



External Memory Interface Handbook Volume 3

Section III. DDR2 and DDR3 SDRAM Controller with UniPHY User Guide



101 Innovation Drive
San Jose, CA 95134
www.altera.com

EMI_DDR3UP_UG-2.0

Document last updated for Altera Complete Design Suite version:
Document publication date:

11.0
June 2011



[Subscribe](#)

© 2011 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX are Reg. U.S. Pat. & Tm. Off. and/or trademarks of Altera Corporation in the U.S. and other countries. All other trademarks and service marks are the property of their respective holders as described at 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.



Chapter 1. About This IP

Release Information	1-1
Device Family Support	1-2
Features	1-3
Unsupported Features	1-3
MegaCore Verification	1-4
Resource Utilization	1-4
System Requirements	1-9

Chapter 2. Getting Started

Installation and Licensing	2-1
Design Flows	2-1
MegaWizard Plug-In Manager Flow	2-3
Specifying Parameters	2-3
Simulate the IP Core	2-4
SOPC Builder Design Flow	2-5
Specify Parameters	2-5
Complete the SOPC Builder System	2-6
Qsys System Integration Tool Design Flow	2-7
Specify Parameters	2-7
Complete the Qsys System	2-8
Simulate the System	2-8
Qsys and SOPC Builder Interfaces	2-9
Generated Files	2-18
MegaWizard Plug-in Manager Flow	2-18
Synthesis	2-19
Simulation	2-19
Example Design Fileset	2-20
SOPC Builder Flow	2-21
Qsys Flow	2-21
Synthesis	2-21
Simulation	2-22

Chapter 3. Parameter Settings

PHY Settings	3-1
FPGA	3-1
Clocks	3-1
Advanced Settings	3-1
Memory Parameters	3-2
Memory Initialization Options—DDR2	3-3
Mode Register 0	3-3
Mode Register 1	3-4
Mode Register 2	3-4
Memory Initialization Options—DDR3	3-4
Mode Register 0	3-4
Mode Register 1	3-5
Mode Register 2	3-5
Memory Timing	3-5

Board Settings	3-6
Setup and Hold Derating	3-7
Intersymbol Interference	3-7
Board Skews	3-8
Controller Settings	3-9
Avalon Interface	3-9
Low Power Mode	3-10
Efficiency	3-10
Configuration, Status, and Error Handling	3-11
Diagnostics	3-11
Simulation Options	3-11
Efficiency Monitor and Protocol Checker Settings	3-12

Chapter 4. Constraining and Compiling

Add Pin and DQ Group Assignments	4-1
Board Settings	4-2
Compile the Design	4-2

Chapter 5. Functional Description—Controller

Memory Controller Architecture	5-1
Avalon-ST Input Interface	5-2
Command Generator	5-2
Timing Bank Pool	5-2
Arbiter	5-3
Arbitration Rules	5-3
Rank Timer	5-3
Read Data Buffer	5-3
Write Data Buffer	5-3
ECC Block	5-3
AFI Interface	5-3
CSR Interface	5-4
Controller Features Descriptions	5-4
Data Reordering	5-4
Pre-emptive Bank Management	5-4
Quasi-1T and Quasi-2T	5-4
User Autoprecharge Commands	5-4
Address and Command Decoding Logic	5-5
Low-Power Logic	5-5
User-Controlled Self-Refresh	5-5
Automatic Power-Down with Programmable Time-Out	5-5
ODT Generation Logic	5-5
DDR2 SDRAM	5-6
DDR3 SDRAM	5-6
ECC	5-8
Partial Writes	5-9
Partial Bursts	5-10
External Interfaces	5-10
Clock and Reset Interface	5-11
Avalon-ST Data Slave Interface	5-11
Controller-PHY Interface	5-11
Memory Side-Band Signals	5-11
Self-Refresh (Low Power) Interface	5-11
User-Controlled Refresh Interface	5-12

Configuration and Status Register (CSR) Interface	5-12
Top-Level Signals Description	5-13
Sequence of Operations	5-19
Write Command	5-19
Read Command	5-20
Read-Modify-Write Command	5-20
AFI 3.0 Specification	5-21
Implementation	5-21
Bus Width and AFI Ratio	5-21
AFI Parameters	5-22
Parameters Affecting Bus Width	5-22
AFI Signals	5-23
Clock and Reset Signals	5-23
Address and Command Signals	5-24
Write Data Signals	5-25
Read Data Signals	5-26
Calibration Status Signals	5-26
Tracking Management Signals	5-27
Register Maps	5-28
UniPHY Register Map	5-28
Controller Register Map	5-30
Efficiency Monitor and Protocol Checker	5-33
Efficiency Monitor	5-33
Protocol Checker	5-33
Read Latency Counter	5-33
Using the Efficiency Monitor and Protocol Checker	5-33
Avalon CSR Slave and JTAG Memory Map	5-34

Chapter 6. Functional Description—UniPHY

Block Description	6-1
I/O Pads	6-1
Reset and Clock Generation	6-2
Address and Command Datapath	6-3
Write Datapath	6-4
Leveling Circuitry	6-4
Read Datapath	6-6
Sequencer	6-7
Function	6-7
Architecture	6-7
SCC Manager	6-8
RW Manager	6-9
PHY Manager	6-9
Data Manager	6-9
Nios II Processor	6-9
DLL Offset Control Block	6-10
Interfaces	6-11
AFI	6-12
The Memory Interface	6-12
The DLL and PLL Sharing Interface	6-12
The OCT Sharing Interface	6-14
UniPHY Signals	6-15
PHY-to-Controller Interfaces	6-18
Using a Custom Controller	6-22
Using a Vendor-Specific Memory Model	6-22

Chapter 7. Functional Description—Example Top-Level Project

Example Driver	7-2
Read and Write Generation	7-3
Individual Read and Write Generation	7-3
Block Read and Write Generation	7-3
Address and Burst Length Generation	7-3
Sequential Addressing	7-3
Random Addressing	7-3
Sequential and Random Interleaved Addressing	7-3
Example Driver Signals	7-4
Example Driver Add-Ons	7-4
User Refresh Generator	7-4

Chapter 8. Latency

Chapter 9. Timing Diagrams

Chapter 10. Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers

Upgrading from DDR2 or DDR3 SDRAM High-Performance Controller II with ALTMEMPHY Designs	10-1
Generating Equivalent Design	10-1
Replacing the ALTMEMPHY Datapath with UniPHY Datapath	10-2
Resolving Port Name Differences	10-2
Creating OCT Signals	10-4
Running Pin Assignments Script	10-4
Removing Obsolete Files	10-4
Simulating your Design	10-4

Additional Information

Document Revision History	Info-1
How to Contact Altera	Info-1
Typographic Conventions	Info-2

The Altera® DDR2 and DDR3 SDRAM controllers with UniPHY provide low latency, high-performance, feature-rich controller interfaces to industry-standard DDR2 and DDR3 SDRAM memory. The DDR2 SDRAM controller with UniPHY offers full-rate and half-rate DDR2 interfaces, and the DDR3 SDRAM controller with UniPHY offers a half-rate DDR3 SDRAM interface.

The MegaWizard™ interface generates an example top-level project, consisting of an example driver, and your DDR2 or DDR3 SDRAM 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.

The UniPHY datapath is an interface between a memory controller and memory devices and performs read and write operations to the memory.



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® II software, but you are then responsible for all aspects of the design including timing analysis and design constraints.

Release Information

Table 1–1 provides information about this release of the DDR2 and DDR3 SDRAM controllers with UniPHY.

Table 1–1. Release Information

Item	Description
Version	11.0
Release Date	May 2011
Ordering Codes	IP-DDR2/UNI IP-DDR3/UNI
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.

Device Family Support

IP cores provide the following levels of support for target Altera device families:

- For FPGA device support:
 - Preliminary—verified with preliminary timing models for this device
 - Final—verified with final timing models for this device
- For ASIC devices (HardCopy families)
 - HardCopy companion—verified with preliminary timing models for HardCopy companion device
 - HardCopy compilation—verified with final timing models for HardCopy device



For information about migrating your design to a HardCopy device, refer to “[HardCopy Migration Design Guidelines](#)”, in section 6 of this volume.

Table 1-2 shows the level of support offered by the DDR2 and DDR3 SDRAM controllers to each of the Altera device families.



For information about features and supported clock rates for external memory interfaces, refer to the [External Memory Interface System Specifications](#) section in volume 1 of the *External Memory Interface Handbook* or use the [External Memory Specification Estimator](#).

Table 1-2. Device Family Support

Device Family	Support
Arria II GZ	Final
HardCopy [®] III	Companion
HardCopy IV	Companion
Stratix [®] III (For DDR3, only $V_{CC} = 1.1V$ supported)	Final
Stratix IV	Final
Stratix V	Preliminary
Other device families	No support

Features

Table 1–3 summarizes key feature support for the DDR2 and DDR3 SDRAM controllers with UniPHY.

Table 1–3. Key Feature Support for DDR2 and DDR3 SDRAM Controllers with UniPHY

Key Feature	DDR2 SDRAM UniPHY	DDR3 SDRAM UniPHY
High-performance controller II (HPC II)	✓	✓
Half-rate core logic and user interface	✓	✓
Full-rate core logic and user interface	✓	—
Dynamically generated Nios II-based sequencer	✓	✓
Available Efficiency Monitor and Protocol Checker	✓	✓
Quarter-rate core logic and user interface (5)	—	✓
DDR3L support (5)	—	✓
UDIMM and RDIMM in any form factor (1) (2)	✓	✓
Multiple components in a single-rank UDIMM or RDIMM layout	✓	✓
Burst length in HPC II (half-rate)	8	—
Burst length in HPC II (full-rate)	4	—
Burst length of 8 and burst chop of 4 (on the fly)	—	✓
With leveling	✓ (240 MHz and above)	✓ (240 MHz and above) (3)
Without leveling	✓ (below 240 MHz)	—
Maximum data width (4)	144 bits	144 bits
Notes for Table 1–3: (1) For DDR3, the DIMM form is not supported in Arria II GX and Arria II GZ devices. (2) 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. (3) The leveling delay on the board between first and last DDR3 SDRAM component laid out as a DIMM must be less than $0.69 t_{CK}$. (4) 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. (5) For Stratix V devices only, beyond 533MHz.		

Unsupported Features

Table 1–4 summarizes unsupported features for the DDR2 and DDR3 SDRAM controllers with UniPHY.

Table 1–4. Unsupported Features for DDR2 and DDR3 SDRAM Controllers with UniPHY (Part 1 of 2)

Memory Protocol	Unsupported Feature
DDR2 SDRAM	Timing simulation
	Arria II GX support
	Cyclone III support
	Cyclone IV support

Table 1–4. Unsupported Features for DDR2 and DDR3 SDRAM Controllers with UniPHY (Part 2 of 2)

Memory Protocol	Unsupported Feature
DDR3 SDRAM	Timing simulation
	Full-rate
	Arria II GZ DIMM in any form factor
	Stratix III ($V_{CC} = 0.9V$)
	Arria II GX support
	Cyclone III support
	Cyclone IV support

MegaCore Verification

Altera has carried out extensive random, directed tests with functional test coverage using industry-standard models to ensure the functionality of the DDR2 and DDR3 SDRAM controllers with UniPHY. Altera's functional verification of DDR2 and DDR3 controllers with UniPHY use modified Denali models, with certain assertions disabled.

Resource Utilization

This section lists resource utilization for the DDR2 and DDR3 SDRAM controllers with UniPHY for supported device families.

Table 1–5 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 11.0 for Arria II GZ devices.

Table 1–5. Resource Utilization in Arria II GZ Devices (Part 1 of 2)

Protocol	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
Controller							
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
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

Table 1–5. Resource Utilization in Arria II GZ Devices (Part 2 of 2)

Protocol	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
PHY							
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
Total							
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 1–6 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 11.0 for Stratix III devices.

Table 1–6. 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)
Controller							
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

Table 1-6. Resource Utilization in Stratix III Devices (Part 2 of 2)

Protocol	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
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
PHY							
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
Total							
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
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 1-7 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 11.0 for Stratix IV devices.

Table 1-7. Resource Utilization in Stratix IV Devices (Part 1 of 2)

Protocol	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
Controller							
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
PHY							
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
Total							
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

Table 1-7. Resource Utilization in Stratix IV Devices (Part 2 of 2)

Protocol	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
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 1-8 shows the typical size of the DDR2 and DDR3 SDRAM controllers with UniPHY in the Quartus II software version 11.0 for Stratix V devices.

Table 1-8. Resource Utilization in Stratix V Devices (Part 1 of 2)

Protocol	Memory Width (Bits)	Combinational LCs	Logic Registers	M20K Blocks	Memory (Bits)
Controller					
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 (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
PHY					
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 (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

Table 1–8. Resource Utilization in Stratix V Devices (Part 2 of 2)

Protocol	Memory Width (Bits)	Combinational LCs	Logic Registers	M20K Blocks	Memory (Bits)
Total					
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 (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

System Requirements

The DDR2 and DDR3 SDRAM controllers with UniPHY 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*.

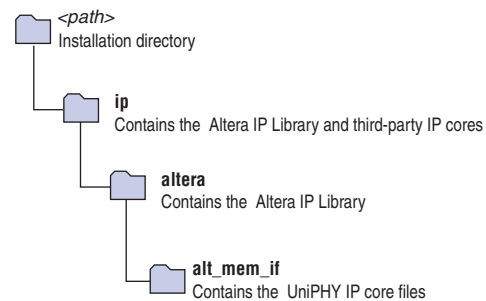
This chapter provides a general overview of the Altera IP core design flow to help you quickly get started with any Altera IP core. The Altera IP Library is installed as part of the Quartus II installation process. You can select and parameterize any Altera IP core from the library. Altera provides an integrated parameter editor that allows you to customize IP cores to support a wide variety of applications. The parameter editor guides you through the setting of parameter values and selection of optional ports. The following sections describe the general design flow and use of Altera IP cores.

Installation and Licensing

The Altera IP Library is distributed with the Quartus II software and downloadable from the Altera website (www.altera.com).

Figure 2-1 shows the directory structure after you install an Altera IP core, where *<path>* is the installation directory. The default installation directory on Windows is `C:\altera\<version number>`; on Linux it is `/opt/altera<version number>`.

Figure 2-1. IP core Directory Structure

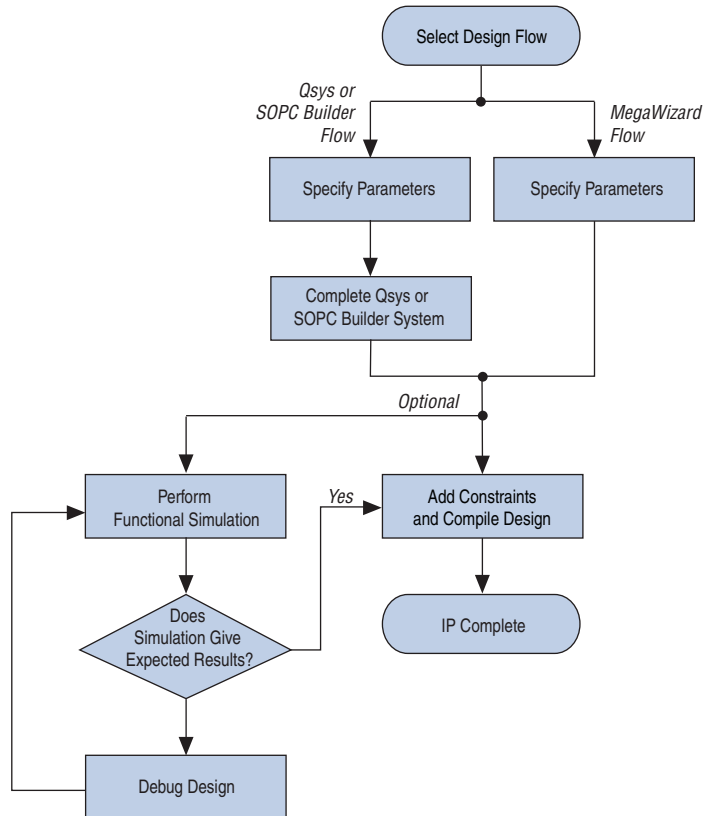


You can evaluate an IP core in simulation and in hardware until you are satisfied with its functionality and performance. Some IP cores require that you purchase a license for the IP core when you want to take your design to production. After you purchase a license for an Altera IP core, you can request a license file from the [Altera Licensing](#) page of the Altera website and install the license on your computer. For additional information, refer to [Altera Software Installation and Licensing](#).

Design Flows

You can use the following flow(s) to parameterize Altera IP cores:

- MegaWizard Plug-In Manager Flow
- SOPC Builder Flow
- Qsys Flow

Figure 2-2. Design Flows *(Note 1)***Note to Figure 2-2:**

(1) Altera IP cores may or may not support the Qsys and SOPC Builder design flows.

The MegaWizard Plug-In Manager flow offers the following advantages:

- Allows you to parameterize an IP core variant and instantiate into an existing design
- For some IP cores, this flow generates a complete example design and testbench.

The SOPC Builder flow offers the following advantages:

- Generates simulation environment
- Allows you to integrate Altera-provided custom components
- Uses Avalon[®] memory-mapped (Avalon-MM) interfaces

The Qsys flow offers the following additional advantages over SOPC Builder:

- Provides visualization of hierarchical designs
- Allows greater performance through interconnect elements and pipelining
- Provides closer integration with the Quartus II software

MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize your IP core and manually integrate the function into your design.

Specifying Parameters

To specify IP core parameters with the MegaWizard Plug-In Manager, follow these steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.
2. In the Quartus II software, launch the **MegaWizard Plug-in Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create or edit a custom IP core variation.
3. To select a specific Altera IP core, click the IP core in the **Installed Plug-Ins** list in the MegaWizard Plug-In Manager.
4. Specify the parameters on the **Parameter Settings** pages. For detailed explanations of these parameters, refer to the “*Parameter Settings*” chapter in this document or the “*Documentation*” button in the MegaWizard parameter editor.



Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor modify the `<installation_directory>/ip/altera/alt_mem_if_interfaces/alt_mem_if_<memory_protocol>_emif/alt_mem_if_<memory_protocol>_mem_model.qprs` file.

5. If the IP core provides a simulation model, specify appropriate options in the wizard to generate a simulation model.



Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models allow for fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model.



For more information about functional simulation models for Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.



Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

6. If the parameter editor includes **EDA** and **Summary** tabs, follow these steps:
 - a. Some third-party synthesis tools can use a netlist that contains the structure of an IP core but no detailed logic to optimize timing and performance of the design containing it. To use this feature if your synthesis tool and IP core support it, turn on **Generate netlist**.
 - b. On the **Summary** tab, if available, select the files you want to generate. A gray checkmark indicates a file that is automatically generated. All other files are optional.



If file selection is supported for your IP core, after you generate the core, a generation report (*<variation_name>.html*) appears in your project directory. This file contains information about the generated files.

7. Click the **Finish** button, the parameter editor generates the top-level HDL code for your IP core, and a simulation directory which includes files for simulation.



The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

8. Click **Yes** if you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

You can now integrate your custom IP core instance in your design, simulate, and compile. While integrating your IP core instance into your design, you must make appropriate pin assignments. You can create a virtual pin to avoid making specific pin assignments for top-level signals while you are simulating and not ready to map the design to hardware.

For some IP cores, the generation process also creates complete example designs. An example design for hardware testing is located in the *<variation_name>_example_design/example_project/* directory. An example design for RTL simulation is located in the *<variation_name>_example_design/simulation/* directory.



For information about the Quartus II software, including virtual pins and the MegaWizard Plug-In Manager, refer to [Quartus II Help](#).

Simulate the IP Core

You can simulate your IP core variation with the functional simulation model and the testbench or example design generated with your IP core. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench.

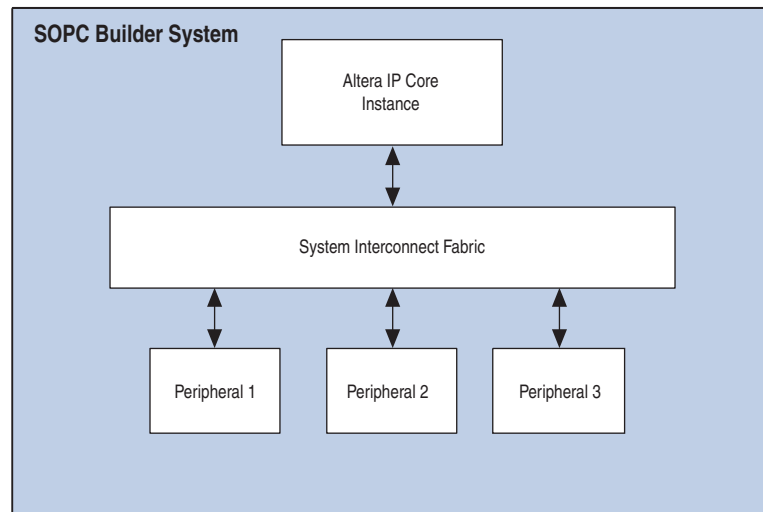
For a complete list of models or libraries required to simulate your IP core, refer to the scripts provided with the testbench.



For more information about simulating Altera IP cores, refer to [Simulating Altera Designs](#) in volume 3 of the *Quartus II Handbook*.

SOPC Builder Design Flow

You can use SOPC Builder to build a system that includes your customized IP core. You easily can add other components and quickly create an SOPC Builder system. SOPC Builder automatically generates HDL files that include all of the specified components and interconnections. SOPC Builder defines default connections, which you can modify. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. [Figure 2-3](#) shows a block diagram of an example SOPC Builder system.

Figure 2-3. SOPC Builder System



-  For more information about system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* and *System Interconnect Fabric for Streaming Interfaces* chapters in the [SOPC Builder User Guide](#) and to the [Avalon Interface Specifications](#).
-  For more information about SOPC Builder and the Quartus II software, refer to the *SOPC Builder Features* and *Building Systems with SOPC Builder* sections in the [SOPC Builder User Guide](#) and to Quartus II Help.

Specify Parameters

To specify IP core parameters in the SOPC Builder flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.

- Specify the required parameters in the parameter editor. For detailed explanations of these parameters, refer to the “*Parameter Settings*” chapter in this document.



Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor modify the `<installation_directory>/ip/altera/alt_mem_if_interfaces/alt_mem_if_<memory_protocol>_emif/alt_mem_if_<memory_protocol>_mem_model.qprs` file.



If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.



You must also turn on **Generate SOPC Builder compatible resets** on the **Controller Settings** tab when parameterizing those cores.

- Click **Finish** to complete the IP core instance and add it to the system.



The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

Complete the SOPC Builder System

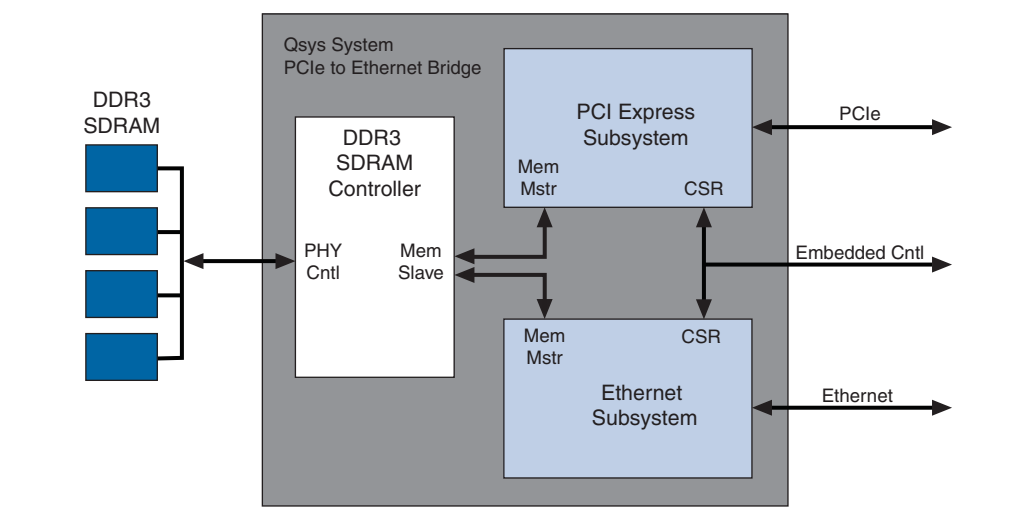
To complete the SOPC Builder system, follow these steps:

- Add and parameterize any additional components. Some IP cores include a complete SOPC Builder system design example.
- Use the Connection panel on the **System Contents** tab to connect the components.
- By default, clock names are not displayed. To display clock names in the **Module Name** column and the clocks in the **Clock** column in the **System Contents** tab, click **Filters** to display the **Filters** dialog box. In the **Filter** list, click **All**.
- Click **Generate** to generate the system. SOPC Builder generates the system and produces the `<system name>.qip` file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
- In the Quartus II software, click **Add/Remove Files in Project** and add the `.qip` file to the project.
- Compile your design in the Quartus II software.

Qsys System Integration Tool Design Flow

You can use the Qsys system integration tool to build a system that includes your customized IP core. You easily can add other components and quickly create a Qsys system. Qsys automatically generates HDL files that include all of the specified components and interconnections. In Qsys, you specify the connections you want. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. Qsys generates Verilog HDL simulation models for the IP cores that comprise your system. Figure 2-4 shows a high level block diagram of an example Qsys system.

Figure 2-4. Example Qsys System



- For more information about the Qsys system interconnect, refer to the *Qsys Interconnect* chapter in volume 1 of the *Quartus II Handbook* and to the *Avalon Interface Specifications*.
- For more information about the Qsys tool and the Quartus II software, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook* and to Quartus II Help.

Specify Parameters

To specify parameters for your IP core using the Qsys flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **Qsys**.
3. On the **System Contents** tab, double-click the name of your IP core to add it to your system. The relevant parameter editor appears.
4. Specify the required parameters in all tabs in the Qsys tool. For detailed explanations of these parameters, refer to the “*Parameter Settings*” chapter in this document.



If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.



Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor modify the `<installation_directory>/ip/altera/alt_mem_if_interfaces/alt_mem_if_<memory_protocol>_emif/alt_mem_if_<memory_protocol>_mem_model.qprs` file.

- Click **Finish** to complete the IP core instance and add it to the system.



The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

Complete the Qsys System

To complete the Qsys system, follow these steps:

- Add and parameterize any additional components.
- Connect the components using the Connection panel on the **System Contents** tab.
- In the **Export As** column, enter the name of any connections that should be a top-level Qsys system port.
- If you intend to simulate your Qsys system, on the **Generation** tab, turn on one or more options under **Simulation** to generate desired simulation files.
- If your system is not part of a Quartus II project and you want to generate synthesis RTL files, turn on **Create synthesis RTL files**.
- Click **Generate** to generate the system. Qsys generates the system and produces the `<system name>.qip` file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
- In the Quartus II software, click **Add/Remove Files in Project** and add the `.qip` file to the project.
- Compile your project in the Quartus II software.





To ensure that the **memory** and **oct** interfaces are exported to the top-level RTL file, be careful not to accidentally rename or delete either of these interfaces in the **Export** column of the **System Contents** tab.

Simulate the System

During system generation, Qsys generates a functional simulation model—or example design that includes a testbench—which you can use to simulate your system in any Altera-supported simulation tool.



For information about the latest Altera-supported simulation tools, refer to the [Quartus II Software Release Notes](#).

-  For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.
-  For information about simulating Qsys systems, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

Qsys and SOPC Builder Interfaces

Table 2-1 and Table 2-2 list the signals available for each interface in Qsys and SOPC Builder and provide a description and guidance on how to connect those interfaces.

Table 2-1. DDR2 SDRAM Controller with UniPHY Interfaces (Part 1 of 5)

Signals in Interface	Interface Type	Description/How to Connect
pll_ref_clk interface		
pll_ref_clk	Clock input	PLL reference clock input.
global_reset interface		
global_reset_n	Reset input	Asynchronous global reset for PLL and all logic in PHY.
soft_reset interface		
soft_reset_n	Reset input	Asynchronous reset input. Resets the PHY, but not the PLL that the PHY uses.
afi_reset interface		
afi_reset_n	Reset output (PLL master/no sharing)	When the interface is in PLL master or no sharing modes, this interface is an asynchronous reset output of the AFI interface. This interface is asserted when the PLL loses lock or the PHY is reset.
afi_reset_in interface		
afi_reset_n	Reset input (PLL slave)	When the interface is in PLL slave mode, this interface is a reset input that you must connect to the afi_reset output of an identically configured memory interface in PLL master mode.
afi_clk interface		
afi_clk	Clock output (PLL master/no sharing)	This AFI interface clock can be full-rate or half-rate memory clock frequency based on the memory interface parameterization. When the interface is in PLL master or no sharing modes, this interface is a clock output.

Table 2-1. DDR2 SDRAM Controller with UniPHY Interfaces (Part 2 of 5)

Signals in Interface	Interface Type	Description/How to Connect
afi_clk_in interface		
afi_clk	Clock input (PLL slave)	<p>This AFI interface clock can be full-rate or half-rate memory clock frequency based on the memory interface parameterization.</p> <p>When the interface is in PLL slave mode, this is a clock input that you must connect to the afi_clk output of an identically configured memory interface in PLL master mode.</p>
afi_half_clk interface		
afi_half_clk	Clock output (PLL master/no sharing)	<p>The AFI half clock that is half the frequency of the afi_clk.</p> <p>When the interface is in PLL master or no sharing modes, this interface is a clock output.</p>
afi_half_clk_in interface		
afi_half_clk	Clock input (PLL slave)	<p>The AFI half clock that is half the frequency of the afi_clk.</p> <p>When the interface is in PLL slave mode, this is a clock input that you must connect to the afi_half_clk output of an identically configured memory interface in PLL master mode.</p>
memory interface		
mem_a	Conduit	Interface signals between the PHY and the memory device.
mem_ba		
mem_ck		
mem_ck_n		
mem_cke		
mem_cs_n		
mem_dm		
mem_ras_n		
mem_cas_n		
mem_we_n		
mem_dq		
mem_dqs		
mem_dqs_n		
mem_odt		
mem_ac_parity		
mem_err_out_n		
mem_parity_error_n		

Table 2-1. DDR2 SDRAM Controller with UniPHY Interfaces (Part 3 of 5)

Signals in Interface	Interface Type	Description/How to Connect
avl interface		
avl_ready	Avalon-MM Slave	Avalon-MM interface signals between the memory interface and user logic.
avl_burst_begin		
avl_addr		
avl_rdata_valid		
avl_rdata		
avl_wdata		
avl_be		
avl_read_req		
avl_write_req		
avl_size		
status interface		
local_init_done	Conduit	Memory interface status signals.
local_cal_success		
local_cal_fail		
oct interface		
rup (Stratix III/IV, Arria II GZ)	Conduit	OCT reference resistor pins for rup/rdn or rzqin.
rdn (Stratix III/IV, Arria II GZ)		
rzq (Stratix V)		
local_powerdown interface		
local_powerdn_ack	Conduit	This powerdown interface for the controller is enabled only when Auto Powerdown is enabled.
local_powerdn_req		
pll_sharing interface		
pll_mem_clk	Conduit	Interface signals for PLL sharing, to connect PLL masters to PLL slaves. This interface is enabled only when you set PLL sharing to master or slave.
pll_write_clk		
pll_addr_cmd_clk		
pll_locked		
pll_avl_clk		
pll_config_clk		
pll_hr_clk		
pll_p2c_read_clk		
pll_c2p_write_clk		
pll_dr_clk		
dll_sharing interface		
dll_delayctrl	Conduit	DLL sharing interface for connecting DLL masters to DLL slaves. This interface is enabled only when you set DLL sharing to master or slave.

Table 2–1. DDR2 SDRAM Controller with UniPHY Interfaces (Part 4 of 5)

Signals in Interface	Interface Type	Description/How to Connect
oct_sharing interface		
seriesterminationcontrol	Conduit	OCT sharing interface for connecting OCT masters to OCT slaves. This interface is enabled only when you set OCT sharing to master or slave.
parallelerminationcontrol		
hcx_dll_reconfig interface		
dll_offset_ctrl_addnsub	Conduit	This DLL reconfiguration interface is enabled only when you enable HCx compatibility mode. You can connect this interface to user-created custom logic to enable DLL reconfiguration.
dll_offset_ctrl_offset		
dll_offset_ctrl_addnsub (1)		
dll_offset_ctrl_offset (1)		
dll_offset_ctrl_offsetctrlout (1)		
dll_offset_ctrl_b_offsetctrlout (1)		
hcx_pll_reconfig interface		
configupdate	Conduit	This PLL reconfiguration interface is enabled only when you enable HCx compatibility mode. You can connect this interface to user-created custom logic to enable PLL reconfiguration.
phasecounterselect		
phasestep		
phaseupdown		
scanclk		
scanclockena		
scandata		
phasedone		
scandataout		
scandone		
hcx_rom_reconfig interface		
hc_rom_config_clock	Conduit	This ROM loader interface is enabled only when you enable HCx compatibility mode. You can connect this interface to user-created custom logic to control loading of the sequencer ROM.
hc_rom_conig_datain		
hc_rom_config_rom_data_ready		
hc_rom_config_init		
hc_rom_config_init_busy		
hc_rom_config_rom_rden		
hc_rom_config_rom_address		
avl_multicast_write interface		
local_multicast	Conduit	Avalon multicast write request for connection to a custom control block. This interface is enabled only when you enable Multi-cast Write Control.
autoprecharge_req interface		
local_autopch_req	Conduit	Precharge interface for connection to a custom control block. This interface is enabled only when you enable Auto-precharge Control.

Table 2–1. DDR2 SDRAM Controller with UniPHY Interfaces (Part 5 of 5)

Signals in Interface	Interface Type	Description/How to Connect
user_refresh interface		
local_refresh_req	Conduit	User refresh interface for connection to a custom control block. This interface is enabled only when you enable User Auto-Refresh Control.
local_refresh_chip		
local_refresh_ack		
self_refresh interface		
local_self_rfsh_req	Conduit	Self refresh interface for connection to a custom control block. This interface is enabled only when you enable Self-refresh Control.
local_self_rfsh_chip		
local_self_rfsh_ack		
ecc_interrupt interface		
ecc_interrupt	Conduit	ECC interrupt signal for connection to a custom control block. This interface is enabled only when you enable Error Detection and Correction Logic.
priority interface		
local_priority	Conduit	Priority interface for commands, for connection to a custom control block.
csr interface		
csr_write_req	Avalon-MM Slave	Configuration and status register signals for the memory interface, for connection to an Avalon_MM master. This interface is enabled only when you enable Configuration and Status Register.
csr_read_req		
csr_waitrequest		
csr_addr		
csr_be		
csr_wdata		
csr_rdata		
csr_rdata_valid		
csr_beginbursttransfer		
csr_burst_count		
Notes:		
(1) Signals available only in DLL master mode.		

Table 2–2. DDR3 SDRAM Controller with UniPHY Interfaces (Part 1 of 6)

Signals in Interface	Interface Type	Description/How to Connect
pll_ref_clk interface		
pll_ref_clk	Clock input	PLL reference clock input.
global_reset interface		
global_reset_n	Reset input	Asynchronous global reset for PLL and all logic in PHY.

Table 2–2. DDR3 SDRAM Controller with UniPHY Interfaces (Part 2 of 6)

Signals in Interface	Interface Type	Description/How to Connect
soft_reset interface		
soft_reset_n	Reset input	Asynchronous reset input. Resets the PHY, but not the PLL that the PHY uses.
afi_reset interface		
afi_reset_n	Reset output (PLL master/no sharing)	When the interface is in PLL master or no sharing modes, this interface is an asynchronous reset output of the AFI interface. This interface is asserted when the PLL loses lock or the PHY is reset.
afi_reset_in interface		
afi_reset_n	Reset input (PLL slave)	When the interface is in PLL slave mode, this interface is a reset input that you must connect to the afi_reset output of an identically configured memory interface in PLL master mode.
afi_clk interface		
afi_clk	Clock output (PLL master/no sharing)	This AFI interface clock can be full-rate or half-rate memory clock frequency based on the memory interface parameterization. When the interface is in PLL master or no sharing modes, this interface is a clock output.
afi_clk_in interface		
afi_clk	Clock input (PLL slave)	This AFI interface clock can be full-rate or half-rate memory clock frequency based on the memory interface parameterization. When the interface is in PLL slave mode, this is a clock input that you must connect to the afi_clk output of an identically configured memory interface in PLL master mode.
afi_half_clk interface		
afi_half_clk	Clock output (PLL master/no sharing)	The AFI half clock that is half the frequency of the afi_clk. When the interface is in PLL master or no sharing modes, this interface is a clock output.

Table 2–2. DDR3 SDRAM Controller with UniPHY Interfaces (Part 3 of 6)

Signals in Interface	Interface Type	Description/How to Connect
afi_half_clk_in interface		
afi_half_clk	Clock input (PLL slave)	<p>The AFI half clock that is half the frequency of the afi_clk.</p> <p>When the interface is in PLL slave mode, this is a clock input that you must connect to the afi_half_clk output of an identically configured memory interface in PLL master mode.</p>
memory interface		
mem_a	Conduit	Interface signals between the PHY and the memory device.
mem_ba		
mem_ck		
mem_ck_n		
mem_cke		
mem_cs_n		
mem_dm		
mem_ras_n		
mem_cas_n		
mem_we_n		
mem_dq		
mem_dqs		
mem_dqs_n		
mem_odt		
mem_reset_n		
mem_ac_parity		
mem_err_out_n		
mem_parity_error_n		
avl interface		
avl_ready	Avalon-MM Slave	Avalon-MM interface signals between the memory interface and user logic.
avl_burst_begin		
avl_addr		
avl_rdata_valid		
avl_rdata		
avl_wdata		
avl_be		
avl_read_req		
avl_write_req		
avl_size		

Table 2–2. DDR3 SDRAM Controller with UniPHY Interfaces (Part 4 of 6)

Signals in Interface	Interface Type	Description/How to Connect
status interface		
local_init_done	Conduit	Memory interface status signals.
local_cal_success		
local_cal_fail		
oct interface		
rup (Stratix III/IV, Arria II GZ)	Conduit	OCT reference resistor pins for rup/rdn or rzqin.
rdn (Stratix III/IV, Arria II GZ)		
rzq (Stratix V)		
local_powerdown interface		
local_powerdn_ack	Conduit	This powerdown interface for the controller is enabled only when Auto Powerdown is enabled.
local_powerdn_req		
pll_sharing interface		
pll_mem_clk	Conduit	Interface signals for PLL sharing, to connect PLL masters to PLL slaves. This interface is enabled only when you set PLL sharing to master or slave.
pll_write_clk		
pll_addr_cmd_clk		
pll_locked		
pll_avl_clk		
pll_config_clk		
pll_hr_clk		
pll_p2c_read_clk		
pll_c2p_write_clk		
pll_dr_clk		
dll_sharing interface		
dll_delayctrl	Conduit	DLL sharing interface for connecting DLL masters to DLL slaves. This interface is enabled only when you set DLL sharing to master or slave.
oct_sharing interface		
seriesterminationcontrol	Conduit	OCT sharing interface for connecting OCT masters to OCT slaves. This interface is inabled only when you set OCT sharing to master or slave.
parallelterminationcontrol		
hcx_dll_reconfig interface		
dll_offset_ctrl_addnsub	Conduit	This DLL reconfiguration interface is enabled only when you enable HCx compatibility mode. You can connect this interface to user-created custom logic to enable DLL reconfiguration.
dll_offset_ctrl_offset		
dll_offset_ctrl_addnsub (1)		
dll_offset_ctrl_offset (1)		
dll_offset_ctrl_offsetctrlout (1)		
dll_offset_ctrl_b_offsetctrlout (1)		

Table 2–2. DDR3 SDRAM Controller with UniPHY Interfaces (Part 5 of 6)

Signals in Interface	Interface Type	Description/How to Connect
hcx_pll_reconfig interface		
configupdate	Conduit	This PLL reconfiguration interface is enabled only when you enable HCx compatibility mode. You can connect this interface to user-created custom logic to enable PLL reconfiguration.
phasecounterselect		
phasesstep		
phaseupdown		
scanclk		
scanclkena		
scandata		
phasedone		
scandataout		
scandone		
hcx_rom_reconfig interface		
hc_rom_config_clock	Conduit	This ROM loader interface is enabled only when you enable HCx compatibility mode. You can connect this interface to user-created custom logic to control loading of the sequencer ROM.
hc_rom_conig_datain		
hc_rom_config_rom_data_ready		
hc_rom_config_init		
hc_rom_config_init_busy		
hc_rom_config_rom_rden		
hc_rom_config_rom_address		
avl_multicast_write interface		
local_multicast	Conduit	Avalon multicast write request for connection to a custom control block. This interface is enabled only when you enable Multi-cast Write Control.
autoprecharge_req interface		
local_autopch_req	Conduit	Precharge interface for connection to a custom control block. This interface is enabled only when you enable Auto-precharge Control.
user_refresh interface		
local_refresh_req	Conduit	User refresh interface for connection to a custom control block. This interface is enabled only when you enable User Auto-Refresh Control.
local_refresh_chip		
local_refresh_ack		
self_refresh interface		
local_self_rfsh_req	Conduit	Self refresh interface for connection to a custom control block. This interface is enabled only when you enable Self-refresh Control.
local_self_rfsh_chip		
local_self_rfsh_ack		

Table 2–2. DDR3 SDRAM Controller with UniPHY Interfaces (Part 6 of 6)

Signals in Interface	Interface Type	Description/How to Connect
ecc_interrupt interface		
ecc_interrupt	Conduit	ECC interrupt signal for connection to a custom control block. This interface is enabled only when you enable Error Detection and Correction Logic.
priority interface		
local_priority	Conduit	Priority interface for commands, for connection to a custom control block.
csr interface		
csr_write_req	Avalon-MM Slave	Configuration and status register signals for the memory interface, for connection to an Avalon_MM master. This interface is enabled only when you enable Configuration and Status Register.
csr_read_req		
csr_waitrequest		
csr_addr		
csr_be		
csr_wdata		
csr_rdata		
csr_rdata_valid		
csr_beginbursttransfer		
csr_burst_count		
Notes:		
(1) Signals available only in DLL master mode.		

Generated Files

When you complete the IP generation flow, there are generated files created in your project directory. The directory structure created varies somewhat, depending on the tool used to parameterize and generate the IP.



The PLL parameters are statically defined in the `<variation_name>_parameters.tcl` at generation time. To ensure timing constraints and timing reports are correct, when you use the GUI to make changes to the PLL component, apply those changes to the PLL parameters in this file.

MegaWizard Plug-in Manager Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the MegaWizard Plug-in Manager flow.

Synthesis

Table 2-3 lists the generated directory structure and key files created by the synthesis flow with the MegaWizard Plug-in Manager.

Table 2-3. Generated Directory Structure and Key Files—MWPIM Synthesis Flow

Directory	File Name	Description
<working_dir>/	<variation_name>.qip	QIP file which refers to all generated files in the synthesis fileset. Include this file in your Quartus II project.
<working_dir>/	<variation_name>.v (for Verilog) or <variation_name>.vhd (for VHDL)	Top-level wrapper for synthesis files.
<working_dir>/<variation_name>/	<variation_name>_0002.v	UniPHY top-level wrapper.
<working_dir>/<variation_name>/	*.v, *.sv, *.tcl, *.sdc, *.ppf	RTL and constraints files.
<working_dir>/<variation_name>/	<variation_name>_p0_pin_assignments.tcl	Pin constraints script to be run after synthesis.

Simulation

Table 2-4 and Table 2-5 list the generated directory structure and key files created by the simulation flow with the MegaWizard Plug-in Manager for Verilog and VHDL, respectively.

Table 2-4. Generated Directory Structure and Key Files—MWPIM Simulation Flow (Verilog)

Directory	File Name	Description
<working_dir>/<variation_name>_sim/	<variation_name>.v	Top-level wrapper.
<working_dir>/<variation_name>_sim/<subcomponent_module>/	*.v, *.sv, *.hex, *.mif	RTL and constraints files.

Table 2-5. Generated Directory Structure and Key Files—MWPIM Simulation Flow (VHDL)

Directory	File Name	Description
<working_dir>/<variation_name>_sim/	<variation_name>.v	Top-level wrapper.
<working_dir>/<variation_name>_sim/<subcomponent_module>/	*.v, *.sv, *.vhd, *.vho, *.hex, *.mif	RTL and constraints files. (.v and .sv files are IEEE Encrypted Verilog, .vho files are generated VHDL.)

Example Design Fileset

Table 2–6 lists the generated directory structure and key files created for the example design for synthesis.

Table 2–6. Generated Directory Structure and Key Files—Example Design for Synthesis

Directory	File Name	Description
<variation_name>_example_design/example_project/	<variation_name>_example.qip	QIP file that refers to all generated files in the synthesizable project.
<variation_name>_example_design/example_project/	<variation_name>_example.qpf	Quartus II project for synthesis flow.
<variation_name>_example_design/example_project/	<variation_name>_example.qsf	Quartus II project for synthesis flow.
<variation_name>_example_design/example_project/ <variation_name>_example/	<variation_name>_example.v	Top-level wrapper.
<variation_name>_example_design/example_project/ <variation_name>_example/submodules/	*.v, *.sv, *.tcl, *.sdc, *.ppf	RTL and constraints files.
<variation_name>_example_design/example_project/ <variation_name>_example/submodules/	<variation_name>_example_if0_p0_pin_assignments.tcl	Pin constraints script to be run after synthesis.

Table 2–7 lists the generated directory structure and key files created for the example design for simulation.

Table 2–7. Generated Directory Structure and Key Files—Example Design for Simulation

Directory	File Name	Description
<variation_name>_example_design/simulation/	<variation_name>_example_sim.qip	QIP file that refers to all generated files in the simulation project.
<variation_name>_example_design/simulation/	<variation_name>_example_sim.qpf	Quartus II project for simulation flow.
<variation_name>_example_design/simulation/	<variation_name>_example_sim.qsf	Quartus II project for simulation flow.
<variation_name>_example_design/simulation/	<variation_name>_example_sim_tb.v	Test bench.
<variation_name>_example_design/simulation/ <variation_name>_sim/	<variation_name>_example_sim.v	Top-level wrapper.
<variation_name>_example_design/simulation/ <variation_name>_sim/submodules/	*.v, *.sv, *.hex, .mif	RTL and ROM data.

SOPC Builder Flow

Table 2-8 lists the generated directory structure and key files created by the SOPC Builder flow.

Table 2-8. Generated Directory Structure and Key Files—SOPC Builder Flow

Directory	File Name ^a	Description
<working_dir>/	<system_name>.qip	QIP file that refers to all generated files in the SOPC Builder project.
<working_dir>/	<system_name>.v	System top-level RTL.
<working_dir>/	<module_name>.v	Module wrapper RTL.
<working_dir>/<module_name>/	*.v, *.sv, *.tcl, *.sdc, *.ppf	Subdirectory of TL and constraints for each system module.

Qsys Flow

The tables in this section list the generated directory structure and key files of interest to users, resulting from the Qsys flow

Synthesis

Table 2-9 lists the generated directory structure and key files created by the synthesis flow with Qsys.

Table 2-9. Generated Directory Structure and Key Files—Qsys Synthesis Flow

Directory	File Name	Description
<working_dir>/<system_name>/synthesis/	<system_name>.qip	QIP file that refers to all generated files in the synthesis filesset.
<working_dir>/<system_name>/synthesis/	<system_name>.v	System top-level RTL.
<working_dir>/<system_name>/synthesis/submodules/	*.v, *.sv, *.tcl, *.sdc, *.ppf	RTL and constraints files.

Simulation

Table 2-10 and Table 2-11 list the generated directory structure and key files created by the simulation flow with Qsys for Verilog and VHDL, respectively.

Table 2-10. Generated Directory Structure and Key Files—Qsys Verilog Simulation

Directory	File Name	Description
<working_dir>/<system_name>/simulation/	<system_name>.v	System top-level RTL.
<working_dir>/<system_name>/simulation/submodules/	*.v, *.sv, *.hex, *.mif	RTL and ROM data.

Table 2-11. Generated Directory Structure and Key Files—Qsys VHDL Simulation

Directory	File Name	Description
<working_dir>/<system_name>/simulation/	<system_name>.vhd	VHDL system top-level RTL.
<working_dir>/<system_name>/simulation/submodules/	*.v, *.sv, *.vhd, *.vho, *.hex, *.mif	RTL and ROM data. (.v and .sv files are IEEE Encrypted Verilog, .vho files are generated VHDL.)

This chapter describes the parameters you can set in the UniPHY GUI.

PHY Settings

Use this tab to apply the PHY settings suitable for your design.

FPGA

Speed grade is the speed grade of the targeted FPGA device, which affects the generated timing constraints and timing reporting.

Clocks

Table 3–1 shows the clock parameters.

Table 3–1. Clock Parameters

Parameter	Description
Memory clock frequency	The frequency of the clock that drives the memory device. Use up to 4 decimal places of precision.
Achieved memory clock frequency	The actual frequency the PLL will generate to drive the external memory interface (memory clock).
PLL reference clock frequency	The frequency of the input clock that feeds the PLL. Use up to 4 decimal places of precision.
Full or half rate on Avalon-MM interface	The width of data bus on the Avalon-MM interface. Full results in a width of 2× the memory data width. Half results in a width of 4× the memory data width.
Achieved local clock frequency	The actual frequency the PLL will generate to drive the local interface for the memory controller (AFI clock).

Advanced Settings

Table 3–2 shows the advanced settings parameters.

Table 3–2. Advanced Settings (Part 1 of 2)

Parameter	Description
Advanced clock phase control	Enables access to clock phases. Default value should suffice for most DIMMs and board layouts, but can be modified if necessary to compensate for larger address and command versus clock skews.
Additional address and command clock phase	Allows you to increase or decrease the amount of phase shift on the address and command clock. The base phase shift center aligns the address and command clock at the memory device, which may not be the optimal setting under all circumstances. Increasing or decreasing the amount of phase shift can improve timing. The default value is 0 degrees.

Table 3–2. Advanced Settings (Part 2 of 2)

Parameter	Description
Additional CK/CK# phase	Allows you to increase or decrease the amount of phase shift on the CK/CK# clock. The base phase shift center aligns the address and command clock at the memory device, which may not be the optimal setting under all circumstances. Increasing or decreasing the amount of phase shift can improve timing. Increasing or decreasing the phase shift on CK/CK# also impacts the read, write, and leveling transfers, which increasing or decreasing the phase shift on the address and command clocks does not.
Supply voltage (DDR3)	The supply voltage and sub-family type of memory. (DDR3L is currently supported only on Stratix V.)
I/O standard	The I/O standard voltage.
PLL sharing mode	When set to “No sharing,” a PLL block is instantiated but no PLL signals are exported. When set to “Master,” a PLL block is instantiated and the signals are exported. When set to “Slave,” a PLL interface is exposed and an external PLL master must be connected to drive them.
DLL sharing mode	When set to “No sharing,” a DLL block is instantiated but no DLL signals are exported. When set to “Master,” a DLL block is instantiated and the signals are exported. When set to “Slave,” a DLL interface is exposed and an external DLL master must be connected to drive them.
OCT sharing mode	When set to “No sharing,” an OCT block is instantiated but no OCT signals are exported. When set to “Master,” an OCT block is instantiated and the signals are exported. When set to “Slave,” an OCT interface is exposed and an external OCT control block must be connected to drive them.
HardCopy compatibility	Enables all required HardCopy compatibility options for the generated IP core. For some parameterizations, a pipeline stage is added to the write datapath to help the more challenging timing closure in HardCopy; the pipeline stage does not affect overall read and write latency.
Reconfigurable PLL location	When the PLL used in the UniPHY memory interface is set to be reconfigurable at run time, the location of the PLL must be specified. This assignment will generate a PLL which can only be placed in the given sides.

Memory Parameters

Use this tab to apply the memory parameters from your memory manufacturer’s data sheet.

Table 3–3 shows the memory parameters.

Table 3–3. Memory Parameters (Part 1 of 2)

Parameter	Description
Memory vendor	The vendor of the memory device, which is set automatically when you select a configuration from the list of memory presets.
Memory format	The format of the memory device.
Memory device speed grade	The maximum frequency at which the memory device can run.

Table 3–3. Memory Parameters (Part 2 of 2)

Parameter	Description
Total interface width	The total number of DQ pins of the memory device. Limited to 144 bits for DDR2 and DDR3 SDRAM (with or without leveling).
DQ/DQS group size	The number of DQ bits per DQS group.
Number of DQS groups	The number of DQS groups is calculated automatically from the Total interface width and the DQ/DQS group size parameters.
Number of chip selects	The number of chip-selects the IP core uses for the current device configuration.
Number of clocks per chip select	The width of the clock bus to each chip select of the memory interface.
Row address width	The width of the row address on the memory interface.
Column address width	The width of the column address on the memory interface.
Bank-address width	The width of the bank address bus on the memory interface.
Enable DM pins	Specifies whether the DM pins of the memory device are driven by the FPGA. You can turn off this option to avoid overusing FPGA device pins when using x4 mode memory devices.
DQS# Enable (DDR2)	Available only for DDR2.

Memory Initialization Options—DDR2

The following tables describe the memory initialization parameters.

Table 3–4 shows the address and command parity setting.

Table 3–4. Address and command parity

Parameter	Description
Address and command parity	Enables address/command parity checking.

Mode Register 0

Table 3–5 shows the mode register 0 parameters.

Table 3–5. Mode Register 0 Parameters

Parameter	Description
Burst length	Specifies the burst length.
READ burst type	Determines whether the controller performs accesses within a given burst in sequential or interleaved order.
DLL precharge power down	Determines whether the DLL in the memory device is in slow exit mode or in fast exit mode during precharge power down.
Memory CAS latency setting	Determines the number of clock cycles between the READ command and the availability of the first bit of output data at the memory device.

Mode Register 1

Table 3-6 shows the mode register 1 parameters.

Table 3-6. Mode Register 1 Parameters

Parameter	Description
Output drive strength setting	Determines the output driver impedance setting at the memory device.
Memory additive CAS latency setting	Determines the posted CAS additive latency of the memory device.
Memory on-die termination (ODT) setting	Determines the on-die termination resistance at the memory device.

Mode Register 2

SRT Enable determines whether selfrefresh temperature (SRT) enable is 1x refresh rate or 2x refresh rate (for high-temperature operations).

Memory Initialization Options—DDR3

Table 3-7 shows the general initialization settings.

Table 3-7. General initialization settings

Parameter	Description
Mirror addressing: 1 per chip select	Specifies the mirror addressing. For example for four CS, 1101 makes CS #3, #2 and #0 mirrored.
Address and command parity	Enables address/command parity checking.

Mode Register 0

Table 3-8 shows the mode register 0 parameters.

Table 3-8. Mode Register 0 Parameters

Parameter	Description
READ burst type	Specifies whether accesses within a given burst are in sequential or interleaved order.
DLL precharge power down	Specifies whether the DLL in the memory device is off or on during precharge power-down.
Memory CAS latency setting	The number of clock cycles between the read command and the availability of the first bit of output data at the memory device.

Mode Register 1

Table 3–9 shows the mode register 1 parameters.

Table 3–9. Mode Register 1 Parameters

Parameter	Description
Output drive strength setting	The output driver impedance setting at the memory device.
Memory additive CAS latency setting	The posted CAS additive latency of the memory device.
ODT Rtt nominal value	The on-die termination resistance at the memory device.

Mode Register 2

Table 3–10 shows the mode register 2 parameters.

Table 3–10. Mode Register 2 Parameters

Parameter	Description
Auto selfrefresh method	Disable or enable auto selfrefresh.
Selfrefresh temperature	Specifies the selfrefresh temperature as Normal or Extended .
Memory write CAS latency setting	The number of clock cycles from the releasing of the internal write to the latching of the first data in, at the memory device.
Dynamic ODT (Rtt_WR) value	The mode of the dynamic ODT feature of the memory device.

Memory Timing

Use this tab to apply the memory timings from your memory manufacturer's data sheet. Table 3–11 shows the memory timing parameters.

Table 3–11. Memory Timing Parameters (Part 1 of 2)

Parameter	Description
tIS (base)	Address and control setup to CK clock rise.
tIH (base)	Address and control hold after CK clock rise.
tDS (base)	Data setup to clock (DQS) rise.
tDH (base)	Data hold after clock (DQS) rise.
tDQSQ	DQS, DQS# to DQ skew, per access.
tQHS (DDR2)	DQ output hold time from DQS, DQS# (absolute time value)
tQH (DDR3)	DQ output hold time from DQS, DQS# (percentage of tCK).
tDQSCK	DQS output access time from CK/CK#.
tDQSS	First latching edge of DQS to associated clock edge (percentage of tCK).
tQSH (DDR3)	DQS Differential High Pulse Width (percentage of tCK). Specifies the minimum high time of the DQS signal received by the memory.
tDQSH (DDR2)	
tDSH	DQS falling edge hold time from CK (percentage of tCK).

Table 3-11. Memory Timing Parameters (Part 2 of 2)

Parameter	Description
tDSS	DQS falling edge to CK setup time (percentage of tCK).
tINIT	Memory initialization time at power-up.
tMRD	Load mode register command period.
tRAS	Active to precharge time.
tRCD	Active to read or write time.
tRP	Precharge command period.
tREFI	Refresh command interval.
tRFC	Auto-refresh command interval.
tWR	Write recovery time.
tWTR	Write to read period.
tFAW	Four active window time.
tRRD	RAS to RAS delay time.
tRTP	Read to precharge time.

Board Settings

Use the **Board Settings** tab to model the board-level effects in the timing analysis.



For accurate timing results, you must enter board settings parameters that are correct for your PCB.

The IP core supports single and multiple chip-select configurations. Altera has determined the effects on the output signalling of these configurations for certain Altera boards, and has stored the effects on the output slew rate and the intersymbol interference (ISI) within the wizard.



These stored values are representative of specific Altera boards. You must change the values to account for the board-level effects for your board. You can use HyperLynx or similar simulators to obtain values that are representative of your board.

The **Board Settings** tab allows you to enter board-related data. In the **Intersymbol Interference** and **Board Skews** sections, you enter information derived during your PCB development process of prelayout (line) and postlayout (board) simulation.



For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *Volume 5: Design Flow Tutorials* of the *External Memory Interface Handbook*. For information about timing deration methodology, refer to *Chapter 3, Timing Deration Methodology for Multiple Chip Select DDR2 and DDR3 SDRAM Designs* in Volume 4 of the *External Memory Interface Handbook*.

Setup and Hold Derating

The slew rate of the output signals affects the setup and hold times of the memory device.

You can specify the slew rate of the output signals to see their effect on the setup and hold times of both the address and command signals and the DQ signals, or specify the setup and hold times directly.



You should enter information derived during your PCB development process of prelayout (line) and postlayout (board) simulation.

Table 3–12 shows the setup and hold derating parameters.

Table 3–12. Setup and Hold Derating Parameters

Parameter	Description
Derating method	Derating method.
CK/CK# slew rate (differential)	CK/CK# slew rate (differential).
Address/Command slew rate	Address and command slew rate.
DQS/DQS# slew rate (Differential)	DQS and DQS# slew rate (differential).
DQ slew rate	DQ slew rate.
tIS	Address/command setup time to CK.
tIH	Address/command hold time from CK.
tDS	Data setup time to DQS.
tDH	Data hold time from DQS.

Intersymbol Interference

Intersymbol interference is the distortion of a signal in which one symbol interferes with subsequent symbols. Typically, when going from a single chip-select configuration to a multiple chip-select configuration there is an increase in intersymbol interference because there are multiple stubs causing reflections.

Table 3–13 shows the intersymbol interference parameters.

Table 3–13. ISI Parameters (Part 1 of 2)

Parameter	Description
Derating method	Choose between default Altera settings (with specific Altera boards) or manually enter board simulation numbers obtained for your specific board.
Address and command eye reduction (setup)	The reduction in the eye diagram on the setup side (or left side of the eye) due to ISI on the address and command signals compared to a case when there is no ISI. (For single rank designs, ISI can be zero; in multirank designs, ISI is necessary for accurate timing analysis.)
Address and command eye reduction (hold)	The reduction in the eye diagram on the hold side (or right side of the eye) due to ISI on the address and command signals compared to a case when there is no ISI.

Table 3-13. ISI Parameters (Part 2 of 2)

Parameter	Description
DQ eye reduction	The total reduction in the eye diagram due to ISI on DQ signals compared to a case when there is no ISI. Altera assumes that the ISI reduces the eye width symmetrically on the left and right side of the eye.
Delta DQS arrival time	The increase in variation on the range of arrival times of DQS compared to a case when there is no ISI. Altera assumes that the ISI causes DQS to further vary symmetrically to the left and to the right.

Board Skews

PCB traces can have skews between them that can reduce timing margins. Furthermore, skews between different chip selects can further reduce the timing margin in multiple chip-select topologies. The **Board Skews** section of the parameter editor allows you to enter parameters to compensate for these variations. Very large board trace skews should be specified in your board trace model.

Table 3-14 shows the board skew parameters.

Table 3-14. Board Skew Parameters (Part 1 of 2)

Parameter	Description
Maximum CK delay to DIMM/device	The delay of the longest CK trace from the FPGA to the memory device, whether on a DIMM or the same PCB as the FPGA.
Maximum DQS delay to DIMM/device	The delay of the longest DQS trace from the FPGA to the memory device, whether on a DIMM or the same PCB as the FPGA.
Minimum delay difference between CK and DQS	The minimum skew (or largest negative skew) between the CK signal and any DQS signal when arriving at the same DIMM over all DIMMs. This value affects the write leveling margin for DDR3 interfaces with leveling in multirank configurations.
Maximum delay difference between CK and DQS	The maximum skew (or largest positive skew) between the CK signal and any DQS signal when arriving at the same DIMM over all DIMMs. This value affects the Write Leveling margin for DDR3 interfaces with leveling in multi-rank configurations.
Maximum skew within DQS group	The largest skew between DQ and DM signals in a DQS group. This value affects the read capture and write margins for DDR2 and DDR3 SDRAM interfaces in all configurations (single or multiple chip-select, DIMM or component).
Maximum skew between DQS groups	The largest skew between DQS signals in different DQS groups. This value affects the resynchronization margin in memory interfaces without leveling such as DDR2 SDRAM and discrete-device DDR3 SDRAM in both single- or multiple chip-select configurations.
Average delay difference between DQ and DQS	The average delay difference between each DQ signal and the DQS signal, calculated by averaging the longest and smallest DQ signal delay values minus the delay of DQS.

Table 3-14. Board Skew Parameters (Part 2 of 2)

Parameter	Description
Maximum skew within address and command bus	The largest skew between the address and command signals.
Average delay difference between address and command and CK	<p>A value equal to the average of the longest and smallest address and command signal delay values, minus the delay of the CK signal. The value can be positive or negative. Positive values represent address and command signals that are longer than CK signals; negative values represent address and command signals that are shorter than CK signals. The Quartus II software uses this skew to optimize the delay of the address and command signals to have appropriate setup and hold margins for DDR2 and DDR3 SDRAM interfaces.</p> <p>You should derive this value through board simulation.</p>

Controller Settings

Use this tab to apply the controller settings suitable for your design.



This section describes parameters for the High Performance Controller II (HPC II) with advanced features introduced in version 11.0 for designs generated in version 11.0. Designs created in earlier versions and regenerated in version 11.0 do not inherit the new advanced features; for information on parameters for HPC II without the version 11.0 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.

Avalon Interface

[Table 3-15](#) shows the Avalon-MM interface parameters.

Table 3-15. Avalon-MM Interface Parameters

Parameter	Description
Generate power-of-2 bus widths for SOPC Builder	Rounds down the Avalon-MM side data bus to the nearest power of 2.
Generate SOPC Builder compatible resets	You must enable this option if the IP core is to be used in an SOPC Builder system. When turned on, the reset inputs become associated with the PLL reference clock and the paths must be cut. This option must be enabled for SOPC Builder, but is not required when using the MegaWizard Plug-in Manager or Qsys.
Maximum Avalon-MM burst length	Specifies the maximum burst length on the Avalon-MM bus. Affects the <code>AVL_SIZE_WIDTH</code> parameter.
Enable Avalon-MM byte-enable signal	When turned on, the controller will add the byte-enable signal for the Avalon-MM bus.
Avalon interface address width	The address width on the Avalon-MM interface.
Avalon interface data width	The data width on the Avalon-MM interface.

Low Power Mode

Table 3-16 shows low-power mode parameters.

Table 3-16. Low-Power mode Parameters

Parameter	Description
Enable self-refresh controls	Enables the self-refresh signals on the controller top-level design. These controls allow you to control when the memory is placed into self-refresh mode.
Enable auto-power down	Allows the controller to automatically place the memory into power-down mode after a specified number of idle cycles. Specifies the number of idle cycles after which the controller powers down the memory in the auto-power down cycles parameter.
Auto power-down cycles	The number of idle controller clock cycles after which the controller automatically powers down the memory. The legal range is from 1 to 65,535 controller clock cycles.

Efficiency

Table 3-17 shows the efficiency parameters.

Table 3-17. Efficiency Parameters

Parameter	Description
Enable user auto-refresh controls	Enables the user auto-refresh control signals on the controller top level. These controller signals allow you to control when the controller issues memory autorefresh commands.
Enable auto-precharge control	Enables the autoprecharge control on the controller top level. Asserting the autoprecharge control signal while requesting a read or write burst allows you to specify whether the controller should close (autoprecharge) the currently open page at the end of the read or write burst.
Local-to-memory address mapping	Allows you to control the mapping between the address bits on the Avalon-MM interface and the chip, row, bank, and column bits on the memory.
Command queue look-ahead depth	Selects a look-ahead depth value to control how many read or writes requests the look-ahead bank management logic examines. Larger numbers are likely to increase the efficiency of the bank management, but at the cost of higher resource usage. Smaller values may be less efficient, but also use fewer resources. The valid range is from 1 to 16.
Enable reordering	Allows the controller to perform command and data reordering to achieve the highest possible efficiency.
Starvation limit for each command	Specifies the number of commands that can be served before a waiting command is served. The valid range is from 1 to 63.

Configuration, Status, and Error Handling

Table 3-18 shows the configuration, status, and error handling parameters.

Table 3-18. Configuration, Status, and Error Handling Parameters

Parameter	Description
Enable Configuration and Status Register Interface	Enables run-time configuration and status interface for the memory controller. This option adds an additional Avalon-MM slave port to the memory controller top level, which you can use to change or read out the memory timing parameters, memory address sizes, mode register settings and controller status. If Error Detection and Correction Logic is enabled, the same slave port also allows you to control and retrieve the status of this logic.
CSR port host interface	Specifies the type of connection to the CSR port. The port can be exported, internally connected to a JTAG Avalon Master, or both.
Enable error detection and correction logic	Enables ECC for single-bit error correction and double-bit error detection. Your memory interface must be a multiple of 40 or 72 bits wide to use ECC.
Enable auto error correction	Allows the controller to perform auto correction when a single-bit error is detected by the ECC logic.

Diagnostics

The **Diagnostics** tab allows you to set parameters for certain diagnostic functions.

Simulation Options

Table 3-19 describes parameters for simulation.

Table 3-19. Simulation Options

Parameter	Description
Auto-calibration mode	<p>Specifies whether you want to improve simulation performance by reducing calibration. There is no change to the generated RTL. The following autocalibration modes are available:</p> <ul style="list-style-type: none">■ Skip calibration—provides the fastest simulation. It loads the settings calculated from the memory configuration and enters user mode.■ Quick calibration—calibrates (without centering) one bit per group before entering user mode.■ Full calibration—calibrates the same as in hardware, and includes all phases, delay sweeps, and centering on every data bit. You can use timing annotated memory models. Be aware that full calibration can take hours or days to complete.
Skip memory initialization	Causes the example testbench to skip the memory initialization sequence. This setting does not change the RTL, but can speed up simulation.
Enable support for Nios II ModelSim flow in Eclipse	Initializes the memory interface for use with the Run as Nios II ModelSim flow with Eclipse.

Efficiency Monitor and Protocol Checker Settings

Table 3-20 describes parameters for the Efficiency Monitor and Protocol Checker.

Table 3-20. Efficiency Monitor and Protocol Checker Settings

Parameter	Description
Enable the Efficiency Monitor and Protocol Checker on the Controller Avalon Interface	Enables efficiency monitor and protocol checker block on the controller Avalon interface.

The wizard generates a Synopsis design constraint (.sdc) script, `<variation_name>.sdc`, and a pin assignment script, `<variation_name>_pin_assignments.tcl`. Both the .sdc and the `<variation_name>_pin_assignments.tcl` support multiple instances. These scripts iterate through all instances of the core and apply the same constraints to all of them.

Add Pin and DQ Group Assignments

The pin assignment script, `<variation_name>_pin_assignments.tcl`, sets up the I/O standards and the input/output termination for the DDR2 or DDR3 SDRAM controller with UniPHY. This script also helps to relate the DQ pin groups together for the Quartus II Fitter to place them correctly.

The pin assignment script does not create a clock for the design. You must create a clock for the design and provide pin assignments for the signals of both the example driver and testbench that the MegaCore variation generates.

Run the `<variation_name>_pin_assignments.tcl` to add the input and output termination, I/O standards, and DQ group assignments to the example design. To run the pin assignment script, follow these steps:

1. On the Processing menu, point to **Start**, and click **Start Analysis and Synthesis**.
2. On the Tools menu click **Tcl Scripts**.
3. Specify the `pin_assignments.tcl` file and click **Run**.



If the PLL input reference clock pin does not have the same I/O standard as the memory interface I/Os, a no-fit might occur because incompatible I/O standards cannot be placed in the same I/O bank.



On Stratix V devices, unused pins in a DQ/DQS group cannot be used as CK or CKn pins; CK and CKn pins must use DQ/DQS-capable pins, but must not be in the same group as any other DQ/DQS pins. Do not place CK or CKn pins in the same group as any other pin that uses onboard memory hardware (such as DQ, DQS, or other CK or CKn pins).

If a given interface contains multiple CK/CKn pairs, they must all reside in the same group. CK/CKn pairs from different interfaces cannot reside in the same I/O bank.



For more information about pin planning, refer to *Device and Pin Planning* in volume 2 of the *External Memory Interface Handbook*. For related information, refer to the *Recommended Design Flow* chapter in volume 1 of the *External Memory Interfaces Handbook*.

Board Settings

The DDR2 or DDR3 SDRAM controller with UniPHY Board Settings tab includes default settings and guideline values based on typical Altera boards. These suggested values are starting points for initial design investigation, but you should verify that the parameters you use are appropriate for your hardware implementation.



Failure to specify appropriate values for setup and hold derating, intersymbol interference, and board skews, or failure to add board trace models representative of your hardware implementation, may result in inaccurate timing estimates and unreliable memory interface operation.


Compile the Design

To compile the design, on the Processing menu, click **Start Compilation**.

After you have compiled the top-level file, you can perform RTL simulation or program your targeted Altera device to verify the top-level file in hardware.



For more information about simulating, refer to the *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

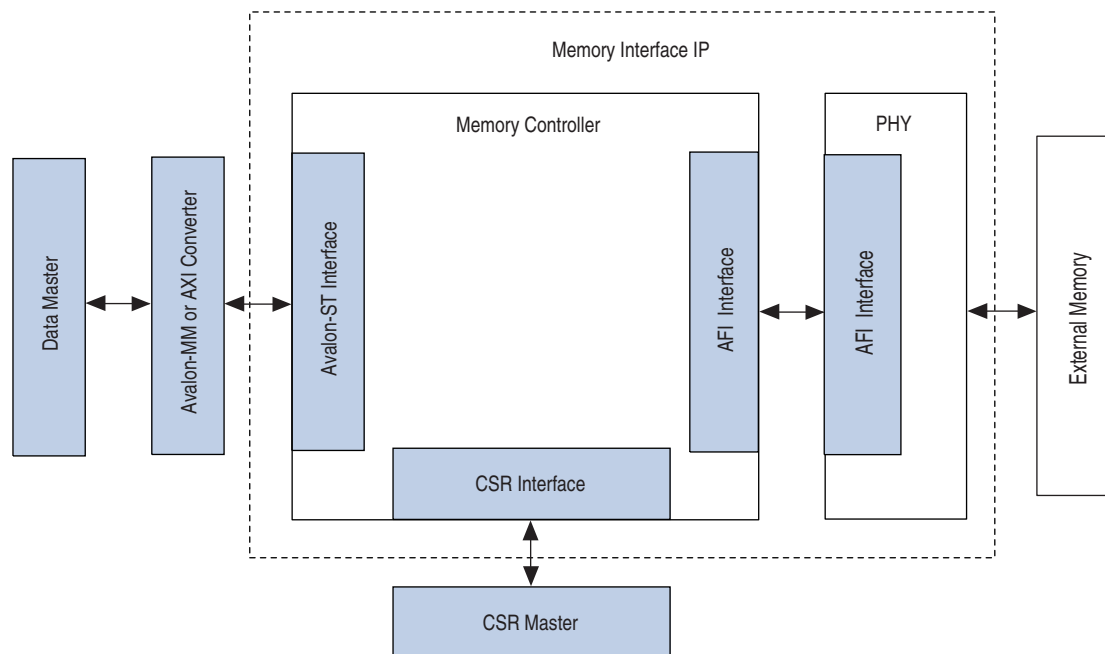
 This chapter describes the High Performance Controller II (HPC II) with advanced features introduced in version 11.0 for designs generated in the Quartus II software version 11.0. Designs created in earlier versions and regenerated in version 11.0 do not inherit the new advanced features; for information on HPC II without the version 11.0 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 memory 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 memory interface architecture.

Figure 5–1. High-Level Diagram of Memory Interface Architecture

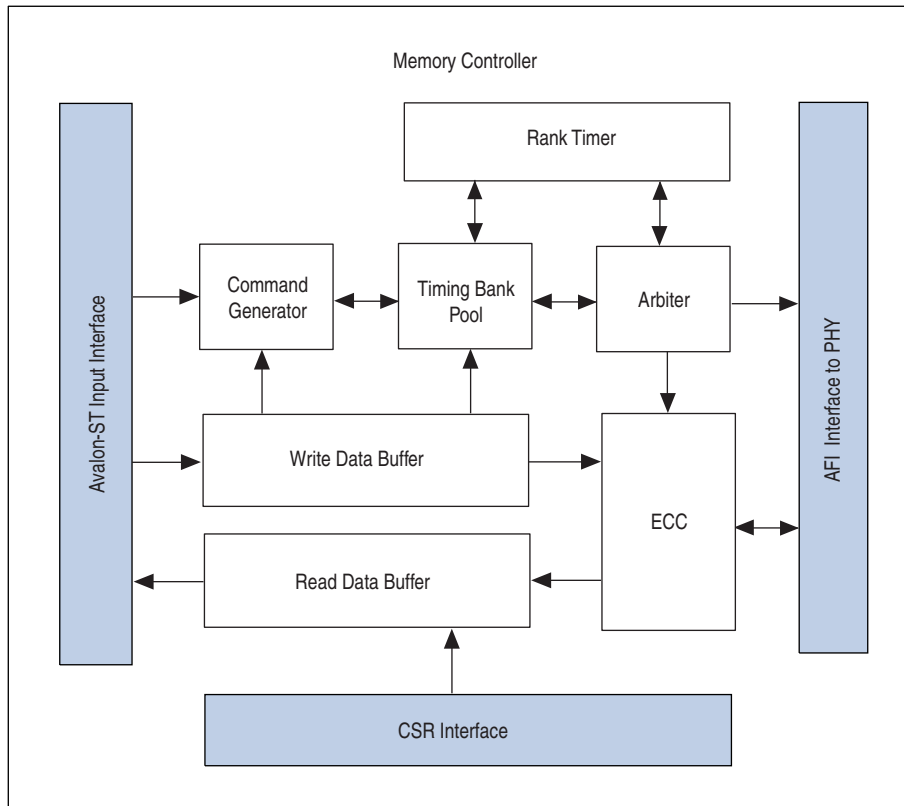


The memory interface consists of the memory controller logic block, the physical (PHY) logic layer, and their associated interfaces.

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](#).

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 follows 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” on page 5-21](#).



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.

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 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.

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 shows 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
mem_cs [0]	mem_odt [0]

Table 5–2 shows 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
mem_cs [0]	mem_odt [0]
mem_cs [1]	mem_odt [1]

Table 5–3 shows 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
mem_cs [0]	mem_odt [1]
mem_cs [1]	mem_odt [0]

Table 5–4 shows 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
mem_cs [0]	mem_odt [2]
mem_cs [1]	mem_odt [3]
mem_cs [2]	mem_odt [0]
mem_cs [3]	mem_odt [1]

DDR3 SDRAM

Table 5–5 shows 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 shows 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 shows 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 shows 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 shows 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]
mem_cs[2]	mem_odt[0] and mem_odt[2]
mem_cs[3]	mem_odt[1] and mem_odt[3]

Table 5–10 shows 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]

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. 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 5-3 and Figure 5-4 show partial write operations for the controller, for full and half rate configurations, respectively.

Figure 5-3. Partial Write for the Controller—Full Rate

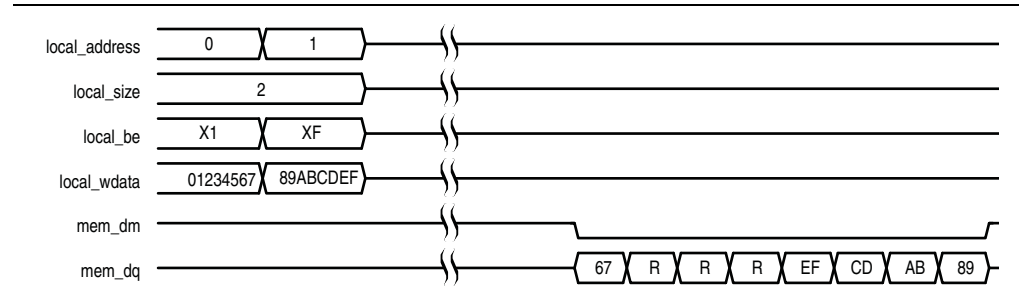
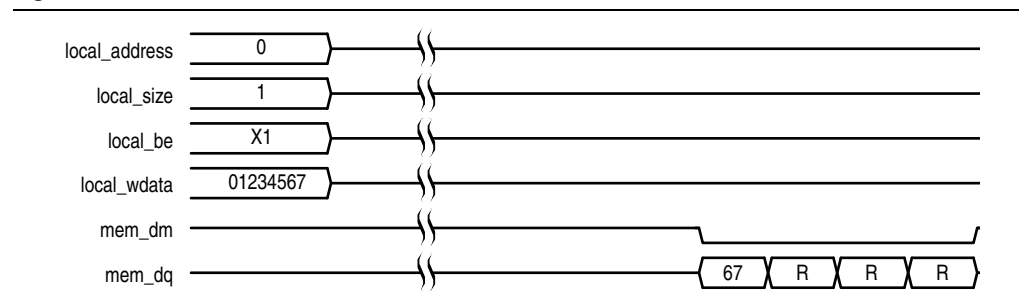


Figure 5-4. Partial Write for the Controller—Half Rate

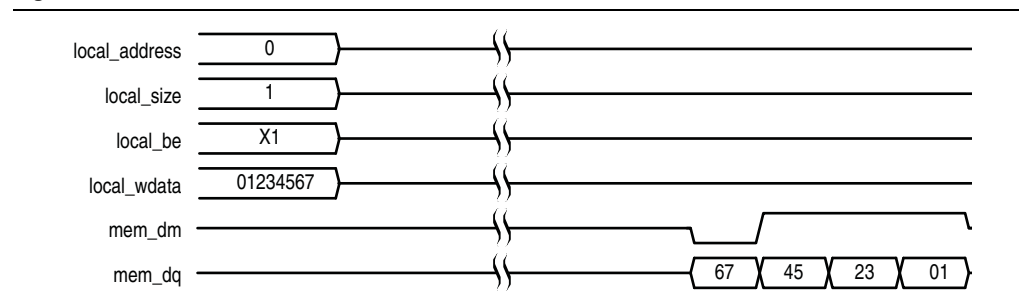


Partial Bursts

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 5-5 shows a partial burst operation for the controller.

Figure 5-5. 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](#).

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](#)” on page 5-21.

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.

User-Controlled Refresh Interface

The optional user-controlled refresh interface consists of a request signal and an acknowledgement signal, which allow you to determine when the controller issues refresh signals to the memory device. This interface gives you 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. Once the controller successfully issues a refresh command to the memory device, the controller asserts the refresh acknowledge signal for one clock cycle.

If you want to send consecutive refresh commands, you should keep the refresh request asserted, which causes the controller to issue another refresh command and again assert the acknowledge signal for a one clock cycle. You can perform up to nine consecutive refresh commands.

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 5-11 summarizes the controller's external interfaces.

Table 5-11. Summary of Controller External Interfaces (Part 1 of 2)

Interface Name	Display Name	Type	Description
Clock and Reset Interface			
Clock and Reset Interface	Clock and Reset Interface	AFI (1)	Clock and reset generated by UniPHY to the controller.
Avalon-ST Data Slave Interface			
Command Channel	Avalon-ST Data Slave Interface	Avalon-ST (2)	Address and command channel for read and write, SCSD.
Write Data Channel	Avalon-ST Data Slave Interface	Avalon-ST (2)	Write Data Channel, SCMD.
Read Data Channel	Avalon-ST Data Slave Interface	Avalon-ST (2)	Read data channel, SCMD with read data error response.
Controller-PHY Interface			
AFI 2.0	AFI Interface	AFI (1)	Interface between controller and PHY.
Memory Side-Band Signals			
Self Refresh (Low Power) Interface	Self Refresh (Low Power) Interface	Avalon Control & Status Interface (2)	SDRAM-specific signals to place memory into low-power mode.
User-Controller Refresh Interface	User-Controller Refresh Interface	Avalon Control & Status Interface (2)	SDRAM-specific signals to request memory refresh.

Table 5-11. Summary of Controller External Interfaces (Part 2 of 2)

Interface Name	Display Name	Type	Description
Configuration and Status Register (CSR) Interface			
CSR	Configuration and Status Register Interface	Avalon-MM (2)	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 5-12 shows the clock and reset signals.



The suffix `_n` denotes active low signals.

Table 5-12. Clock and Reset Signals (Part 1 of 2)

Name	Direction	Description
<code>global_reset_n</code>	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.
<code>pll_ref_clk</code>	Input	The reference clock input to PLL.
<code>phy_clk</code>	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.
<code>reset_phy_clk_n</code>	Output	The reset signal that the PHY provides to the user. The IP core asserts <code>reset_phy_clk_n</code> asynchronously and deasserts synchronously to <code>phy_clk</code> clock domain.
<code>aux_full_rate_clk</code>	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 <code>phy_clk</code> and you can use it whenever you require a 2x clock. In full-rate designs, the same PLL output as the <code>phy_clk</code> signal drives this clock.
<code>aux_half_rate_clk</code>	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 <code>phy_clk</code> and you can use it, for example to clock the user side of a half-rate bridge. In half-rate designs, or if the Enable Half Rate Bridge option is turned on. The same PLL output that drives the <code>phy_clk</code> signal drives this clock.
<code>dll_reference_clk</code>	Output	Reference clock to feed to an externally instantiated DLL.
<code>reset_request_n</code>	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.
<code>soft_reset_n</code>	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.

Table 5–12. Clock and Reset Signals (Part 2 of 2)

Name	Direction	Description
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.
parallelerminationcontrol	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.

Table 5–13 shows the controller local interface signals.

Table 5-13. Local Interface Signals (Part 1 of 4)

Signal Name	Direction	Description
avl_addr[]	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 Local-to-Memory Address Mapping option in the Controller Settings page.</p> <p>The IP core sizes the width of this bus according to the following equations:</p> <ul style="list-style-type: none"> ■ Full rate controllers <p>For one chip select: width = row bits + bank bits + column bits – 1</p> <p>For multiple chip selects: width = chip bits + row bits + bank bits + column bits – 1</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 local_address to bank, row and column address:</p> <p>avl_addr is 24 bits wide</p> <p>avl_addr[23:11] = row address[12:0]</p> <p>avl_addr[10:9] = bank address[1:0]</p> <p>avl_addr[8:0] = column address[9:1]</p> <p>The IP core ignores the least significant bit (LSB) of the column address (multiples of four) 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"> ■ Half rate controllers <p>For one chip select: width = row bits + bank bits + column bits – 2</p> <p>For multiple chip selects: width = chip bits + row bits + bank bits + column bits – 2</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 local_address to bank, row and column address:</p> <p>avl_addr is 23 bits wide</p> <p>avl_addr[22:10] = row address[12:0]</p> <p>avl_addr[9:8] = bank address[1:0]</p> <p>avl_addr[7:0] = column address[9:2]</p> <p>The IP core ignores two LSBs of the column address on the memory side, because the local data width is four times that of the memory data bus width.</p>
avl_be[]	Input	<p>Byte enable signal, which you use to mask off individual bytes during writes. avl_be is active high; mem_dm is active low.</p> <p>To map avl_wdata and avl_be to mem_dq and mem_dm, consider a full-rate design with 32-bit avl_wdata and 16-bit mem_dq.</p> <p>avl_wdata = < 22334455 > 667788AA > BBCCDDEE ></p> <p>avl_be = < 1100 > 0110 > 1010 ></p> <p>These values map to:</p> <p>Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC></p> <p>Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 ></p>

Table 5-13. Local Interface Signals (Part 2 of 4)

Signal Name	Direction	Description
avl_burstbegin	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 avl_ready 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 avl_ready. The IP core samples this signal at the rising edge of phy_clk when avl_write_req is asserted. After the slave deasserts the avl_ready signal, the master keeps all the write request signals asserted until avl_ready signal becomes high again.</p> <p>For read transactions, assert this signal for one clock cycle when read request is asserted and avl_addr from which the data should be read is given to the memory. After the slave deasserts avl_ready (waitrequest_n in Avalon interface), the master keeps all the read request signals asserted until avl_ready becomes high again.</p>
avl_read_req	Input	Read request signal. You cannot assert read request and write request signals at the same time. The controller must deassert reset_phy_clk_n before you can assert local_autopch_req.
local_refresh_req	Input	User-controlled refresh request. If Enable User Auto-Refresh Controls option is turned on, local_refresh_req 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 local_refresh_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_refresh_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>
avl_size[]	Input	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 Local Maximum Burst Count option. With the derived width, you specify a value ranging from 1 to the local maximum burst count specified.
avl_wdata[]	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.
avl_write_req	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.

Table 5-13. Local Interface Signals (Part 3 of 4)

Signal Name	Direction	Description
local_autopch_req	Input	<p>User control of autoprecharge. If you turn on Enable Auto-Precharge Control, 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>
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	<p>User control of the self-refresh feature. If you turn on Enable Self-Refresh Controls, 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.</p>
local_init_done	Output	<p>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.</p> <p>The controller still accepts read and write requests before local_init_done is asserted, however it does not issue them to the memory until it is safe to do so.</p> <p>This signal does not indicate that the calibration is successful.</p>
avl_rdata[]	Output	<p>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.</p>
avl_rdata_error	Output	<p>Asserted if the current read data has an error. This signal is only available if you turn on Enable Error Detection and Correction Logic. The controller asserts this signal with the avl_rdata_valid signal.</p> <p>If the controller encounters double-bit errors, no correction is made and the controller asserts this signal.</p>
avl_rdata_valid	Output	<p>Read data valid signal. The avl_rdata_valid signal indicates that valid data is present on the read data bus.</p>

Table 5-13. Local Interface Signals (Part 4 of 4)

Signal Name	Direction	Description
avl_ready	Output	The avl_ready signal indicates that the controller is ready to accept request signals. If controller asserts the avl_ready signal in the clock cycle that it asserts a read or write request, the controller accepts that request. The controller deasserts the avl_ready signal to indicate that it cannot accept any more requests. The controller can buffer eight read or write requests, after which the avl_ready 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 Enable User Auto-Refresh Controls , local_refresh_ack 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 local_self_rfsh_req 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	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.

Table 5-14 shows the controller interface signals.

Table 5-14. Interface Signals (Part 1 of 2)

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 DDR3 SDRAM 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 (1)	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 (1)	Output	Active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset. DDR3 SDRAM only

Table 5-14. Interface Signals (Part 2 of 2)

Signal Name	Direction	Description
mem_err_out_n (1)	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 5-14:

(1) This signal is for registered DIMMs only.

Table 5-15 shows the CSR interface signals.

Table 5-15. CSR Interface Signals

Signal Name	Direction	Description
csr_addr[]	Input	Register map address. The width of <code>csr_addr</code> is 16 bits.
csr_be[]	Input	Byte-enable signal, which you use to mask off individual bytes during writes. <code>csr_be</code> is active high.
csr_wdata[]	Input	Write data bus. The width of <code>csr_wdata</code> is 32 bits.
csr_write_req	Input	Write request signal. You cannot assert <code>csr_write_req</code> and <code>csr_read_req</code> signals at the same time.
csr_read_req	Input	Read request signal. You cannot assert <code>csr_read_req</code> and <code>csr_write_req</code> signals at the same time.
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.

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.

AFI 3.0 Specification

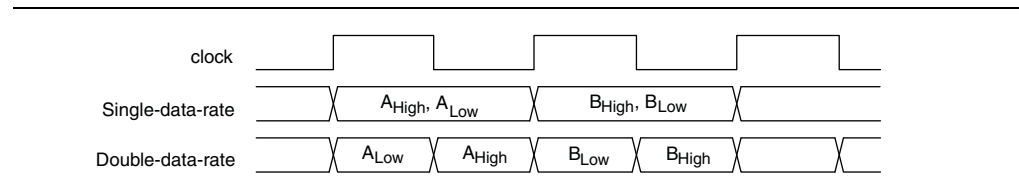
The Altera AFI interface, which is loosely based on the DDR-PHY Interface (DFI) specification, defines communication between the controller and physical layer (PHY) in the external memory interface. The following sections describe the AFI implementation and the AFI signals.

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 5-6](#).

Figure 5-6. 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 5-16 through Table 5-24 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 5-16. Ratio Parameters

Parameter Name	Description
AFI_RATE_RATIO	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.
DATA_RATE_RATIO	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.
ADDR_RATE_RATIO	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 5-17. 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).

Table 5-18. Derived AFI Parameters (Part 1 of 2)

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

Table 5–18. Derived AFI Parameters (Part 2 of 2)

Parameter Name	Derivation Equation
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 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 5–19. 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 5–20. Address and Command Signals

Signal Name	Direction	Width	Description
afi_ba	Input	AFI_BANKADDR_WIDTH	Bank address.
afi_cke	Input	AFI_CS_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 use 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.)
afi_we_n	Input	AFI_CONTROL_WIDTH	WE# (for DDR2, DDR3, and RDRAM 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 RDRAM II memory devices.)
afi_rst_n	Input	AFI_CONTROL_WIDTH	RESET# (for DDR3 memory devices.)
afi_odt	Input	AFI_CS_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 5–21. 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 and DDR3 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, 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.)

Read Data Signals

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

Table 5-22. 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).

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 5-23. Calibration Status Signals (Part 1 of 2)

Signal Name	Direction	Width	Description
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.

Table 5-23. Calibration Status Signals (Part 2 of 2)

Signal Name	Direction	Width	Description
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 5-24. Tracking Management Signals

Signal Name	Direction	Width ^a	Description
afi_ctl_refresh_done	Input	MEM_IF_CS_WIDTH	Signal from controller to tracking manager, indicating that a refresh has occurred; also acts as an acknowledge signal.
afi_seq_busy	Output	MEM_IF_CS_WIDTH	Stall request and acknowledge signal from sequencer to controller when DQS tracking is needed and in progress.
afi_ctl_long_idle	Input	MEM_IF_CS_WIDTH	Signal from controller to tracking manager, indicating that it has exited low power state without a refresh; also acts as an acknowledge signal.

Register Maps

Table 5–25 shows the overall register mapping for the DDR2 and DDR3 SDRAM Controller with UniPHY.

Table 5–25. Register Map

Address	Description
UniPHY Register Map	
0x001	Reserved.
0x004	UniPHY status register 0.
0x005	UniPHY status register 1.
0x006	UniPHY status register 2.
0x007	UniPHY memory initialization parameters register 0.
Controller Register Map	
0x100	ALTMEMPHY status and control register.
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 5–26 shows the register map for UniPHY.

Table 5–26. UniPHY Register Map (Part 1 of 2)

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 5–26. UniPHY Register Map (Part 2 of 2)

Address	Bit	Name	Default	Access	Description
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 <code>afi_cal_success</code> . Writing to this bit has no effect.
	25	AFI_CAL_FAIL	—	Read only	Reports the value of the UniPHY <code>afi_cal_fail</code> . 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 <code>AFI_CAL_FAIL=1</code> .
	15:8	Reserved.	0	—	Reserved for future use.
	23:16	INIT_FAILING_GROUP	—	Read only	Initial failing error group of calibration. Only applicable if <code>AFI_CAL_FAIL=1</code> .
	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.

Controller Register Map

The controller register map allows you to control the memory controller settings. Table 5-27 shows the register map for the controller.

Table 5-27. Controller Register Map (Part 1 of 4)

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.
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.

Table 5–27. Controller Register Map (Part 2 of 4)

Address	Bit	Name	Default	Access	Description
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.
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.
	19:16	Burst Length	—	Read write	Value must match memory burst length.
	31:20	Reserved.	0	—	Reserved for future use.

Table 5–27. Controller Register Map (Part 3 of 4)

Address	Bit	Name	Default	Access	Description
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 autoconnections 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	The address of the most recent ECC error. This address is a memory burst-aligned local address.

Table 5–27. Controller Register Map (Part 4 of 4)

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.

Efficiency Monitor and Protocol Checker

The Efficiency Monitor and Protocol Checker is an available feature that 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 the *Debugging* section in volume 4 of the *External Memory Interface Handbook*. For information about the Avalon interface, refer to *Avalon Interface Specifications*.

Avalon CSR Slave and JTAG Memory Map

Table 5–28 summarizes 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.

Table 5–28. Avalon CSR Slave and JTAG Memory Map (Part 1 of 2)

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.
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).

Table 5–28. Avalon CSR Slave and JTAG Memory Map (Part 2 of 2)

Address	Bit	Name	Default	Access	Description
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.
0x49	31:0	Read latency total [63:32]	—	Read Only	The upper 32 bits of the total read latency.
0x50	7:0	Illegan 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.

This chapter describes the PHY part of the DDR2 and DDR3 SDRAM controllers with UniPHY.

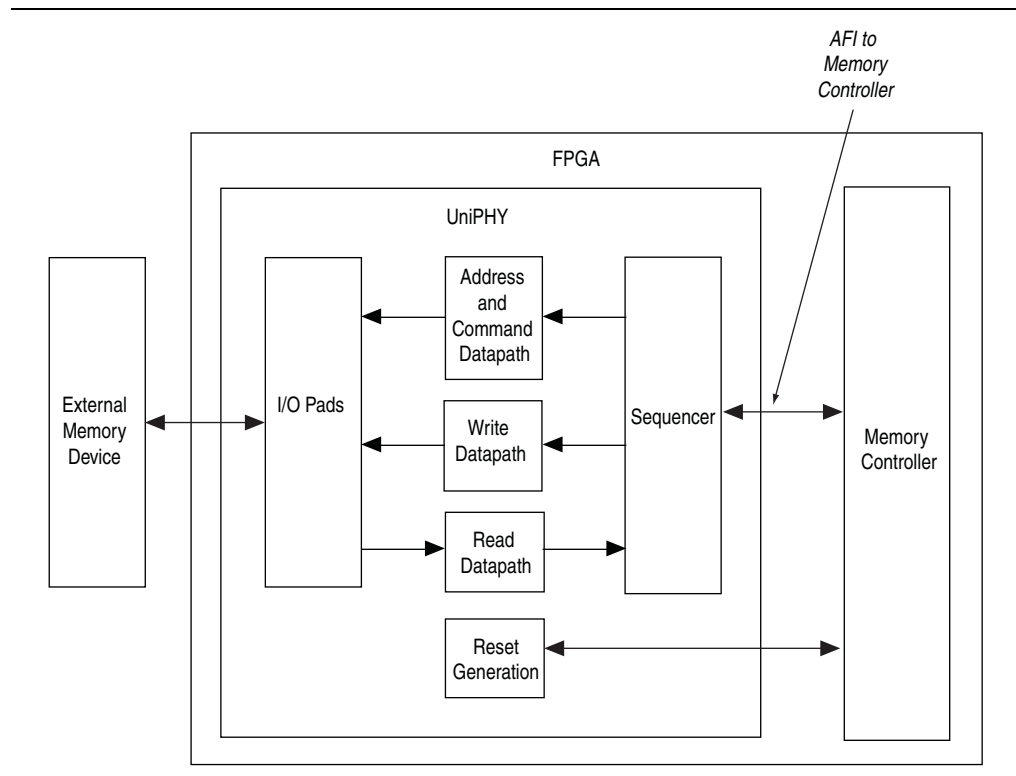
Block Description

The PHY comprises the following major functional units:

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

Figure 6–1 shows the PHY block diagram.

Figure 6–1. PHY Block Diagram



I/O Pads

The I/O pads contain all the I/O instantiations. The bulk of the UniPHY I/O circuitry is encapsulated in the **ALTDQ_DQS** megafunction for IV-series device families and earlier, and in the **ALTDQ_DQS2** megafunction for Stratix V devices.

Reset and Clock Generation

The clocking operation 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 is always at full-rate. The PHY-AFI domain interfaces with the memory controller and can be either a full-rate or half-rate clock based on the choice of the controller. Table 6-1 lists the clocks required for half-rate designs.

Table 6-1. Clocks—Half-Rate Designs

Clock	Source	Clock Rate	Phase	Clock Network Type	Description
pll_mem_clk	PLL: C1	Full	0° (1) -45° (2)	Dual-regional (4)	Output clock to memory.
pll_write_clk	PLL: C2	Full	90° (1) -135° (2) 45° (3)	Dual-regional (4)	Clock for the write data out to memory (data is center aligned with the delayed pll_write_clk).
pll_addr_cmd_clk	PLL: C3	Half	Set in wizard (default 270°)	Dual-regional	Clock for the address and command out to memory (address and command is center aligned with memory clock).
pll_avl_clk	PLL:C5	—	0°	Dual-regional	NIOS sequencer clock.
pll_config_clk	PLL:C6	—	0°	Dual-regional	Scan chain clock.
DQS	Memory	Full	90°	Local	The read data strobe.
Notes for Table 6-1: (1) For memory frequencies >240 MHz (2) For memory frequencies <=240 MHz (3) For memory frequencies >=240 MHz, for Stratix V devices only. (4) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.					

Table 6-2 lists the clocks required for full-rate designs.

Table 6-2. Clocks—Full-Rate Designs

Clock	Source	Clock Rate	Phase	Clock Network Type	Description
pll_mem_clk	PLL: C1	Full	90° (1) 0° (2)	Dual-regional (3)	Output clock to memory.
pll_write_clk	PLL: C2	Full	180° (1) -90° (2)	Dual-regional (3)	Clock for write data out to memory (data is center aligned with the delayed pll_write_clk).
pll_addr_cmd_clk	PLL: C3	Full	Set in wizard (default 225°)	Dual-regional	Clock for the address and command out to memory. 180° gives address and command center aligned with memory clock; 225° produces best overall timing results.
pll_avl_clk	PLL:C5	—	0°	Dual-regional	NIOS sequencer clock.
pll_config_clk	PLL:C6	—	0°	Dual-regional	Scan chain clock.

Table 6–2. Clocks—Full-Rate Designs

Clock	Source	Clock Rate	Phase	Clock Network Type	Description
DQS	Memory	Full	90°	Local	The read data strobe.
Notes for Table 6–2: (1) For memory frequencies >240 MHz (2) For memory frequencies ≤240 MHz (3) For parameterizations with interface width >36, pll_mem_clk and pll_write_clk are assigned to use the global network.					

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. The number of synchronization pipeline stages is set to 4.

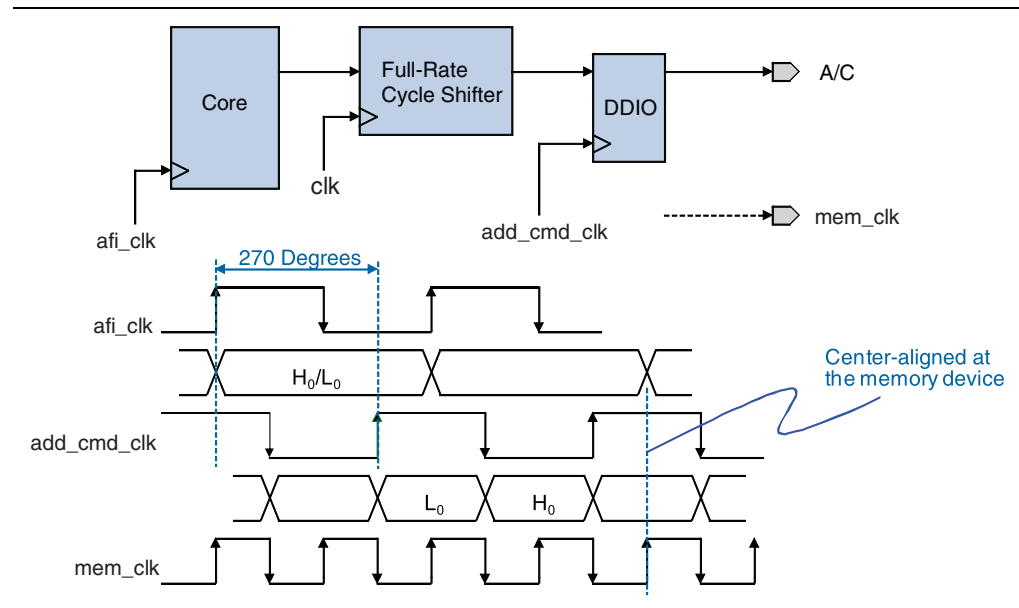
Address and Command Datapath

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. Address and command signals are generated in the AFI clock domain and sent to the memory device in the address and command clock domain. The DDIO stage converts the half-rate signals into full-rate signals, when the AFI clock runs at 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 6–2, this offset is 270 degrees. The fitter can further optimize margins based on the actual delays and clock skews. In half-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 6-2 illustrates the address and command datapath.

Figure 6-2. 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 DQ pins are bidirectional and shared between read and write. The write data valid signal from the memory controller generates the output enable signal to control the output buffer. It also generates the dynamic termination control signal, which selects between series (output mode) and parallel (input mode) termination. An ALT_OCT megafunction (instantiated automatically in the top-level file) configures the termination values.

Leveling Circuitry

For DDR2, leveling circuitry is invoked automatically for frequencies above 240 MHz; no leveling is used for frequencies below 240 MHz. For DDR3, leveling is always invoked.

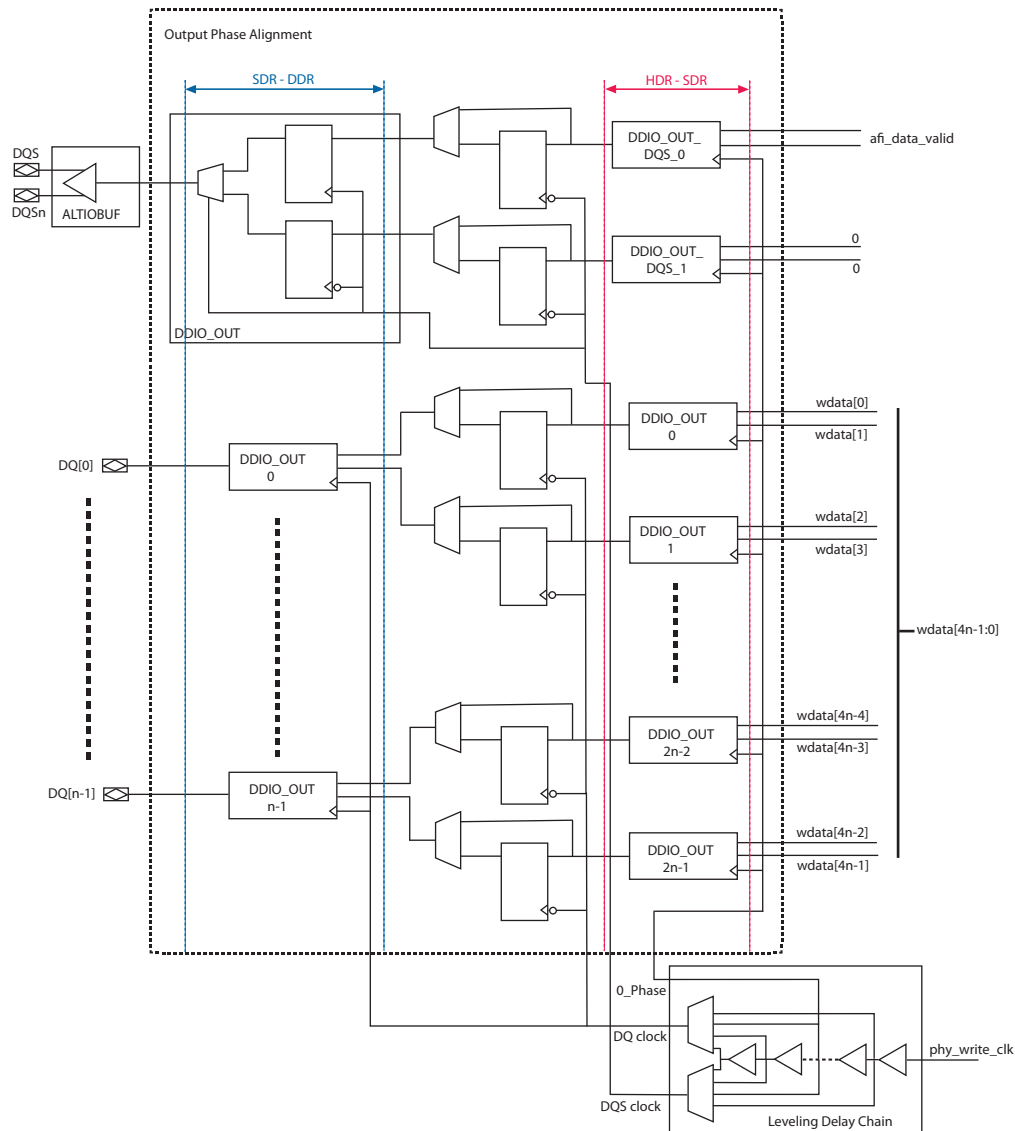
For frequencies below 240 MHz, you should use a tree-style layout. For frequencies above 240 MHz, you can choose either a leveled or balanced-T or Y topology, as the leveled PHY will calibrate correctly to either implementation. For DDR3 implementations at higher frequencies, a fly-by topology is recommended for best signal integrity.

Although a levelling PHY is generated for DDR2 and DDR3 above 240MHz, a balanced-T topology for address/command/clock should be used for DDR2 for all supported FPGAs. For DDR3 with Arria II GZ devices, a levelling PHY is always generated, but a balanced-T PCB topology for address/command/clock should be used for DDR2 and DDR3.

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

Figure 6-3 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 peripheral 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 ALTIOBUF buffer which creates a pair of pseudo differential clocks that connect to the memory. In full-rate designs, the HDR-SDR portion of the output phase alignment is bypassed. The `<variation_name>_pin_assignments.tcl` script automatically specifies the logic option that associates all DQ pins to the DQS pin. The Quartus II Fitter treats the pins as a DQS/DQ pin group.

Figure 6-3. Write Datapath for a Leveling Interface



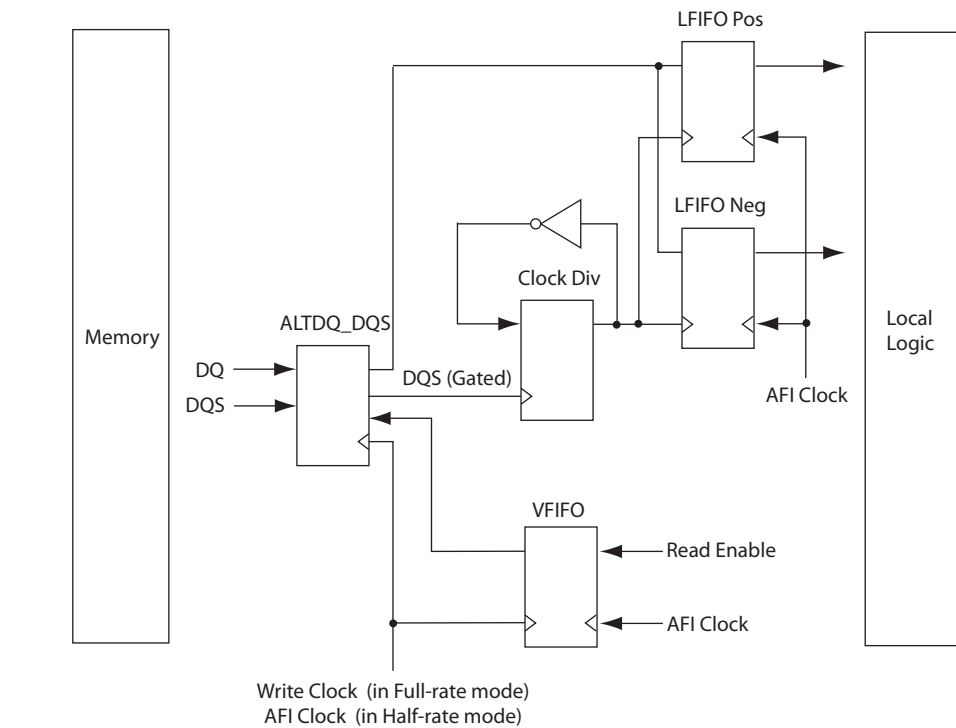
Read Datapath

Figure 6-4 shows the read datapath. The read data is captured in the input mode ALTDQ_DQS in the I/O. The captured data is then forwarded to the read datapath. The read datapath synchronizes read data from the read capture clock domain to the AFI clock domain and converts data from single data rate to half data rate (for half-rate designs only).

The VFIFO and LFIFO buffers are part of the PHY; the VFIFO buffer determines when data from memory is ready to be sampled, while the LFIFO buffer stores the sampled data. The VFIFO buffer has settings that determine when the read valid signal is generated, and the LFIFO buffer has settings that determine read latency.

In half-rate designs, the write side of the FIFO buffer should be half the width and double the depth of the read side of the FIFO buffer. The read side only reads one entry after the IP core writes the write side into two entries, which effectively converts data from SDR to HDR. In full-rate designs, the size of the FIFO buffer is the same for both write and read as both sides operate at the same rate. For half-rate designs, the FIFO operates at half-rate on both read and write sides, and contains 4 half-rate entries; for full-rate designs, the FIFO operates at full-rate on both read and write sides, and contains 8 full-rate entries.

Figure 6-4. Read Datapath



Sequencer

Commencing with ACDS 11.0, the DDR2 and DDR3 SDRAM controllers with UniPHY employ a new Nios[®] II-based sequencer that is parameterizable and is dynamically generated at run time.

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.

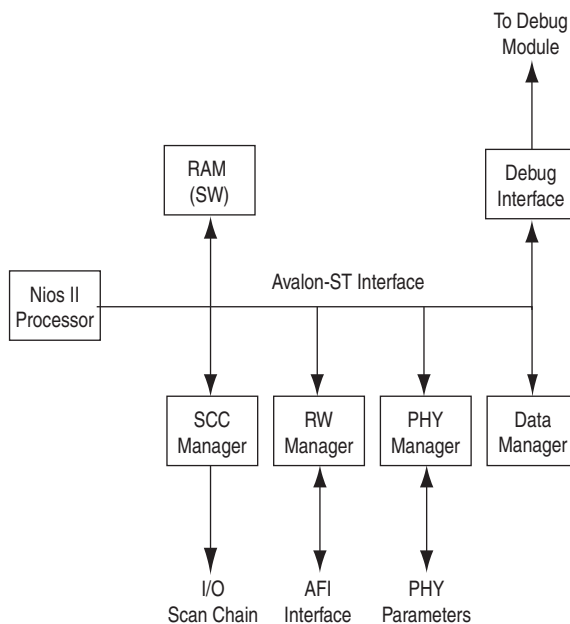


For more information about calibration, refer to the *Calibration Stages* chapter in Volume 4 of the *External Memory Interface Handbook*.

Architecture

Figure 6-5 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 6-5. Sequencer Block Diagram

The sequencer is not a static entity; it is assembled dynamically each time you generate the DDR2 and DDR3 SDRAM controllers with UniPHY IP. Consequently, the C code that defines calibration routines is compiled during generation, resulting in the display of many compilation-related messages during IP generation. These messages are normal, and you may ignore them.

The C code that defines the calibration routines is available for your reference in the `\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-ST 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-ST 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 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)—some memory devices provide a special register that contains calibration specific patterns that you can read. This command enables or disables access to this 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—write 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—which the initialization sequence requires.

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.

Nios II Processor

The Nios II processor manages the calibration algorithm; the NIOS II processor is unavailable once 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 comprises 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.

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 6-6 shows the major blocks of the UniPHY and how it interfaces with the external memory device and the controller.


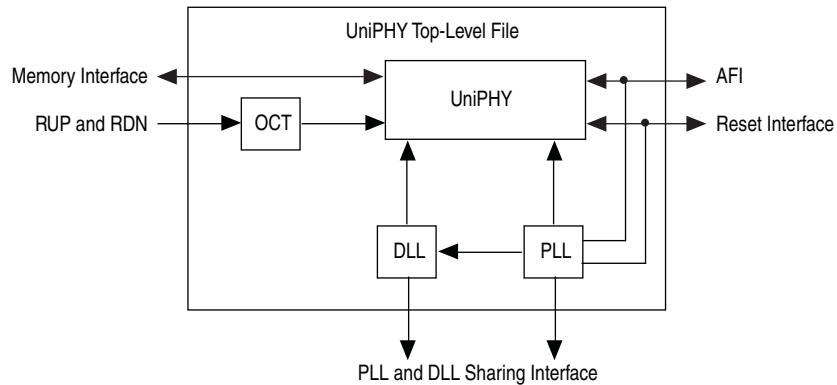
 Instantiating the delay-locked loop (DLL) and the phase-locked loop (PLL) on the same level as the UniPHY eases DLL and PLL sharing.

Figure 6-6. UniPHY Interfaces with the Controller and the External Memory



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

- AFI
- Memory interface
- DLL and 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” on page 5-21](#).

The Memory Interface

For more information on the memory interface, refer to [“UniPHY Signals” on page 6-15](#).

The DLL and PLL Sharing Interface

You can generate the UniPHY memory interface as a master or as a slave, depending on the settings of the **PLL sharing mode** and **DLL sharing mode** options on the **PHY Settings** tab of the parameter editor. The top level of a generated UniPHY variant contains both a PLL and a DLL; the PLL produces a variety of required clock signals derived from the reference clock, and the DLL produces a delay codeword. The top-level defines the PLL and DLL output signals as outputs for the master and as inputs for the slave. When you instantiate master and slave variants into your HDL code, you must connect the PLL outputs from the master to the clock inputs of the slaves.



The master **.qip** file must appear before the slave **.qip** file in the Quartus II Settings File (**.qsf**).

The UniPHY memory interface requires one PLL and one DLL to produce the clocks and delay codeword. The PLL and DLL can be shared using a master and slave scenario. When the PLL or DLL sharing mode is set to **Master** or **Slave**, the PLL or DLL signals are available as outputs or inputs, respectively, in the top-level file. When you set the sharing mode to **Master**, the RTL code instantiates the PLL or DLL as a master which drives the clock or DLL codeword. If you set the sharing mode to **No Sharing**, then the PLL or DLL signals are not exposed in the top-level file. When PLL sharing is set to **Slave**, the clocks and reset signals `afi_clk`, `afi_half_clk`, and `afi_reset_n` become inputs and must be connected to the corresponding outputs of a PLL master or a stand-alone PLL core.



If you generate a slave IP core, you must modify the timing scripts to allow the timing analysis to correctly resolve clock names and analyze the IP core. Otherwise the software issues critical warnings and an incorrect timing report.

To modify the timing script, follow these steps:

1. In a text editor, open your system's Tcl timing scripts file, as follows:
 - For systems generated with the MegaWizard Plug-In Manager:
Open the `<IP core name>/<IP core name>_p0_timing.tcl` file.
 - For systems generated with Qsys or SOPC Builder:
Open the `<HDL Path>/<submodules>/<master_corename>_timing.tcl` file.
2. Search for the following 2 lines:
 - `set ::master_corename "_MASTER_CORE_"`
 - `set ::master_instname "_MASTER_INST_"`
3. Replace `_MASTER_CORE_` with the core name and `_MASTER_INST_` with instance name of the UniPHY master to which the slave is connected.



For systems generated with Qsys or SOPC Builder, the core name can be determined from the name of the Tcl timing file, for example: `<HDL Path>/<submodules>/<master_corename>_timing.tcl`, where `<master_corename>` is the name of the core.



The instance name is the full path to the instance and is in the `<IP core name>_all_pins.txt` file that is automatically generated after the `<IP core name>_pin_assignments.tcl` script runs.

4. If the slave component is connected to a user-defined PLL rather than a UniPHY master, you must manually enter all clock names.
 - In the <instance_name>_timing.tcl file, remove the master_corename and master_instname variables with the checks performed in the eight lines following them.
 - In the <instance_name>.sdc file, manually define all local pll clock name variables. For example:

```
set local_pll_afi_clk "mycomponent|mypll|my_afi_clk"
```

 where "mycomponent" is the path to where the PLL is instantiated from the top level, "mypll" is the PLL name, and "my_afi_clk" is the name of the wire connected to the PLL. If you are unsure of the exact clock name and path, use the Node Finder in the TimeQuest timing analyzer to find it.



You must be extremely 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 failures.



The DLL and PLL Sharing interface is not available with SOPC Builder.

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 6-7 and Figure 6-8, respectively, show the PHY architecture with and without Master for OCT Control Block

Figure 6-7. PHY Architecture with Master for OCT Control Block

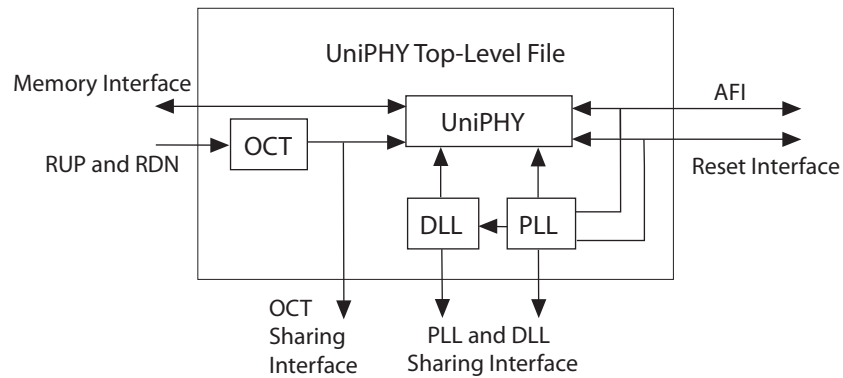
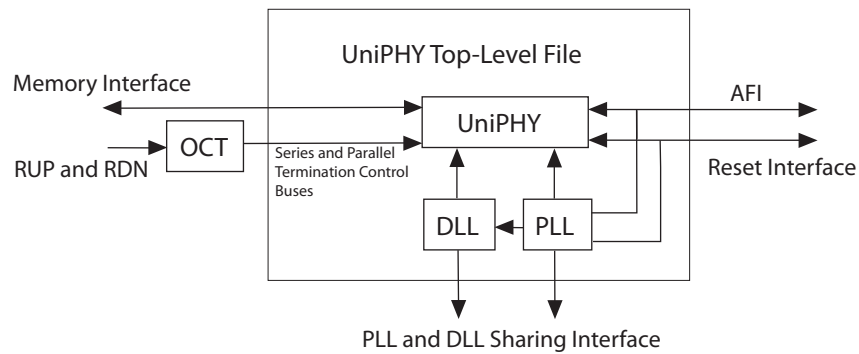



Figure 6-8. PHY Architecture without Master for OCT Control Block



 The OCT Sharing Interface and OCT slave mode are not available with SOPC Builder.

UniPHY Signals

This section describes the UniPHY signals.

Table 6-3 shows the clock and reset signals.


Table 6-3. 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.
soft_reset_n	Input	1	Holding <code>soft_reset_n</code> low holds the PHY in a reset state. However it does not reset the PLL, which keeps running. It also holds the <code>afi_reset_n</code> output low. Mainly for use by SOPC Builder.
reset_request_n	Output	1	When the PLL is locked, <code>reset_request_n</code> is high. When the PLL is out of lock, <code>reset_request_n</code> is low.

Table 6–4 shows the DDR2 and DDR3 SDRAM interface signals.

Table 6–4. DDR2 and DDR3 SDRAM Interface Signals

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.
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 (RDIMM only)	Output	MEM_CONTROL_WIDTH	Address/command parity bit.
mem_err_out_n (RDIMM only)	Input	MEM_CONTROL_WIDTH	Address/command parity error.

 For information about the AFI signals, refer to “AFI 3.0 Specification” on page 5–21.

 For information about top-level HardCopy migration signals, refer to “HardCopy Migration Design Guidelines” in section 6 of this volume.

Table 6–5 shows parameters relating to UniPHY signals.

Table 6–5. Parameters (Part 1 of 3)

Parameter Name	Description
AFI_RATIO	AFI_RATIO is 1 in full-rate designs. AFI_RATIO is 2 for half-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	—

Table 6–5. Parameters (Part 2 of 3)

Parameter Name	Description
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.
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	—

Table 6–5. Parameters (Part 3 of 3)

Parameter Name	Description
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 PHY and the port name prefixes each module uses. 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 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, drive both input bits (of the address and command signal) identically in half-rate designs.

Figure 6–9 shows the half-rate write operation.

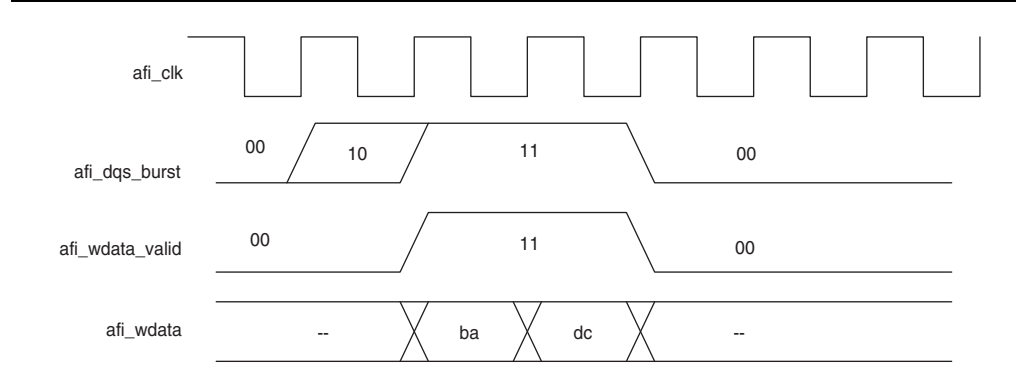
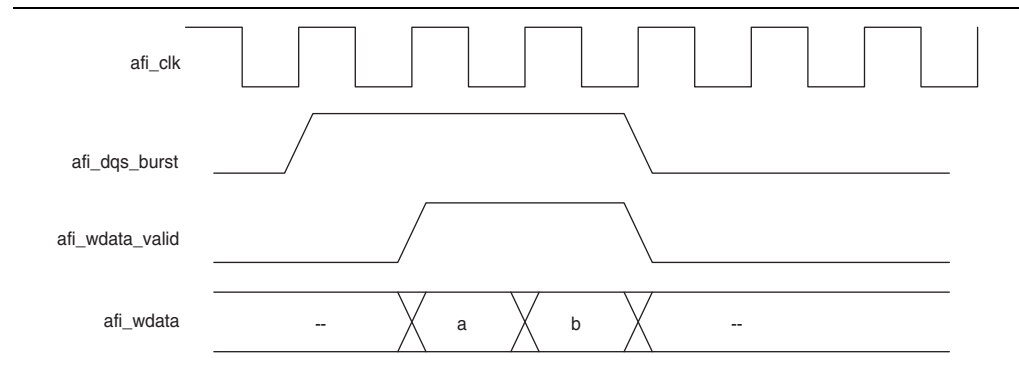
Figure 6–9. Half-Rate Write with Word-Aligned Data

Figure 6-10 shows a full-rate write.

Figure 6-10. Full-Rate Write



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

Figure 6-11 and Figure 6-12 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:


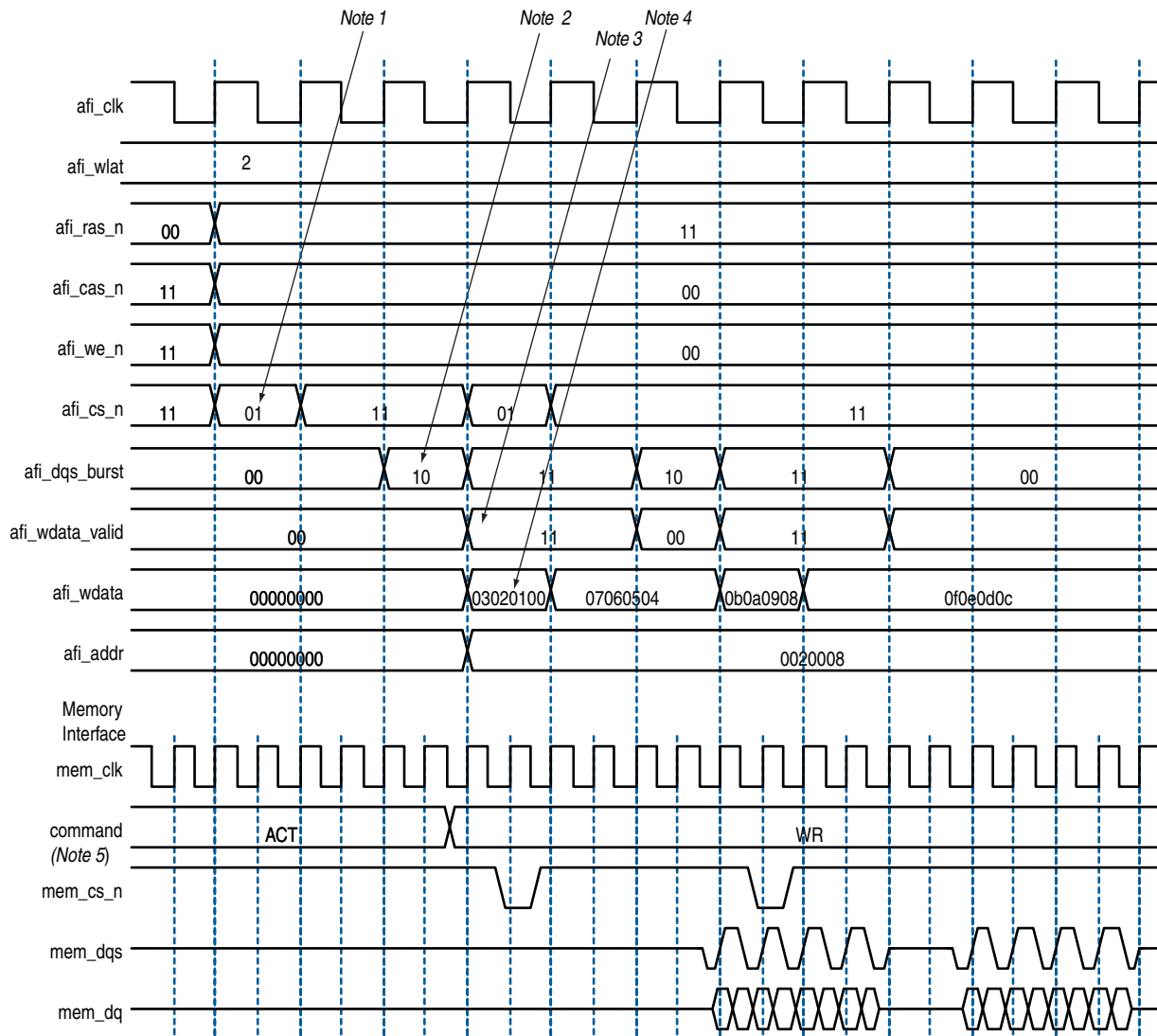
- With the AFI, high and low signals are combined in one signal, so for a single chip select (`afi_cs_n`) interface, `afi_cs_n[1:0]`, location 0 appears on the memory bus on one `mem_clk` cycle and location 1 on the next `mem_clk` cycle.
-  This convention is maintained for all signals so for an 8 bit 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 6-11 through Figure 6-12 assume the following general points:

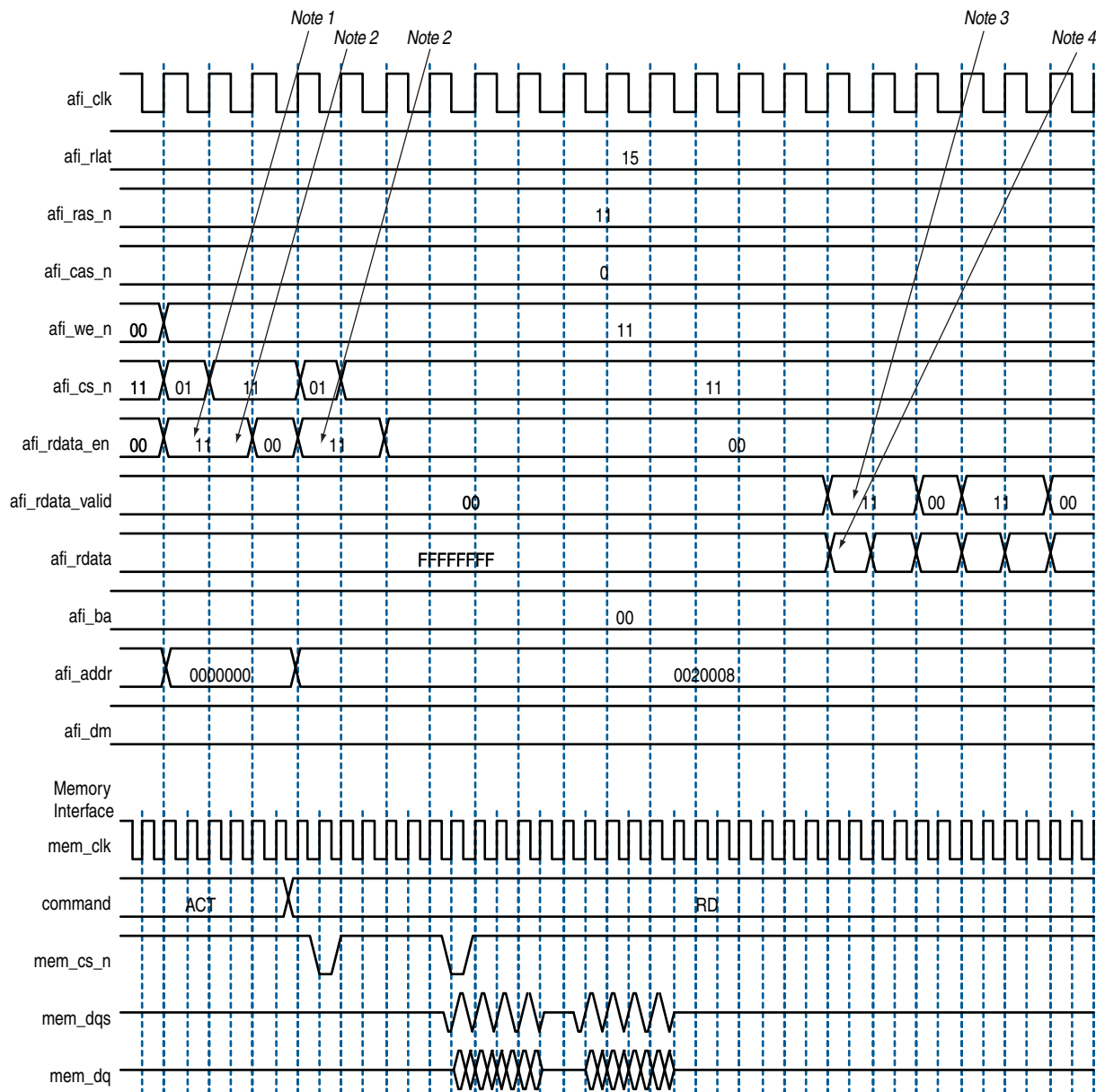
- The burst length is four.
- An 8-bit interface with one chip select.

- The data for one controller clock (`afi_clk`) cycle represents data for two memory clock (`mem_clk`) cycles (half-rate interface).

Figure 6–11. Word-Aligned Writes**Notes to Figure 6–11:**

- (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_wlat` 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_clks` 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 6-12. Word-Aligned Reads



Notes to Figure 6-12:

- (1) For AFI, **afi_rdata_en** is required to be asserted one memory clock cycle before chip select (**afi_cs_n**) is asserted. In the half-rate **afi_clk** domain, this requirement manifests as the controller driving 11 (as opposed to the 01) on **afi_rdata_en**.
- (2) AFI requires that **afi_rdata_en** is driven for the duration of the read. In this example, it is driven to 11 for two half-rate **afi_clks**, which equates to driving to 1, for the four memory clock cycles of this four-beat burst.
- (3) The **afi_rdata_valid** returns 15 (**afi_rlat**) controller clock (**afi_clk**) cycles after **afi_rdata_en** is asserted. Returned is when the **afi_rdata_valid** signal is observed at the output of a register within the controller. A controller can use the **afi_rlat** value to determine when to register returned data, but this is unnecessary as the **afi_rdata_valid** is provided for the controller to use as an enable when registering read data.
- (4) Observe the alignment of returned read data with respect to data on the bus.


Using a Custom Controller

The UniPHY-based memory interface IP cores are delivered with both the PHY and the memory controller integrated. To replace the Altera high-performance memory controller with a custom memory controller, perform the following steps:


1. Parameterize and generate your variation of the UniPHY-based memory controller IP.


This step creates a top-level HDL file called `<variation_name>.v` (or `<variation_name>.vhd`), and a sub-directory called `<variation_name>`.

The top-level module instantiates the `<variation_name>_<stamp>_controller_phy` module in the `<variation_name>` subdirectory. The `<variation_name>_<stamp>_controller_phy` module instantiates the PHY and the controller.

 `<stamp>` is a unique identifier determined by the MegaWizard Plug-in Manager, SOPC Builder, or Qsys, during generation.

2. Open the `<variation_name>/<variation_name>_<stamp>_controller_phy.sv` file.
3. Replace the `<variation_name>_<stamp>_alt_ddrx_controller` module with your custom controller module.
4. Delete the ports of the Altera high-performance memory controller, and add the top-level ports of your custom controller.
5. Similarly, update the port names in the top-level module in the `<variation_name>.v` or `<variation_name>.vhd` file.
6. Compile and simulate the design to confirm correct operation.

 Regenerating the UniPHY memory interface IP erases all modifications made to the HDL files. The parameters you select in the MegaWizard are stored in the top-level `<variation_name>` module; hence, you must repeat the above steps every time you regenerate the IP variation.

 For half-rate controllers, AFI signals are double the bus width of the memory interface. Half rate controllers have double the width of the signal and run at half the speed. Hence, the overall bandwidth is maintained. Such double-width signals are divided into two signals for transmission to the memory interface, with a higher order bits representing the most-significant bit (MSB) and a lower order bits representing the least-significant bit (LSB). The LSB is transmitted first, and is followed by the MSB.

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 the *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

IP generation creates an example top-level project that shows you how to instantiate and connect the controller.

The example top-level project contains a testbench, which is for use with Verilog HDL only language simulators such as ModelSim-AE Verilog, and shows simple operation of the memory interface.



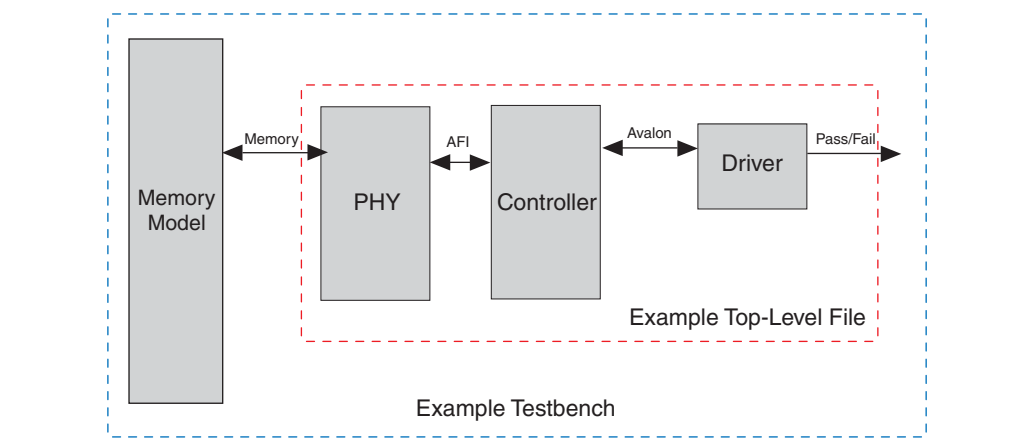
For a VHDL-only simulation, use the VHDL IP functional simulation model.

The testbench contains the following blocks:

- A synthesizable Avalon-MM example driver, which acts as a traffic generator block and implements a pseudo-random pattern of reads and writes to a parameterized number of addresses. The driver also monitors the data read from the memory to ensure it matches the written data and asserts a failure otherwise.
- An instance of the controller, which interfaces between the Avalon-MM interface and the AFI.
- The UniPHY IP, which serves as an interface between the memory controller and external memory device(s) to perform read and write operations to the memory.
- A memory model, which acts as a generic model that adheres to the memory protocol specifications. Memory vendors also provide simulation models for specific memory components that can be downloaded from their websites. This block is available in Verilog HDL only.

Figure 7–1 shows the testbench and the example top-level file.

Figure 7–1. Testbench and Example Top-Level File

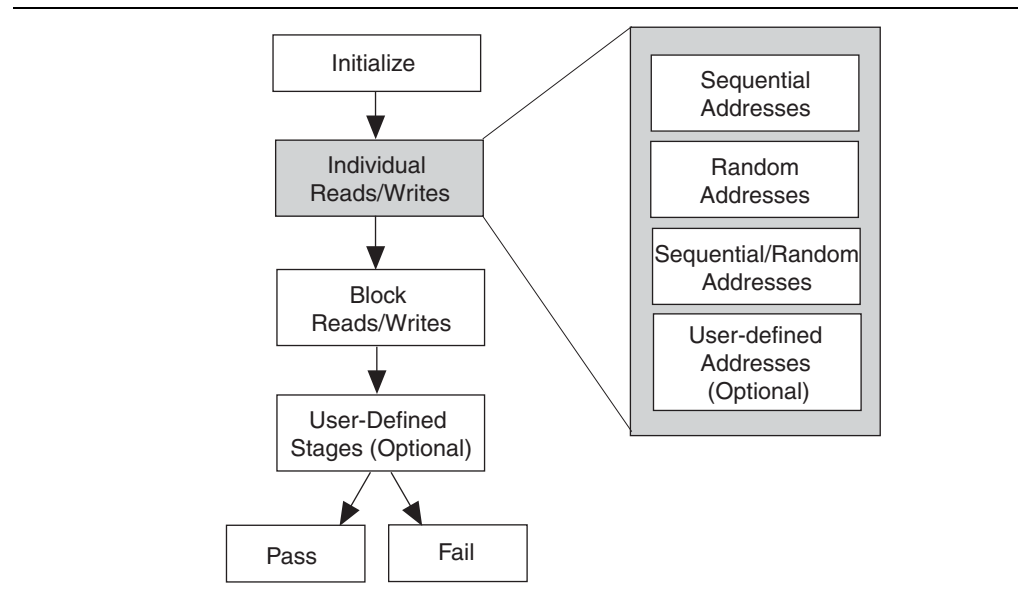


Example Driver

The example driver for Avalon-MM memory interfaces generates Avalon-MM traffic on an Avalon-MM master interface. As the read and write traffic is generated, the expected read response is stored internally and compared 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 is asserted.

Each operation generated by the driver 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 driver 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 driver enters a fail state. If all patterns have been generated and compared successfully, the driver enters a pass state.

Figure 7-2. Example Driver Operations



Within the driver, 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 perform individual or block read and write generation.

Individual Read and Write Generation

During the individual read and write generation stage of the driver, the traffic generation block generates individual write followed by individual read Avalon-MM transactions, where the address for the transactions are 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 block read and write generation state of the driver, 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.

Example Driver Signals

Table 7-1 lists the signals used by the example driver.

Table 7-1. Driver 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*.

Example Driver 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 example driver.

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.

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.



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.

Table 8–1 shows the DDR2 SDRAM latency in full rate memory clock cycles.

Table 8–1. DDR2 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) (Note 3)

Rate	Controller Address and Command (4)	PHY Address and Command	Memory Maximum Read	PHY Read Return	Round Trip	Round Trip without Memory
Full	5	1	3–7	4	13–17	10
Half	10	3 (1) 4 (2)	3–7	8	24–28 (1) 26–28 (2)	21 (1) 22 (2)
Notes to Table 8–1: (1) Even write latency. (2) Odd write latency. (3) Latency is the number of cycles between the first register of the current stage capturing cmd/data, and the first register in the next stage capturing cmd/data. (4) Latency shown is best case, for maximum performance specifications. Latency may be higher due to protocol requirements; controller latency may be lower for slower frequencies.						

Table 8-2 shows the DDR3 SDRAM latency in full rate memory clock cycles.

Table 8-2. DDR3 SDRAM Controller Latency (In Full-Rate Memory Clock Cycles) (Note 5)

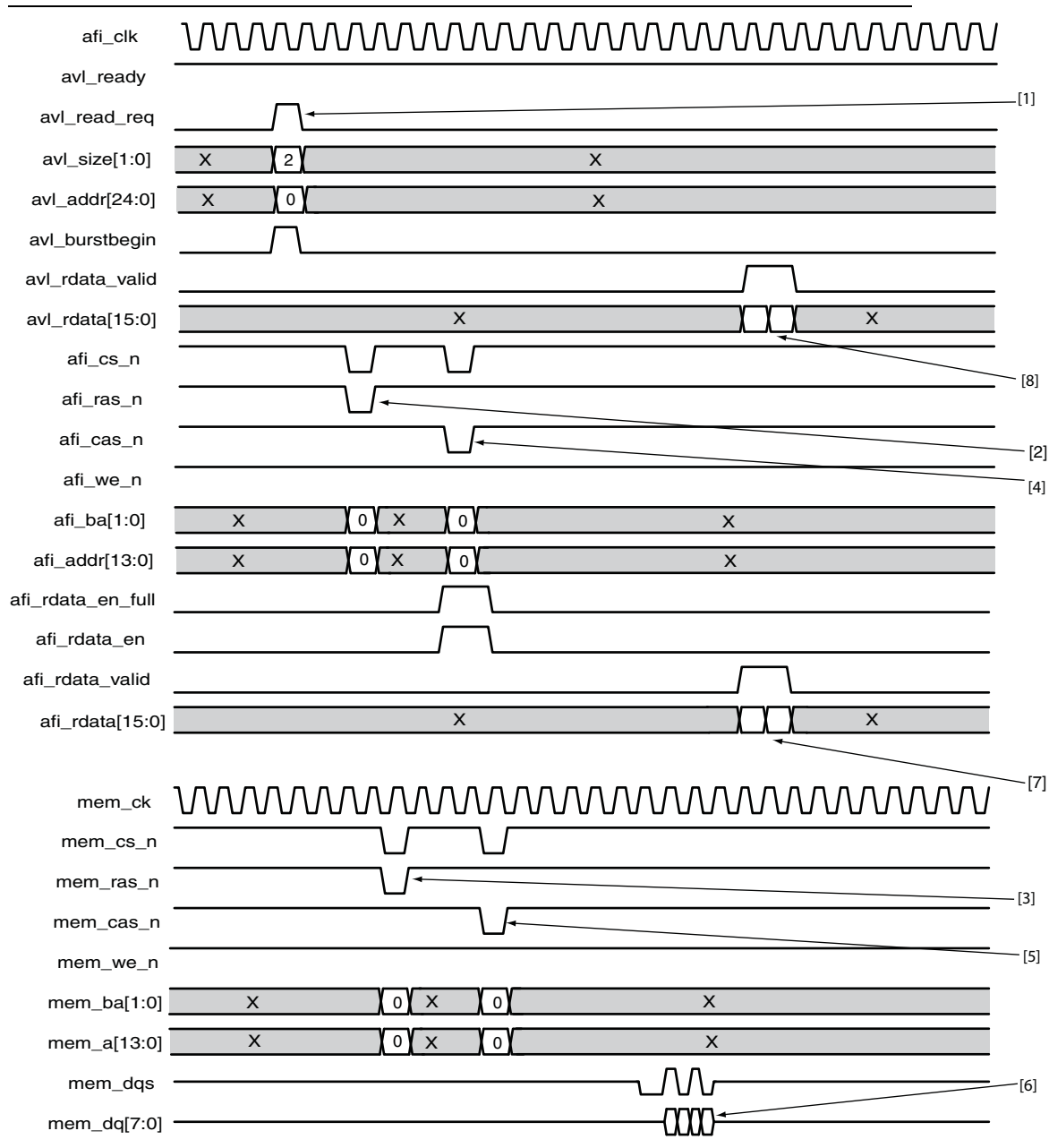
Rate	Controller Address and Command (6)	PHY Address and Command	Memory Maximum Read	PHY Read Return	Round Trip	Round Trip without Memory
Half	10	3 (1) (2) 4 (3) (4)	5-11	7 (1) 8 (2) 8 (3) 7 (4)	26-30 (1) 26-32 (2) 28-32 (3) 26-32 (4)	20 (1) 21 (2) 22 (3) 21 (4)

Notes to Table 8-2:

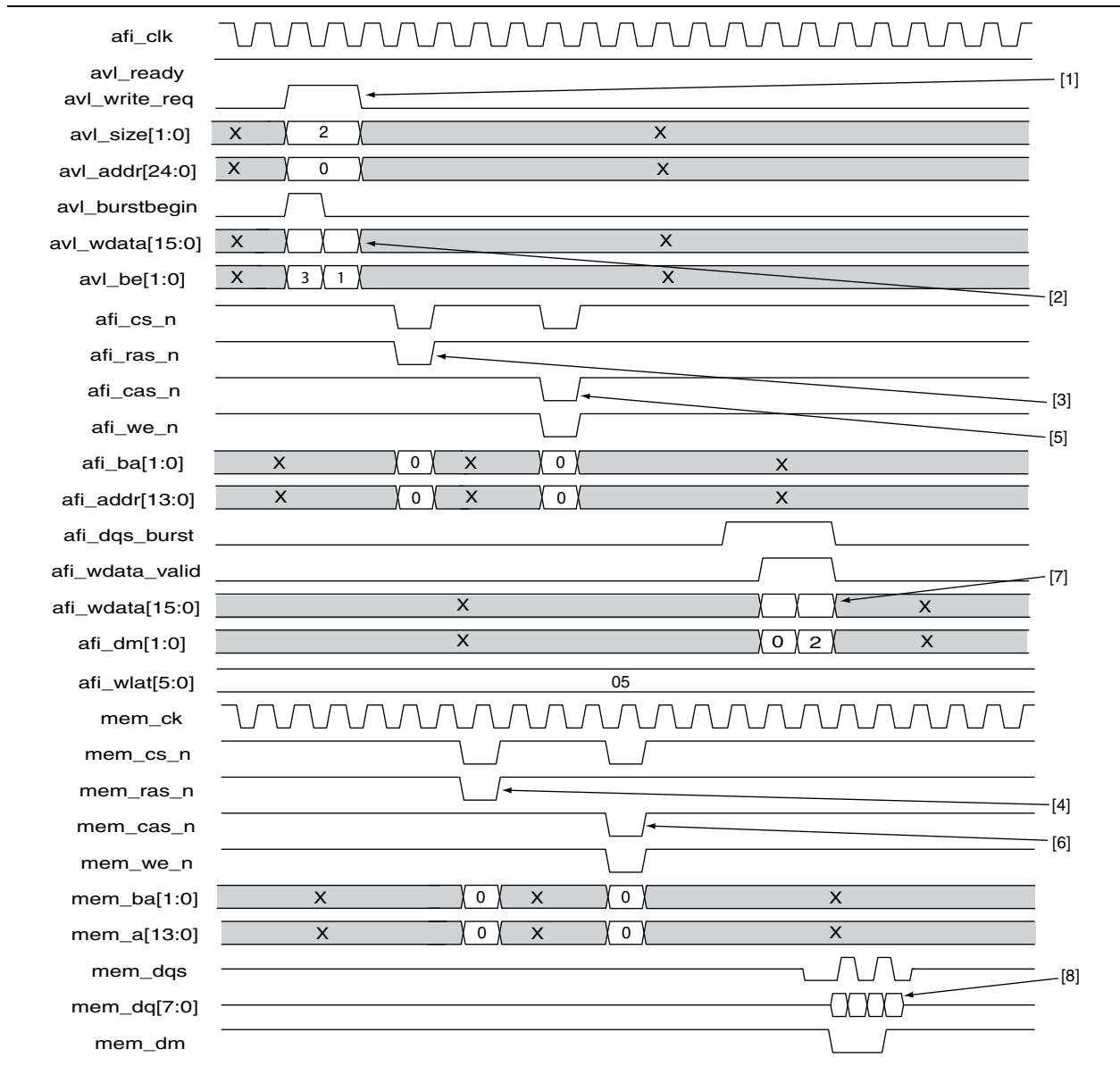
- (1) Even write latency and even read latency.
- (2) Even write latency and odd read latency.
- (3) Odd write latency and even read latency.
- (4) Odd write latency and odd read latency.
- (5) Latency is the number of cycles between the first register of the current stage capturing cmd/data, and the first register in the next stage capturing cmd/data.
- (6) Latency shown is best case, for maximum performance specifications. Latency may be higher due to protocol requirements; controller latency may be lower for slower frequencies.

Figure 9–1 through Figure 9–6 present the following timing diagrams, based on a Stratix III device:

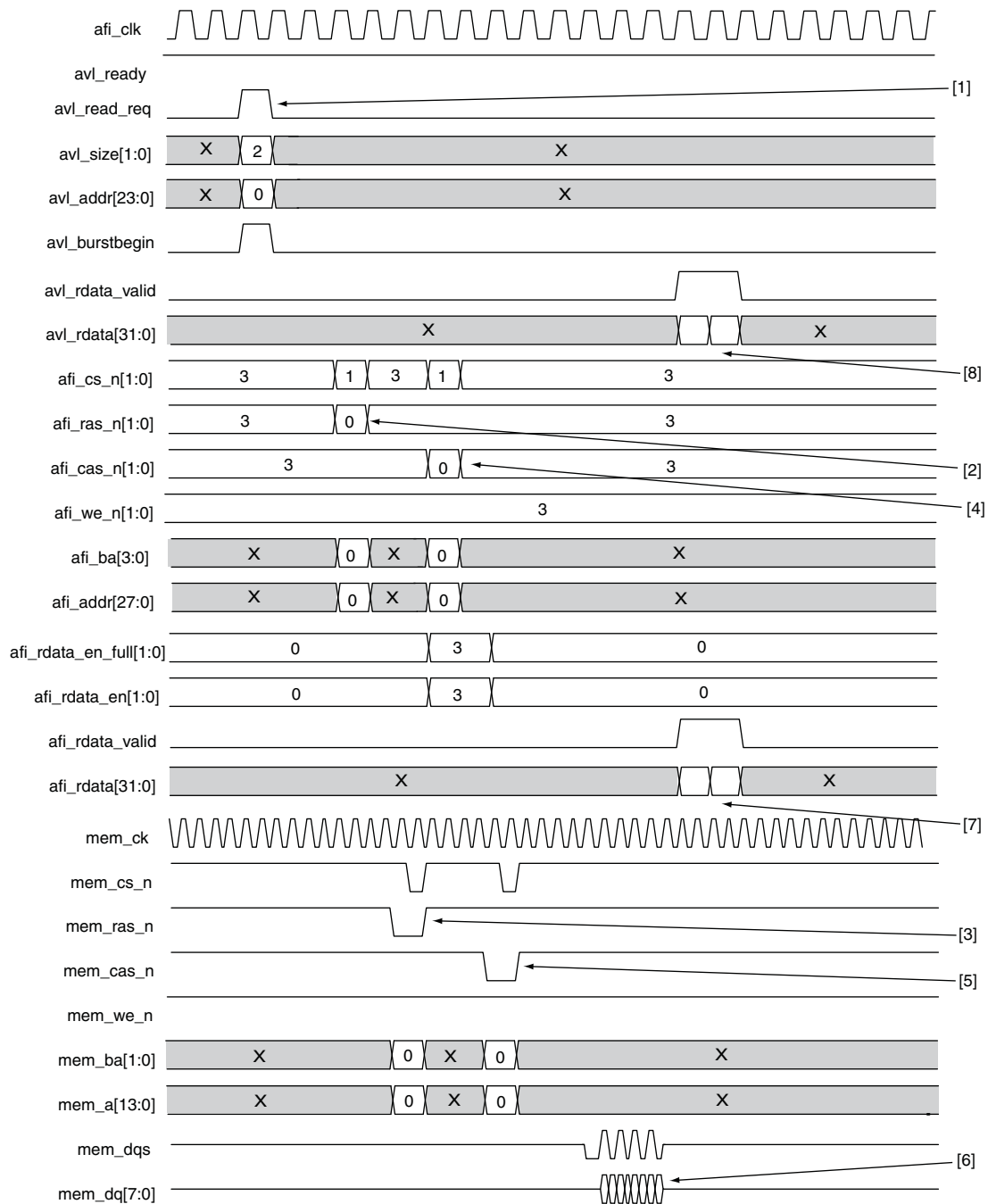
- Full-Rate DDR2 SDRAM Read
- Full-Rate DDR2 SDRAM Write
- Half-Rate DDR2 SDRAM Read
- Half-Rate DDR2 SDRAM Write
- Half-Rate DDR3 SDRAM Read
- Half-Rate DDR3 SDRAM Writes

Figure 9–1. Full-Rate DDR2 SDRAM Read

- (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.

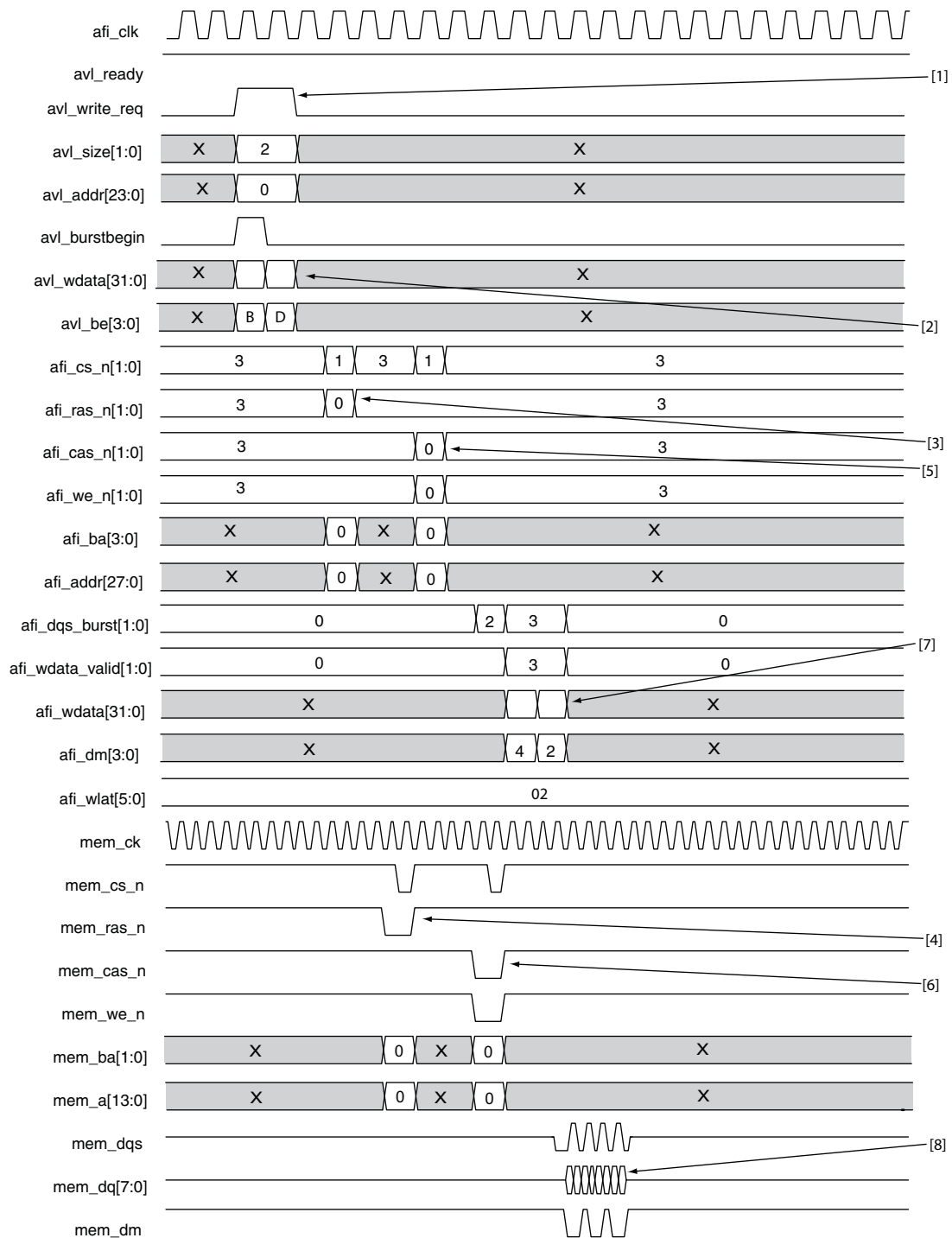
Figure 9–2. Full-Rate DDR2 SDRAM Write

- (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 9–3. Half-Rate DDR2 SDRAM Read

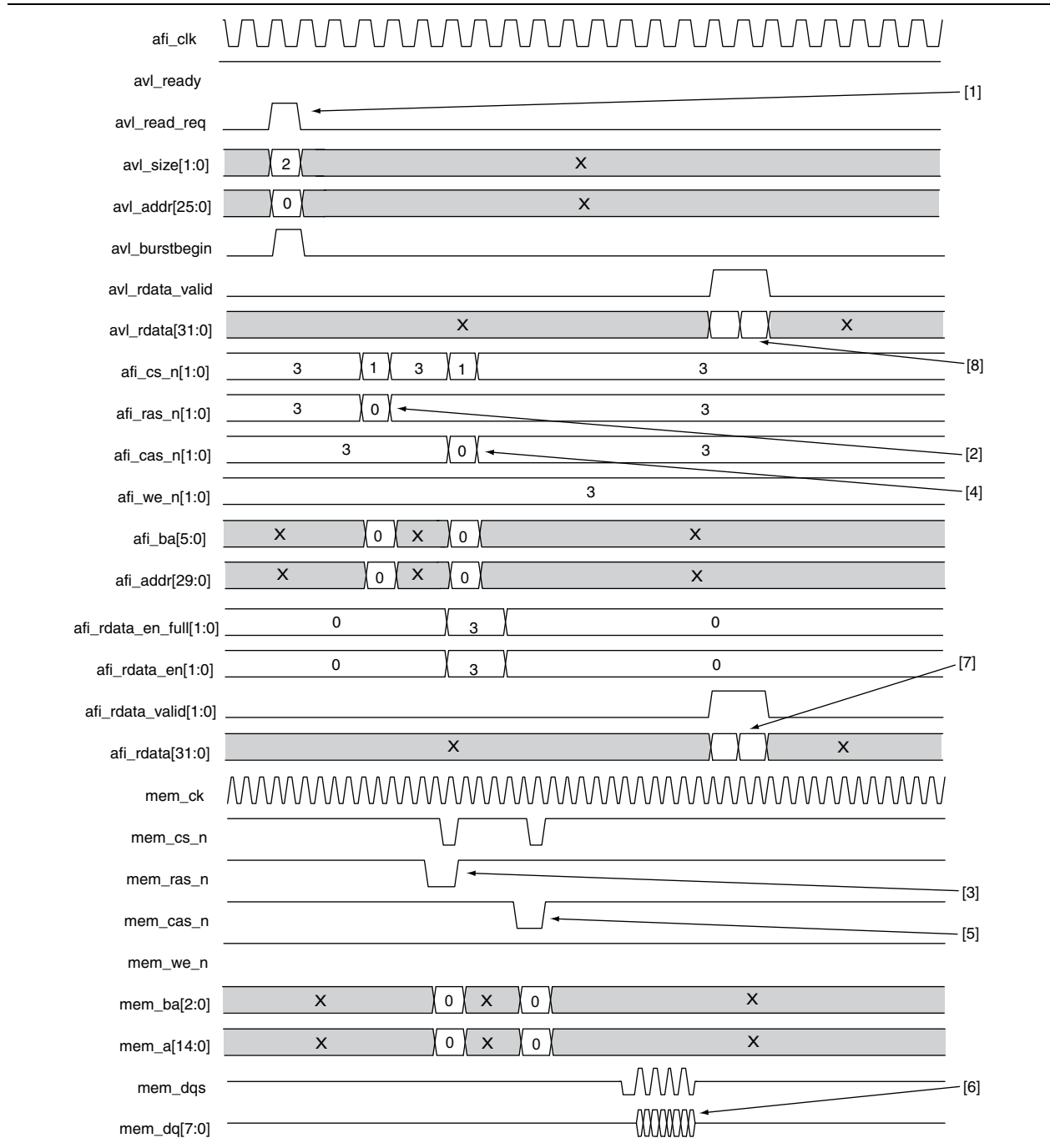
- (1) Controller receives read command.
- (2) Controller issues activate command to PHY.
- (3) PHY issues activate command to memory.
- (4) Controller issues read command to PHY.
- (5) PHY issues read command to memory.
- (6) PHY receives read data from memory.

- (7) Controller receives read data from PHY.
 (8) User logic receives read data from controller.

Figure 9-4. Half-Rate DDR2 SDRAM Write

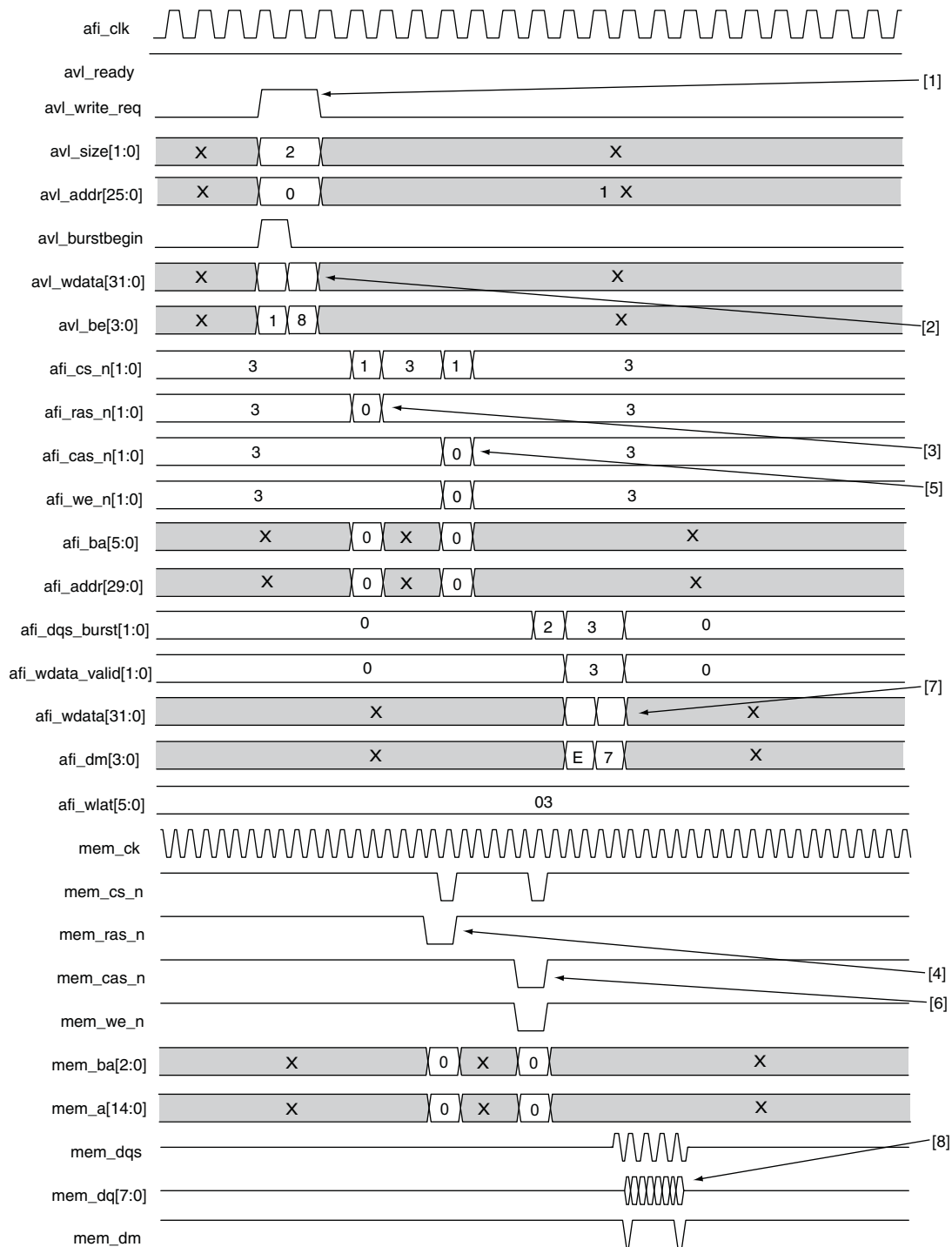
- (1) Controller receives write command.
 (2) Controller receives write data.

- (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 9-5. Half-Rate DDR3 SDRAM Read

- (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 9-6. Half-Rate DDR3 SDRAM Writes

- (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.

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 [“MegaWizard Plug-In Manager Flow” on page 2–3](#) 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 10-1 shows the changes you must make.

Table 10-1. Changes to ALTMEMPHY Port Names

ALTMEMPHY Port	Changes
aux_full_rate_clk	Rename to pll_mem_clk.
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	Rename to pll_mem_clk.
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 ("Creating OCT Signals" on page 10-4).
oct_ctl_rt_value	Remove from design ("Creating OCT Signals" on page 10-4).
phy_clk	Rename to afi_clk.
reset_phy_clk_n	Rename to afi_reset_n.
local_refresh_ack local_wdata_req	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><project directory>/uniphy/rtl/<variation name>_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” on [page 6–14](#).

Running Pin Assignments Script

Remap your design by running analysis and synthesis. When analysis and synthesis completes, run the pin assignments Tcl script (“[Add Pin and DQ Group Assignments](#)” on [page 4–1](#)), 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`, `afi_cal_success`, and `afi_cal_fail` signals are at the top-level entity so that an instantiating testbench can refer to those signals.
2. 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:
 - `\uniphy\<variation name>\rtl_sim\<variation name>_example_top_tb.v`
 - `\uniphy\<variation name>\example_project\mem_model.sv`
3. Open up the UniPHY testbench file and find the following line in the testbench:


```
force dut.mem_if.controller_phy_inst... = SEQ_CALIB_INIT;
```
4. Change the PHY instance name from the default `mem_if` to the instance name in which you instantiated the UniPHY datapath.

5. Update the following port names of the example design in the UniPHY-generated testbench (Table 10-2).

Table 10-2. Example Design Port Names

Example Design Name	New Name
pll_ref_clk	Rename to clock_source.
mem_a	Rename to mem_addr.
mem_ck	Rename to mem_clk.
mem_ck_n	Rename to mem_clk_n.
mem_dqs_n	Rename to mem_dqsn.
pass	Rename to pnf.

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
June 2011	2.0	<ul style="list-style-type: none"> ■ Updated Generated Files lists. ■ Added Qsys and SOPC Builder Interfaces section. ■ Added new AFI 3.0 Specification. ■ Added Efficiency Monitor and Protocol Checker section. ■ Added new Sequencer section. ■ Revised Address and Command Datapath description and figure. ■ Added DLL Offset Control Block section. ■ Updated Timing Diagrams section. ■ Moved HardCopy Migration Design Guidelines to section 6.
December 2010	1.1	<ul style="list-style-type: none"> ■ Updated Device Family Support, Features list, and Resource Utilization tables. ■ Updated Design Flows, HardCopy Migration Design Guidelines, and Generated Files information. ■ Updated Parameter Settings chapter. ■ Updated ODT Generation Logic, Reset and Clock Generation, and Write Datapath. ■ Added The OCT Sharing Interface, and expanded Using a Custom Controller information. ■ Updated Latency data. ■ Updated Creating OCT Signals in Upgrading to UniPHY-based Controllers from ALTMEMPHY-based Controllers.
July 2010	1.0	First published.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General)	Email	nacomp@altera.com







Contact (1) (Software Licensing)	Contact Method	Address
	Email	authorization@altera.com

Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, Save As dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, \qdesigns directory, D: drive, and chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Indicate document titles. For example, <i>Stratix IV Design Guidelines</i> .
<i>italic type</i>	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, <file name> and <project name>.pof file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
“Subheading Title”	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, data1, tdi, and input. The suffix n denotes an active-low signal. For example, resetn. Indicates command line commands and anything that must be typed exactly as it appears. For example, c:\qdesigns\tutorial\chiptrip.gdf. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), and logic function names (for example, TRI).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	A question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.