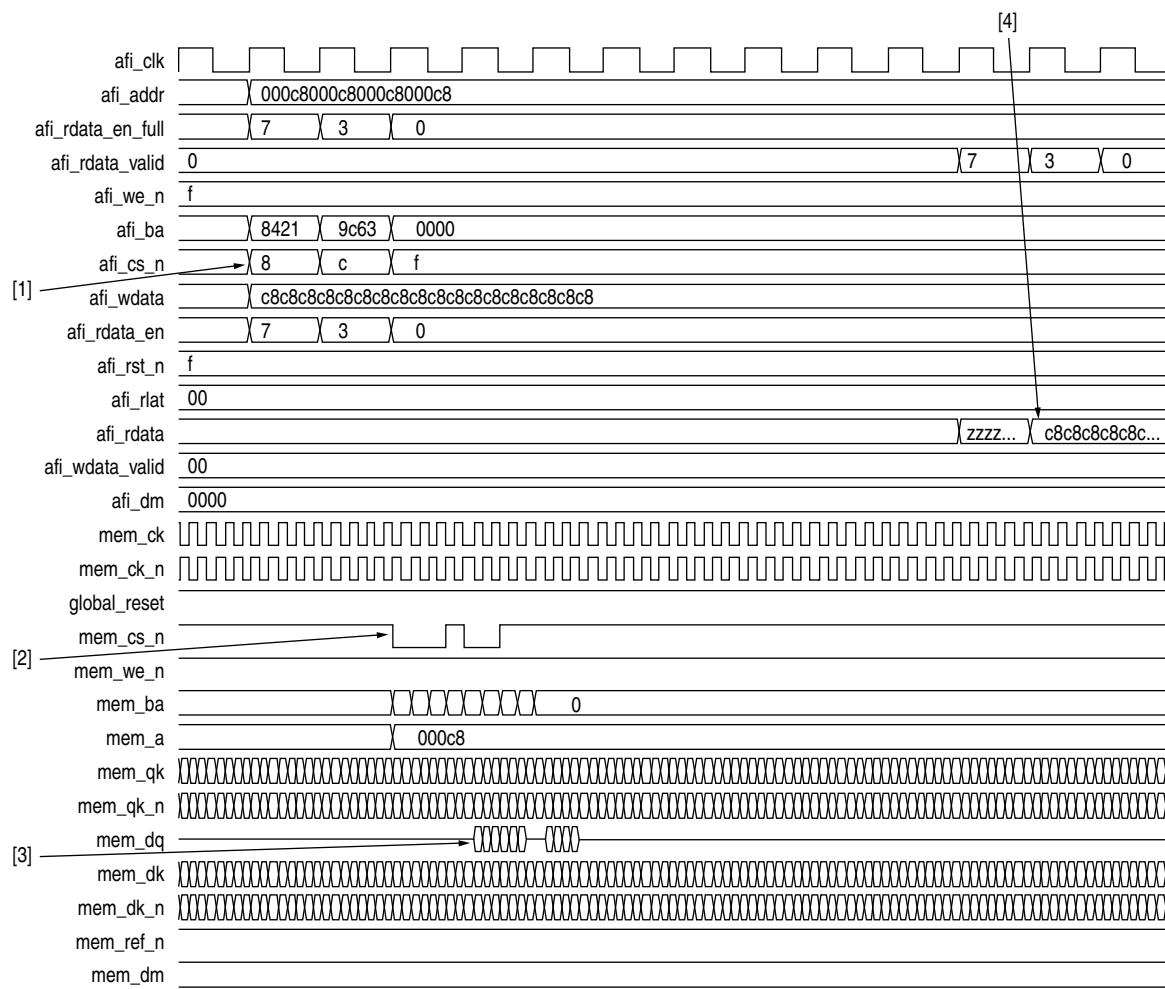
## RLDRAM 3 Timing Diagrams

This section contains timing diagrams for RLDRAm 3 protocols.

Figure 12-21 through Figure 12-22 present the following timing diagrams:

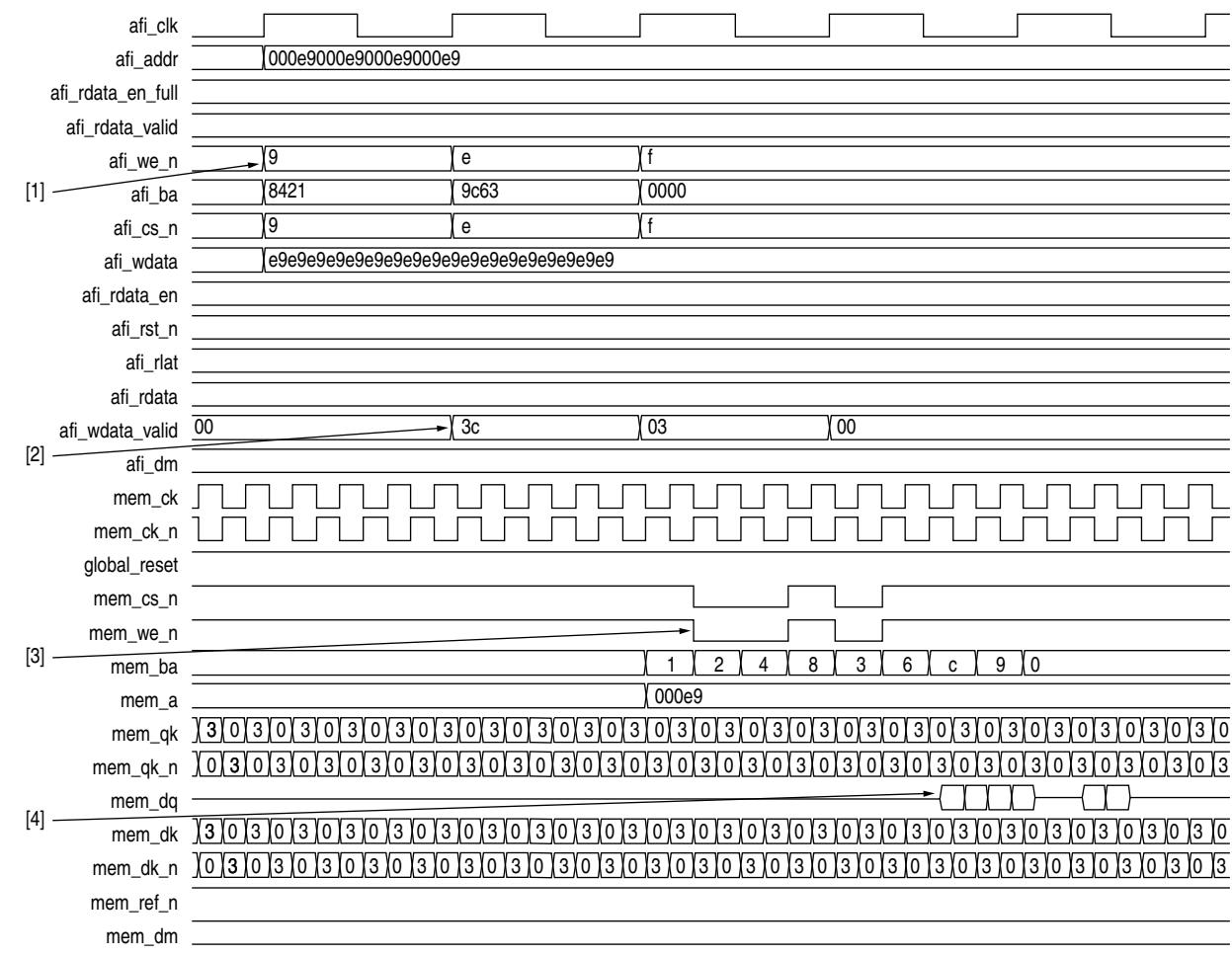
- Quarter-Rate RLDRAm 3 Read
- Quarter-Rate RLDRAm 3 Write
- Half-rate RLDRAm 3 Read
- Half-Rate RLDRAm 3 Write

**Figure 12-21. Quarter-Rate RLDRAm 3 Read**



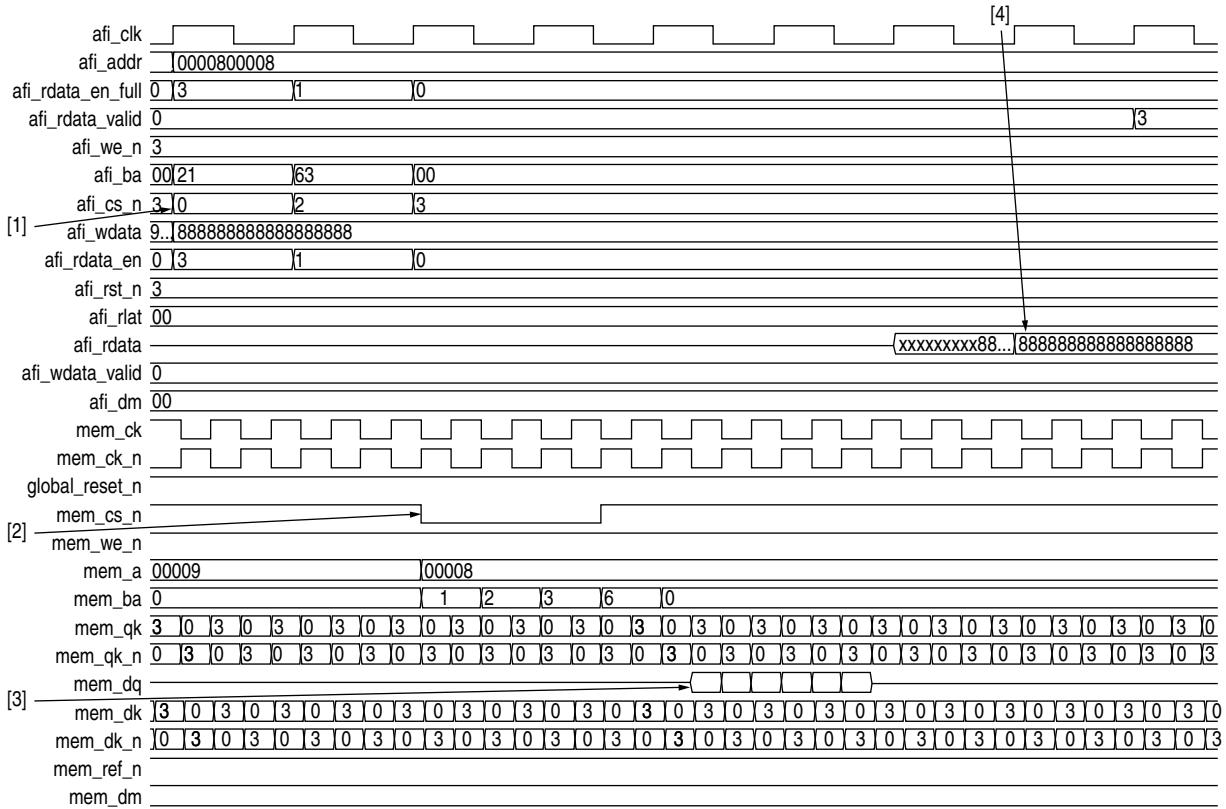
### Notes for Figure 12-21:

- (1) Controller issues read command to PHY.
- (2) PHY issues read command to memory.
- (3) PHY receives data from memory.
- (4) Controller receives read data from PHY.

**Figure 12–22. Quarter-Rate RLDRAM 3 Write****Notes for Figure 12–22:**

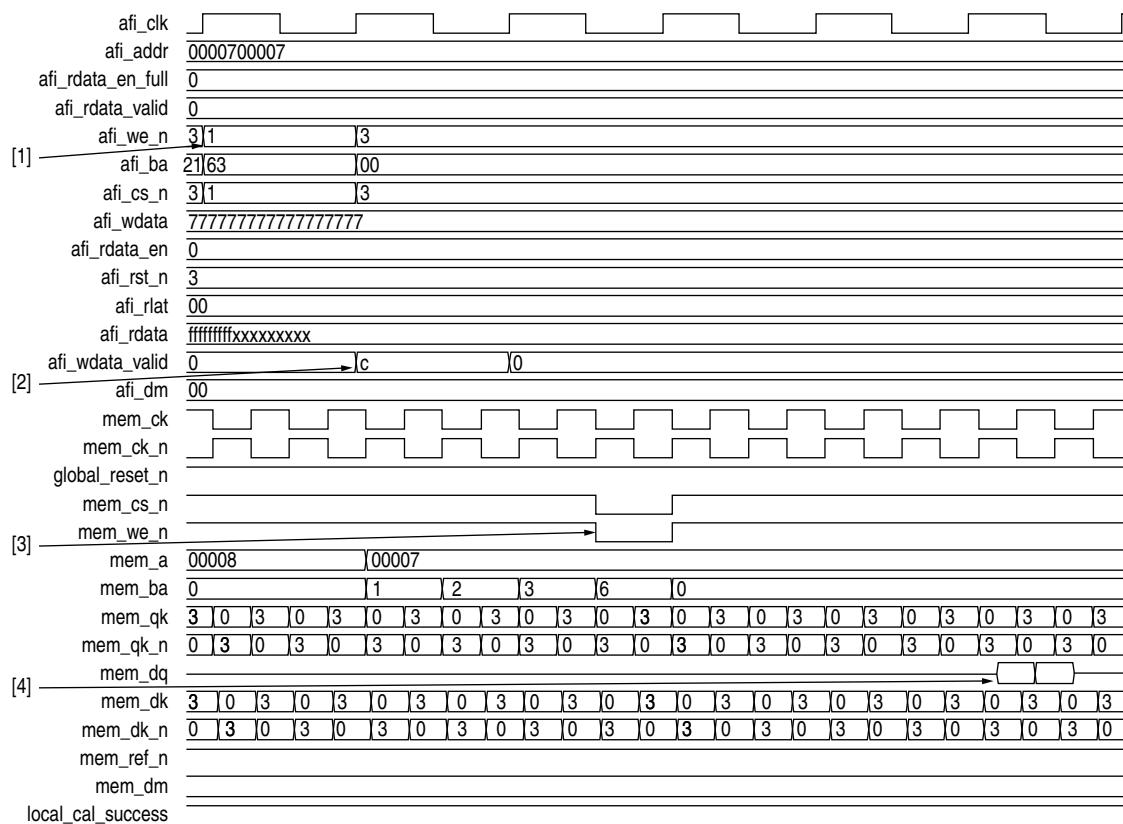
- (1) Controller issues write command to PHY.
- (2) Data ready from controller for PHY.
- (3) PHY issues write command to memory.
- (4) PHY sends read data to memory.

**Figure 12–23. Half-rate RLDRAM 3 Read**



### **Notes for Figure 12–23:**

- (1) Controller issues read command to PHY.
  - (2) PHY issues read command to memory.
  - (3) PHY receives data from memory.
  - (4) Controller receives read data from PHY.

**Figure 12-24. Half-Rate RLDRAM 3 Write****Notes for Figure 12-24:**

- (1) Controller issues write command to PHY.
- (2) Data ready from controller for PHY.
- (3) PHY issues write command to memory.
- (4) PHY sends read data to memory.

## Document Revision History

Table 12–1 lists the revision history for this document.

**Table 12–1. Document Revision History**

Date	Version	Changes
November 2012	2.1	<ul style="list-style-type: none"><li>■ Added timing diagrams for RLDRAM 3.</li><li>■ Changed chapter number from 10 to 12.</li></ul>
June 2012	2.0	<ul style="list-style-type: none"><li>■ Added timing diagrams for LPDDR2.</li><li>■ Added Feedback icon.</li></ul>
November 2011	1.1	<ul style="list-style-type: none"><li>■ Consolidated timing diagrams from <a href="#">11.0 DDR2 and DDR3 SDRAM Controller with UniPHY User Guide</a>, <a href="#">QDR II and QDR II+ SRAM Controller with UniPHY User Guide</a>, and <a href="#">RLDRAM II Controller with UniPHY IP User Guide</a>.</li><li>■ Added Read and Write diagrams for DDR3 quarter-rate.</li></ul>



This chapter describes the UniPHY External Memory Interface Toolkit. It explains the architecture and workflow, as well as how to launch the toolkit, link your project to a device, and establish communications over a selected connection. The chapter also discusses several operational considerations to enhance your productivity with the toolkit.

### Introduction

The EMIF Toolkit lets you diagnose and debug calibration problems and produce margining reports for your external memory interface. The toolkit is compatible with UniPHY-based external memory interfaces that use the Nios II-based sequencer, with toolkit communication enabled. Toolkit communication is on by default in versions 10.1 and 11.0 of UniPHY IP; for version 11.1 and later, toolkit communication is on whenever debugging is enabled on the Diagnostics tab of the IP core interface.

The EMIF Toolkit can communicate with several different memory interfaces on the same device, but can communicate with only one memory device at a time.

### Architecture

The EMIF toolkit provides a graphical user interface for communication with connections, however all functions provided in the toolkit are also available directly from the quartus\_sh TCL shell, through the external\_memif\_toolkit TCL package. The availability of TCL support allows you to create scripts to run automatically from TCL. You can find information about specific TCL commands by running help -pkg external\_memif\_toolkit from the quartus\_sh TCL shell.

If you want, you can begin interacting with the toolkit through the GUI, and later automate your workflow by creating TCL scripts. The toolkit GUI records a history of the commands that you run. You can see the command history on the History tab in the toolkit GUI.

### Communication

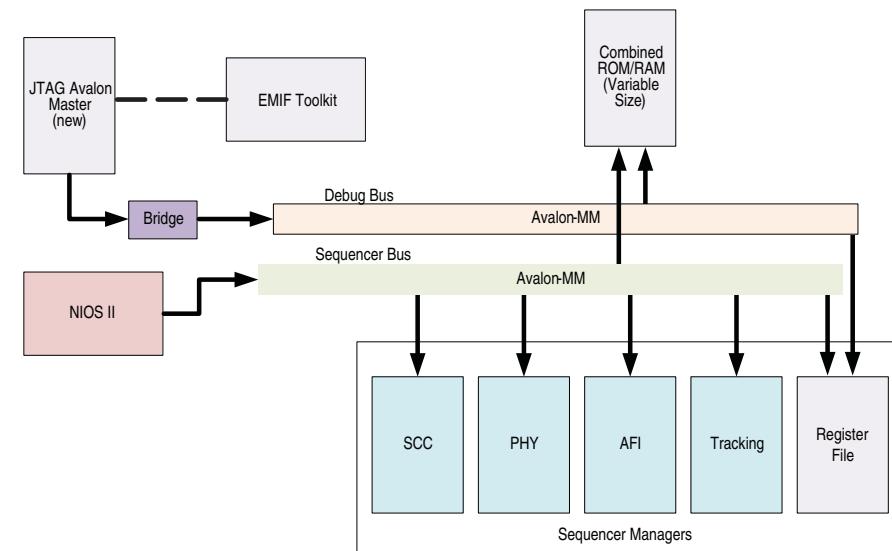
Communication between the EMIF Toolkit and external memory interface connections varies, depending on the connection type and version. In versions 10.1 and 11.0 of the EMIF IP, communication is achieved using direct communication to the Nios II-based sequencer. In version 11.1 and later, communication is achieved using a JTAG Avalon-MM master attached to the sequencer bus.

Figure 13–1 shows the structure of UniPHY-based IP version 11.1 and later, with JTAG Avalon-MM master attached to sequencer bus masters.

© 2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



**Figure 13-1. UniPHY IP Version 11.1 and Later, with JTAG Avalon-MM Master**



## Calibration and Report Generation

For versions 10.1 and 11.0 UniPHY-based interfaces, the EMIF Toolkit causes the memory interface to calibrate several times, to produce the data from which the toolkit generates its reports. In version 11.1 and later, report data is generated during calibration, without need to repeat calibration. For version 11.1 and later, generated reports reflect the result of the previous calibration, without need to recalibrate unless you choose to do so.

## Setup and Use

Before using the EMIF Toolkit, you should compile your design and program the target device with the resulting SRAM Object File (**.sof**). For designs compiled in the Quartus II software version 12.0 or earlier, debugging information is contained in the JTAG Debugging Information file (**.jdi**); however, for designs compiled in the Quartus II software version 12.1 or later, all debugging information resides in the **.sof** file.

You can run the toolkit using all your project files, or using only the Quartus II Project File (**.qpf**), Quartus II Settings File (**.qsf**), and **.sof** file; the **.jdi** file is also required for designs compiled prior to version 12.1. To ensure that all debugging information is correctly synchronized for designs compiled prior to version 12.1, ensure that the **.sof** and **.jdi** files that you used are generated during the same run of the Quartus II Assembler.

After you have programmed the target device, you can run the EMIF Toolkit and open your project. You can then use the toolkit to create connections to the external memory interface.

## General Workflow

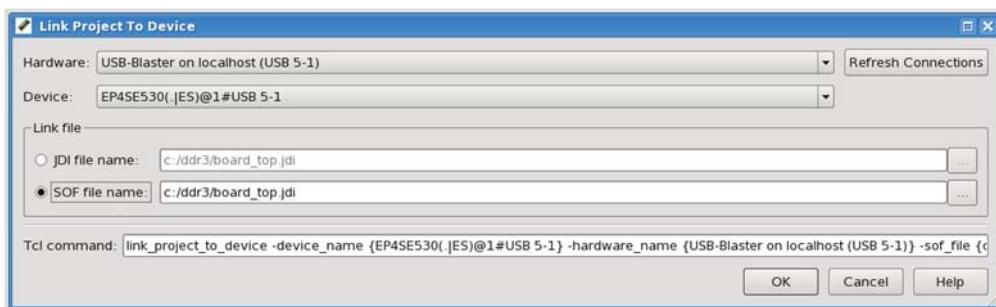
To use the EMIF Toolkit, you must link your compiled project to a device, and create a communication channel to the connection that you want to examine. This section explains how to perform these major steps.

### Linking the Project to a Device

1. To launch the toolkit, select External Memory Interface Toolkit from the Tools menu in the Quartus II software.
2. After you have launched the toolkit, open your project and click the **Initialize connections** task in the **Tasks** window, to initialize a list of all known connections.
3. To link your project to a specific device on specific hardware, perform the following steps:
  - a. Click the **Link Project to Device** task in the **Tasks** window.
  - b. Select the desired hardware from the **Hardware** dropdown menu in the **Link Project to Device** dialog box.
  - c. Select the desired device on the hardware from the **Device** dropdown menu in the **Link Project to Device** dialog box.
  - d. Select the correct **Link file type**, depending on the version of Quartus II software in which your design was compiled:
    - If your design was compiled in the Quartus II software version 12.0 or earlier, select **JDI** as the **Link file type**, verify that the **.jdi** file is correct for your **.sof** file, and click **Ok**.
    - If your design was compiled in the Quartus II software version 12.1 or later, select **SOF** as the **Link file type**, verify that the **.sof** file is correct for your programmed device, and click **Ok**.

Figure 13–2 illustrates the **Link Project to Device** dialog box.

**Figure 13–2. Link Project to Device Dialog Box**



When you link your project to the device, the toolkit verifies all connections on the device against the information in the JDI or SOF file, as appropriate. If the toolkit detects any mismatch between the JDI file and the device connections, an error message is displayed.

For designs compiled using the Quartus II software version 12.1 or later, the SOF file contains a design hash to ensure the SOF file used to program the device matches the SOF file specified for linking to a project. If the hash does not match, an error message appears.

If the toolkit successfully verifies all connections, it then attempts to determine the connection type for each connection. Connections of a known type are listed in the Linked Connections report, and are available for the toolkit to use.

## Establishing Communication to Connections

After you have completed linking the project, you can establish communication to the connections

1. In the Tasks window,
  - Click **Establish Memory Interface Connection** to create a connection to the external memory interface.
  - Click **Establish Efficiency Monitor Connection** to create a connection to the efficiency monitor.
2. To create a communication channel to a connection, select the desired connection from the displayed pulldown menu of connections, and click **Ok**.

The toolkit establishes a communication channel to the connection, creates a report folder for the connection, and creates a folder of tasks for the connection.

 By default, the connection and the reports and tasks folders are named according to the hierarchy path of the connection. If you want, you can specify a different name for the connection and its folders.

3. You can run any of the tasks in the folder for the connection; any resulting reports appear in the reports folder for the connection.

## Reports

The toolkit can generate a variety of reports, including summary, calibration, and margining reports for external memory interface connections. To generate a supported type of report for a connection, you run the associated task in the tasks folder for that connection.

### Summary Report

The Summary Report provides an overview of the memory interface; it consists of the following tables:

- Summary table. Provides a high-level summary of calibration results. This table lists details about the connection, IP version, IP protocol, and basic calibration results, including calibration failures. This table also lists the estimated average read and write data valid windows, and the calibrated read and write latencies.
- Interface Details table. Provides details about the parameterization of the memory IP. This table allows you to verify that the parameters in use match the actual memory device in use.

- Groups Masked from Calibration table. Lists any groups that were masked from calibration when calibration occurred. Masked groups are ignored during calibration.
- Ranks Masked from Calibration tables (DDR2 and DDR3 only). Lists any ranks that were masked from calibration when calibration occurred. Masked ranks are ignored during calibration.

## Calibration Report

The Calibration Report provides detailed information about the margins observed before and after calibration, and the settings applied to the memory interface during calibration; it consists of the following tables:

- Per DQS Group Calibration table. Lists calibration results for each group. If a group fails calibration, this table also lists the reason for the failure.
  - ☞ If a group fails calibration, the calibration routine skips all remaining groups. You can deactivate this behaviour by running the **Enable Calibration for All Groups On Failure** command in the toolkit.
- DQ Pin Margins Observed Before Calibration table. Lists the DQ pin margins observed before calibration occurs. You can refer to this table to see the per-bit skews resulting from the specific silicon and board that you are using.
- DQS Group Margins Observed During Calibration table. Lists the DQS group margins observed during calibration.
- DQ Pin Settings After Calibration and DQS Group Settings After Calibration table. Lists the settings made to all dynamically controllable parts of the memory interface as a result of calibration. You can refer to this table to see the modifications made by the calibration algorithm.

## Margin Report

The Margin Report lists the post-calibration margins for each DQ and data mask pin, keeping all other pin settings constant; it consists of the following tables:

- DQ Pin Post Calibration Margins table. Lists the margin data in tabular format.
- Read Data Valid Windows report. Shows read data valid windows in graphical format.
- Write Data Valid Windows report. Shows write data valid windows in graphical format.

## Operational Considerations

This section provides additional information about specific features and considerations.

## Specifying a Particular JDI File

Correct operation of the EMIF Toolkit depends on the correct JDI being used when linking the project to the device. The JDI file is produced by the Quartus II Assembler, and contains a list of all system level debug nodes and their hierarchy path names. If the default **.jdi** file name is incorrect for your project, you must specify the correct **.jdi** file. The **.jdi** file is supplied during the link-project-to-device step, where the **revision\_name.jdi** file in the project directory is used by default. To supply an alternative **.jdi** file, click on the ellipse then select the correct **.jdi** file.

## PLL Status

When connecting to DDR-based external memory interface connections, the PLL status appears in the Establish Connection dialog box when the IP is generated to use the CSR controller port, allowing you to immediately see whether the PLL status is locked. If the PLL is not locked, no communication can occur until the PLL becomes locked and the memory interface reset is deasserted.

When you are linking your project to a device, an error message will occur if the toolkit detects that a JTAG Avalon-MM master has no clock running. You can run the **Reindex Connections** task to have the toolkit rescan for connections and update the status and type of found connections in the Linked Connections report.

## Margining Reports

The EMIF Toolkit can display margining information showing the post-calibration data-valid windows for reads and writes. Margining information is determined by individually modifying the input and output delay chains for each data and strobe/clock pin to determine the working region. The toolkit can display margining data in both tabular and hierachial formats.

## Group Masks

To aid in debugging your external memory interface, the EMIF Toolkit allows you to mask individual groups and ranks from calibration. Masked groups and ranks are skipped during the calibration process, meaning that only unmasked groups and ranks are included in calibration. Subsequent mask operations overwrite any previous masks.



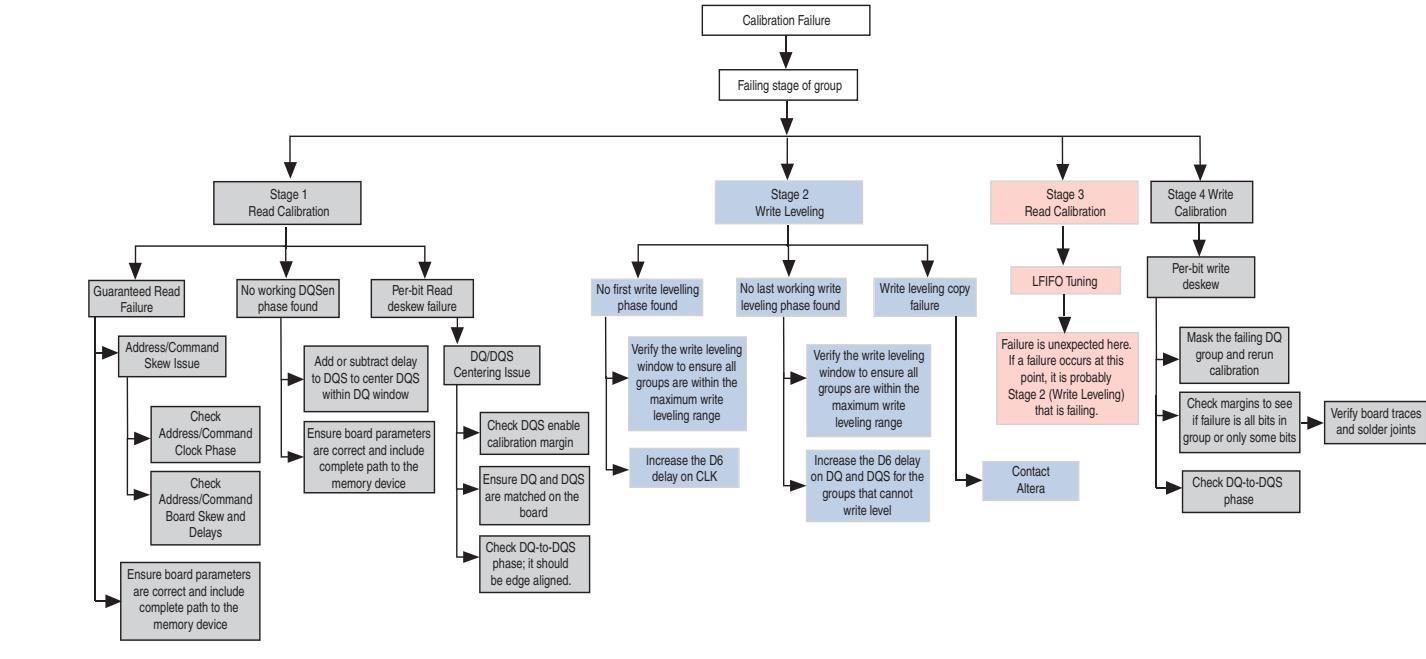
For information about calibration stages, refer to [UniPHY Calibration Stages](#) in section 1 of this volume.

## Troubleshooting

In the event of calibration failure, refer to [Figure 13–3](#) to assist in troubleshooting your design. Calibration results and failing stages are available through the external memory interface toolkit.

Figure 13–3 shows the recommended flow for troubleshooting calibration failure.

**Figure 13–3. Debugging Tips**



## EMIF On-Chip Debug Toolkit

The EMIF On-Chip Debug Toolkit allows user logic to access the same calibration data used by the EMIF Toolkit, and allows user logic to send commands to the sequencer.

### Introduction

You can use the EMIF On-Chip Debug Toolkit to access calibration data for your design and to send commands to the sequencer just as the EMIF Toolkit would. The following information is available:

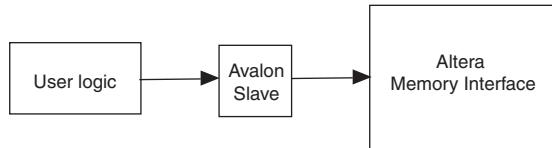
- Pass/fail status for each DQS group
- Read and write data valid windows for each group

In addition, user logic can request the following commands from the sequencer:

- Destructive recalibration of all groups
- Masking of groups and ranks
- Generation of per-DQ pin margining data as part of calibration

The user logic communicates through an Avalon-MM slave interface as shown in Figure 13–4.

**Figure 13–4. User Logic Access**



### Access Protocol

The EMIF On-Chip Debug Toolkit provides access to calibration data through an Avalon-MM slave interface. To send a command to the sequencer, user logic sends a command code to the command space in sequencer memory. The sequencer polls the command space for new commands after each group completes calibration, and continuously after overall calibration has completed.

The communication protocol to send commands from user logic to the sequencer uses a multistep handshake with a data structure as shown below, and an algorithm as shown in Figure 13–5.

```

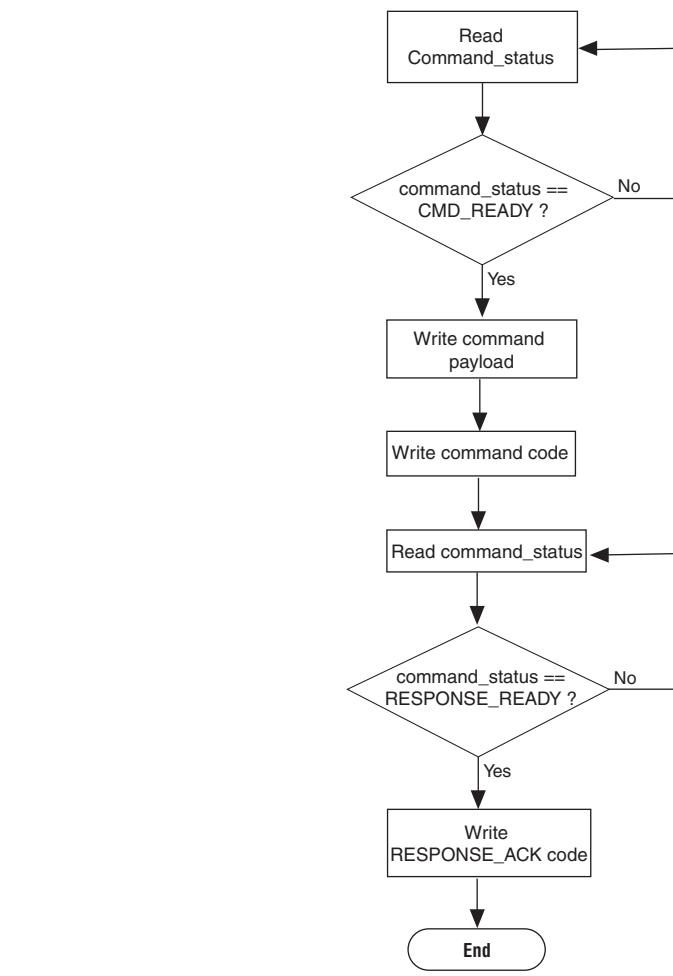
typedef struct_debug_data_struct {
    ...
    // Command interaction
    alt_u32 requested_command;
    alt_u32 command_status;
    alt_u32 command_parameters[COMMAND_PARAM_WORDS];
    ...
}
  
```

To send a command to the sequencer, user logic must first poll the `command_status` word for a value of `TCLDBG_TX_STATUS_CMD_READY`, which indicates that the sequencer is ready to accept commands. When the sequencer is ready to accept commands, user logic must write the command parameters into `command_parameters`, and then write the command code into `requested_command`.

The sequencer detects the command code and replaces `command_status` with `TCLDBG_TX_STATUS_CMD_EXE`, to indicate that it is processing the command. When the sequencer has finished running the command, it sets `command_status` to `TCLDBG_TX_STATUS_RESPONSE_READY` to indicate that the result of the command is available to be read. (If the sequencer rejects the requested command as illegal, it sets `command_status` to `TCLDBG_TX_STATUS_ILLEGAL_CMD`.)

User logic acknowledges completion of the command by writing `TCLDBG_CMD_RESPONSE_ACK` to `requested_command`. The sequencer responds by setting `command_status` back to `STATUS_CMD_READY`. (If an illegal command is received, it must be cleared using `CMD_RESPONSE_ACK`.)

**Figure 13–5. Debugging Algorithm Flowchart**



## Command Codes Reference

Table 13–1 lists the supported command codes for the On-Chip Debug Toolkit.

**Table 13–1. Supported Command Codes**

Command	Parameters	Description
TCLDBG_RUN_MEM_CALIBRATE	None	Runs the calibration routine.
TCLDBG_MARK_ALL_DQS_GROUPS_AS_VALID	None	Marks all groups as valid for calibration.
TCLDBG_MARK_GROUP_AS_SKIP	Group to skip	Mark the specified group to be skipped by calibration.
TCLDBG_MARK_ALL_RANKS_AS_VALID	None	Mark all ranks as valid for calibration
TCLDBG_MARK_RANK_AS_SKIP	Rank to skip	Mark the specified rank to be skipped by calibration.
TCLDBG_ENABLE_MARGIN_REPORT	None	Enables generation of the margin report.

## Header Files

The UniPHY-based external memory interface IP generates header files which identify the debug data structures and memory locations used with the EMIF On-Chip Debug Toolkit. You should refer to these header files for information required for use with your core user logic. It is highly recommended to use a software component (such as a Nios II processor) to access the calibration debug data.

The header files are unique to your IP parameterization and version, therefore you must ensure that you are referring to the correct version of header for your design. The names of the header files are: **core\_debug.h** and **core\_debugDefines.h**.

## Generating UniPHY IP With the Debug Port

The following steps summarize the procedure for implementing your IP with the EMIF On-Chip Debug Toolkit enabled:

1. Add the variable `alt_mem_if_enable_core_debug_data=1` to the **quartus.ini** file.
2. Start the Quartus II software and generate a new DDR3 external memory interface with UniPHY.
3. On the **Diagnostics** tab of the parameter editor, turn on **Enable core sequencer debug access**.
4. Ensure that the **Core sequencer debug interface type** is set to **Avalon-MM Slave**.
5. Click **Finish** to generate your IP.
6. Find the Avalon interface in the top-level generated file. Connect this interface to your debug component.

```

input wire [19:0] seq_debug_addr,      // seq_debug.address
input wire      seq_debug_read_req,   // .read
output wire [31:0] seq_debug_rdata,    // .readdata
input wire      seq_debug_write_req,   // .write
input wire [31:0] seq_debug_wdata,     // .writedata
output wire     seq_debug_waitrequest, // .waitrequest
input wire [3:0]  seq_debug_be,        // .byteenable
output wire     seq_debug_rdata_valid // .readdatavalid

```

7. Find the **core\_debug.h** and **core\_debugDefines.h** header files in `<design_name>/<design_name>_s0_software/sequencer` and include these files in your debug component code.
8. Write your debug component using the supported command codes, to read and write to the Avalon-MM interface.

The debug data structure resides at the memory address `SEQ_CORE_DEBUG_BASE`, which is defined in the **core\_debugDefines.h** header file.

## Example C Code for Accessing Debug Data

A typical use of the EMIF On-Chip Debug Toolkit might be to recalibrate the external memory interface, and then access the reports directly using the `summary_report_ptr`, `cal_report_ptr`, and `margin_report_ptr` pointers, which are part of the debug data structure.

The following code sample illustrates:

```

/*
 * DDR3 UniPHY sequencer core access example
 */

#include <stdio.h>
#include <unistd.h>
#include <io.h>
#include "core_debugDefines.h"

int send_command(volatile debug_data_t* debug_data_ptr, int command, int
args[], int num_args)
{
    volatile int i, response;

    // Wait until command_status is ready
    do {
        response = IORD_32DIRECT(&(debug_data_ptr->command_status), 0);
    } while(response != TCLDBG_TX_STATUS_CMD_READY);

    // Load arguments
    if(num_args > COMMAND_PARAM_WORDS)
    {

```

```

        // Too many arguments
        return 0;
    }
    for(i = 0; i < num_args; i++)
    {
        IOWR_32DIRECT(&(debug_data_ptr->command_parameters[i]), 0,
args[i]);
    }

    // Send command code
    IOWR_32DIRECT(&(debug_data_ptr->requested_command), 0, command);

    // Wait for acknowledgment
    do {
        response = IORD_32DIRECT(&(debug_data_ptr->
command_status), 0);
    } while(response != TCLDBG_TX_STATUS_RESPONSE_READY && response !=
TCLDBG_TX_STATUS_ILLEGAL_CMD);

    // Acknowledge response
    IOWR_32DIRECT(&(debug_data_ptr->requested_command), 0,
TCLDBG_CMD_RESPONSE_ACK);

    // Return 1 on success, 0 on illegal command
    return (response != TCLDBG_TX_STATUS_ILLEGAL_CMD);
}

int main()
{
    volatile debug_data_t* my_debug_data_ptr;
    volatile debug_summary_report_t* my_summary_report_ptr;
    volatile debug_cal_report_t* my_cal_report_ptr;
    volatile debug_margin_report_t* my_margin_report_ptr;

    volatile debug_cal_observed_dq_margins_t*
cal_observed_dq_margins_ptr;

    int i, j, size;
    int args[COMMAND_PARAM_WORDS];

    // Initialize pointers to the debug reports
    my_debug_data_ptr = (debug_data_t*)(SEQ_CORE_DEBUG_BASE);
    my_summary_report_ptr =
(debug_summary_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr->
summary_report_ptr), 0));
    my_cal_report_ptr =
(debug_cal_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr->
cal_report_ptr), 0));
}

```

```

        my_margin_report_ptr =
(debug_margin_report_t*)(IORD_32DIRECT(&(my_debug_data_ptr->
margin_report_ptr), 0));

        // Activate all groups and ranks
        send_command(my_debug_data_ptr,
TCLDBG_MARK_ALL_DQS_GROUPS_AS_VALID, 0, 0);
        send_command(my_debug_data_ptr, TCLDBG_MARK_ALL_RANKS_AS_VALID,
0, 0);
        send_command(my_debug_data_ptr, TCLDBG_ENABLE_MARGIN_REPORT, 0,
0);

        // Mask group 4
        args[0] = 4;
        send_command(my_debug_data_ptr, TCLDBG_MARK_GROUP_AS_SKIP,
args, 1);

        send_command(my_debug_data_ptr, TCLDBG_RUN_MEM_CALIBRATE, 0, 0);

        // SUMMARY
printf("SUMMARY REPORT\n");
printf("mem_address_width: %u\n",
IORD_32DIRECT(&(my_summary_report_ptr->mem_address_width), 0));
printf("mem_bank_width: %u\n",
IORD_32DIRECT(&(my_summary_report_ptr->mem_bank_width), 0));
        // etc...

        // CAL REPORT
printf("CALIBRATION REPORT\n");
// DQ read margins
for(i = 0; i < RW_MGR_MEM_DATA_WIDTH; i++)
{
    cal_observed_dq_margins_ptr = &(my_cal_report_ptr->
cal_dq_in_margins[i]);

        printf("0x%08x DQ %d Read Margin (taps): -%d : %d\n",
(unsigned int)cal_observed_dq_margins_ptr, i,
IORD_32DIRECT(&(cal_observed_dq_margins_ptr->
left_edge), 0),
IORD_32DIRECT(&(cal_observed_dq_margins_ptr->
right_edge), 0));
    }
    // etc...

    return 0;
}

```

## Document Revision History

Table 13–2 lists the revision history for this document.

**Table 13–2. Document Revision History**

Date	Version	Changes
November 2012	2.2	<ul style="list-style-type: none"><li>■ Changes to <a href="#">Setup and Use</a> and <a href="#">General Workflow</a> sections.</li><li>■ Added <a href="#">EMIF On-Chip Debug Toolkit</a> section</li><li>■ Changed chapter number from 11 to 13.</li></ul>
August 2012	2.1	<ul style="list-style-type: none"><li>■ Added table of debugging tips.</li></ul>
June 2012	2.0	<ul style="list-style-type: none"><li>■ Revised content for new UniPHY EMIF Toolkit.</li><li>■ Added Feedback icon.</li></ul>
November 2011	1.0	Harvested 11.0 <a href="#">DDR2 and DDR3 SDRAM Controller</a> with <a href="#">UniPHY EMIF Toolkit</a> content.

This chapter describes upgrading the following ALTMEMPHY-based controller designs to UniPHY-based controllers:

- DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY designs
- DDR2 or DDR3 SDRAM High-Performance Controller with ALTMEMPHY designs



Altera does not support upgrading designs that do not use the AFI.

If your design uses non-AFI IP cores, Altera recommends that you start a new design with the UniPHY IP core. In addition, Altera recommends that any new designs targeting Stratix III, Stratix IV, or Stratix V use the UniPHY datapath.

### Upgrading from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY Designs

To upgrade to the DDR2 or DDR3 SDRAM controller with UniPHY IP core from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY designs, follow these steps:

1. Generating Equivalent Design
2. Replacing the ALTMEMPHY Datapath with UniPHY Datapath
3. Resolving Port Name Differences
4. Creating OCT Signals
5. Running Pin Assignments Script
6. Removing Obsolete Files
7. Simulating your Design

The following sections describes these steps.

#### Generating Equivalent Design

Create a new DDR2 or DDR3 SDRAM controller with UniPHY IP core, by following the steps in volume 2, section 1, chapter 8, *Implementing and Parameterizing Memory IP* and use the following guidelines:

- Specify the same variation name as the ALTMEMPHY variation.

- Specify a directory different than the ALTMEMPHY design directory to prevent files from overwriting each other during generation.

To ease the migration process, ensure the UniPHY-based design you create is as similar as possible to the existing ALTMEMPHY-based design. In particular, you should ensure the following settings are the same in your UniPHY-based design:

- **PHY settings** tab
  - FPGA speed grade
  - PLL reference clock
  - Memory clock frequency
  - There is no need to change the default **Address and command clock phase settings**; however, if you have board skew effects in your ALTMEMPHY design, enter the difference between that clock phase and the default clock phase into the **Address and command clock phase settings**.
- **Memory Parameters** tab—all parameters must match.
- **Memory Timing** tab—all parameters must match.
- **Board settings** tab—all parameters must match.
- **Controller settings** tab—all parameters must match



In ALTMEMPHY-based designs you can turn off dynamic OCT. However, all UniPHY-based designs use dynamic parallel OCT and you cannot turn it off.

## Replacing the ALTMEMPHY Datapath with UniPHY Datapath

To replace the ALTMEMPHY datapath with the UniPHY datapath, follow these steps:

1. In the Quartus II software, open the Assignment Editor, on the Assignments menu click **Assignment Editor**.
2. Manually, delete all of the assignments related to the external memory interface pins, except for the location assignments if you are preserving the pinout. By default, these pin names start with the `mem` prefix, though in your design they may have a different name.
3. Remove the old ALTMEMPHY `.qip` file from the project:
  - On the Assignments menu click **Settings**.
  - Specify the old `.qip`, and click **Remove**.

Your design now uses the UniPHY datapath.

## Resolving Port Name Differences

Several port names in the ALTMEMPHY datapath are different than in the UniPHY datapath. The different names may cause compilation errors. This section describes the changes you must make in the RTL for the entity that instantiates the memory IP core. Each change applies to a specific port in the ALTMEMPHY datapath. Unconnected ports require no changes.

In some instances, multiple ports in ALTMEMPHY-based designs are mapped to a single port in UniPHY-based designs. If you use both ports in ALTMEMPHY-based designs, assign a temporary signal to the common port and connect it to the original wires. **Table 14–1** shows the changes you must make.

**Table 14–1. Changes to ALTMEMPHY Port Names**

ALTMEMPHY Port	Changes
aux_full_rate_clk	The UniPHY-based design does not generate this signal. You can generate it if you require it.
aux_scan_clk	The UniPHY-based design does not generate this signal. You can generate it if you require it.
aux_scan_clk_reset_n	The UniPHY-based design does not generate this signal. You can generate it if you require it.
dll_reference_clk	The UniPHY-based design does not generate this signal. You can generate it if you require it.
dqs_delay_ctrl_export	This signal is for DLL sharing between ALTMEMPHY instances and is not applicable for UniPHY-based designs.
local_address	Rename to avl_addr.
local_be	Rename to avl_be.
local_burstbegin	Rename to avl_burstbegin.
local_rdata	Rename to avl_rdata.
local_rdata_valid	Rename to avl_rdata_valid.
local_read_req	Rename to avl_read_req.
local_ready	Rename to avl_ready.
local_size	Rename to avl_size.
local_wdata	Rename to avl_wdata.
local_write_req	Rename to avl_write_req.
mem_addr	Rename to mem_a.
mem_clk	Rename to mem_ck.
mem_clk_n	Rename to mem_ck_n.
mem_dqsn	Rename to mem_dqs_n.
oct_ctl_rs_value	Remove from design (“ <a href="#">Creating OCT Signals</a> ” on page 14–4).
oct_ctl_rt_value	Remove from design (“ <a href="#">Creating OCT Signals</a> ” on page 14–4).
phy_clk	Rename to afi_clk.
reset_phy_clk_n	Rename to afi_reset_n.
local_refresh_ack local_wdata_req reset_request_n	The controller no longer exposes these signals to the top-level design, so comment out these outputs. If you need it, bring the wire out from the high-performance II controller entity in <i>&lt;project directory&gt;/uniphy/rtl/&lt;variation name&gt;_controller_phy.sv</i> .

## Creating OCT Signals

In ALTMEMPHY-based designs, the Quartus II Fitter creates the `alt_oct` block outside the IP core and connects it to the `oct_ctl_rs_value` and `oct_ctl_rt_value` signals. In UniPHY-based designs, the OCT block is part of the IP core, so the design no longer requires these two ports. Instead, the UniPHY-based design requires two additional ports, `oct_rup` and `oct_rdn`. You must create these ports in the instantiating entity as input pins and connect to the UniPHY instance. Then route these pins to the top-level design and connect to the OCT  $R_{UP}$  and  $R_{DOWN}$  resistors on the board.

For information on OCT control block sharing, refer to “[The OCT Sharing Interface](#)” in this volume.

## Running Pin Assignments Script

Remap your design by running analysis and synthesis. When analysis and synthesis completes, run the pin assignments Tcl script and then verify the new pin assignments in the Assignment Editor.

## Removing Obsolete Files

After you upgrade the design, you may remove the unnecessary ALTMEMPHY design files from your project. To identify these files, examine the original ALTMEMPHY-generated `.qip` file in any text editor.

## Simulating your Design

You must use the UniPHY memory model to simulate your new design. To use the UniPHY memory model, follow these steps:

1. Edit your instantiation of the UniPHY datapath to ensure the `local_init_done`, `local_cal_success`, `local_cal_fail`, `soft_reset_n`, `oct_rdn`, `oct_rup`, `reset_phy_clk_n`, and `phy_clk` signals are at the top-level entity so that an instantiating testbench can refer to those signals.
2. To use the UniPHY testbench and memory model, generate the example design when generating your IP instantiation.

3. Specify that your third-party simulator should use the UniPHY testbench and memory model instead of the ALTMEMPHY memory model:
  - a. On the Assignments menu, point to **Settings** and click the **Project Settings** window.
  - b. Select the **Simulation** tab, click **Test Benches**, click **Edit**, and replace the ALTMEMPHY testbench files with the following files:
    - `\<project directory>\<variation name>_example_design\simulation\verilog\submodules\altera_avalon_clock_source.sv`  
or  
`\<project directory>\<variation name>_example_design\simulation\vhdl\submodules\altera_avalon_clock_source.vhd`
    - `\<project directory>\<variation name>_example_design\simulation\verilog\submodules\altera_avalon_reset_source.sv`  
or  
`\<project directory>\<variation name>_example_design\simulation\vhdl\submodules\altera_avalon_reset_source.vhd`
    - `\<project directory>\<variation name>_example_design\simulation\verilog\<variation name>_example_sim.v`  
or  
`\uniphy\<variation name>_example_design\simulation\vhdl\<variation name>_example_sim.vhd`
    - `\<project directory>\<variation name>_example_design\simulation\verilog\submodules\verbosity_pkg.sv`
    - `\<project directory>\<variation name>_example_design\simulation\verilog\submodules\status_checker_no_ifdef_params.sv`  
or  
`\<project directory>\<variation name>_example_design\simulation\vhdl\submodules\status_checker_no_ifdef_params.sv`
    - `\<project directory>\<variation name>_example_design\simulation\verilog\submodules\alt_mem_if_common_ddr_mem_model_ddr3_mem_if_dm_pins_en_mem_if_dqsn_en.sv`  
or  
`\<project directory>\<variation name>_example_design\simulation\vhdl\submodules\alt_mem_if_common_ddr_mem_model_ddr3_mem_if_dm_pins_en_mem_if_dqsn_en.sv`
    - `\<project directory>\<variation name>_example_design\simulation\verilog\submodules\alt_mem_if_ddr3_mem_model_top_ddr3_mem_if_dm_pins_en_mem_if_dqsn_en`  
or  
`\<project directory>\<variation name>_example_design\simulation\vhdl\submodules\alt_mem_if_ddr3_mem_model_top_ddr3_mem_if_dm_pins_en_mem_if_dqsn_en`
4. Open the `<variation name>_example_sim.v` file and find the UniPHY-generated simulation example design module name below: `<variation name>_example_sim_e0`.
5. Change the module name above to the name of your top-level design module.

6. Update the following port names of the example design in the UniPHY-generated `<variation name>_example_sim.v` file. ([Table 14-2](#)).

**Table 14-2. Example Design Port Names**

Example Design Name	New Name
<code>pll_ref_clk</code>	Rename to <code>clock_source</code> .
<code>mem_a</code>	Rename to <code>mem_addr</code> .
<code>mem_ck</code>	Rename to <code>mem_clk</code> .
<code>mem_ck_n</code>	Rename to <code>mem_clk_n</code> .
<code>mem_dqs_n</code>	Rename to <code>mem_dqsn</code> .
<code>drv_status_pass</code>	Rename to <code>pnf</code> .
<code>afi_clk</code>	Rename to <code>phy_clk</code> .
<code>afi_reset_n</code>	Rename to <code>reset_phy_clk_n</code> .
<code>drv_status_fail</code>	This signal is not available, so comment out this output.
<code>afi_half_clk</code>	This signal is not exposed to the top-level design, so comment out this output.



For more information about generating example simulation files, refer to [Simulating Memory IP](#), in volume 2 of the *External Memory Interface Handbook*.

## Document Revision History

[Table 14-3](#) lists the revision history for this document.

**Table 14-3. Document Revision History**

Date	Version	Changes
November 2012	2.3	Changed chapter number from 12 to 14.
June 2012	2.2	Added Feedback icon.
November 2011	2.1	Revised <a href="#">Simulating your Design</a> section.

This section provides reference information about the ALTMEMPHY-based external memory interface IP.

This section includes the following chapters:

- [Chapter 15, Introduction to ALTMEMPHY IP](#)
- [Chapter 16, Latency for ALTMEMPHY IP](#)
- [Chapter 17, Timing Diagrams for ALTMEMPHY IP](#)
- [Chapter 18, ALTMEMPHY External Memory Interface Debug Toolkit](#)



For information about the revision history for chapters in this section, refer to “Document Revision History” in each individual chapter.

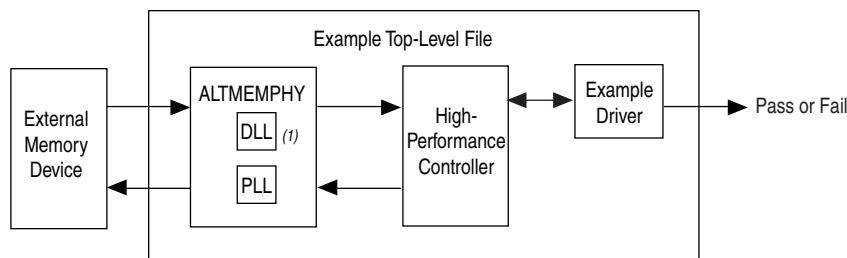


The Altera® DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP provide simplified interfaces to industry-standard DDR, DDR2, and DDR3 SDRAM. The ALTMEMPHY megafunction is an interface between a memory controller and the memory devices, and performs read and write operations to the memory. The DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP work in conjunction with the Altera ALTMEMPHY megafunction.

The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP and ALTMEMPHY megafunction offer full-rate or half-rate DDR and DDR2 SDRAM interfaces. The DDR3 SDRAM Controller with ALTMEMPHY IP and ALTMEMPHY megafunction support DDR3 SDRAM interfaces in half-rate mode. The DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP offer the high-performance controller II (HPC II), which provides high efficiency and advanced features.

**Figure 15–1** shows a system-level diagram including the example top-level file that the DDR, DDR2, or DDR3 SDRAM Controller with ALTMEMPHY IP creates for you.

**Figure 15–1. System-Level Diagram**



**Note to Figure 15–1:**

- (1) When you choose **Instantiate DLL Externally**, delay-locked loop (DLL) is instantiated outside the ALTMEMPHY megafunction.

The MegaWizard™ Plug-In Manager generates an example top-level file, consisting of an example driver, and your DDR, DDR2, or DDR3 SDRAM high-performance controller custom variation. The controller instantiates an instance of the ALTMEMPHY megafunction which in turn instantiates a phase-locked loop (PLL) and DLL. You can also instantiate the DLL outside the ALTMEMPHY megafunction to share the DLL between multiple instances of the ALTMEMPHY megafunction. You cannot share a PLL between multiple instances of the ALTMEMPHY megafunction, but you may share some of the PLL clock outputs between these multiple instances.

The example top-level file is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass or fail, and test complete signals.

The ALTMEMPHY megafunction creates the datapath between the memory device and the memory controller. The megafunction is available as a stand-alone product or can be used in conjunction with the Altera high-performance memory controller. When using the ALTMEMPHY megafunction as a stand-alone product, use with either custom or third-party controllers.



For new designs, Altera recommends using a UniPHY-based external memory interface, such as the DDR2 and DDR3 SDRAM controllers with UniPHY, QDR II and QDR II+ SRAM controllers with UniPHY, or RLDRAM II controller with UniPHY.

## Release Information

**Table 15–1** provides information about this release of the DDR3 SDRAM Controller with ALTMEMPHY IP.

**Table 15–1. Release Information**

Item	Description
Version	11.1
Release Date	November 2011
Ordering Codes	IP-SDRAM/HPDDR (DDR SDRAM HPC) IP-SDRAM/HPDDR2 (DDR2 SDRAM HPC) IP-HPMCII (HPC II)
Product IDs	00BE (DDR SDRAM) 00BF (DDR2 SDRAM) 00C2 (DDR3 SDRAM) 00CO (ALTMEMPHY Megafunction)
Vendor ID	6AF7

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release. For information about issues on the DDR, DDR2, or DDR3 SDRAM high-performance controller and the ALTMEMPHY megafunction in a particular Quartus II version, refer to the *Quartus II Software Release Notes*.

## Device Family Support

Table 15–2 defines the device support levels for Altera IP cores.

**Table 15–2. Altera IP Core Device Support Levels**

FPGA Device Families	HardCopy Device Families
<b>Preliminary support</b> —The IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.	<b>HardCopy Companion</b> —The IP core is verified with preliminary timing models for the HardCopy companion device. The IP core meets all functional requirements, but might still be undergoing timing analysis for the HardCopy device family. It can be used in production designs with caution.
<b>Final support</b> —The IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.	<b>HardCopy Compilation</b> —The IP core is verified with final timing models for the HardCopy device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 15–3 shows the level of support offered by the DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP for Altera device families.

**Table 15–3. Device Family Support**

Device Family	Protocol	
	DDR and DDR2	DDR3
Arria® GX	Final	No support
Arria II GX	Final	Final
Cyclone® III	Final	No support
Cyclone III LS	Final	No support
Cyclone IV E	Final	No support
Cyclone IV GX	Final	No support
HardCopy II	Refer to the <a href="#">What's New in Altera IP</a> page of the Altera website.	No support
Stratix® II	Final	No support
Stratix II GX	Final	No support
Other device families	No support	No support

## Features

### ALTMEMPHY Megafunction

Table 15–4 summarizes key feature support for the ALTMEMPHY megafunction.

**Table 15–4. ALTMEMPHY Megafunction Feature Support**

Feature	DDR and DDR2	DDR3
Support for the Altera PHY Interface (AFI) on all supported devices.	✓	✓
Automated initial calibration eliminating complicated read data timing calculations.	✓	✓
Voltage and temperature (VT) tracking that guarantees maximum stable performance for DDR, DDR2, and DDR3 SDRAM interfaces.	✓	✓
Self-contained datapath that makes connection to an Altera controller or a third-party controller independent of the critical timing paths.	✓	✓
Full-rate interface	✓	—
Half-rate interface	✓	✓
Easy-to-use parameter editor	✓	✓

In addition, the ALTMEMPHY megafunction supports DDR3 SDRAM components without leveling:

- The ALTMEMPHY megafunction supports DDR3 SDRAM components without leveling for Arria II GX devices using T-topology for clock, address, and command bus:
  - Supports multiple chip selects.
- The DDR3 SDRAM PHY without leveling  $f_{MAX}$  is 400 MHz for single chip selects.
- No support for data-mask (DM) pins for  $\times 4$  DDR3 SDRAM DIMMs or components, so select **No** for **Drive DM pins from FPGA** when using  $\times 4$  devices.
- The ALTMEMPHY megafunction supports half-rate DDR3 SDRAM interfaces only.

### High-Performance Controller II

Table 15–5 summarizes key feature support for the DDR, DDR2, and DDR3 SDRAM HPC II.

**Table 15–5. Feature Support (Part 1 of 2)**

Feature	DDR and DDR2	DDR3
Half-rate controller	✓	✓
Support for AFI ALTMEMPHY	✓	✓
Support for Avalon®Memory Mapped (Avalon-MM) local interface	✓	✓

**Table 15–5. Feature Support (Part 2 of 2)**

Feature	DDR and DDR2	DDR3
Configurable command look-ahead bank management with in-order reads and writes	✓	✓
Additive latency	✓	✓
Support for arbitrary Avalon burst length	✓	✓
Built-in flexible memory burst adapter	✓	✓
Configurable Local-to-Memory address mappings	✓	✓
Optional run-time configuration of size and mode register settings, and memory timing	✓	✓
Partial array self-refresh (PASR)	✓	✓
Support for industry-standard DDR3 SDRAM devices		
Optional support for self-refresh command	✓	✓
Optional support for user-controlled power-down command	✓	✓
Optional support for automatic power-down command with programmable time-out	✓	✓
Optional support for auto-precharge read and auto-precharge write commands	✓	✓
Optional support for user-controller refresh	✓	✓
Optional multiple controller clock sharing in SOPC Builder Flow		
Integrated error correction coding (ECC) function 72-bit	✓	✓
Integrated ECC function, 16, 24, and 40-bit	✓	✓
Support for partial-word write with optional automatic error correction	✓	✓
SOPC Builder ready		
Support for OpenCore Plus evaluation	✓	✓
IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulator	✓	✓

**Notes to Table 15–5:**

- (1) HPC II supports additive latency values greater or equal to  $t_{RCD}-1$ , in clock cycle unit ( $t_{CK}$ ).
- (2) This feature is not supported with DDR3 SDRAM with leveling.

## Unsupported Features

**Table 15–6** summarizes unsupported features for Altera’s ALTMEMPHY-based external memory interfaces.

**Table 15–6. Unsupported Features**

Memory Protocol	Unsupported Feature
DDR and DDR2 SDRAM	Timing simulation
	Burst length of 2
	Partial burst and unaligned burst in ECC and non-ECC mode when DM pins are disabled
DDR3 SDRAM	Timing simulation
	Partial burst and unaligned burst in ECC and non-ECC mode when DM pins are disabled
	Stratix III and Stratix IV
	DIMM support
	Full-rate interfaces

## MegaCore Verification

Altera performs extensive random, directed tests with functional test coverage using industry-standard Denali models to ensure the functionality of the DDR, DDR2, and DDR3 SDRAM Controllers with ALTMEMPHY IP.

## Resource Utilization

This section provides typical resource utilization information for the external memory controllers with ALTMEMPHY for supported device families. This information is provided as a guideline only; for precise resource utilization data, you should generate your IP core and refer to the reports generated by the Quartus II software.

**Table 15–7** shows resource utilization data for the ALTMEMPHY megafunction, and the DDR3 high-performance controller II for Arria II GX devices.

**Table 15–7. Resource Utilization in Arria II GX Devices (Part 1 of 2)**

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
<b>Controller</b>							
DDR3 (Half rate)	8	1,883	1,505	10	2	0	4,352
	16	1,893	1,505	10	4	0	8,704
	64	1,946	1,521	18	15	0	34,560
	72	1,950	1,505	10	17	0	39,168

**Table 15–7. Resource Utilization in Arria II GX Devices (Part 2 of 2)**

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
<b>Controller+PHY</b>							
DDR3 (Half rate)	8	3,389	2,760	12	4	0	4,672
	16	3,457	2,856	12	7	0	9,280
	64	3,793	3,696	20	24	0	36,672
	72	3,878	3,818	12	26	0	41,536

Table 15–8 shows resource utilization data for the DDR2 high-performance controller and controller plus PHY, for half-rate and full-rate configurations for Arria II GX devices.

**Table 15–8. DDR2 Resource Utilization in Arria II GX Devices**

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
<b>Controller</b>							
DDR2 (Half rate)	8	1,971	1,547	10	2	0	4,352
	16	1,973	1,547	10	4	0	8,704
	64	2,028	1,563	18	15	0	34,560
	72	2,044	1,547	10	17	0	39,168
DDR2 (Full rate)	8	2,007	1,565	10	2	0	2,176
	16	2,013	1,565	10	2	0	4,352
	64	2,022	1,565	10	8	0	17,408
	72	2,025	1,565	10	9	0	19,584
<b>Controller+PHY</b>							
DDR2 (Half rate)	8	3,481	2,722	12	4	0	4,672
	16	3,545	2,862	12	7	0	9,280
	64	3,891	3,704	20	24	0	36,672
	72	3,984	3,827	12	26	0	41,536
DDR2 (Full rate)	8	3,337	2,568	29	2	0	2,176
	16	3,356	2,558	11	4	0	4,928
	64	3,423	2,836	31	12	0	19,200
	72	3,445	2,827	11	14	0	21,952

Table 15–9 shows resource utilization data for the DDR2 high-performance controller and controller plus PHY, for half-rate and full-rate configurations for Cyclone III devices.

**Table 15–9. DDR2 Resource Utilization in Cyclone III Devices**

Protocol	Memory Width (Bits)	Logic Registers	Logic Cells	M9K Blocks	Memory (Bits)
<b>Controller</b>					
DDR2 (Half rate)	8	1,513	3,015	4	4,464
	16	1,513	3,034	6	8,816
	64	1,513	3,082	18	34,928
	72	1,513	3,076	19	39,280
DDR2 (Full rate)	8	1,531	3,059	4	2,288
	16	1,531	3,108	4	4,464
	64	1,531	3,134	10	17,520
	72	1,531	3,119	11	19,696
<b>Controller+PHY</b>					
DDR2 (Half rate)	8	2,737	5,131	6	4,784
	16	2,915	5,351	9	9,392
	64	3,969	6,564	27	37,040
	72	4,143	6,786	28	41,648
DDR2 (Full rate)	8	2,418	4,763	6	2,576
	16	2,499	4,919	6	5,008
	64	2,957	5,505	15	19,600
	72	3,034	5,608	16	22,032

## System Requirements

The DDR3 SDRAM Controller with ALTMEMPHY IP is a part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, [www.altera.com](http://www.altera.com).

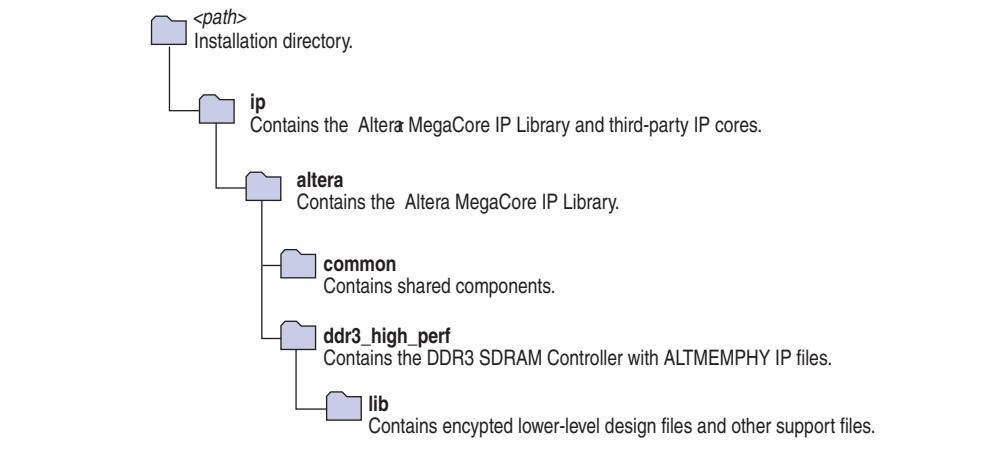


For system requirements and installation instructions, refer to *Altera Software Installation & Licensing*.

## Installation and Licensing

Figure 15–2 shows the directory structure after you install the DDR3 SDRAM Controller with ALTMEMPHY IP, where *<path>* is the installation directory. The default installation directory on Windows is `c:\altera\<version>`; on Linux it is `/opt/altera<version>`.

**Figure 15–2. Directory Structure**



You need a license for the MegaCore function only when you are completely satisfied with its functionality and performance, and want to take your design to production.

To use the DDR3 SDRAM HPC, you can request a license file from the Altera web site at [www.altera.com/licensing](http://www.altera.com/licensing) and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local representative.

To use the DDR3 SDRAM HPC II, contact your local sales representative to order a license.

## Free Evaluation

Altera's OpenCore Plus evaluation feature is only applicable to the DDR3 SDRAM HPC. With the OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP<sup>SM</sup> megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include MegaCore functions.
- Program a device and verify your design in hardware.

You need to purchase a license for the megafunction only when you are completely satisfied with its functionality and performance, and want to take your design to production.

## OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation can support the following two modes of operation:

- Untethered—the design runs for a limited time
- Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time-out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.



For MegaCore functions, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

Your design stops working after the hardware evaluation time expires and the `local_ready` output goes low.

## Document Revision History

Table 15–10 lists the revision history for this document.

**Table 15–10. Document Revision History**

Date	Version	Changes
November 2012	1.2	Changed chapter number from 13 to 15.
June 2012	1.1	Added Feedback icon.
November 2011	1.0	Combined <a href="#">Release Information</a> , <a href="#">Device Family Support</a> , <a href="#">Features</a> list, and <a href="#">Unsupported Features</a> list for DDR, DDR2, and DDR3.

Latency is defined using the local (user) side frequency and absolute time (ns). There are two types of latencies that exists while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.



For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.

Altera defines read and write latencies in terms of the local interface clock frequency and by the absolute time for the memory controllers. These latencies apply to supported device families with half-rate and full-rate HPC II memory controllers.

The latency defined in this section uses the following assumptions:

- The row is already open, there is no extra bank management needed.
- The controller is idle, there is no queued transaction pending, indicated by the `local_ready` signal asserted high.
- No refresh cycles occur before the transaction.

## Latency Stages

The latency for the high-performance controller II comprises many different stages of the memory interface. Figure 16–1 shows a typical memory interface read latency path showing read latency from the time a `local_read_req` signal assertion is detected by the controller up to data available to be read from the dual-port RAM (DPRAM) module.

**Figure 16–1. Typical Latency Path**

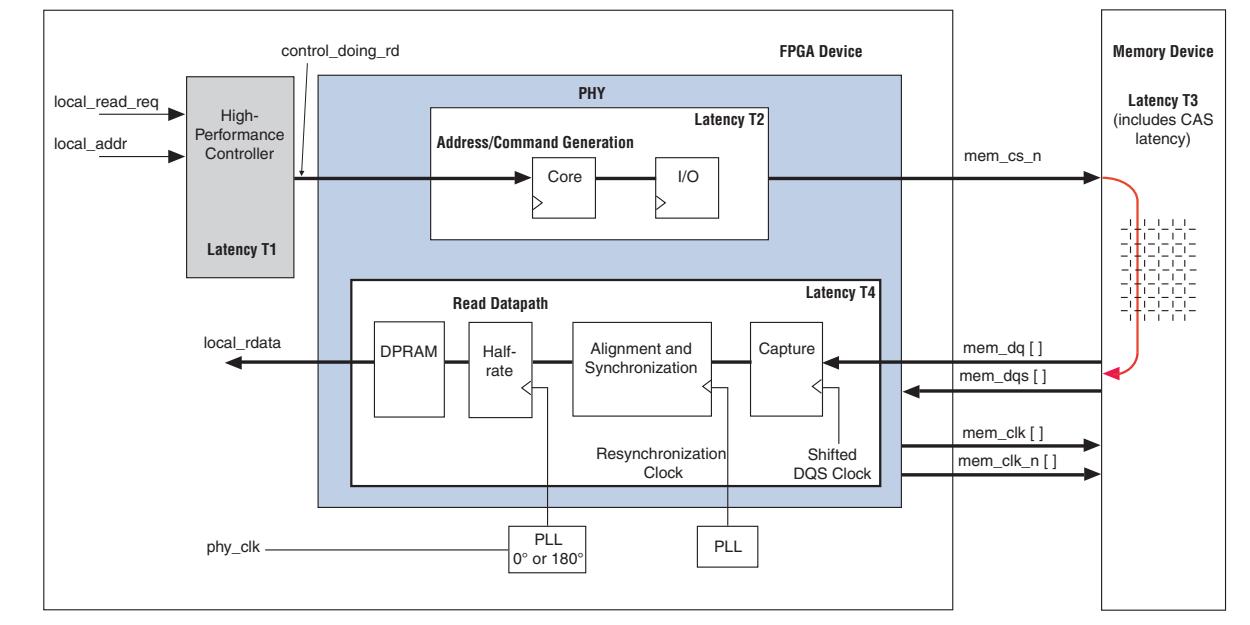


Table 16–1 shows the different stages that make up the whole read and write latency that Figure 16–1 shows.

**Table 16–1. High-Performance Controller Latency Stages and Descriptions**

Latency Number	Latency Stage	Description
T1	Controller	<code>local_read_req</code> or <code>local_write_req</code> signal assertion to <code>ddr_cs_n</code> signal assertion.
T2	Command Output	<code>ddr_cs_n</code> signal assertion to <code>mem_cs_n</code> signal assertion.
T3	CAS or WL	Read command to DQ data from the memory or write command to DQ data to the memory.
T4	ALTMEMPHY read data input	Read data appearing on the local interface.
T2 + T3	Write data latency	Write data appearing on the memory interface.

From Figure 16–1, the read latency in the high-performance controllers is made up of four components:

$$\text{read latency} = \text{controller latency (T1)} + \text{command output latency (T2)} + \text{CAS latency (T3)} + \text{PHY read data input latency (T4)}$$

Similarly, the write latency in the high-performance controller II is made up of three components:

$$\text{write latency} = \text{controller latency (T1)} + \text{write data latency (T2+T3)}$$

You can separate the controller and ALTMEMPHY read data input latency into latency that occurred in the I/O element (IOE) and latency that occurred in the FPGA fabric.

**Table 16–2** shows the minimum and maximum supported CAS latency for the DDR and DDR2 SDRAM high-performance controller II.

**Table 16–2. Supported CAS Latency** <sup>(1)</sup>

<b>Device Family</b>	<b>Minimum Supported CAS Latency</b>		<b>Maximum Supported CAS Latency</b>	
	<b>DDR</b>	<b>DDR2</b>	<b>DDR</b>	<b>DDR2</b>
Arria GX	3.0	3.0	3.0	6.0
Arria II GX	3.0	3.0	3.0	6.0
Cyclone III	2.0	3.0	3.0	6.0
Cyclone IV	2.0	3.0	3.0	6.0
HardCopy III	3.0	3.0	3.0	6.0
HardCopy IV	3.0	3.0	3.0	6.0
Stratix II	3.0	3.0	3.0	6.0
Stratix III	3.0	3.0	3.0	6.0
Stratix IV	3.0	3.0	3.0	6.0

**Note to Table 16–2:**

- (1) The registered DIMMs, where supported, effectively introduce one extra cycle of CAS latency. For the registered DIMMs, you need to subtract 1.0 from the CAS figures to determine the minimum supported CAS latency, and add 1.0 to the CAS figures to determine the maximum supported CAS latency.

**Table 16–3** and **Table 16–4** show a typical latency that can be achieved in Arria GX, Arria II GX, Cyclone III, Cyclone IV, Stratix IV, Stratix III, Stratix II, and Stratix II GX devices. The exact latency for your memory controller depends on your precise configuration. You can obtain precise latency from simulation, but this figure can vary slightly in hardware because of the automatic calibration process.

The actual memory CAS and write latencies shown are halved in half-rate designs as the latency calculation is based on the local clock.

The read latency also depends on your board trace delay. The latency found in simulation can be different from that found in board testing as functional simulation does not take into account the board trace delays. For a given design on a given board, the latency may change by one clock cycle (for full-rate designs) or two clock cycles (for half-rate designs) upon resetting the board. Different boards could also show different latencies even with the same design.

The CAS and write latencies are different between DDR and DDR2 SDRAM interfaces. To calculate latencies for DDR SDRAM interfaces, use the numbers from DDR2 SDRAM listed below and replace the CAS and write latency with the DDR SDRAM values.

**Table 16–3. Typical Read Latency in HPC II (1), (2)**

Device	Frequency (MHz)	Interface	Controller Latency (3)	Address and Command Latency		CAS Latency (4)	Read Data Latency		Total Read Latency (5)	
				FPGA	I/O		FPGA	I/O	Local Clock Cycles	Time (ns)
Arria GX	233	Half-rate	5	3	1	2	4.5	1	18	154
	167	Full-rate	5	2	1	4	5	1	19	114
Arria II GX	233	Half-rate	5	3	1	2.5	5.5	1	18	154
	167	Full-rate	5	2	1	4	6	1	20	120
Cyclone III and Cyclone IV	200	Half-rate	5	3	1	2	4.5	1	18	180
	167	Full-rate	5	2	1	4	5	1	19	114
Stratix II and Stratix II GX	333	Half-rate	5	3	1	2	4.5	1	18	108
	267	Half-rate	5	3	1	2	4.5	1	18	135
	200	Full-rate	5	2	1	4	5	1	19	95
Stratix III and Stratix IV	400	Half-rate	5	3	1	2.5	7.125	1.5	20	100
	267	Full-rate	4	2	1.5	4	7	1	20	75

**Notes to Table 16–3:**

- (1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.
- (2) Numbers shown may have been rounded up to the nearest higher integer.
- (3) The controller latency value is from the Altera high-performance controller.
- (4) CAS latency is per memory device specification and is programmable in the MegaWizard Plug-In Manager.
- (5) Total read latency is the sum of controller, address and command, CAS, and read data latencies.

**Table 16–4. Typical Write Latency in HPC II (1), (2) (Part 1 of 2)**

Device	Frequency (MHz)	Interface	Controller Latency (3)	Address and Command Latency		Memory Write Latency (4)	Total Write Latency (5)	
				FPGA	I/O		Local Clock Cycles	Time (ns)
Arria GX	233	Half-rate	5	3	1	1.5	12	103
	167	Full-rate	5	2	1	3	12	72
Arria II GX	233	Half-rate	5	3	1	2.5	12	103
	167	Full-rate	5	2	1	4	12	72
Cyclone III and Cyclone IV	200	Half-rate	5	3	1	1.5	12	120
	167	Full-rate	5	2	1	3	12	72

**Table 16–4. Typical Write Latency in HPC II (1), (2) (Part 2 of 2)**

Device	Frequency (MHz)	Interface	Controller Latency (3)	Address and Command Latency		Memory Write Latency (4)	Total Write Latency (5)	
				FPGA	I/O		Local Clock Cycles	Time (ns)
Stratix II and Stratix II GX	333	Half-rate	5	3	1	1.5	12	72
	267	Half-rate	5	3	1	1.5	12	90
	200	Full-rate	5	2	1	3	12	60
Stratix III and Stratix IV	400	Half-rate	5	3	1	2	12	60
	267	Full-rate	5	2	1.5	3	13	49

**Notes to Table 16–4:**

- (1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.
- (2) Numbers shown may have been rounded up to the nearest higher integer.
- (3) The controller latency value is from the Altera high-performance controller.
- (4) Memory write latency is per memory device specification. The latency from when you provide the command to write to when you need to provide data at the memory device.
- (5) Total write latency is the sum of controller, address and command, and memory write latencies.



To see the latency incurred in the IOE for both read and write paths for ALTMEMPHY variations in Stratix IV and Stratix III devices refer to the IOE figures in the *External Memory Interfaces in Stratix III Devices* chapter of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter of the *Stratix IV Device Handbook*.

## Document Revision History

Table 16–5 lists the revision history for this document.

**Table 16–5. Document Revision History**

Date	Version	Changes
November 2012	1.2	Changed chapter number from 14 to 16.
June 2012	1.1	Added Feedback icon.
November 2011	1.0	Consolidated latency information from <b>11.0 DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide</b> , and <b>DDR3 SDRAM Controller with ALTMEMPHY IP User Guide</b> .



This chapter shows timing diagrams for the DDR, DDR2, and DDR3 SDRAM high-performance controllers II (HPC II).

## DDR and DDR2 High-Performance Controllers II

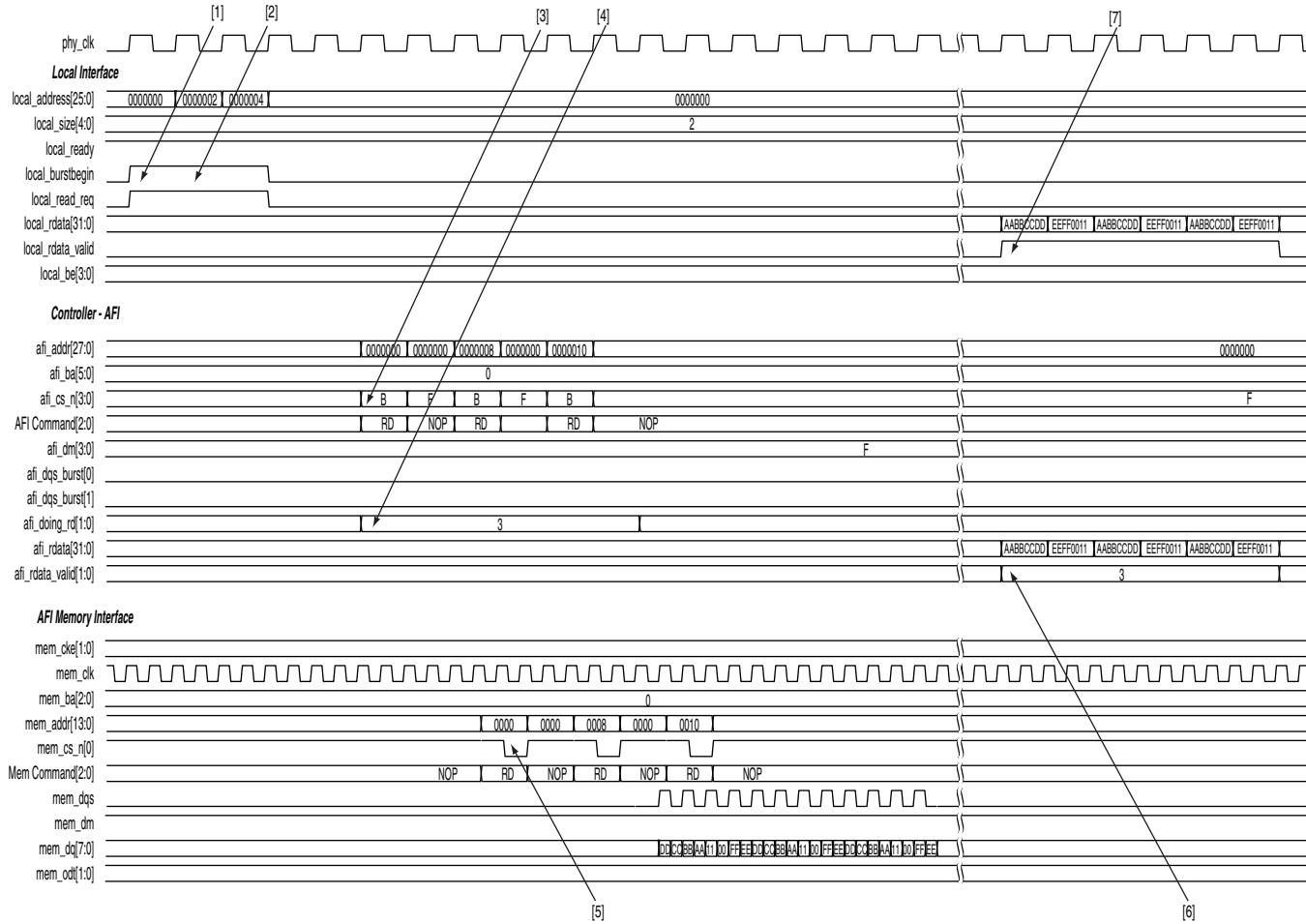
This section discusses the following timing diagrams for the DDR and DDR2 HPC II:

- “Half-Rate Read”
- “Half-Rate Write”
- “Full-Rate Read”
- “Full-Rate Write”



## Half-Rate Read

**Figure 17-1. Half-Rate Read Operation for HPC II**



The following sequence corresponds with the numbered items in [Figure 17-1](#):

1. The user logic requests the first read by asserting the local\_read\_req signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address 0x000000. This local address is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x000000  
mem_col_address = 0x0000  
mem_bank_address = 0x00
```

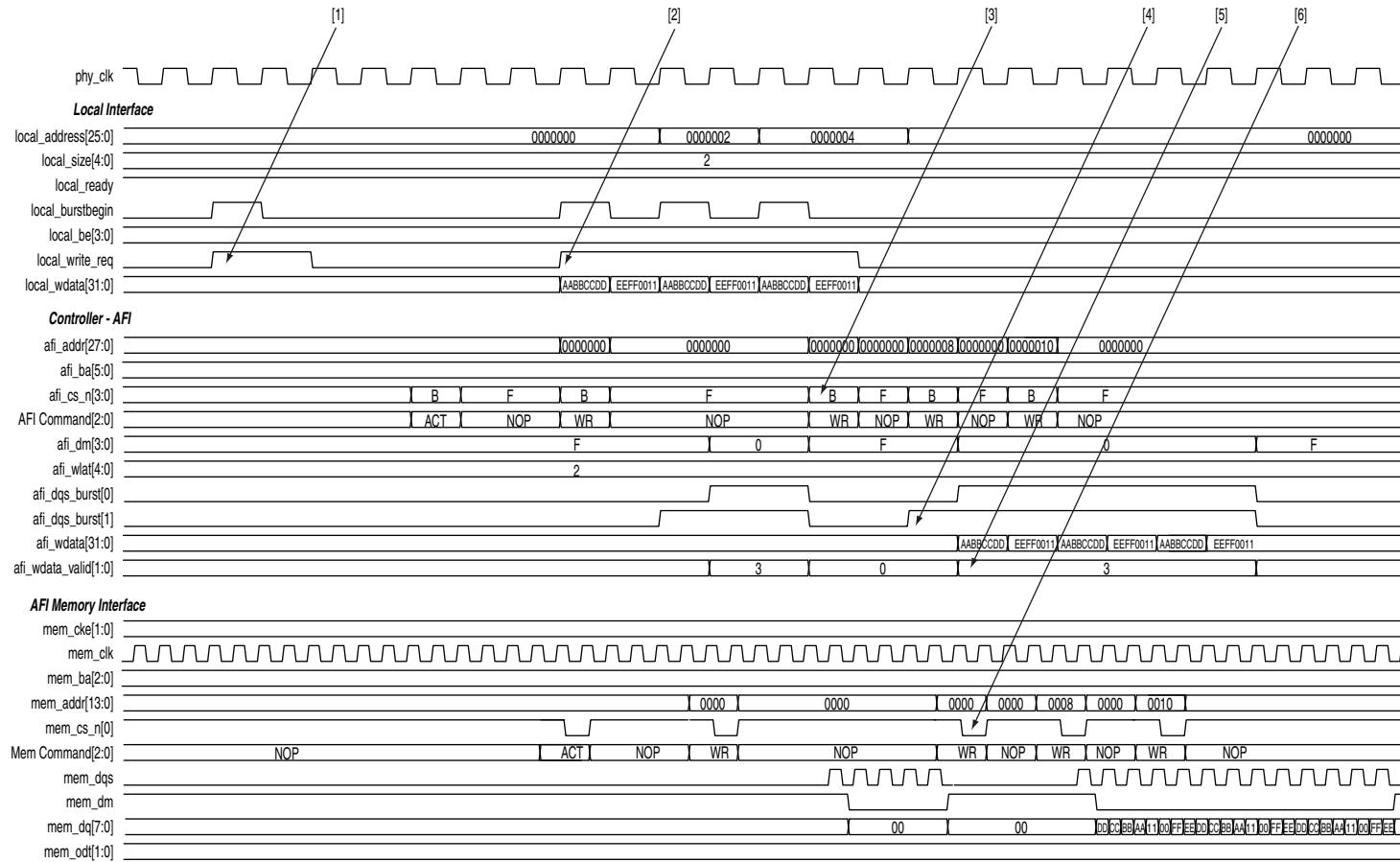
2. The user logic initiates a second read to a different memory column within the same row. The request for the second write is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the local\_ready signal. The starting local address 0x000002 is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000  
mem_col_address = 0x0002<<2 = 0x0008  
mem_bank_address = 0x00
```

3. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the afi\_doing\_rd signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the afi\_doing\_rd signal to enable its capture registers for the expected duration of memory burst.
5. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
6. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the phy\_clk domain, by asserting the afi\_rdata\_valid signal when there is valid read data on the afi\_rdata bus.
7. The controller returns the first read data to the user by asserting the local\_rdata\_valid signal when there is valid read data on the local\_rdata bus. If the ECC logic is disabled, there is no delay between the afi\_rdata and the local\_rdata buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the afi\_rdata and local\_rdata buses.

## Half-Rate Write

**Figure 17–2. Half-Rate Write Operation for HPC II**

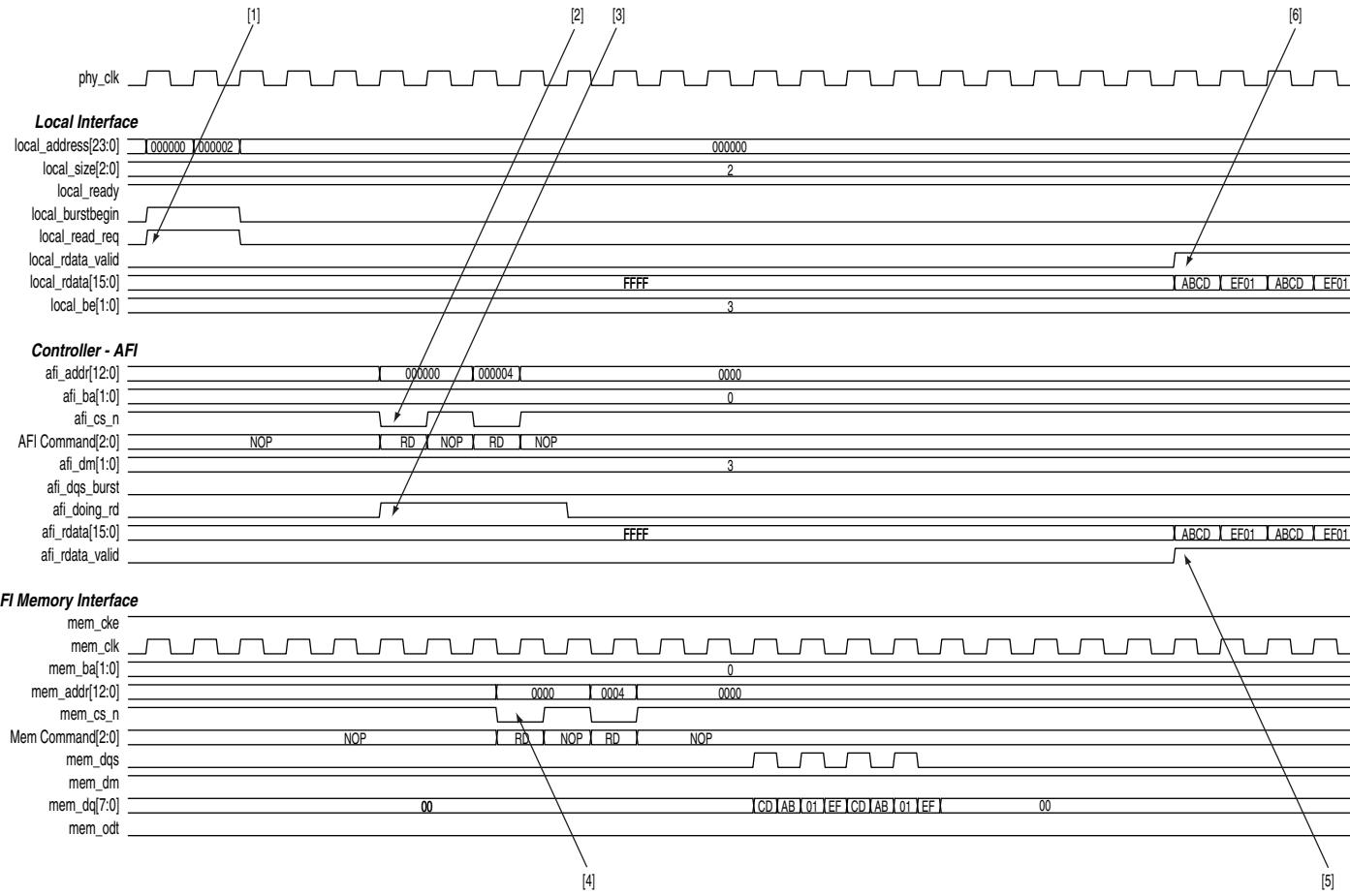


The following sequence corresponds with the numbered items in [Figure 17-2](#):

1. The user logic asserts the first write request to row 0 so that row 0 is open before the next transaction.
2. The user logic asserts a second `local_write_req` signal with size of 2 and address of 0 (`col = 0`, `row = 0`, `bank = 0`, `chip = 0`). The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

## Full-Rate Read

**Figure 17-3. Full-Rate Read Operation for HPC II**



The following sequence corresponds with the numbered items in [Figure 17-3](#):

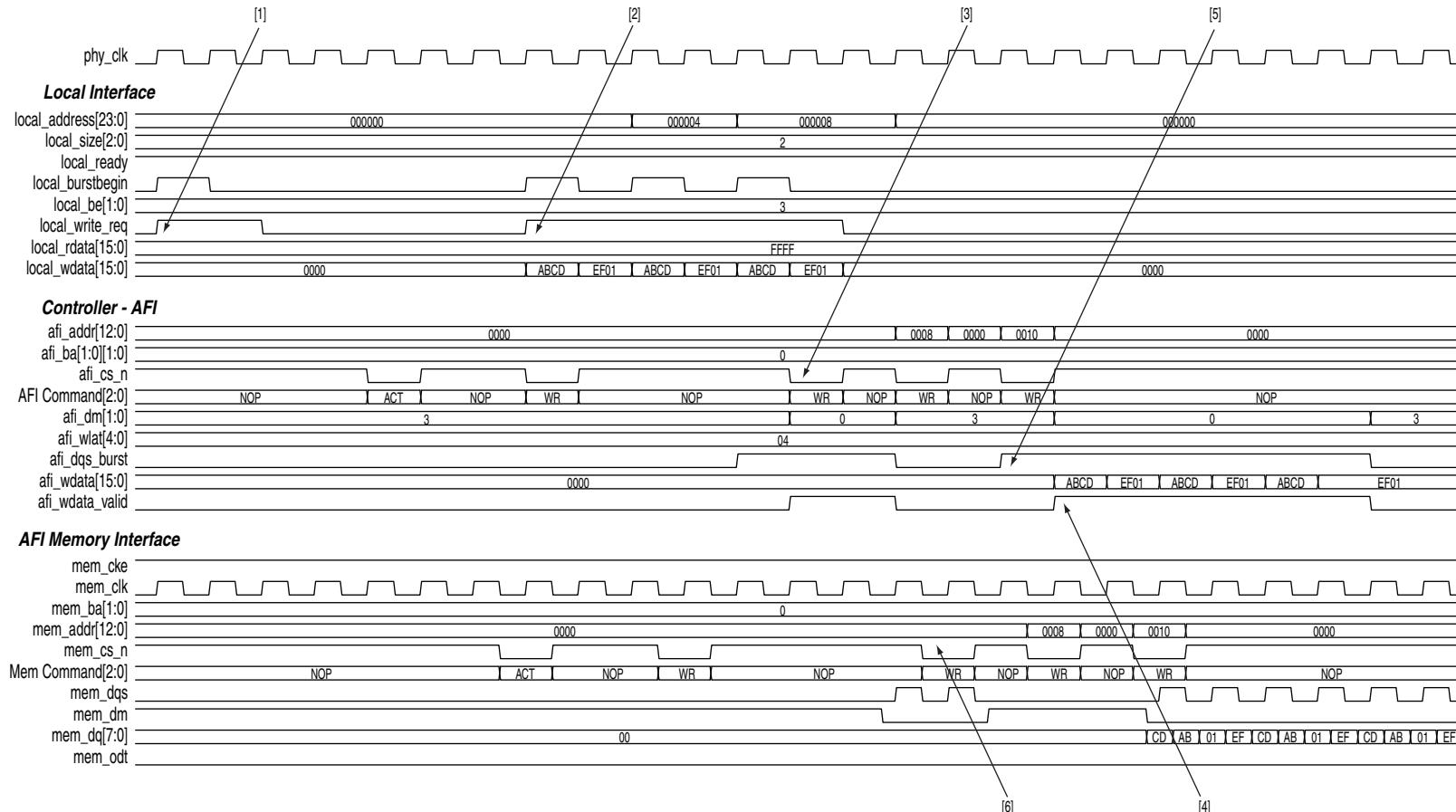
1. The user logic requests the first read by asserting local\_read\_req signal, and the size and address for this read. In this example, the request is a burst length of 2 to a local address 0x000000. This local address is mapped to the following memory address in full-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x0000<<2 = 0x0000
mem_bank_address = 0x00
```

2. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the afi\_doing\_rd signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the afi\_doing\_rd signal to enable its capture registers for the expected duration of memory burst.
4. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
5. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the phy\_clk domain, by asserting the afi\_rdata\_valid signal when there is valid read data on the afi\_rdata bus.
6. The controller returns the first read data to the user by asserting the local\_rdata\_valid signal when there is valid read data on the local\_rdata bus. If the ECC logic is disabled, there is no delay between the afi\_rdata and the local\_rdata buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the afi\_rdata and local\_rdata buses.

## Full-Rate Write

**Figure 17–4. Full-Rate Write Operation for HPC II**



The following sequence corresponds with the numbered items in [Figure 17-4](#):

1. The user logic asserts the first write request to row 0 so that row 0 is open before the next transaction.
2. The user logic asserts a second `local_write_req` signal with a size of 2 and address of 0 (`col = 0`, `row = 0`, `bank = 0`, `chip = 0`). The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

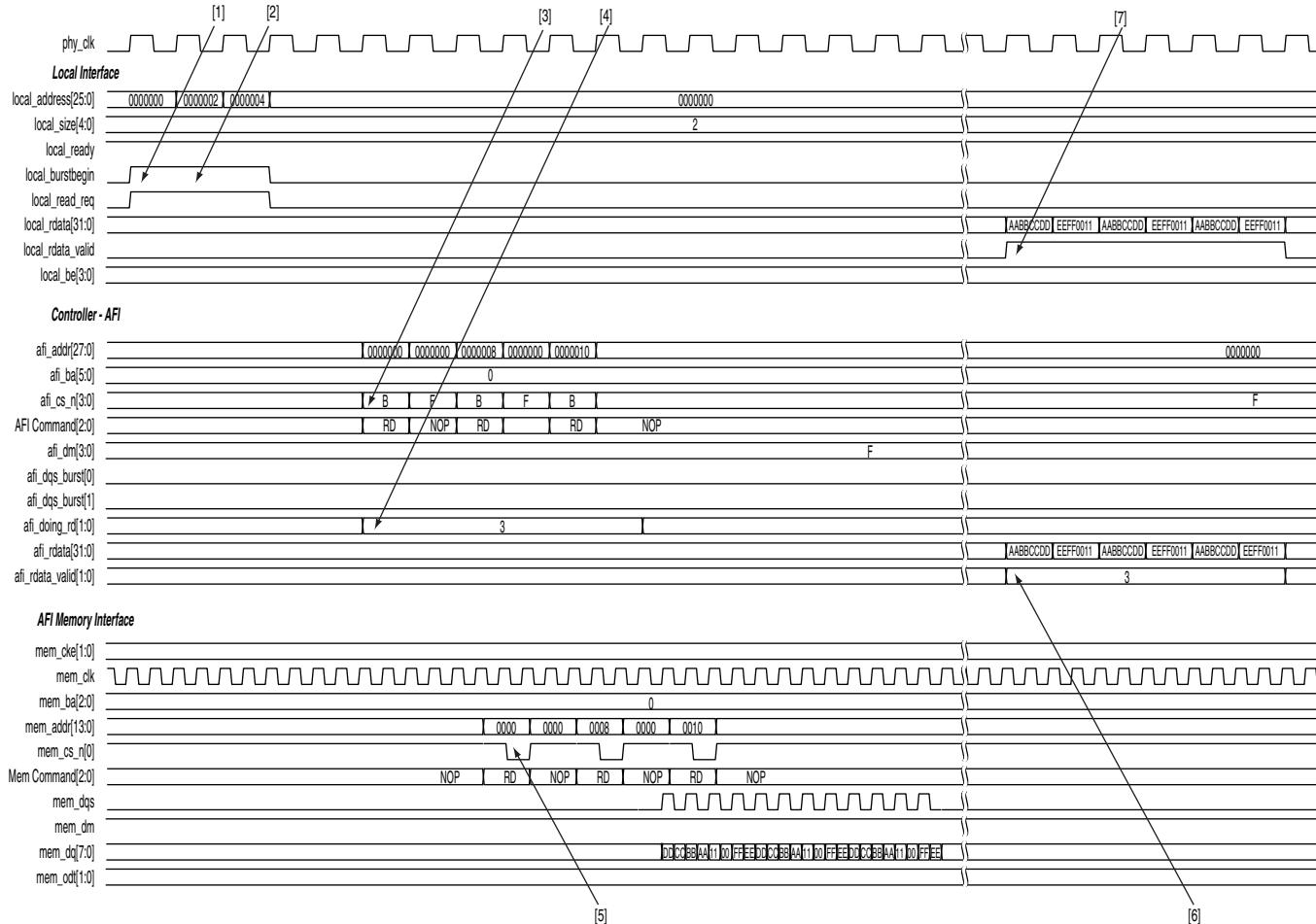
## DDR3 High-Performance Controller II

This section discusses the following timing diagrams for the DDR3 HPC II:

- “Half-Rate Read”
- “Half-Rate Write”
- “Half-Rate Read (Non Burst-Aligned Address)”
- “Half-Rate Write (Non Burst-Aligned Address)”
- “Half-Rate Read With Gaps”
- “Full-Rate Read”
- “Half-Rate Write Operation (Merging Writes)”
- “Write-Read-Write-Read Operation”

## Half-Rate Read (Burst-Aligned Address)

**Figure 17-5. Half-Rate Read Operation for HPC II—Burst-Aligned Address**



The following sequence corresponds with the numbered items in [Figure 17-1](#):

1. The user logic requests the first read by asserting the local\_read\_req signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address 0x000000. This local address is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x000000  
mem_col_address = 0x0000  
mem_bank_address = 0x00
```

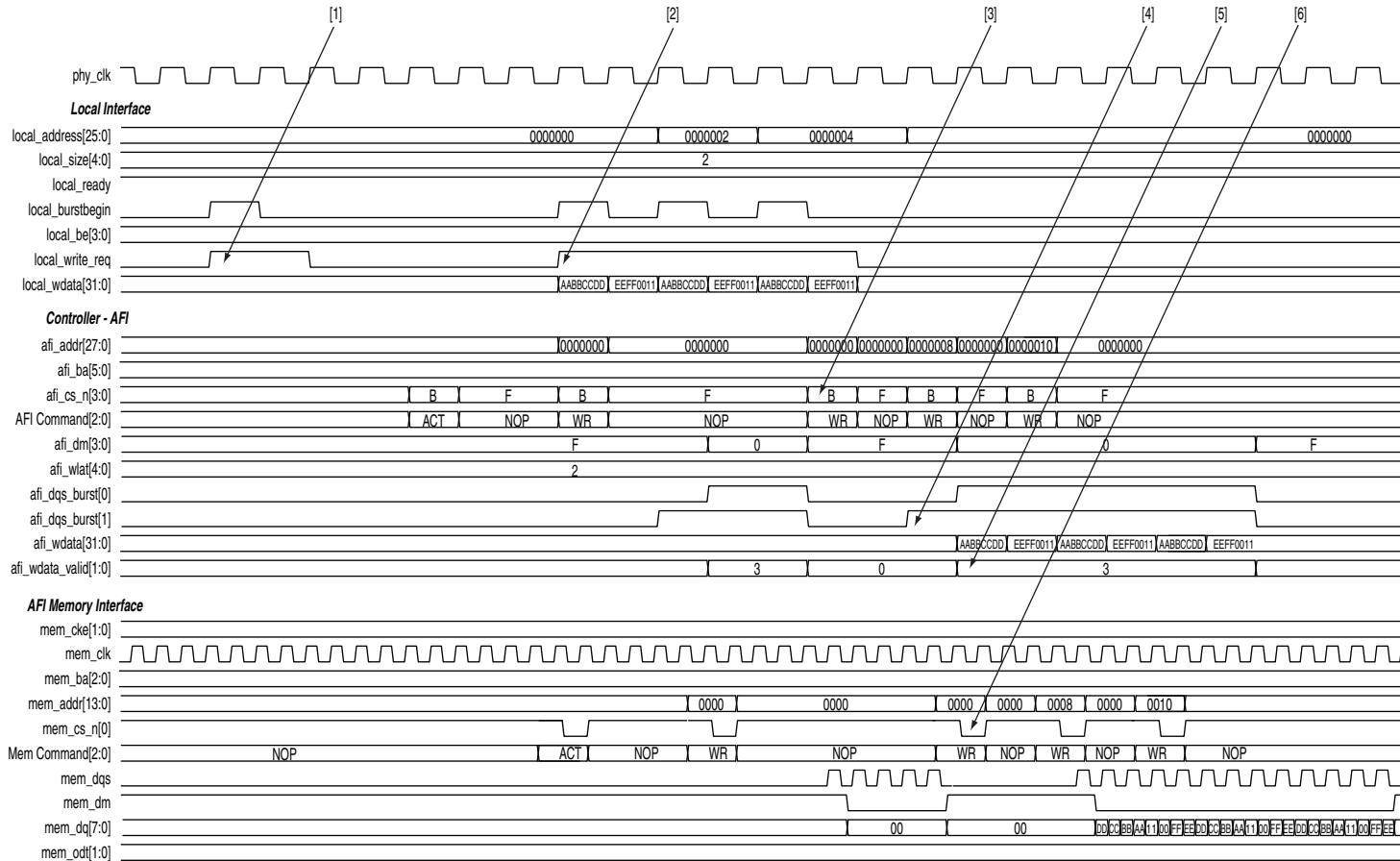
2. The user logic initiates a second read to a different memory column within the same row. The request for the second read is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the local\_ready signal. The starting local address 0x000002 is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000  
mem_col_address = 0x0002<<2 = 0x0008  
mem_bank_address = 0x00
```

3. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the afi\_doing\_rd signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the afi\_doing\_rd signal to enable its capture registers for the expected duration of memory burst.
5. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
6. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the phy\_clk domain, by asserting the afi\_rdata\_valid signal when there is valid read data on the afi\_rdata bus.
7. The controller returns the first read data to the user by asserting the local\_rdata\_valid signal when there is valid read data on the local\_rdata bus. If the ECC logic is disabled, there is no delay between the afi\_rdata and the local\_rdata buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the afi\_rdata and local\_rdata buses.

## Half-Rate Write (Burst-Aligned Address)

Figure 17-6. Half-Rate Write Operation for HPC II—Burst-Aligned Address

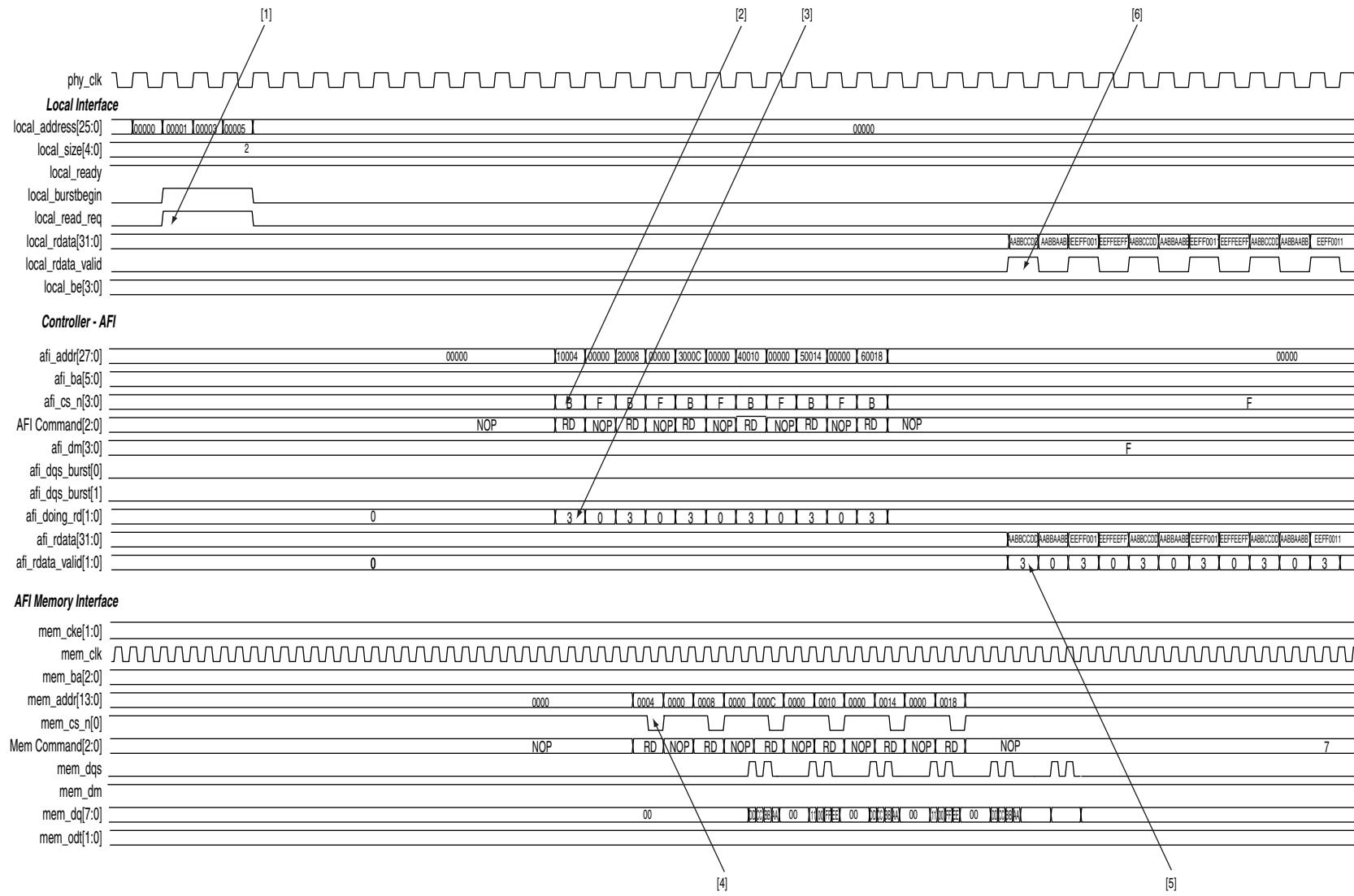


The following sequence corresponds with the numbered items in [Figure 17-2](#):

1. The user logic asserts the first write request to row 0 so that row 0 is open before the next transaction.
2. The user logic asserts a second `local_write_req` signal with size of 2 and address of 0 (`col = 0`, `row = 0`, `bank = 0`, `chip = 0`). The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

### **Half-Rate Read (Non Burst-Aligned Address)**

**Figure 17–7. Half-Rate Read Operation for HPC II—Non Burst-Aligned Address**



The following sequence corresponds with the numbered items in [Figure 17-7](#):

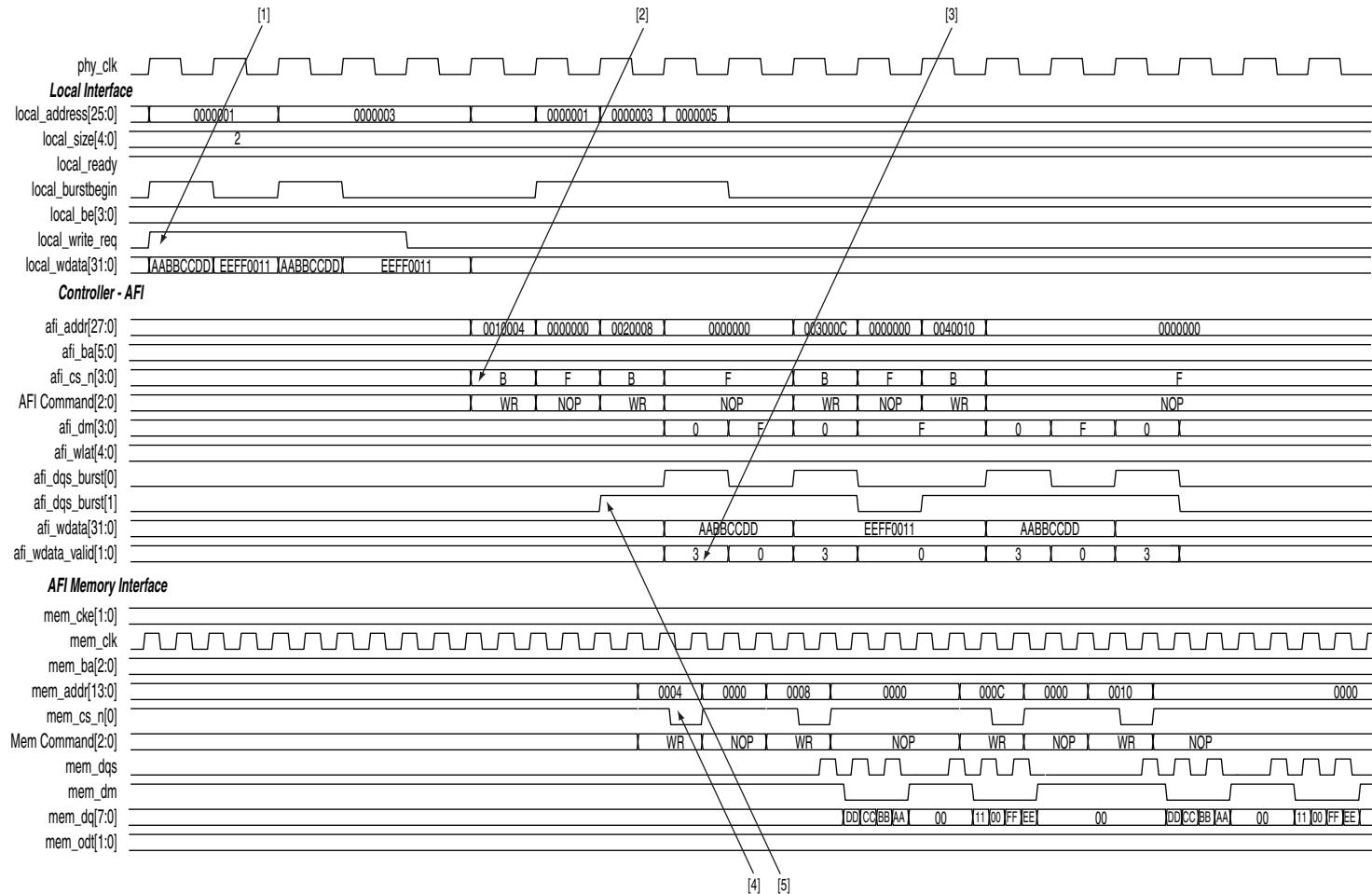
1. The user logic requests the first read by asserting the local\_read\_req signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address 0x000001. This local address is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x0001<<2 = 0x0004
mem_bank_address = 0x00
```

2. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the afi\_doing\_rd signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the afi\_doing\_rd signal to enable its capture registers for the expected duration of memory burst.
4. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
5. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the phy\_clk domain, by asserting the afi\_rdata\_valid signal when there is valid read data on the afi\_rdata bus.
6. The controller returns the first read data to the user by asserting the local\_rdata\_valid signal when there is valid read data on the local\_rdata bus. If the ECC logic is disabled, there is no delay between the afi\_rdata and the local\_rdata buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the afi\_rdata and local\_rdata buses.

## Half-Rate Write (Non Burst-Aligned Address)

Figure 17-8. Half-Rate Write Operation for HPC II—Non Burst-Aligned Address



The following sequence corresponds with the numbered items in [Figure 17-8](#):

1. The user logic asserts the first `local_write_req` signal with a size of 2 and an address of `0x000001`. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high. The local address `0x000001` is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x000001<<2 = 0x000004
mem_bank_address = 0x00
```

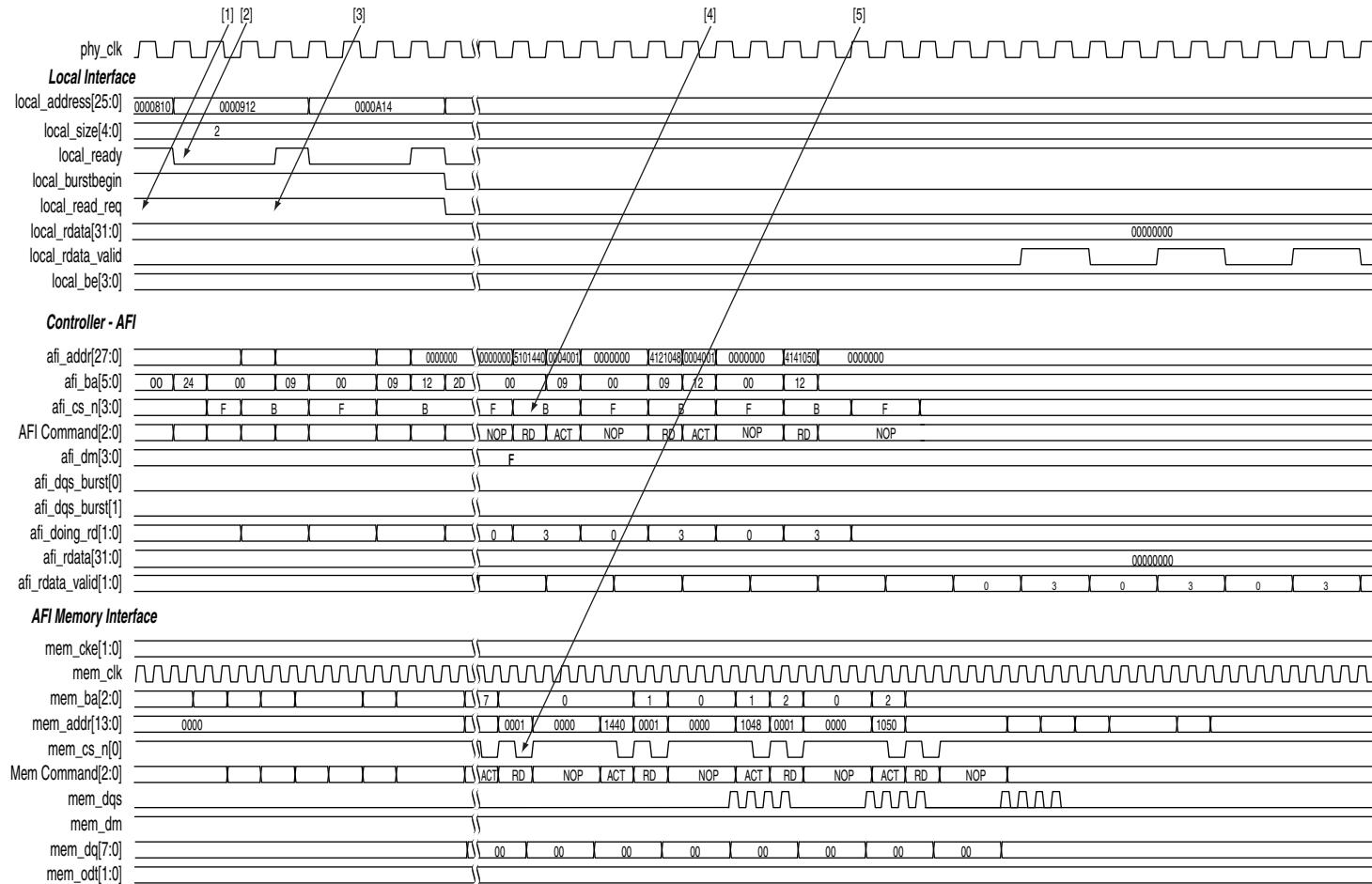
2. The user logic asserts the second `local_write_req` signal with a size of 2 and an address of `0x000003`. The local address `0x000003` is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x000003<<2 = 0x00000C
mem_bank_address = 0x00
```

3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
7. The controller generates another write because the first write is to a non-aligned memory address, `0x0004`. The controller performs the second write burst at the memory address of `0x0008`.

## Half-Rate Read With Gaps

Figure 17-9. Half-Rate Read Operation for HPC II—With Gaps



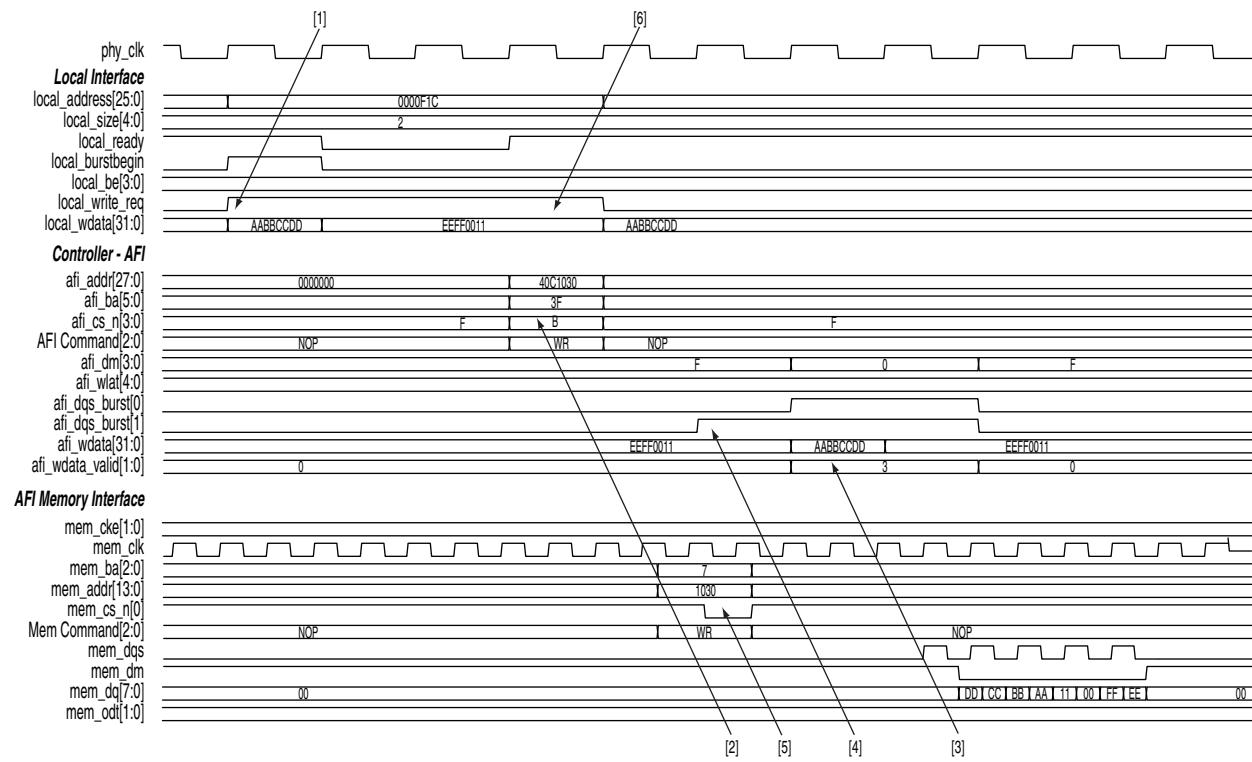
The following sequence corresponds with the numbered items in Figure 17-9:

1. The user logic requests the first read by asserting the local\_read\_req signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address 0x0000810. This local address is mapped to the following memory address in half-rate mode:
 

```
mem_row_address = 0x0001
mem_col_address = 0x0010<<2 = 0x0040
mem_bank_address = 0x00
```
2. When the command queue is full, the controller deasserts the local\_ready signal to indicate that the controller has not accepted the command. The user logic must keep the read request, size, and address signal until the local\_ready signal is asserted again.
3. The user logic asserts a second local\_read\_req signal with a size of 2 and address of 0x0000912.
4. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
5. The ALTMEMPHY megafunction issues the read command to the memory and captures the read data from the memory.

## Half-Rate Write With Gaps

Figure 17-10. Half-Rate Write Operation for HPC II—With Gaps

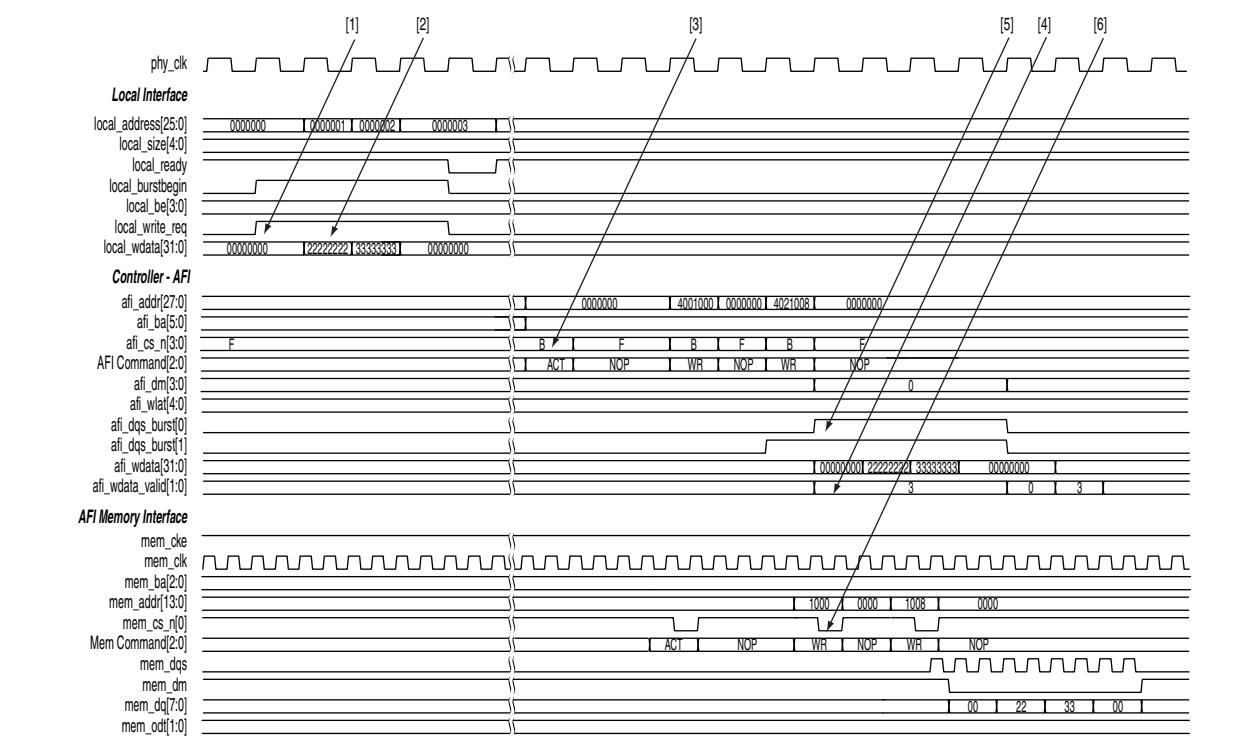


The following sequence corresponds with the numbered items in [Figure 17-10](#):

1. The user logic asserts a `local_write_req` signal with a size of 2 and an address of `0x0000F1C`.
2. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
4. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
5. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
6. For transactions with a local size of two, the `local_write_req` and `local_ready` signals must be high for two clock cycles so that all the write data can be transferred to the controller.

## Half-Rate Write Operation (Merging Writes)

**Figure 17-11. Write Operation for HPC II—Merging Writes**



The following sequence corresponds with the numbered items in [Figure 17-11](#):

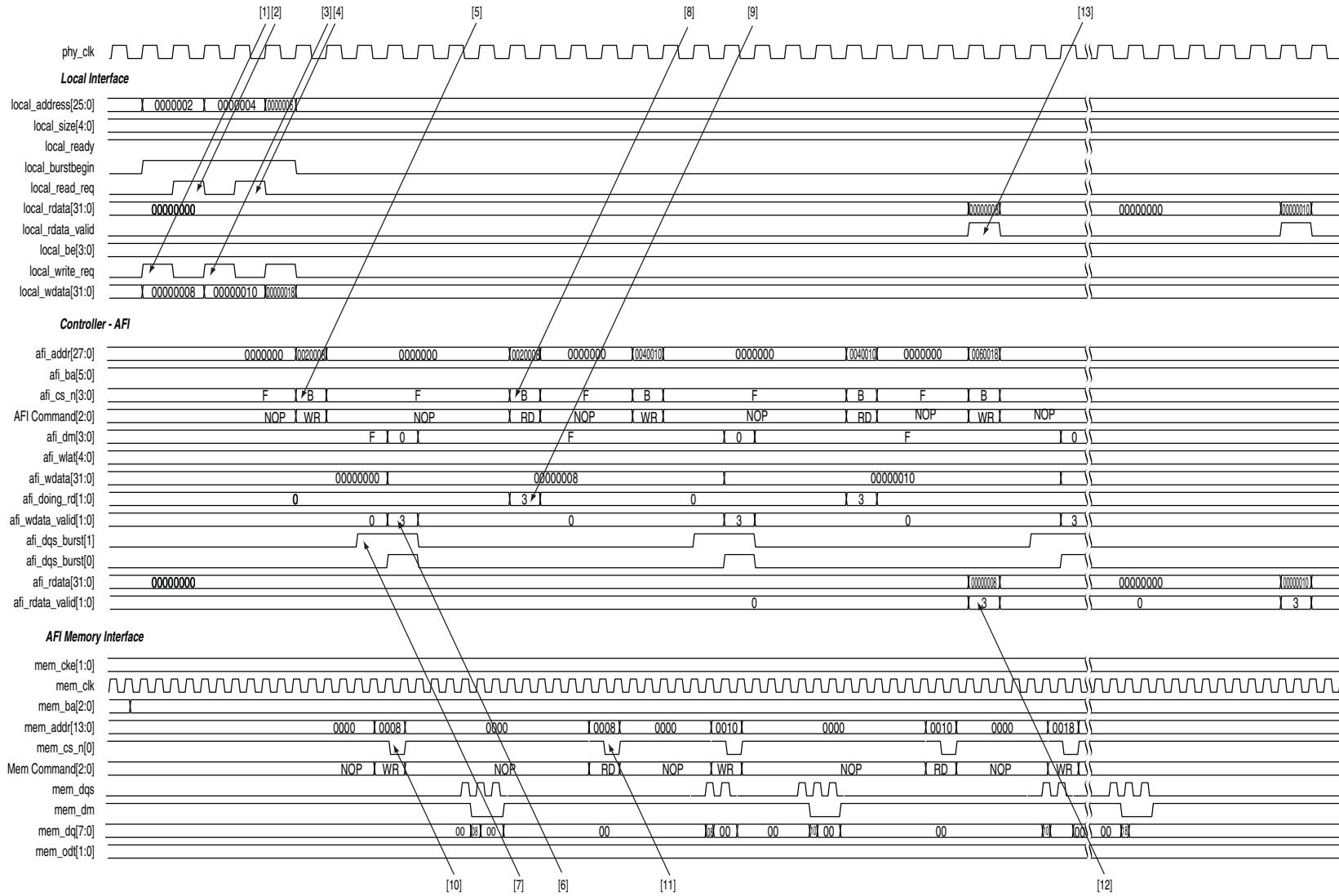
1. The user logic asserts the first local\_write\_req signal with a size of 1 and an address of 0x000000. The local\_ready signal is asserted along with the local\_write\_req signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the local\_ready signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the local\_ready signal is registered high. The local address 0x000000 is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x0000<<2 = 0x0000
mem_bank_address = 0x00
```

2. The user logic asserts a second local\_write\_req signal with a size of 1 and address of 1. The local\_ready signal is asserted along with the local\_write\_req signal, which indicates that the controller has accepted this request. Since the second write request is to a sequential address (same row, same bank, and column increment by 1), this write and the first write can be merged at the memory transaction.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the afi\_wdata\_valid signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the afi\_dqs\_burst signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

## Write-Read-Write-Read Operation

**Figure 17–12. Write-Read Sequential Operation for HPC II**



The following sequence corresponds with the numbered items in [Figure 17-12](#):

1. The user logic requests the first write by asserting the `local_write_req` signal, and the size and address for this write. In this example, the request is a burst length of 1 to a local address 0x000002. This local address is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x0002<<2 = 0x0008
mem_bank_address = 0x00
```

2. The user logic initiates the first read to the same address as the first write. The request for the read is a burst length of 1. The controller continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the `local_ready` signal. The starting local address 0x000002 is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
mem_col_address = 0x0002<<2 = 0x0008
mem_bank_address = 0x00
```

3. The user logic asserts a second `local_write_req` signal with a size of 1 and address of 0x000004.
4. The user logic asserts a second `local_read_req` signal with a size of 1 and address of 0x000004.
5. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
6. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
7. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signals that the ALTMEMPHY megafunction issues to the memory.
8. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
9. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
10. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
11. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
12. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.

13. The controller returns the first read data to the user by asserting the local\_rdata\_valid signal when there is valid read data on the local\_rdata bus. If the ECC logic is disabled, there is no delay between the afi\_rdata and the local\_rdata buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the afi\_rdata and local\_rdata buses.

## Document Revision History

Table 17-1 lists the revision history for this document.

**Table 17-1. Document Revision History**

Date	Version	Changes
November 2012	1.3	Changed chapter number from 15 to 17.
June 2012	1.2	Added Feedback icon.
November 2011	1.1	Consolidated timing diagrams from 11.0 version <b>DDR and DDR2 SDRAM Controller with ALTMEMPHY IP User Guide</b> and <b>DDR3 SDRAM Controller with ALTMEMPHY IP User Guide</b> .

This chapter describes a debug toolkit for ALTMEMPHY-based high performance controllers. The debug toolkit uses a JTAG connection to a Windows PC. The debug toolkit supports the following Altera AFI-based IP:

- ALTMEMPHY megafunction
- DDR2 and DDR3 SDRAM High-Performance Controller and High Performance Controller II

The debug toolkit supports all FPGA device families supported by the high-performance controller (HPC) and high-performance controller II (HPC II) and ALTMEMPHY.



The debug toolkit does not support the QDR II and II+ SRAM, RLDRAM II with UniPHY controllers.

The debug toolkit provides detailed information regarding the calibration process. The debug toolkit and the SignalTap II logic analyzer can be run at the same time. However using **Autorun Analysis** in the SignalTap II logic analyzer slows down the JTAG communication with the debug toolkit.

This chapter provides the following information:

- “Debug Toolkit Overview”
- “Install the Debug Toolkit”
- “Modify the Example Top-Level File to use the Debug Toolkit”
- “Use the Debug Toolkit”
- “Interpret the Results”
- “Understand the Checksum and Failure Code”



The debug toolkit provides information on the failures and calibration results that assist and direct the hardware debug process. The debug toolkit does not fix a failing design. Before you use the debug toolkit, refer to **Debugging Memory IP** in volume 2, section 1, of the *External Memory Interface Handbook*.

## Debug Toolkit Overview

The debug TOOLKIT provides the following information:

- Lists the various calibration stages and indicates whether each stage was successful or not.
- States an error code specific to the exact type of calibration failure.

© 2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at [www.altera.com/common/legal.html](http://www.altera.com/common/legal.html). Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



- Provides possible causes for calibration failures.
- Provides graphics to visualize the following parameters:
  - Resync clock phase setup (per pin)
  - Read deskew multipurpose registers (MPR) (prime DQ pins only)
  - Read deskew block training pattern (per pin)
  - Write deskew (per pin)
  - Write leveling report

You can export a **.doom** file from the debug toolkit. This file provides a record of your calibration results and the ALTMEMPHY IOE configuration specific to your system, allowing you to refer to this data offline later.

## Install the Debug Toolkit

To install the debug toolkit, follow these steps:

1. Download the debug toolkit, **debug-toolkit.zip**, file from Altera [website](#).
2. Unzip the **debug-toolkit.zip** file.
3. To start the debug toolkit, navigate to the directory where you unzipped the **.zip** file and run **debug-toolkit.exe**.

To install the debug toolkit on a Quartus II production programming PC, follow these steps:

1. On a PC running the Windows OS, copy the **debug-toolkit.zip** file to your project directory or a common programming directory that you also use to program your test platform using a USB-Blaster™ download cable.
2. Unzip the **debug-toolkit.zip** file to either your project folder or a common programming folder.

## Modify the Example Top-Level File to use the Debug Toolkit

Before you use the debug toolkit, you must modify your design's example-top-level file, by following these steps:

- Verify the Design
- Regenerate the IP
- Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project
- Add Additional Signals
- Add `alt_jtagavalon.v` to your Quartus II Project Settings Files List
- Recompile your Quartus II Test Design
- Program Hardware with Debug Enabled .sof



Your design must follow the recommended flow; refer to the *Recommended Design Flow* chapter in volume 1 of the *External Memory Interface Handbook*.

## Verify the Design

Ensure your design meets the following conditions:

- The parameters entered into the IP are correct for the memory and data rate.
- The design passes functional simulation.
- The Quartus II project has the correct board trace models specified for the PCB you are using.
- For Cyclone III devices, ensure that the **set t(additional\_addresscmd\_tpd)** parameter is correctly specified in your **.sdc** file.
- For Arria II GX devices, ensure that the **Address and Command to CK skew** parameter is correctly specified in the **Board Settings** tab of the IP wizard.
- The address and command clock phase is correct, to ensure optimum balanced setup and hold times.
- The Quartus II design successfully closes timing.
- The Quartus II project has the correct pin location assignments for the PCB that you are using.
- The autogenerated IP assignments are correctly applied to the example top-level file.
- The **.sdc** constraint files are correctly applied to the example top-level file.
- The Quartus II settings are correctly applied.
- The R<sub>UP</sub>/R<sub>DN</sub> pin locations are correctly specified in the example top-level file if required.
- The SignalTap II logic analyzer is added to the example top-level file.

Before you use the debug toolkit, follow these steps:

1. Edit the example top-level file to enable debugging:
  - a. Open the *<variation name>.v* or **.vhdl** and find the **export\_debug\_port** private value.

 Do not edit this value in the file *<variation name>\_phy.v* or **.vhdl** file.

The value is at the bottom of the file:

```
// =====
// DDR3 High Performance Controller Wizard Data
// =====
// DO NOT EDIT FOLLOWING DATA
// @Altera, IP Toolbench@
...
...
// Retrieval info: <PRIVATE name = "export_debug_port" value="false"
type="STRING" enable="1" />
```

- b. Edit the **export\_debug\_port** private value to true:

```
// Retrieval info: <PRIVATE name = "export_debug_port" value="true"
type="STRING" enable="1" />
```

## Regenerate the IP

To regenerate the IP, follow these steps:

1. Open the MegaWizard Plug In Manager, and select **Edit an existing custom megafunction variation**.
2. Select your modified high-performance controller.
3. Click **Next** to open the IP.
4. Click **Finish** to regenerate the IP.

You now have a version of the design with debug enabled. Seven new ports with the prefix `dbg_*` are added to the controller instance through the design hierarchy up to the `<variation name>_example_top.v` or `.vhdl`.

## Instantiate the JTAG Avalon-MM port in to the Example-Top Level Project

To instantiate the JTAG Avalon-MM port, follow these steps:

1. Declare the following wires in `<variation name>_example_top.v` or `.vhdl`.

```
wire [12: 0] av_address;
wire av_write_n;
wire [31: 0] av_writedata;
wire av_read_n;
wire av_waitrequest;
wire [31: 0] av_readdata;
```

2. Add the following instances in `<variation name>_example_top.v` or `.vhdl`.

```
// inst jtag avalon:
alt_jtagavalon alt_jtagavalon(
    .clk (phy_clk),
    .rst_n (reset_phy_clk_n),
    .av_address (av_address),
    .av_write_n (av_write_n),
    .av_writedata (av_writedata),
    .av_read_n (av_read_n),
    .av_readdata (av_readdata),
    .av_waitrequest (av_waitrequest)
);
defparam alt_jtagavalon.SLD_NODE_INFO = 203976192;
defparam alt_jtagavalon.ADDR_WIDTH = 13;
defparam alt_jtagavalon.DATA_WIDTH = 32;
defparam alt_jtagavalon.MODE_WIDTH = 3;
```

3. Update the following port connections in the DDR2 or DDR3 SDRAM instance in *<variation name>\_example\_top.v* or *.vhd*:

- a. Locate the PHY or controller instance in the top-level file and locate the following debug port connections:

```
//<< START MEGAWIZARD INSERT WRAPPER_NAME
<variation_name> <variation_name>_inst
(
    .dbg_addr (13'b0),
    .dbg_cs (1'b0),
    .dbg_rd (1'b0),
    .dbg_rd_data (dbg_rd_data_sig),
    .dbg_waitrequest (dbg_waitrequest_sig),
    .dbg_wr (1'b0),
    .dbg_wr_data (32'b0),
```

- b. Change the following debug port connections to:

```
//<< START MEGAWIZARD INSERT WRAPPER_NAME
<variation_name> <variation_name>_inst
(
    .dbg_addr (av_address),
    .dbg_cs (1'b1),
    .dbg_rd (~av_read_n),
    .dbg_rd_data (av_readdata),
    .dbg_waitrequest (av_waitrequest),
    .dbg_wr (~av_write_n),
    .dbg_wr_data (av_writedata),
```

The debug toolkit is added to your example top-level file.

## Add Additional Signals

In addition to the standard SignalTap II signals, you can add the following signals during debug to understand the following situations:

- Where calibration failed:
  - \*ctl\_init\_fail -phy\_inst
  - \*ctl\_init\_success -phy\_inst
  - ctl\_cal\_fail -phy\_inst
  - ctl\_cal\_success -phy\_inst
- How much resynchronization margin is available:
  - \*cal\_codvw\_phase \*DT-phy\_inst
  - \*cal\_codvw\_size \*DT-phy\_inst
  - \*codvw\_trk\_shift \*DT-phy\_inst

- What the read and write latency is calibrated as:
    - `ctl_rlat *DT-phy_inst`
    - `ctl_wlat *DT-phy_inst`
  - If the PLL is locked and phase stepping as expected:
    - `Locked -altpll_component`
    - `Phasecounterselect *DT-altpll_component`
    - `Phaseupdown -altpll_component`
    - `Phasestep -altpll_component`
    - `phasedone -altpll_component`
    - `dqs_delay_ctrl_export *DT-phy_inst`
-  For signals marked with \*DT, disable trigger enable in the SignalTap II logic analyzer to reduce memory requirement.

Table 18–1 shows sequencer signals that you can also probe using the SignalTap II logic analyzer, to help you understand where calibration failure is occurring. The signals are in the `<variation_name>_alt_mem_phy_seq.vhd` file.

 All signals are active high.

**Table 18–1. Sequencer Signals (Part 1 of 2)**

Port Name	Description
<code>Flag_done_timeout</code>	Calibration stage timeout failure, memory did not respond.
<code>Flag_ack_timeout</code>	Sequencer failed to respond.
<code>state.s_phy_initialise</code>	PHY initialization Stage: wait for DLL lock and <code>init_done</code> .
<code>state.s_init_dram</code>	DRAM initialization stage: reset sequence.
<code>State.s_prog_cal_mrs</code>	DRAM initialization stage: programming mode registers (once per chip select).
<code>state.s_write_ihi</code>	Write internal RAM header initialization.
<code>state.s_cal</code>	Calibration required stage.
<code>state.s_write_btp</code>	Write block training pattern stage: 00001111.
<code>state.s_write_mtp</code>	Write memory training patterns: 00110101.
<code>state.s_rrp_reset</code>	Read resynchronization phase reset: PLL initial condition.
<code>state.s_rrp_sweep</code>	Read resynchronization phase sweep: sweep PLL phases per chip select.
<code>state.s_read_mtp</code>	Read memory training patterns to find correct alignment.
<code>State.s_rrp_seek</code>	Read resynchronization phase setup stage: set PLL to center of valid window.
<code>state.s_rdv</code>	Read data valid stage.
<code>state.s_poa</code>	Postamble calibration stage.
<code>state.s_was</code>	Write datapath setup: write data to DRAM so that latency can be determined.
<code>state.s_adv_rd_lat</code>	Advertise read latency stage.

**Table 18-1. Sequencer Signals (Part 2 of 2)**

Port Name	Description
state.s_adv_wr_lat	Advertise write latency stage.
state.s_tracking_setup	Tracking setup stage (first pass to setup mimic window).
state.s_prep_customer_mr_setup	Set custom mode register settings (admin).
state.s_tracking	Tracking stage (mimic path tracking in user mode).
state.s_operational	Calibration success: user mode.
state.s_non_operational	Calibration failed or tracking failed in user mode.
state.s_reset	Reset stage.
<hr/>	
dgrb_ctrl.command_err	Error in the data gather read bias block.
dgrb_ctrl.command_result[7..0]	Data gather read block (DGRB) error code.
dgwb_ctrl.command_err	Error in the data gather write bias block.
dgwb_ctrl.command_result[7..0]	Data gather write block (DGWB) error code.
admin_ctrl.command_err	Error in the admin (DRAM initialization and control) block.
admin_ctrl.command_result[7..0]	Admin block error code.

## Add alt\_jtagavalon.v to your Quartus II Project Settings Files List

Before you compile your design, you must add the `alt_jtagavalon.v` file to your projects file list. This `alt_jtagavalon.v` file is included with the debug toolkit.

## Recompile your Quartus II Test Design

You must compile your modified design to generate a new `.sof` for testing that includes the debug toolkit code. Altera recommends you ensure that this modified design continues to pass timing analysis. Any timing failures should be assessed and corrected before using the debug toolkit.

## Program Hardware with Debug Enabled .sof

To program hardware with the debug enabled `.sof`, program the device using the SignalTap II logic analyzer. Then click **Run Analysis** to run once. Typically, the SignalTap II logic analyzer is initially configured to trigger on the signal `test_complete`, which is fine for working designs.

For designs that are failing calibration, Altera recommends modifying the trigger based on the observed results. Combine this SignalTap II trigger fault isolation activity with use of the debug toolkit. For example:

1. Initially trigger on `test_complete`, if the interface works first time.
2. Trigger on `cal_fail`, if the PHY is failing calibration.
3. Trigger on the same `state.s_*` or error code that is reported as the calibration failure point in the debug toolkit.
4. Trigger on `init_fail`, if the memory is failing to initialize.
5. Trigger on `pll_locked`, if the PLL is operating incorrectly.

## Use the Debug Toolkit

To use the debug toolkit, follow these steps:

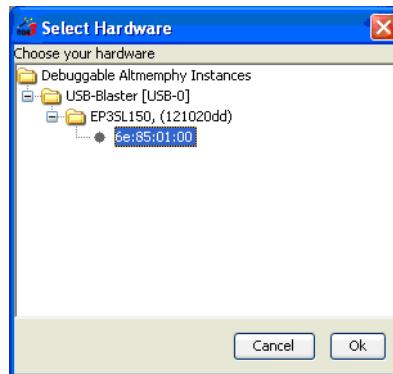
1. Double-click **debug-toolkit.exe**.
2. On the File menu, click **Connect via JTAG** (Figure 18–1).

**Figure 18–1. Connect to JTAG**



3. Navigate down the hierarchy and click on the Avalon-MM JTAG node (Figure 18–2).
- Tip** If you encounter connection problems, Altera recommends that you have only a single USB-Blaster™ download cable programming adaptor connected to your PC.

**Figure 18–2. Select Hardware**



4. If you receive a prompt stating the following message, verify you have the latest debug toolkit, and click **Yes** (Figure 18–3).

```
Hardware Version Number Mismatch -  
this_debug_GUI_release_doesnt_match_the_altmemphy_version_used_in_g  
eneration
```

**Figure 18–3. Hardware Mismatch**



## Interpret the Results

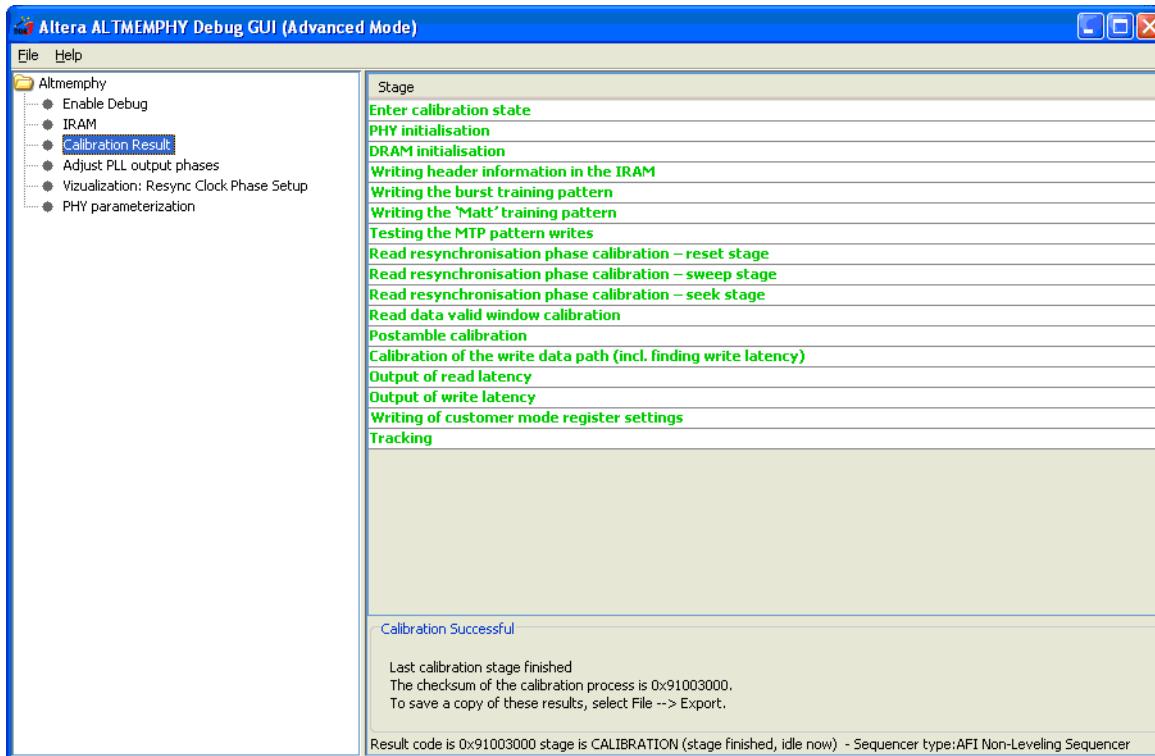
This topic discusses:

- Calibration Successful
- Calibration Fails

### Calibration Successful

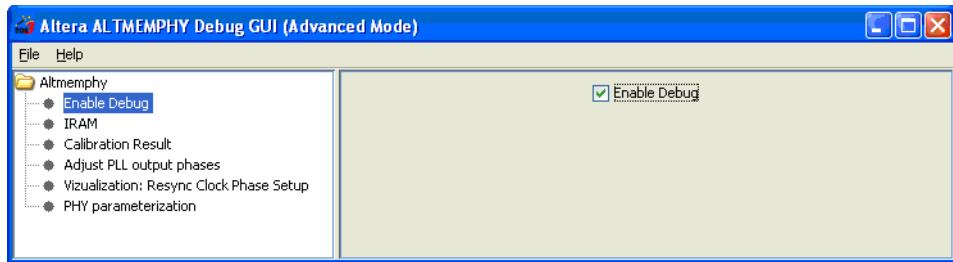
If calibration is successful, you see the following screen (Figure 18–4).

**Figure 18–4. Calibration Successful**



For optimum operation of the debug toolkit, ensure that you turn on **Enable Debug** (Figure 18–5).

**Figure 18–5. Enable Debug**



Click **internal RAM** to display the calibration memory results (Figure 18–6).

This setting is not typically used.

**Figure 18–6. Calibration Memory Results**

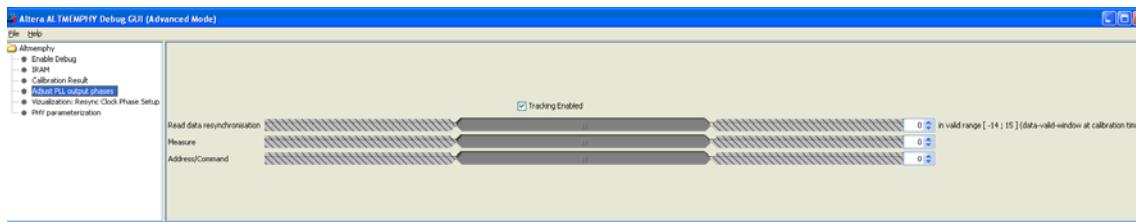
The screenshot shows the 'Altmemphy' section of the debug GUI. Under 'Altmemphy', the 'IRAM' option is selected, indicated by a blue highlight. To the right, a table displays calibration memory results for IRAM:

	3	2	1	0
0	02	00	05	02
1	01	09	09	48
2	DE	AD	DE	AD
3	00	44	0A	63
4	00	00	00	00
5	00	00	00	00
6	00	00	00	09
7	32	00	00	00
8	08	04	00	4A
9	00	07	FF	FF
A	FF	F8	00	00
B	5A	5A	5A	5A
C	06	04	01	4A
D	00	00	02	20
E	5A	5A	5A	5A
F	08	0C	02	4A
10	00	00	00	00
11	00	00	00	00
12	5A	5A	5A	5A
13	06	0C	03	4A
14	FF	FF	00	00
15	5A	5A	5A	5A
16	08	00	04	4A
17	00	07	FF	FF
18	00	07	FF	FF
19	00	07	FF	FF
1A	00	07	FF	FF

The debug toolkit can dynamically alter the PLL clock phases (Figure 18–7).

 This setting is not typically used.

**Figure 18–7. Altering PLL Clock Phases**



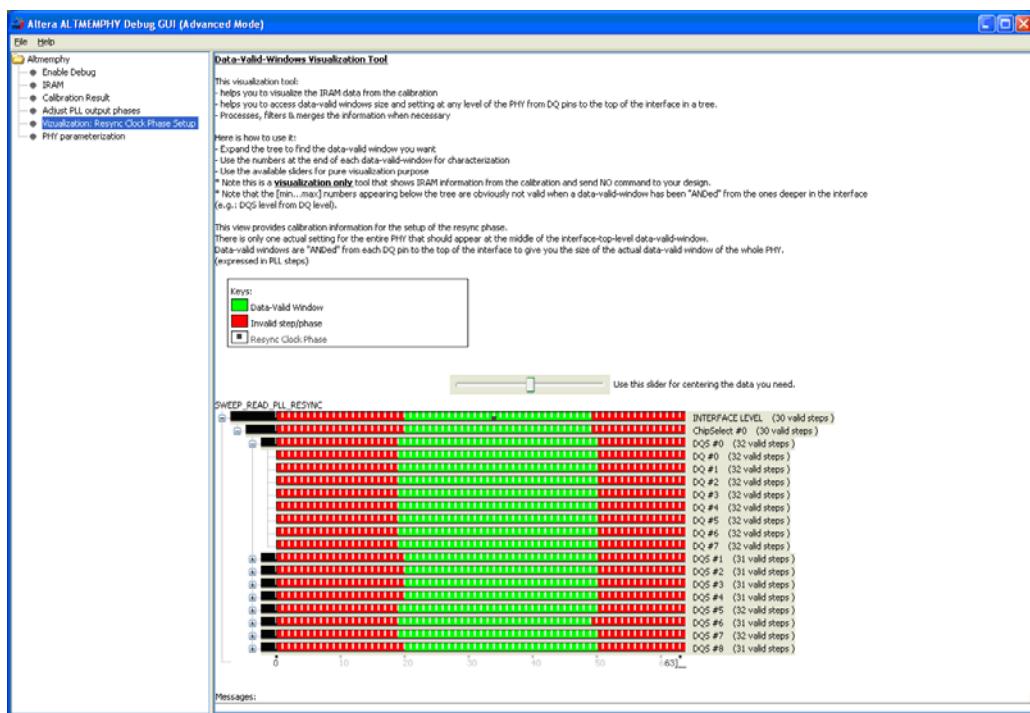
The debug toolkit states the number of resynchronization clock phase steps that are valid at calibration time. For example, the resynchronization window size in PLL phase steps at calibration in [Figure 18–7](#) is 26 PLL phase steps wide.

 The address and command phase sweep is limited to either address and command pin margins or address and command core-to-I/O transfer margins.

 In [Figure 18–7](#), moving the slider to the left increases the value, while moving the slider to the right decreases the value.

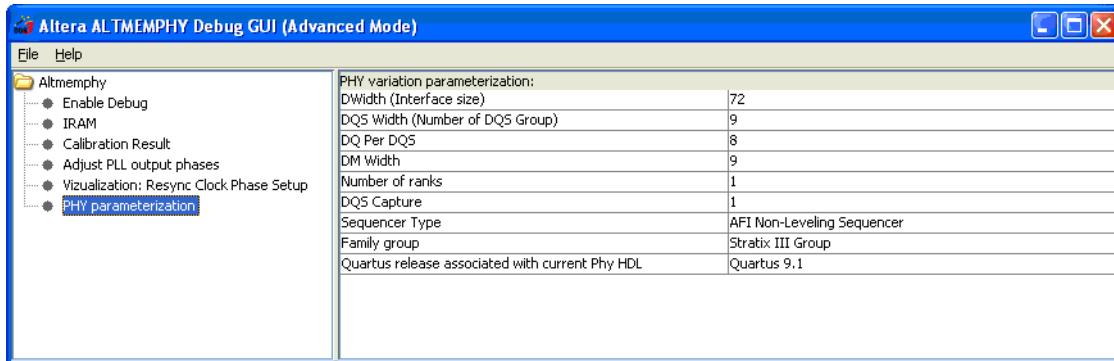
Click **Visualization: resync clock phase setup** ([Figure 18–8](#)) to show the PHY resynchronization pass and fail results in an expandable tree structure:

- For the whole interface including the chosen phase (black dot)
- On a DQS group basis
- On a per DQ pin basis

**Figure 18–8. Visualization**

The debug toolkit additionally states the number of passing phase steps that it finds during calibration. Thus to calculate the resynchronization margin in this design, the resynchronization clock (C6) from the PLL has a phase-shift step resolution of 78.12 ps or 5.62 degrees. So 30 valid steps means that the window size = 2.343 ns or 168.6 degrees.

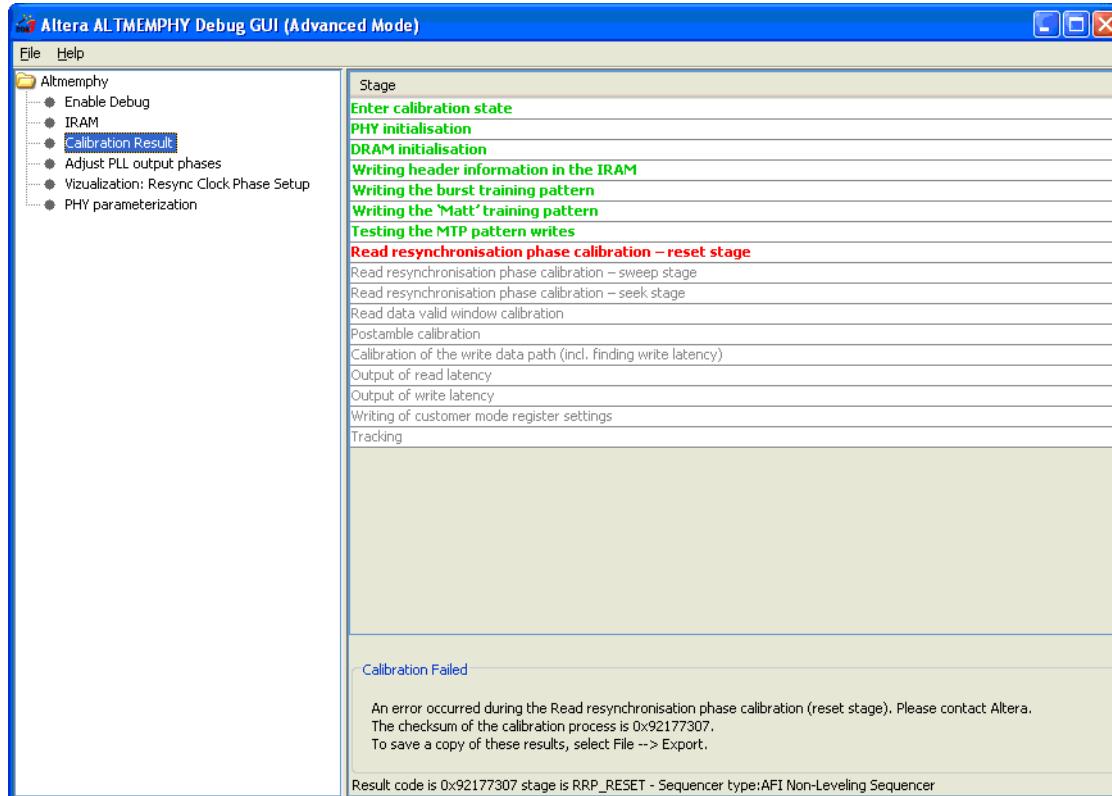
Click **PHY parameterization** (Figure 18–9), to show the exact calibration configuration of the generated IP.

**Figure 18–9. PHY Parameterization**

## Calibration Fails

If calibration fails, you see the following screen (Figure 18-10).

**Figure 18-10. Calibration Fails**



The stage at which calibration fails is highlighted in red; the stages that have successfully passed are in green. When possible, the debug toolkit also provides a possible cause for the failure code.

This failure code is: 0x92177307, for more information on failure codes, refer to “Understand the Checksum and Failure Code” on page 18-15.

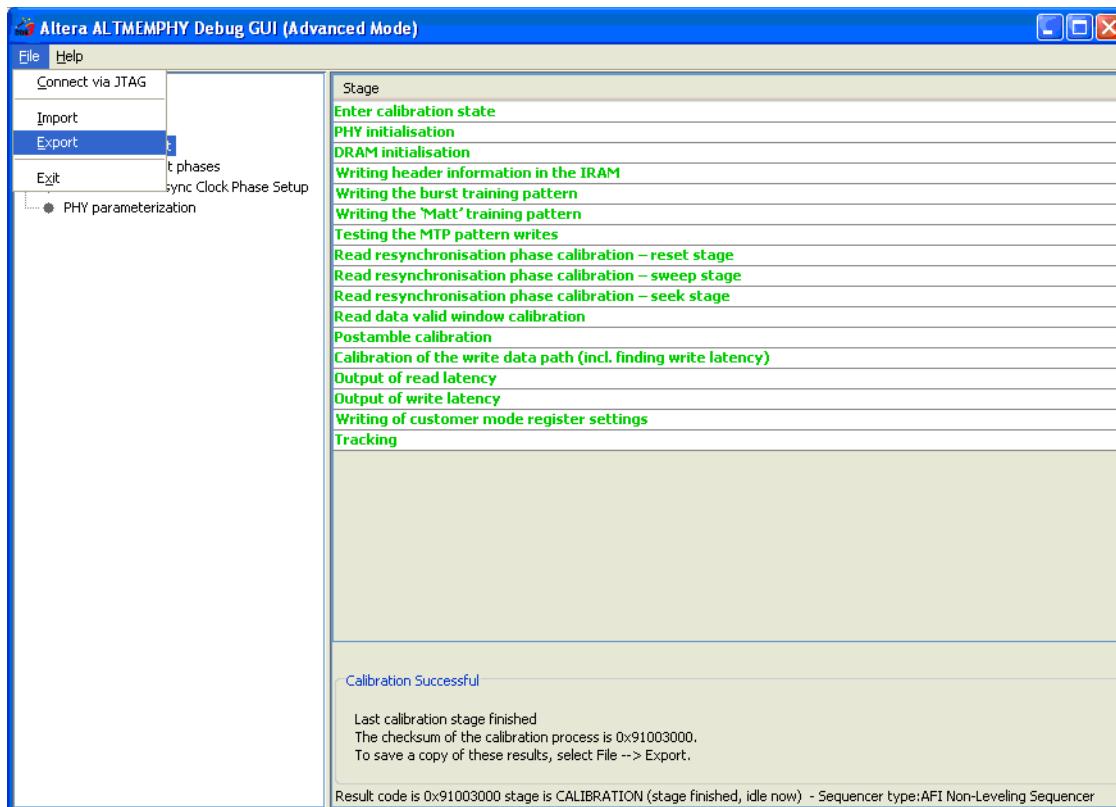
## Save the Calibration Results

Often the calibration failure stage, the reported suggested failure cause, or the combined debug toolkit result and waveforms viewed in the SignalTap II logic analyzer provide enough detail to resolve the failure directly. However, you may wish to save your calibration results, so that you can refer to them later.

With your calibration process results still displayed on the File menu, click **Export** (Figure 18–11).

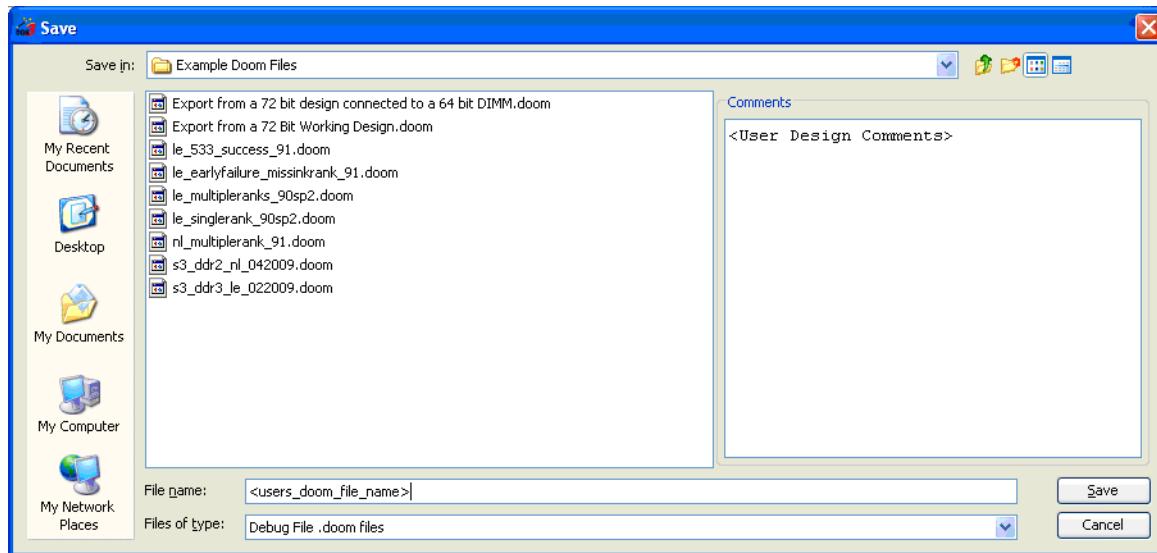
**Figure 18–11. Export**

---



In the **Save** dialog box (Figure 18–12), specify a file name and any comments to help with the identification and understanding of the controller configuration that you have evaluated, and details on what you may have tested.

**Figure 18–12. Save**



You can save this **.doom** file with a Quartus II archive (**.qar**) file of the test design, and a copy of the captured SignalTap II waveform files, as a single design archive. This archive provides a record of your calibration results and the ALTMEMPHY IOE configuration specific to your system, so you can refer to this data at a later date.

## Understand the Checksum and Failure Code

The debug toolkit checksum provides a direct correlation to the exact stage that calibration failed and the error code for that failure.

For example, the hexadecimal code in the format 0xAABBCCDD represents the full 32-bit contents of the calibration status register.

The code and the subcode have the following definitions:

- The code or calibration stage is the first byte (DD) or [7..0]
- The subcode or error code is the third byte (BB) or [23..16]

Take the hexadecimal value of each code and convert that to decimal. When you have these two numbers in decimal format, you can open the **failuremessages\_nl.csv** spreadsheet file and look up the likely causes of your calibration failure.



In a passing interface, these two numbers are zero.

Table 18–2 shows the codes that correspond to the indicated calibration stages.

**Table 18–2. Calibration Stages**

Stage	Code
Enter calibration state	0
PHY initialization	1
DRAM initialization	2
Writing header information in the internal RAM	3
Writing the burst training pattern	4
Writing more training patterns	5
Testing more training pattern writes	6
Read resynchronization phase calibration—reset stage	7
Read resynchronization phase calibration—sweep stage	8
Read resynchronization phase calibration—seek stage	9
Read data valid window calibration	10
Postamble calibration	11
Calibration of the write datapath (including finding write latency)	12
Output of read latency	13
Output of write latency	14
Writing of customer mode register settings	15
Tracking	16

You can find the same information from the SignalTap II logic analyzer if the `*_ctrl.command_err`, `*_ctrl.command_result` \* and `state.s_*` signals are added. The command error and state signals identify within which calibration stage the interface fails. The corresponding command result then includes the same information as the subcode.

For more information on the stages of calibration, refer to “[ALTMEMPHY Calibration Stages](#)” in section 1, chapter 2, of this volume.

## Document Revision History

Table 18–3 lists the revision history for this document.

**Table 18–3. Document Revision History**

Date	Version	Changes
November 2012	1.2	Changed chapter number from 16 to 18.
June 2012	1.1	Added Feedback icon.
November 2011	1.0	Harvested 11.0 Debug Toolkit for DDR2 and DDR3 SDRAM Controllers with ALTMEMPHY IP content.