



External Memory Interface Handbook Volume 3

Section I. DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP User Guide



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

EMI_DDR_UG-3.0

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	1-2
Device Family Support	1-2
Features	1-3
Unsupported Features	1-4
MegaCore Verification	1-4
Resource Utilization	1-5
System Requirements	1-6
Installation and Licensing	1-7
Free Evaluation	1-7
OpenCore Plus Time-Out Behavior	1-8

Chapter 2. Getting Started

Design Flow	2-1
SOPC Builder Flow	2-2
Specifying Parameters	2-2
Completing the SOPC Builder System	2-3
Qsys Flow	2-4
Specifying Parameters	2-4
Completing the Qsys System	2-5
MegaWizard Plug-In Manager Flow	2-6
Specifying Parameters	2-6
HardCopy Guidelines	2-7
Generated Files	2-8

Chapter 3. Parameter Settings

ALTMEMPHY Parameter Settings	3-1
Memory Settings	3-2
Using the Preset Editor to Create a Custom Memory Preset	3-3
Derating Memory Setup and Hold Timing	3-9
PHY Settings	3-10
Board Settings	3-12
DDR or DDR2 SDRAM Controller with ALTMEMPHY Parameter Settings	3-13
Controller Settings	3-14

Chapter 4. Compiling and Simulating

Compiling the Design	4-1
Simulating the Design	4-4

Chapter 5. Functional Description—ALTMEMPHY

Block Description	5-2
Calibration	5-3
Address and Command Datapath	5-3
Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices	5-3
Stratix III and Stratix IV Devices	5-5
Clock and Reset Management	5-5
Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices	5-5
Cyclone III Devices	5-13

Stratix III and Stratix IV Devices	5-16
Read Datapath	5-20
Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices	5-20
Cyclone III Devices	5-23
Stratix III and Stratix IV Devices	5-24
Write Datapath	5-25
Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices	5-25
Stratix III and Stratix IV Devices	5-26
ALTMEMPHY Signals	5-27
PHY-to-Controller Interfaces	5-34
Using a Custom Controller	5-43
Preliminary Steps	5-43
Design Considerations	5-43
Clocks and Resets	5-43
Calibration Process Requirements	5-44
Other Local Interface Requirements	5-44
Address and Command Interfacing	5-44
Handshake Mechanism Between Read Commands and Read Data	5-44
Handshake Mechanism Between Write Commands and Write Data	5-45
Partial Write Operations	5-46
Using a Custom Controller with the TimeQuest Timing Analyzer	5-46

Chapter 6. Functional Description—High-Performance Controller II

Memory Controller Architecture	6-1
Avalon-ST Input Interface	6-2
Command Generator	6-2
Timing Bank Pool	6-2
Arbiter	6-3
Arbitration Rules	6-3
Rank Timer	6-3
Read Data Buffer	6-3
Write Data Buffer	6-3
ECC Block	6-3
AFI Interface	6-3
CSR Interface	6-4
Controller Features Descriptions	6-4
Data Reordering	6-4
Pre-emptive Bank Management	6-4
Quasi-1T and Quasi-2T	6-4
User Autoprecharge Commands	6-4
Address and Command Decoding Logic	6-5
Low-Power Logic	6-5
User-Controlled Self-Refresh	6-5
Automatic Power-Down with Programmable Time-Out	6-5
ODT Generation Logic	6-5
DDR2 SDRAM	6-6
ECC	6-6
Partial Writes	6-7
Partial Bursts	6-8
External Interfaces	6-9
Clock and Reset Interface	6-9
Avalon-ST Data Slave Interface	6-9
Controller-PHY Interface	6-9
Memory Side-Band Signals	6-10

Self-Refresh (Low Power) Interface	6-10
User-Controller Refresh Interface	6-10
Configuration and Status Register (CSR) Interface	6-10
Top-Level Signals Description	6-11
Sequence of Operations	6-17
Write Command	6-17
Read Command	6-18
Read-Modify-Write Command	6-18
Example Top-Level File	6-19
Example Driver	6-20
Register Maps	6-22
ALTMEMPHY Register Map	6-22
Controller Register Map	6-24

Chapter 7. Latency

Chapter 8. Timing Diagrams

DDR and DDR2 High-Performance Controllers II	8-1
Half-Rate Read	8-2
Half-Rate Write	8-4
Full-Rate Read	8-6
Full-Rate Write	8-8

Additional Information

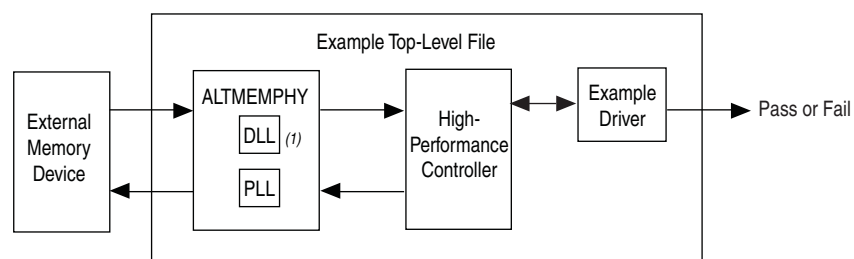
Document Revision History	Info-1
How to Contact Altera	Info-1
Typographic Conventions	Info-2

The Altera® DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP provide simplified interfaces to industry-standard DDR SDRAM and DDR2 SDRAM. The ALTMEMPHY megafunction is an interface between a memory controller and the memory devices, and performs read and write operations to the memory. The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP work in conjunction with the Altera ALTMEMPHY megafunction.

The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP and ALTMEMPHY megafunction offer full-rate or half-rate DDR and DDR2 SDRAM interfaces. The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP offer the high-performance controller II (HPC II), which provides high efficiency and advanced features.

Figure 1–1 shows a system-level diagram including the example top-level file that the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP creates for you.

Figure 1–1. System-Level Diagram



Note to Figure 1–1:

(1) When you choose **Instantiate DLL Externally**, delay-locked loop (DLL) is instantiated outside the ALTMEMPHY megafunction.

The MegaWizard™ Plug-In Manager generates an example top-level file, consisting of an example driver and your DDR or DDR2 SDRAM high-performance controller custom variation. The controller instantiates an instance of the ALTMEMPHY megafunction which in turn instantiates a phase-locked loop (PLL) and DLL. You can also instantiate the DLL outside the ALTMEMPHY megafunction to share the DLL between multiple instances of the ALTMEMPHY megafunction. You cannot share a PLL between multiple instances of the ALTMEMPHY megafunction, but you may share some of the PLL clock outputs between these multiple instances.

The example top-level file is a fully-functional design that you can simulate, synthesize, and use in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass or fail, and test complete signals.

The ALTMEMPHY megafunction creates the datapath between the memory device and the memory controller. The megafunction is available as a stand-alone product or can be used in conjunction with Altera high-performance memory controllers. When using the ALTMEMPHY megafunction as a stand-alone product, use with either custom or third-party controllers.

Release Information

Table 1–1 provides information about this release of the DDR and DDR2 SDRAM high-performance controller and ALTMEMPHY IP.

Table 1–1. Release Information

Item	Description
Version	11.0
Release Date	May 2011
Ordering Codes	IP-SDRAM/HPDDR (DDR SDRAM HPC) IP-SDRAM/HPDDR2 (DDR2 SDRAM HPC) IP-HPMCII (HPC II)
Product IDs	00BE (DDR SDRAM) 00BF (DDR2 SDRAM) 00C0 (ALTMEMPHY Megafunction)
Vendor ID	6AF7

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release. For information about issues on the DDR and DDR2 SDRAM high-performance controllers and the ALTMEMPHY megafunction in a particular Quartus II version, refer to the *Quartus II Software Release Notes*.

Device Family Support

The MegaCore functions provide either final or preliminary support for target Altera device families:

- **Final support** means the core is verified with final timing models for this device family. The core meets all functional and timing requirements for the device family and can be used in production designs.
- **Preliminary support** means the core is verified with preliminary timing models for this device family. The core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
- **HardCopy Compilation** means the core is verified with final timing models for the HardCopy® device family. The core meets all functional and timing requirements for the device family and can be used in production designs.
- **HardCopy Companion** means the core is verified with preliminary timing models for the HardCopy companion device. The core meets all functional requirements, but might still be undergoing timing analysis for HardCopy device family. It can be used in production designs with caution.

Table 1–2 shows the level of support offered by the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP to each of the Altera device families.

Table 1–2. Device Family Support

Device Family	Support
Arria® GX	Final
Arria II GX	Final
Cyclone® III	Final
Cyclone III LS	Final
Cyclone IV E	Final
Cyclone IV GX	Final
HardCopy II	HardCopy Companion
Stratix® II	Final
Stratix II GX	Final
Other device families	No support

Features

The ALTMEMPHY megafunction offers the following features:

- Simple setup.
- Support for the Altera PHY Interface (AFI) for DDR and DDR2 SDRAM on all supported devices.
- Automated initial calibration eliminating complicated read data timing calculations.
- Voltage and temperature (VT) tracking that guarantees maximum stable performance for DDR and DDR2 SDRAM interfaces.
- Self-contained datapath that makes connection to an Altera controller or a third-party controller independent of the critical timing paths.
- Full-rate and half-rate DDR and DDR2 SDRAM interfaces.
- Easy-to-use parameter editor.

In addition, Table 1–3 shows the features provided by the DDR and DDR2 SDRAM HPC II.

Table 1–3. DDR and DDR2 SDRAM HPC and HPC II Features (Part 1 of 2)

Features	HPC II
Half-rate controller	✓
Support for AFI ALTMEMPHY	✓
Support for Avalon®Memory Mapped (MM) local interface	✓
Configurable command look-ahead bank management with in-order reads and writes	✓
Additive latency	✓ (1)
Support for arbitrary Avalon burst length	✓

Table 1–3. DDR and DDR2 SDRAM HPC and HPC II Features (Part 2 of 2)

Features	HPC II
Memory burst length of 4	✓
Memory burst length of 8	✓
Built-in flexible memory burst adapter	✓
Configurable Local-to-Memory address mappings	✓
Optional run-time configuration of size and mode register settings, and memory timing	✓
Partial array self-refresh (PASR)	✓
Support for industry-standard DDR and DDR2 SDRAM devices; and DIMMs	✓
Optional support for self-refresh command	✓
Optional support for user-controlled power-down command	—
Optional support for automatic power-down command with programmable time-out	✓
Optional support for auto-precharge read and auto-precharge write commands	—
Optional support for user-controller refresh	✓
Optional multiple controller clock sharing in Qsys Flow	✓
Integrated error correction coding (ECC) function 72-bit	✓
Integrated ECC function, 16, 24, and 40-bit	✓
Support for partial-word write with optional automatic error correction	✓
Support for OpenCore Plus evaluation	—
IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulator	✓

Note to Table 1–3:

(1) HPC II supports additive latency values greater or equal to $t_{RCD}-1$, in clock cycle unit (t_{CK}).

Unsupported Features

The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP do not support the following features:

- Timing simulation.
- Burst length of 2.
- Partial burst and unaligned burst in ECC and non-ECC mode when DM pins are disabled.

MegaCore Verification

Altera performs extensive random, directed tests with functional test coverage using industry-standard Denali models to ensure the functionality of the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP.

Resource Utilization

Table 1–4 through Table 1–6 show typical resource utilization data for the DDR2 high-performance controller and controller plus PHY, for half-rate and full-rate configurations.

Table 1–4. DDR2 Resource Utilization in Arria II GX Devices

Protocol	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
Controller							
DDR2 (Half rate)	8	1,971	1,547	10	2	0	4,352
	16	1,973	1,547	10	4	0	8,704
	64	2,028	1,563	18	15	0	34,560
	72	2,044	1,547	10	17	0	39,168
DDR2 (Full rate)	8	2,007	1,565	10	2	0	2,176
	16	2,013	1,565	10	2	0	4,352
	64	2,022	1,565	10	8	0	17,408
	72	2,025	1,565	10	9	0	19,584
Controller+PHY							
DDR2 (Half rate)	8	3,481	2,722	12	4	0	4,672
	16	3,545	2,862	12	7	0	9,280
	64	3,891	3,704	20	24	0	36,672
	72	3,984	3,827	12	26	0	41,536
DDR2 (Full rate)	8	3,337	2,568	29	2	0	2,176
	16	3,356	2,558	11	4	0	4,928
	64	3,423	2,836	31	12	0	19,200
	72	3,445	2,827	11	14	0	21,952

Table 1–5. DDR2 Resource Utilization in Cyclone III Devices (Part 1 of 2)

Protocol	Memory Width (Bits)	Logic Registers	Logic Cells	M9K Blocks	Memory (Bits)
Controller					
DDR2 (Half rate)	8	1,513	3,015	4	4,464
	16	1,513	3,034	6	8,816
	64	1,513	3,082	18	34,928
	72	1,513	3,076	19	39,280
DDR2 (Full rate)	8	1,531	3,059	4	2,288
	16	1,531	3,108	4	4,464
	64	1,531	3,134	10	17,520
	72	1,531	3,119	11	19,696

Table 1–5. DDR2 Resource Utilization in Cyclone III Devices (Part 2 of 2)


Protocol	Memory Width (Bits)	Logic Registers	Logic Cells	M9K Blocks	Memory (Bits)
Controller+PHY					
DDR2 (Half rate)	8	2,737	5,131	6	4,784
	16	2,915	5,351	9	9,392
	64	3,969	6,564	27	37,040
	72	4,143	6,786	28	41,648
DDR2 (Full rate)	8	2,418	4,763	6	2,576
	16	2,499	4,919	6	5,008
	64	2,957	5,505	15	19,600
	72	3,034	5,608	16	22,032

Table 1–6. DDR2 Resource Utilization in Stratix II Devices

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	M4K Blocks	M512 Blocks	Memory (Bits)
Controller						
DDR2 (Half rate)	8	1,998	1,519	2	2	4,464
	16	2,002	1,519	4	2	8,816
	64	2,058	1,519	16	2	34,928
	72	2,064	1,518	17	2	39,280
DDR2 (Full rate)	8	2,021	1,541	2	2	2,288
	16	2,017	1,537	2	2	4,464
	64	2,035	1,538	8	2	17,520
	72	2,033	1,536	9	2	19,696
Controller+PHY						
DDR2 (Half rate)	8	3,575	2,754	3	6	5,196
	16	3,639	2,910	6	6	9,804
	64	3,984	3,831	23	7	37,452
	72	4,071	3,987	25	6	42,060
DDR2 (Full rate)	8	3,387	2,496	2	7	2,988
	16	3,400	2,553	4	5	5,420
	64	3,472	2,880	13	5	20,012
	72	3,491	2,935	14	5	22,444

System Requirements

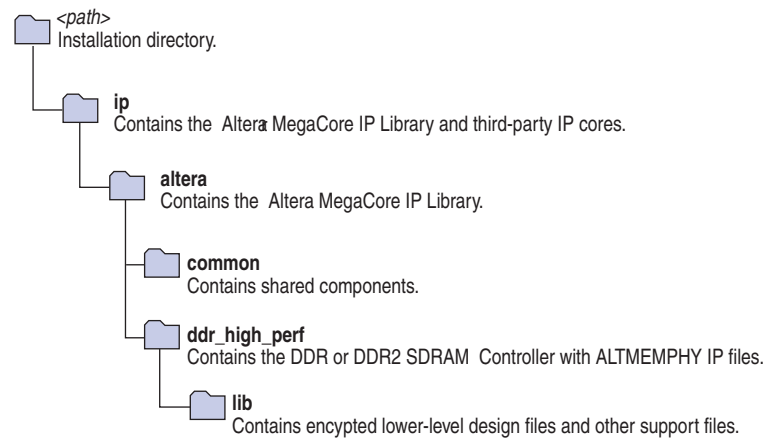
The DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP are part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

 For system requirements and installation instructions, refer to *Altera Software Installation & Licensing*.

Installation and Licensing

Figure 1-2 shows the directory structure after you install the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP, where *<path>* is the installation directory. The default installation directory on Windows is `c:\altera\<version>`; on Linux it is `/opt/altera<version>`.

Figure 1-2. Directory Structure



You need a license for the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP only when you are completely satisfied with its functionality and performance, and want to take your design to production.

To use the DDR or DDR2 SDRAM HPC, you can request a license file from the Altera web site at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local representative.

To use the DDR or DDR2 HPC II, contact your local sales representative to order a license.

Free Evaluation

Altera's OpenCore Plus evaluation feature is only applicable to the DDR or DDR2 SDRAM HPC. With the OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPPSM megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include MegaCore functions.
- Program a device and verify your design in hardware.

You need to purchase a license for the megafunction only when you are completely satisfied with its functionality and performance, and want to take your design to production.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation can support the following two modes of operation:

- Untethered—the design runs for a limited time
- Tethered—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely

All megafunctions in a device time-out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.



For MegaCore functions, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

Your design stops working after the hardware evaluation time expires and the `local_ready` output goes low.

Design Flow

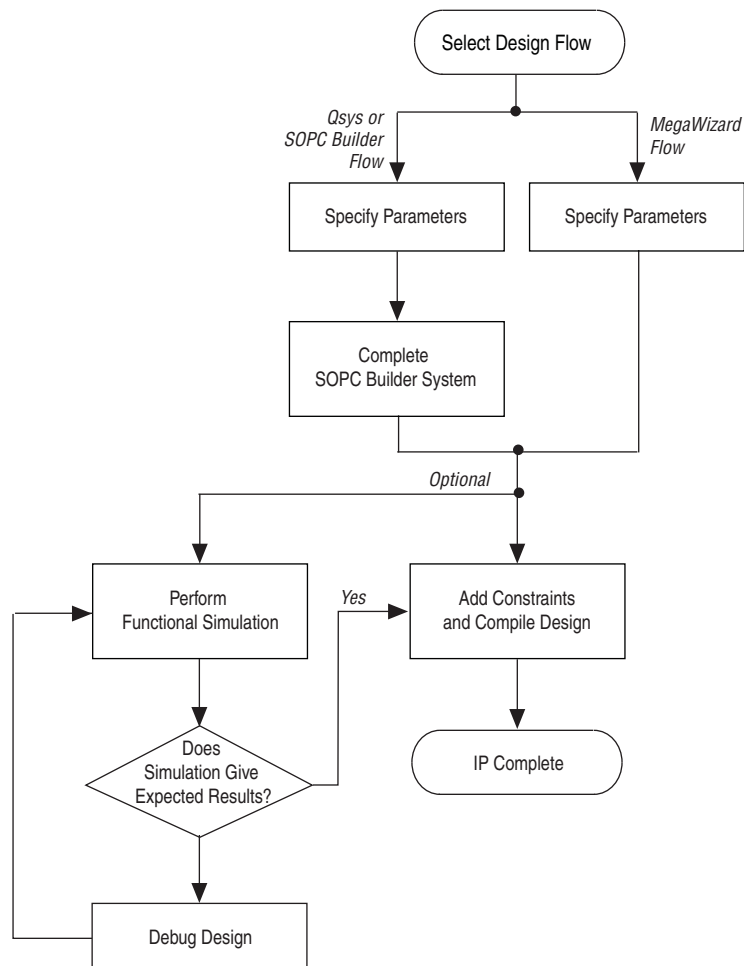
You can implement the DDR and DDR2 SDRAM Controllers with ALTMEMPHY IP using any of the following flows:

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

You can only instantiate the ALTMEMPHY megafunction using the MegaWizard Plug-In Manager flow.

Figure 2–1 shows the stages for creating a system in the Quartus II software using the available flows.

Figure 2–1. Design Flow



The SOPC Builder flow offers the following advantages:

- Generates simulation environment
- Creates custom components and integrates them via the component wizard
- Interconnects all components with the Avalon-MM interface

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

- Allows you to design directly from the DDR or DDR2 SDRAM interface to peripheral device or devices
- Achieves higher-frequency operation

SOPC Builder Flow

The SOPC Builder flow allows you to add the DDR and DDR2 SDRAM high-performance controllers directly to a new or existing SOPC Builder system.

You can also easily add other available components to quickly create an SOPC Builder system with a DDR or DDR2 SDRAM controller, such as the Nios II processor and scatter-gather direct memory access (SDMA) controllers. SOPC Builder automatically creates the system interconnect logic and system simulation environment.



For more information about SOPC Builder, refer to [volume 4](#) of the *Quartus II Handbook*. For more information about how to use controllers with SOPC Builder, refer to the [ALTMEMPHY Design Tutorials](#) section in volume 5 of the *External Memory Interface Handbook*. For more information on the Quartus II software, refer to the Quartus II Help.

Specifying Parameters

To specify the parameters for the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP, using the SOPC Builder flow, perform the following steps:

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. Add **DDR or DDR2 SDRAM Controller with ALTMEMPHY** to your system from the **System Contents** tab.



The **DDR or DDR2 SDRAM Controller with ALTMEMPHY** is in the **SDRAM** folder under the **Memories and Memory Controllers** folder.

5. Specify the required parameters on all pages in the **Parameter Settings** tab.



To avoid simulation failure, you must set **Local-to-Memory Address Mapping** to **CHP-BANK-ROW-COL** if you select **High Performance Controller II** for **Controller Architecture**.



For detailed explanation of the parameters, refer to the [“Parameter Settings” on page 3–1](#).

- Click **Finish** to complete parameterizing the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP and add it to the system.

Completing the SOPC Builder System

To complete the SOPC Builder system, perform the following steps:

- In the **System Contents** tab, select **Nios II Processor** and click **Add**.
- On the **Nios II Processor** page, in the **Core Nios II** tab, select **altmemddr** for **Reset Vector** and **Exception Vector**.
- Change the **Reset Vector Offset** and the **Exception Vector Offset** to an Avalon address that is not written to by the ALTMEMPHY megafunction during its calibration process.



The ALTMEMPHY megafunction performs memory interface calibration every time it is reset, and in doing so, writes to a range of addresses. If you want your memory contents to remain intact through a system reset, you should avoid using these memory addresses. This step is not necessary if you reload your SDRAM memory contents from flash every time you reset your system.

To calculate the Avalon-MM address equivalent of the memory address range 0x0 to 0x1f, multiply the memory address by the width of the memory interface data bus in bytes. Refer to [Table 2-1](#) for more Avalon-MM addresses.

Table 2-1. Avalon-MM Addresses for AFI Mode

External Memory Interface Width	Reset Vector Offset	Exception Vector Offset
8	0x40	0x60
16	0x80	0xA0
32	0x100	0x120
64	0x200	0x220

- Click **Finish**.
- On the **System Contents** tab, expand **Interface Protocols** and expand **Serial**.
- Select **JTAG UART** and click **Add**.
- Click **Finish**.



If there are warnings about overlapping addresses, on the System menu, click **Auto Assign Base Addresses**.

If you enable ECC and there are warnings about overlapping IRQs, on the System menu click **Auto Assign IRQs**.

- For this example system, ensure all the other modules are clocked on the `altmemddr_sysclk`, to avoid any unnecessary clock-domain crossing logic.
- Click **Generate**.



Among the files generated by SOPC Builder is the Quartus II IP File (.qip). This file contains information about a generated IP core or system. In most cases, the .qip file contains all of the necessary assignments and information required to process the MegaCore function or system in the Quartus II compiler. Generally, a single .qip file is generated for each SOPC Builder system. However, some more complex SOPC Builder components generate a separate .qip file. In that case, the system .qip file references the component .qip file.

10. Compile your design, refer to [“Compiling and Simulating” on page 4-1](#).

Qsys Flow

The Qsys flow allows you to add the DDR and DDR2 SDRAM high-performance controllers directly to a new or existing Qsys system.

You can also easily add other available components to quickly create a Qsys system with a DDR or DDR2 SDRAM controller, such as the Nios II processor and scatter-gather direct memory access (SDMA) controllers. Qsys automatically creates the system interconnect logic and system simulation environment.



For more information about Qsys, refer to [volume 1](#) of the *Quartus II Handbook*. For more information about how to use controllers with Qsys, refer to the [ALTMEMPHY Design Tutorials](#) section in volume 6 of the *External Memory Interface Handbook*. For more information on the Quartus II software, refer to the Quartus II Help.

Specifying Parameters

To specify the parameters for the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP, using the Qsys flow, perform the following steps:

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click Qsys.
3. For a new system, specify the system name and language.
4. Add **DDR or DDR2 SDRAM Controller with ALTMEMPHY** to your system from the **System Contents** tab.



The **DDR or DDR2 SDRAM Controller with ALTMEMPHY** is in the **SDRAM** folder under the **Memories and Memory Controllers** folder.

5. In the DDR or DDR2 SDRAM Controller with ALTMEMPHY parameter editor, specify the required parameters on all pages in the **Parameter Settings** tab.



To avoid simulation failure, you must set **Local-to-Memory Address Mapping to CHP-BANK-ROW-COL** if you select **High Performance Controller II** for **Controller Architecture**.



For detailed explanation of the parameters, refer to the [“Parameter Settings” on page 3-1](#).

6. Click **Finish** to complete parameterizing the DDR or DDR2 SDRAM Controller with ALTMEMPHY IP and add it to the system.

Completing the Qsys System

To complete the Qsys system, perform the following steps:

1. In the **Component Library** tab, select **Nios II Processor** and click **Add**.
2. On the **Nios II Processor** page, in the **Core Nios II** tab, select **altmemddr** for **Reset Vector** and **Exception Vector**.
3. Change the **Reset Vector Offset** and the **Exception Vector Offset** to an Avalon address that is not written to by the ALTMEMPHY megafunction during its calibration process.



The ALTMEMPHY megafunction performs memory interface calibration every time it is reset, and in doing so, writes to a range of addresses. If you want your memory contents to remain intact through a system reset, you should avoid using these memory addresses. This step is not necessary if you reload your SDRAM memory contents from flash every time you reset your system.

To calculate the Avalon-MM address equivalent of the memory address range 0x0 to 0x1f, multiply the memory address by the width of the memory interface data bus in bytes. Refer to [Table 2-2](#) for more Avalon-MM addresses.

Table 2-2. Avalon-MM Addresses for AFI Mode

External Memory Interface Width	Reset Vector Offset	Exception Vector Offset
8	0x40	0x60
16	0x80	0xA0
32	0x100	0x120
64	0x200	0x220


4. Click **Finish**.
5. On the **Component Library** tab, expand **Interface Protocols** and expand **Serial**.
6. Select **JTAG UART** and click **Add**.
7. Click **Finish**.




If there are warnings about overlapping addresses, on the System menu, click **Auto Assign Base Addresses**.

If you enable ECC and there are warnings about overlapping IRQs, on the System menu click **Auto Assign IRQs**.

8. For this example system, ensure all the other modules are clocked on the `altmemddr_sysclk`, to avoid any unnecessary clock-domain crossing logic.
9. Click **Generate**.

 To ensure that the **external connections** and **memory** 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.

 Among the files generated by Qsys is the Quartus II IP File (**.qip**). This file contains information about a generated IP core or system. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the MegaCore function or system in the Quartus II compiler. Generally, a single **.qip** file is generated for each Qsys system. However, some more complex Qsys components generate a separate **.qip** file. In that case, the system **.qip** file references the component **.qip** file.

Compile your design, refer to [“Compiling and Simulating” on page 4–1](#).

MegaWizard Plug-In Manager Flow


The MegaWizard Plug-In Manager flow allows you to customize the DDR or DDR2 SDRAM Controller with ALTMEMPHY or the stand-alone PHY with the ALTMEMPHY megafunction, and manually integrate the function into your design.

 For more information about the MegaWizard Plug-In Manager, refer to the Quartus II Help.


Specifying Parameters

To specify parameters using the MegaWizard Plug-In Manager flow, perform the following steps:

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click **MegaWizard Plug-In Manager** to start the MegaWizard Plug-In Manager.
 - The DDR or DDR2 SDRAM Controller with ALTMEMPHY is in the **Interfaces** folder under the **External Memory** folder.
 - The ALTMEMPHY megafunction is in the **I/O** folder.

 The *<variation name>* must be a different name from the project name and the top-level design entity name.

3. Specify the parameters on all pages in the **Parameter Settings** tab.

 For detailed explanation of the parameters, refer to the [“Parameter Settings” on page 3–1](#).

4. On the **EDA** tab, turn on **Generate simulation model** to generate an IP functional simulation model for the MegaCore function in the selected language.

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.



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



Some third-party synthesis tools can use a netlist that contains only the structure of the MegaCore function, but not detailed logic, to optimize performance of the design that contains the MegaCore function. If your synthesis tool supports this feature, turn on **Generate netlist**.

When targeting a VHDL simulation model, the MegaWizard Plug-In Manager still generates the `<variation_name>_alt_mem_phy.v` file for the Quartus II synthesis. Do not use this file for simulation. Use the `<variation_name>.vho` file for simulation instead.

The ALTMEMPHY megafunction only supports functional simulation. You cannot perform timing or gate-level simulation when using the ALTMEMPHY megafunction.

5. On the **Summary** tab, select the files you want to generate. A gray checkmark indicates a file that is automatically generated. All other files are optional.
6. Click **Finish** to generate the MegaCore function and supporting files. A generation report appears.
7. If you generate the MegaCore function instance in a Quartus II project, you are prompted to add the **.qip** files to the current Quartus II project. When prompted to add the **.qip** files to your project, click **Yes**. The addition of the **.qip** files enables their visibility to Nativelink. Nativelink requires the **.qip** files to include libraries for simulation.



The **.qip** file is generated by the parameter editor and contains information about the generated IP core. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the MegaCore function or system in the Quartus II compiler. The parameter editor generates a single **.qip** file for each MegaCore function.

8. After you review the generation report, click **Exit** to close the MegaWizard Plug-In Manager.
9. For the high-performance controller (HPC or HPC II), set the `<variation name>_example_top.v` or `.vhd` file to be the project top-level design file.
 - a. On the File menu, click **Open**.
 - b. Browse to `<variation name>_example_top` and click **Open**.
 - c. On the Project menu, click **Set as Top-Level Entity**.

HardCopy Guidelines

For information about targeting a HardCopy device, refer to “HardCopy Migration Design Guidelines” in Volume 3, Section VI of the *External Memory Interface Handbook*.

Generated Files

Table 2-3 shows the ALTMEMPHY generated files.

Table 2-3. ALTMEMPHY Generated Files (Part 1 of 2)

File Name	Description
alt_mem_phy_defines.v	Contains constants used in the interface. This file is always in Verilog HDL regardless of the language you chose in the MegaWizard Plug-In Manager.
<variation_name>.ppf	Pin planner file for your ALTMEMPHY variation.
<variation_name>.qip	Quartus II IP file for your ALTMEMPHY variation, containing the files associated with this megafunction.
<variation_name>.v/.vhd	Top-level file of your ALTMEMPHY variation, generated based on the language you chose in the MegaWizard Plug-In Manager.
<variation_name>.vho	Contains functional simulation model for VHDL only.
<variation_name>_alt_mem_phy_seq_wrapper.vo/.vho	A wrapper file, for simulation only, that calls the sequencer file, created based on the language you chose in the MegaWizard Plug-In Manager.
<variation_name>.html	Lists the top-level files created and ports used in the megafunction.
<variation_name>_alt_mem_phy_seq_wrapper.v/.vhd	A wrapper file, for compilation only, that calls the sequencer file, created based on the language you chose in the MegaWizard Plug-In Manager.
<variation_name>_alt_mem_phy_seq.vhd	Contains the sequencer used during calibration. This file is always in VHDL language regardless of the language you chose in the MegaWizard Plug-In Manager.
<variation_name>_alt_mem_phy.v	Contains all modules of the ALTMEMPHY variation except for the sequencer. This file is always in Verilog HDL language regardless of the language you chose in the MegaWizard Plug-In Manager. The DDR3 SDRAM sequencer is included in the <variation_name>_alt_mem_phy_seq.vhd file.
<variation_name>_alt_mem_phy_pll_<device>.ppf	This XML file describes the MegaCore pin attributes to the Quartus II Pin Planner.
<variation_name>_alt_mem_phy_pll.v/.vhd	The PLL megafunction file for your ALTMEMPHY variation, generated based on the language you chose in the MegaWizard Plug-In Manager.
<variation_name>_alt_mem_phy_delay.vhd	Includes a delay module for simulation. This file is only generated if you choose VHDL as the language of your MegaWizard Plug-In Manager output files.
<variation_name>_alt_mem_phy_dq_dqs.vhd or .v	Generated file that contains DQ/DQS I/O atoms interconnects and instance. Arria II GX devices only.
<variation_name>_alt_mem_phy_dq_dqs_clearbox.txt	Specification file that generates the <variation_name>_alt_mem_phy_dq_dqs file using the clearbox flow. Arria II GX devices only.

Table 2-3. ALTMEMPHY Generated Files (Part 2 of 2)

File Name	Description
<code><variation_name>_alt_mem_phy_pll.qip</code>	Quartus II IP file for the PLL that your ALTMEMPHY variation uses that contains the files associated with this megafunction.
<code><variation_name>_alt_mem_phy_pll_bb.v/.cmp</code>	Black box file for the PLL used in your ALTMEMPHY variation. Typically unused.
<code><variation_name>_alt_mem_phy_reconfig.qip</code>	Quartus II IP file for the PLL reconfiguration block. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.
<code><variation_name>_alt_mem_phy_reconfig.v/.vhd</code>	PLL reconfiguration block module. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.
<code><variation_name>_alt_mem_phy_reconfig_bb.v/cmp</code>	Black box file for the PLL reconfiguration block. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.
<code><variation_name>_bb.v/.cmp</code>	Black box file for your ALTMEMPHY variation, depending whether you are using Verilog HDL or VHDL language.
<code><variation_name>_ddr_pins.tcl</code>	Contains procedures used in the <code><variation_name>_ddr_timing.sdc</code> and <code><variation_name>_report_timing.tcl</code> files.
<code><variation_name>_pin_assignments.tcl</code>	Contains I/O standard, drive strength, output enable grouping, DQ/DQS grouping, and termination assignments for your ALTMEMPHY variation. If your top-level design pin names do not match the default pin names or a prefixed version, edit the assignments in this file.
<code><variation_name>_ddr_timing.sdc</code>	Contains timing constraints for your ALTMEMPHY variation.
<code><variation_name>_report_timing.tcl</code>	Script that reports timing for your ALTMEMPHY variation during compilation.

Table 2-4 shows the modules that are instantiated in the `<variation_name>_alt_mem_phy.v/.vhd` file. A particular ALTMEMPHY variation may or may not use any of the modules, depending on the memory standard that you specify.

Table 2-4. Modules in `<variation_name>_alt_mem_phy.v` File (Part 1 of 2)

Module Name	Usage	Description
<code><variation_name>_alt_mem_phy_addr_cmd</code>	All ALTMEMPHY variations	Generates the address and command structures.
<code><variation_name>_alt_mem_phy_clk_reset</code>	All ALTMEMPHY variations	Instantiates PLL, DLL, and reset logic.
<code><variation_name>_alt_mem_phy_dp_io</code>	All ALTMEMPHY variations	Generates the DQ, DQS, DM, and QVLD I/O pins.
<code><variation_name>_alt_mem_phy_mimic</code>	DDR2/DDR SDRAM ALTMEMPHY variation	Creates the VT tracking mechanism for DDR and DDR2 SDRAM PHYs.

Table 2-4. Modules in <variation_name>_alt_mem_phy.v File (Part 2 of 2)

Module Name	Usage	Description
<variation_name>_alt_mem_phy_oct_delay	DDR2/DDR SDRAM ALTMEMPHY variation when dynamic OCT is enabled.	Generates the proper delay and duration for the OCT signals.
<variation_name>_alt_mem_phy_postamble	DDR2/DDR SDRAM ALTMEMPHY variations	Generates the postamble enable and disable scheme for DDR and DDR2 SDRAM PHYs.
<variation_name>_alt_mem_phy_read_dp	All ALTMEMPHY variations (unused for Stratix III or Stratix IV devices)	Takes read data from the I/O through a read path FIFO buffer, to transition from the resynchronization clock to the PHY clock.
<variation_name>_alt_mem_phy_read_dp_group	DDR2/DDR SDRAM ALTMEMPHY variations (Stratix III and Stratix IV devices only)	A per DQS group version of <variation_name>_alt_mem_phy_read_dp.
<variation_name>_alt_mem_phy_readata_valid	DDR2/DDR SDRAM ALTMEMPHY variations	Generates read data valid signal to sequencer and controller.
<variation_name>_alt_mem_phy_seq_wrapper	All ALTMEMPHY variations	Generates sequencer for DDR and DDR2 SDRAM.
<variation_name>_alt_mem_phy_write_dp	All ALTMEMPHY variations	Generates the demultiplexing of data from half-rate to full-rate DDR data.
<variation_name>_alt_mem_phy_write_dp_fr	DDR2/DDR SDRAM ALTMEMPHY variations	A full-rate version of <variation_name>_alt_mem_phy_write_dp.

Table 2-5 shows the additional files generated by the high-performance controller II that may be in your project directory.

Table 2-5. Controller-Generated Files (Part 1 of 2)

Filename	Description
alt_mem_ddrx_addr_cmd.v	Decodes internal protocol-related signals into memory address and command signals.
alt_mem_ddrx_addr_cmd_wrap.v	A wrapper that instantiates the alt_mem_ddrx_addr_cmd.v file.
alt_mem_ddrx_ddr2_odt_gen.v	Generates the on-die termination (ODT) control signal for DDR2 memory interfaces.
alt_mem_ddrx_ddr3_odt_gen.v	Generates the on-die termination (ODT) control signal for DDR3 memory interfaces.
alt_mem_ddrx_odt_gen.v	Wrapper that instantiates alt_mem_ddrx_ddr2_odt_gen.v and alt_mem_ddrx_ddr3_odt_gen.v. This file also controls the ODT addressing scheme.
alt_mem_ddrx_rdwr_data_tmg.v	Decodes internal data burst related signals to memory data signals.
alt_mem_ddrx_arbiter.v	Contains logic that determines which command to execute based on certain schemes.
alt_mem_ddrx_burst_gen.v	Converts internal DRAM-aware commands to AFI signals.
alt_mem_ddrx_cmd_gen.v	Converts user requests to DRAM-aware commands.
alt_mem_ddrx_csr.v	Contains configuration registers.
alt_mem_ddrx_buffer.v	Contains buffer for local data.
alt_mem_ddrx_buffer_manager.v	Manages the allocation of buffers.

Table 2-5. Controller-Generated Files (Part 2 of 2)

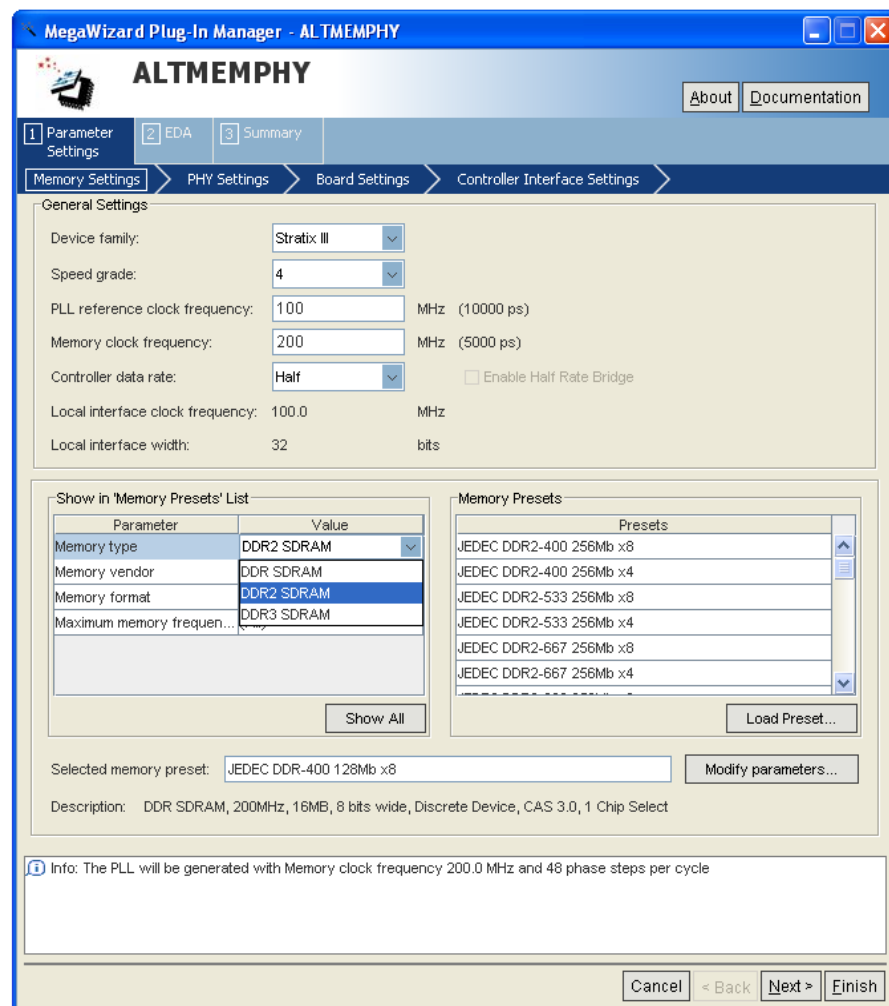
Filename	Description
alt_mem_ddrx_burst_tracking.v	Tracks data received per local burst command.
alt_mem_ddrx_dataid_manager.v	Manages the IDs associated with data stored in buffer.
alt_mem_ddrx_fifo.v	Contains the FIFO buffer to store local data to create a link; is also used in rdata_path to store the read address and error address.
alt_mem_ddrx_list.v	Tracks the DRAM commands associated with the data stored internally.
alt_mem_ddrx_rdata_path.v	Contains read data path logic.
alt_mem_ddrx_wdata_path.v	Contains write data path logic.
alt_mem_ddrx_define.iv	Defines common parameters used in the RTL files.
alt_mem_ddrx_ecc_decoder.v	Instantiates appropriate width ECC decoder logic.
alt_mem_ddrx_ecc_decoder_32_syn.v	Contains synthesizable 32-bit version of ECC decoder.
alt_mem_ddrx_ecc_decoder_64_syn.v	Contains synthesizable 64-bit version of ECC decoder.
alt_mem_ddrx_ecc_encoder.v	Instantiates appropriate width ECC encoder logic.
alt_mem_ddrx_ecc_encoder_32_syn.v	Contains synthesizable 32-bit version of ECC decoder.
alt_mem_ddrx_ecc_encoder_64_syn.v	Contains synthesizable 64-bit version of ECC decoder.
alt_mem_ddrx_ecc_encoder_decoder_wrapper.v	Wrapper that instantiates all ECC logic.
alt_mem_ddrx_input_if.v	Contains local input interface logic.
alt_mem_ddrx_mm_st_converter.v	Contains supporting logic for Avalon-MM interface.
alt_mem_ddrx_rank_timer.v	Contains a timer associated with rank timing.
alt_mem_ddrx_sideband.v	Contains supporting logic for user-controlled refresh and precharge signals.
alt_mem_ddrx_tbp.v	Contains command queue and associated logic for reordering features.
alt_mem_ddrx_timing_param.v	Contains timer logic associated with nonrank timing.
alt_mem_ddrx_controller_st_top.v	Wrapper that instantiates all submodules and configuration registers.
alt_mem_ddrx_controller_top.v	Wrapper that contains memory controller with Avalon-MM interface.
alt_mem_ddrx_controller.v	Wrapper that instantiates all submodules.

ALTMEMPHY Parameter Settings

The **Parameter Settings** page in the ALTMEMPHY parameter editor (Figure 3–1) allows you to parameterize the following settings:

- Memory Settings
- PHY Settings
- Board Settings

Figure 3–1. ALTMEMPHY Parameter Settings Page



The text window at the bottom of the MegaWizard Plug-In Manager displays information about the memory interface, warnings, and errors if you are trying to create something that is not supported. The **Finish** button is disabled until you correct all the errors indicated in this window.

The following sections describe the four tabs of the **Parameter Settings** page in more detail.

Memory Settings

In the **Memory Settings** tab, you can select a particular memory device for your system and choose the frequency of operation for the device. Under **General Settings**, you can choose the device family, speed grade, and clock information. In the middle of the page (left-side), you can filter the available memory device listed on the right side of the **Memory Presets** dialog box, refer to [Figure 3-1](#). If you cannot find the exact device that you are using, choose a device that has the closest specifications, then manually modify the parameters to match your actual device by clicking **Modify parameters**, next to the **Selected memory preset** field.

[Table 3-1](#) describes the **General Settings** available on the **Memory Settings** page of the ALTMEMPHY parameter editor.

Table 3-1. General Settings

Parameter Name	Description
Device family	Targets device family (for example, Stratix III). Table 1-2 on page 1-3 shows supported device families. The device family selected here must match the device family selected on the MegaWizard page 2a.
Speed grade	Selects a particular speed grade of the device (for example, 2, 3, or 4 for the Stratix III device family).
PLL reference clock frequency	Determines the clock frequency of the external input clock to the PLL. Ensure that you use three decimal points if the frequency is not a round number (for example, 166.667 MHz or 100 MHz) to avoid a functional simulation or a PLL locking problem.
Memory clock frequency	Determines the memory interface clock frequency. If you are operating a memory device below its maximum achievable frequency, ensure that you enter the actual frequency of operation rather than the maximum frequency achievable by the memory device. Also, ensure that you use three decimal points if the frequency is not a round number (for example, 333.333 MHz or 400 MHz) to avoid a functional simulation or a PLL locking issue.
Controller data rate	Selects the data rate for the memory controller. Sets the frequency of the controller to equal to either the memory interface frequency (full-rate) or half of the memory interface frequency (half-rate).
Enable half rate bridge	This option is only available for HPC II. Turn on to keep the controller in the memory full clock domain while allowing the local side to run at half the memory clock speed, so that latency can be reduced.
Local interface clock frequency	Value that depends on the memory clock frequency and controller data rate, and whether or not you turn on the Enable Half Rate Bridge option.
Local interface width	Value that depends on the memory clock frequency and controller data rate, and whether or not you turn on the Enable Half Rate Bridge option.

Table 3–2 describes the options available to filter the **Memory Presets** that are displayed. This set of options is where you indicate whether you are creating a datapath for DDR or DDR2 SDRAM.

Table 3–2. Memory Presets List

Parameter Name	Description
Memory type	You can filter the type of memory to display, for example, DDR2 SDRAM. The ALTMEMPHY megafunction supports DDR SDRAM and DDR2 SDRAM.
Memory vendor	You can filter the memory types by vendor. JEDEC is also one of the options, allowing you to choose the JEDEC specifications. If your chosen vendor is not listed, you can choose JEDEC for the DDR and DDR2 SDRAM interfaces. Then, pick a device that has similar specifications to your chosen device and check the values of each parameter. Make sure you change the each parameter value to match your device specifications.
Memory format	You can filter the type of memory by format, for example, discrete devices or DIMM packages.
Maximum frequency	You can filter the type of memory by the maximum operating frequency.

Using the Preset Editor to Create a Custom Memory Preset

Pick a device in the **Memory Presets** list that is closest or the same as the actual memory device that you are using. Then, click the **Modify Parameters** button to parameterize the following settings in the **Preset Editor** dialog box:

- Memory attributes—These are the settings that determine your system's number of DQ, DQ strobe (DQS), address, and memory clock pins.
- Memory initialization options—These settings are stored in the memory mode registers as part of the initialization process.
- Memory timing parameters—These are the parameters that create and time-constrain the PHY.



Even though the device you are using is listed in **Memory Presets**, ensure that the settings in the **Preset Editor** dialog box are accurate, as some parameters may have been updated in the memory device datasheets.

You can change the parameters with a white background to reflect your system. You can also change the parameters with a gray background so the device parameters match the device you are using. These parameters in gray background are characteristics of the chosen memory device and changing them creates a new custom memory preset. If you click **Save As** (at the bottom left of the page) and save the new settings in the `<quartus_install_dir>\quartus\common\ip\altera\altmemphy\lib\` directory, you can use this new memory preset in other Quartus II projects created in the same version of the software.

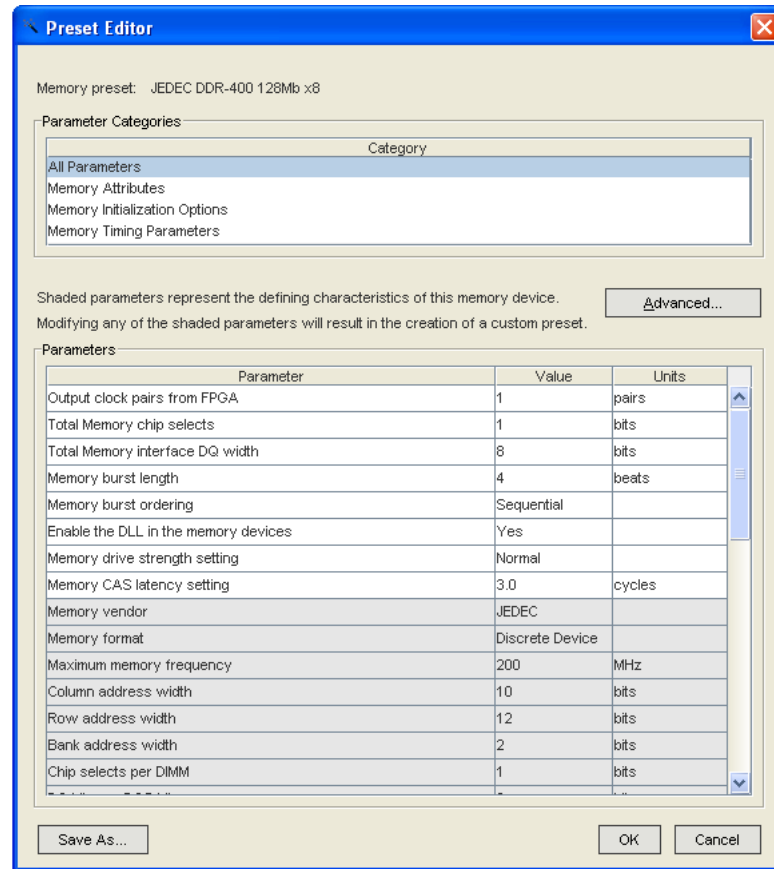
When you click **Save**, the new memory preset appears at the bottom of the **Memory Presets** list in the **Memory Settings** tab.



If you save the new settings in a directory other than the default directory, click **Load Preset** in the **Memory Settings** tab to load the settings into the **Memory Presets** list.

Figure 3-2 shows the **Preset Editor** dialog box for a DDR2 SDRAM.

Figure 3-2. DDR2 SDRAM Preset Editor



The **Advanced** option is only available for Arria II GX and Stratix IV devices. This option shows the percentage of memory specification that is calibrated by the FPGA. The percentage values are estimated by Altera based on the process variation.

Table 3–3 through Table 3–5 describe the DDR2 SDRAM parameters available for memory attributes, initialization options, and timing parameters. DDR SDRAM has the same parameters, but their value ranges are different than DDR2 SDRAM.

Table 3–3. DDR2 SDRAM Attributes Settings (Part 1 of 2)

Parameter Name	Range (1)	Units	Description
Output clock pairs from FPGA	1–6	pairs	Defines the number of differential clock pairs driven from the FPGA to the memory. More clock pairs reduce the loading of each output when interfacing with multiple devices. Memory clock pins use the signal splitter feature in Arria II GX, Stratix III, and Stratix IV devices for differential signaling.
Total Memory chip selects	1, 2, 4, or 8	bits	Sets the number of chip selects in your memory interface. The number of chip selects defines the depth of your memory. You are limited to the range shown as the local side binary encodes the chip select address. You can set this value to the next higher number if the range does not meet your specifications. However, the highest address space of the ALTMEMPHY megafunction is not mapped to any of the actual memory address. The ALTMEMPHY megafunction works with multiple chip selects and calibrates against all chip select, <code>mem_cs_n</code> signals.
Memory interface DQ width	4–288	bits	Defines the total number of DQ pins on the memory interface. If you are interfacing with multiple devices, multiply the number of devices with the number of DQ pins per device. Even though the GUI allows you to choose 288-bit DQ width, the interface data width is limited by the number of pins on the device. For best performance, have the whole interface on one side of the device.
Memory vendor	JEDEC, Micron, Qimonda, Samsung, Hynix, Elpida, Nanya, other	—	Lists the name of the memory vendor for all supported memory standards.
Memory format	Discrete Device, Unbuffered DIMM, Registered DIMM	—	Specifies whether you are interfacing with devices or modules. SODIMM is supported under unbuffered or registered DIMMs.
Maximum memory frequency	See the memory device datasheet	MHz	Sets the maximum frequency supported by the memory.
Column address width	9–11	bits	Defines the number of column address bits for your interface.
Row address width	13–16	bits	Defines the number of row address bits for your interface.
Bank address width	2 or 3	bits	Defines the number of bank address bits for your interface.
Chip selects per DIMM	1 or 2	bits	Defines the number of chip selects on each DIMM in your interface.

Table 3–3. DDR2 SDRAM Attributes Settings (Part 2 of 2)

Parameter Name	Range (1)	Units	Description
DQ bits per DQS bit	4 or 8	bits	Defines the number of data (DQ) bits for each data strobe (DQS) pin.
Precharge address bit	8 or 10	bits	Selects the bit of the address bus to use as the precharge address bit.
Drive DM pins from FPGA	Yes or No	—	Specifies whether you are using DM pins for write operation. Altera devices do not support DM pins in ×4 mode.
Maximum memory frequency for CAS latency 3.0	80–533	MHz	Specifies the frequency limits from the memory data sheet per given CAS latency. The ALTMEMPHY parameter editor generates a warning if the operating frequency with your chosen CAS latency exceeds this number.
Maximum memory frequency for CAS latency 4.0			
Maximum memory frequency for CAS latency 5.0			
Maximum memory frequency for CAS latency 6.0			

Note to Table 3–3:

(1) The range values depend on the actual memory device used.

Table 3–4. DDR2 SDRAM Initialization Options

Parameter Name	Range	Units	Description
Memory burst length	4 or 8	beats	Sets the number of words read or written per transaction. Memory burst length of four equates to local burst length of one in half-rate designs and to local burst length of two in full-rate designs.
Memory burst ordering	Sequential or Interleaved	—	Controls the order in which data is transferred between memory and the FPGA during a read transaction. For more information, refer to the memory device datasheet.
Enable the DLL in the memory devices	Yes or No	—	Enables the DLL in the memory device when set to Yes . You must always enable the DLL in the memory device as Altera does not guarantee any ALTMEMPHY operation when the DLL is turned off. All timings from the memory devices are invalid when the DLL is turned off.
Memory drive strength setting	Normal or Reduced	—	Controls the drive strength of the memory device's output buffers. Reduced drive strength is not supported on all memory devices. The default option is normal.
Memory ODT setting	Disabled, 50, 75, 150	W	Sets the memory ODT value. Not available in DDR SDRAM interfaces.
Memory CAS latency setting	3, 4, 5, 6	cycles	Sets the delay in clock cycles from the read command to the first output data from the memory.

Table 3–5. DDR2 SDRAM Timing Parameter Settings (Note 1) (Part 1 of 2)

Parameter Name	Range	Units	Description
t_{INIT}	0.001–1000	μs	Minimum memory initialization time. After reset, the controller does not issue any commands to the memory during this period.
t_{MRD}	2–39	ns	Minimum load mode register command period. The controller waits for this period of time after issuing a load mode register command before issuing any other commands. t_{MRD} is specified in ns in the DDR2 SDRAM high-performance controller and in terms of t_{CK} cycles in Micron's device datasheet. Convert t_{MRD} to ns by multiplying the number of cycles specified in the datasheet times t_{CK} , where t_{CK} is the memory operation frequency and not the memory device's t_{CK} .
t_{RAS}	8–200	ns	Minimum active to precharge time. The controller waits for this period of time after issuing an active command before issuing a precharge command to the same bank.
t_{RCD}	4–65	ns	Minimum active to read-write time. The controller does not issue read or write commands to a bank during this period of time after issuing an active command.
t_{RP}	4–65	ns	Minimum precharge command period. The controller does not access the bank for this period of time after issuing a precharge command.
t_{REFI}	1–65534	μs	Maximum interval between refresh commands. The controller performs regular refresh at this interval unless user-controlled refresh is turned on.
t_{RFC}	14–1651	ns	Minimum autorefresh command period. The length of time the controller waits before doing anything else after issuing an auto-refresh command.
t_{WR}	4–65	ns	Minimum write recovery time. The controller waits for this period of time after the end of a write transaction before issuing a precharge command.
t_{WTR}	1–3	t_{CK}	Minimum write-to-read command delay. The controller waits for this period of time after the end of a write command before issuing a subsequent read command to the same bank. This timing parameter is specified in clock cycles and the value is rounded off to the next integer.
t_{AC}	300–750	ps	DQ output access time from CK/CK# signals.
t_{DQSK}	100–750	ps	DQS output access time from CK/CK# signals.
t_{DQSQ}	100–500	ps	The maximum DQS to DQ skew; DQS to last DQ valid, per group, per access.
t_{DQSS}	0–0.3	t_{CK}	Positive DQS latching edge to associated clock edge.

Table 3–5. DDR2 SDRAM Timing Parameter Settings (Note 1) (Part 2 of 2)

Parameter Name	Range	Units	Description
t_{DS}	10–600	ps	DQ and DM input setup time relative to DQS, which has a derated value depending on the slew rate of the DQS (for both DDR and DDR2 SDRAM interfaces) and whether DQS is single-ended or differential (for DDR2 SDRAM interfaces). Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}(dc)$, not $V_{IH}(ac)$ min or $V_{IL}(ac)$ max. Refer to “Derating Memory Setup and Hold Timing” on page 3–9 for more information about how to derate this specification.
t_{DH}	10–600	ps	DQ and DM input hold time relative to DQS, which has a derated value depending on the slew rate of the DQS (for both DDR and DDR2 SDRAM interfaces) and whether DQS is single-ended or differential (for DDR2 SDRAM interfaces). Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}(dc)$, not $V_{IH}(dc)$ min or $V_{IL}(dc)$ max. Refer to “Derating Memory Setup and Hold Timing” on page 3–9 for more information about how to derate this specification.
t_{DSH}	0.1–0.5	t_{CK}	DQS falling edge hold time from CK.
t_{DSS}	0.1–0.5	t_{CK}	DQS falling edge to CK setup.
t_{IH}	100–1000	ps	Address and control input hold time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}(dc)$, not $V_{IH}(dc)$ min or $V_{IL}(dc)$ max. Refer to “Derating Memory Setup and Hold Timing” on page 3–9 for more information about how to derate this specification.
t_{IS}	100–1000	ps	Address and control input setup time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to $V_{REF}(dc)$, not $V_{IH}(ac)$ min or $V_{IL}(ac)$ max. Refer to “Derating Memory Setup and Hold Timing” on page 3–9 for more information about how to derate this specification.
t_{QHS}	100–700	ps	The maximum data hold skew factor.
t_{RRD}	2.06–64	ns	The activate command to activate time, per device, RAS to RAS delay timing parameter.
t_{FAW}	7.69–256	ns	The four-activate window time, per device.
t_{RTP}	2.06–64	ns	Read to precharge time.

Note to Table 3–5:

- (1) See the memory device data sheet for the parameter range. Some of the parameters may be listed in a clock cycle (t_{CK}) unit. If the MegaWizard Plug-In Manager requires you to enter the value in a time unit (ps or ns), convert the number by multiplying it with the clock period of your interface (and not the maximum clock period listed in the memory data sheet).

Derating Memory Setup and Hold Timing

Because the base setup and hold time specifications from the memory device datasheet assume input slew rates that may not be true for Altera devices, derate and update the following memory device specifications in the **Preset Editor** dialog box:

- t_{DS}
- t_{DH}
- t_{IH}
- t_{IS}



For Arria II GX and Stratix IV devices, you need not derate using the Preset Editor. You only need to enter the parameters referenced to V_{REF} , and the deration is done automatically when you enter the slew rate information on the **Board Settings** tab.

After derating the values, you then need to normalize the derated value because Altera input and output timing specifications are referenced to V_{REF} . However, JEDEC base setup time specifications are referenced to V_{IH}/V_{IL} AC levels; JEDEC base hold time specifications are referenced to V_{IH}/V_{IL} DC levels.

When the memory device setup and hold time numbers are derated and normalized to V_{REF} , update these values in the **Preset Editor** dialog box to ensure that your timing constraints are correct.

For example, according to JEDEC, 400-MHz DDR2 SDRAM has the following specifications, assuming 1V/ns DQ slew rate rising signal and 2V/ns differential slew rate:

- Base $t_{DS} = 50$
- Base $t_{DH} = 125$
- $V_{IH}(ac) = V_{REF} + 0.2 \text{ V}$
- $V_{IH}(dc) = V_{REF} + 0.125 \text{ V}$
- $V_{IL}(ac) = V_{REF} - 0.2 \text{ V}$
- $V_{IL}(dc) = V_{REF} - 0.125 \text{ V}$



JEDEC lists two different sets of base and derating numbers for t_{DS} and t_{DH} specifications, whether you are using single-ended or differential DQS signaling, for any DDR2 SDRAM components with a maximum frequency up to 267 MHz. In addition, the $V_{IL}(ac)$ and $V_{IH}(ac)$ values may also be different for those devices.

The V_{REF} referenced setup and hold signals for a rising edge are:

$$t_{DS}(V_{REF}) = \text{Base } t_{DS} + \text{delta } t_{DS} + (V_{IH}(ac) - V_{REF})/\text{slew_rate} = 50 + 0 + 200 = 250 \text{ ps}$$

$$t_{DH}(V_{REF}) = \text{Base } t_{DH} + \text{delta } t_{DH} + (V_{IH}(dc) - V_{REF})/\text{slew_rate} = 125 + 0 + 67.5 = 192.5 \text{ ps}$$

If the output slew rate of the write data is different from 1V/ns, you have to first derate the t_{DS} and t_{DH} values, then translate these AC/DC level specs to V_{REF} specification.

For a 2V/ns DQ slew rate rising signal and 2V/ns DQS-DQSn slew rate:

$$t_{DS}(V_{REF}) = \text{Base } t_{DS} + \text{delta } t_{DS} + (V_{IH(ac)} - V_{REF})/\text{slew_rate} = 25 + 100 + 100 = 225 \text{ ps}$$

$$t_{DH}(V_{REF}) = \text{Base } t_{DH} + \text{delta } t_{DH} + (V_{IH(dc)} - V_{REF})/\text{slew_rate} = 100 + 45 + 62.5 = 207.5 \text{ ps}$$

For a 0.5V/ns DQ slew rate rising signal and 1V/ns DQS-DQSn slew rate:

$$t_{DS}(V_{REF}) = \text{Base } t_{DS} + \text{delta } t_{DS} + (V_{IH(ac)} - V_{REF})/\text{slew_rate} = 25 + 0 + 400 = 425 \text{ ps}$$

$$t_{DH}(V_{REF}) = \text{Base } t_{DH} + \text{delta } t_{DH} + (V_{IH(dc)} - V_{REF})/\text{slew_rate} = 100 - 65 + 250 = 285 \text{ ps}$$

A similar approach can be taken to address/command slew rate derating. For t_{IS}/t_{IH} the slew rate used in the derating equations is the address/command slew rate; for t_{DS}/t_{DH} the DQ slew rate is used.

PHY Settings

Click **Next** or the **PHY Settings** tab to set the options described in [Table 3-6](#). The options are available if they apply to the target Altera device.

Table 3-6. ALTMEMPHY PHY Settings (Part 1 of 2)

Parameter Name	Applicable Device Families	Description
Use dedicated PLL outputs to drive memory clocks	HardCopy II and Stratix II (prototyping for HardCopy II)	Turn on to use dedicated PLL outputs to generate the external memory clocks, which is required for HardCopy II ASICs and their Stratix II FPGA prototypes. When turned off, the DDIO output registers generate the clock outputs. When you use the DDIO output registers for the memory clock, both the memory clock and the DQS signals are well aligned and easily meets the t_{DQSS} specification. However, when the dedicated clock outputs are for the memory clock, the memory clock and the DQS signals are not aligned properly and requires a positive phase offset from the PLL to align the signals together.
Dedicated memory clock phase	HardCopy II and Stratix II (prototyping for HardCopy II)	The required phase shift to align the CK/CK# signals with DQS/DQS# signals when using dedicated PLL outputs to drive memory clocks.
Use differential DQS	Arria II GX, Stratix III, and Stratix IV	Enable this feature for better signal integrity. Recommended for operation at 333 MHz or higher. An option for DDR2 SDRAM only, as DDR SDRAM does not support differential DQSS.
Enable external access to reconfigure PLL prior to calibration	HardCopy II, Stratix II, Stratix III, and Stratix IV (prototyping for HardCopy II)	When enabling this option for HardCopy II, Stratix II, Stratix III, and Stratix IV devices, the inputs to the ALTPLL_RECONFIG megafunction are brought to the top level for debugging purposes. This option allows you to reconfigure the PLL before calibration to adjust, if necessary, the phase of the memory clock (<code>mem_clk_2x</code>) before the start of the calibration of the resynchronization clock on the read side. The calibration of the resynchronization clock on the read side depends on the phase of the memory clock on the write side.

Table 3–6. ALTMEMPHY PHY Settings (Part 2 of 2)

Parameter Name	Applicable Device Families	Description
Instantiate DLL externally	All supported device families, except for Cyclone III devices	Use this option with Stratix III, Stratix IV, HardCopy III, or HardCopy IV devices, if you want to apply a non-standard phase shift to the DQS capture clock. The ALTMEMPHY DLL offsetting I/O can then be connected to the external DLL and the Offset Control Block. As Cyclone III devices do not have DLLs, this feature is not supported.
Enable dynamic parallel on-chip termination	Stratix III and Stratix IV	This option provides I/O impedance matching and termination capabilities. The ALTMEMPHY megafunction enables parallel termination during reads and series termination during writes with this option checked. Only applicable for DDR and DDR2 SDRAM interfaces where DQ and DQS are bidirectional. Using the dynamic termination requires that you use the OCT calibration block, which may impose a restriction on your DQS/DQ pin placements depending on your R _{UP} /R _{DN} pin locations. Although DDR SDRAM does not support ODT, dynamic OCT is still supported in Altera FPGAs. For more information, refer to either the <i>External Memory Interfaces in Stratix III Devices</i> chapter in volume 1 of the <i>Stratix III Device Handbook</i> or the <i>External Memory Interfaces in Stratix IV Devices</i> chapter in volume 1 of the <i>Stratix IV Device Handbook</i> .
Clock phase	Arria II GX, Arria GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX	Adjusting the address and command phase can improve the address and command setup and hold margins at the memory device to compensate for the propagation delays that vary with different loadings. You have a choice of 0°, 90°, 180°, and 270°, based on the rising and falling edge of the <code>phy_clk</code> and <code>write_clk</code> signals. In Stratix IV and Stratix III devices, the clock phase is set to dedicated .
Dedicated clock phase	Stratix III and Stratix IV	When you use a dedicated PLL output for address and command, you can choose any legal PLL phase shift to improve setup and hold for the address and command signals. You can set this value to between 180° and 359°, the default is 240°. However, generally PHY timing requires a value of greater than 240° for half-rate designs and 270° for full-rate designs.
Board skew	All supported device families except Arria II GX and Stratix IV devices	Maximum skew across any two memory interface signals for the whole interface from the FPGA to the memory (either a discrete memory device or a DIMM). This parameter includes all types of signals (data, strobe, clock, address, and command signals). You need to input the worst-case skew, whether it is within a DQS/DQ group, or across all groups, or across the address and command and clocks signals. This parameter generates the timing constraints in the <code>.sdc</code> file.
Autocalibration simulation options	All supported device families	Choose between Full Calibration (long simulation time), Quick Calibration , or Skip Calibration . For more information, refer to the <i>Simulation</i> section in volume 4 of the <i>External Memory Interface Handbook</i> .

Board Settings

Click **Next** or the **Board Settings** tab to set the options described in [Table 3-7](#). The board settings parameters are set to model the board level effects in the timing analysis. The options are available if you choose Arria II GX or Stratix IV device for your interface. Otherwise, the options are disabled.

Table 3-7. ALTMEMPHY Board Settings

Parameter Name	Units	Description
Number of slots/discrete devices	—	Sets the single-rank or multi-rank configuration.
CK/CK# slew rate (differential)	V/ns	Sets the differential slew rate for the CK and CK# signals.
Addr/command slew rate	V/ns	Sets the slew rate for the address and command signals.
DQ/DQS# slew rate (differential)	V/ns	Sets the differential slew rate for the DQ and DQS# signals.
DQ slew rate	V/ns	Sets the slew rate for the DQ signals.
Addr/command eye reduction (setup)	ns	Sets the reduction in the eye diagram on the setup side due to the ISI on the address and command signals.
Addr/command eye reduction (hold)	ns	Sets the reduction in the eye diagram on the hold side due to the ISI on the address and command signals.
DQ eye reduction	ns	Sets the total reduction in the eye diagram on the setup side due to the ISI on the DQ signals.
Delta DQS arrival time	ns	Sets the increase of variation on the range of arrival times of DQS due to ISI.
Max skew between DIMMs/devices	ns	Sets the largest skew or propagation delay on the DQ signals between ranks, especially true for DIMMs in different slots. This value affects the Resynchronization margin for the DDR2 interfaces in multi-rank configurations for both DIMMs and devices.
Max skew within DQS group	ns	Sets the largest skew between the DQ pins in a DQS group. This value affects the Read Capture and Write margins for the DDR2 interfaces in all configurations (single- or multi-rank, DIMM or device).
Max skew between DQS groups	ns	Sets the largest skew between DQS signals in different DQS groups. This value affects the Resynchronization margin for the DDR2 interfaces in both single- or multi-rank configurations.
Addr/command to CK skew	ns	Sets the skew or propagation delay between the CK signal and the address and command signals. The positive values represent the address and command signals that are longer than the CK signals, and the negative values represent the address and command signals that are shorter than the CK signals. This skew is used by the Quartus II software to optimize the delay of the address/command signals to have appropriate setup and hold margins for the DDR2 interfaces.

DDR or DDR2 SDRAM Controller with ALTMEMPHY Parameter Settings

The **Parameter Settings** page in the DDR or DDR2 SDRAM Controller with ALTMEMPHY parameter editor (Figure 3-3) allows you to parameterize the following settings:

- Memory Settings
- PHY Settings
- Board Settings
- Controller Settings

The **Memory Settings**, **PHY Settings**, and **Board Settings** tabs provide the same options as in the ALTMEMPHY **Parameter Settings** page.

Figure 3-3. DDR2 SDRAM Controller with ALTMEMPHY Settings

MegaWizard Plug-In Manager - DDR2 SDRAM Controller with ALTMEMPHY

DDR2 SDRAM Controller with ALTMEMPHY

About Documentation

1 Parameter Settings 2 EDA 3 Summary

Memory Settings > PHY Settings > Board Settings > **Controller Settings** >

Controller Architecture: ☒ High Performance Controller II ☐ High Performance Controller Help

Low Power Mode

☐ Enable Self-Refresh Controls

☐ Enable Power Down Controls

☐ Enable Auto Power Down. Auto Power Down Cycles: 0

Efficiency

☐ Enable User Auto-Refresh Controls

☐ Enable Auto-Precharge Control

☒ Enable Reordering Starvation limit for each command: 10 commands

Local-to-Memory Address Mapping: CHIP-ROW-BANK-COL

Command Queue Look-Ahead Depth: 4

Local Maximum Burst Count: 4

Reduce Controller Latency By: 0

Advanced Features

☐ Enable Configuration and Status Register Interface

☐ Enable Error Detection and Correction Logic

☐ Enable Auto Error Correction

☐ Enable Multi-cast Write Control

☐ Enable Reduced Bank Tracking for Area Optimization. Number of Banks to Track: 4

☐ Enable Burst Merge

Multiple Controller Clock Sharing

☐ Use clocks from another controller

Local Interface Protocol

☒ Avalon Memory-Mapped interface ☐ Native interface

Info: The PLL will be generated with Memory clock frequency 200.0 MHz and 24 phase steps per cycle

Cancel < Back Next > Finish

Controller Settings




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.

Table 3–8 shows the options provided in the **Controller Settings** tab.

Table 3–8. Controller Settings (Part 1 of 2)

Parameter	Description
Controller architecture	Specifies the controller architecture.
Enable self-refresh controls	Turn on to enable the controller to allow you to have control on when to place the external memory device in self-refresh mode, refer to “ User-Controlled Self-Refresh ” on page 6–5.
Enable power down controls	Turn on to enable the controller to allow you to have control on when to place the external memory device in power-down mode.
Enable auto power down	Turn on to enable the controller to automatically place the external memory device in power-down mode after a specified number of idle controller clock cycles is observed in the controller. You can specify the number of idle cycles after which the controller powers down the memory in the Auto Power Down Cycles field, refer to “ Automatic Power-Down with Programmable Time-Out ” on page 6–5.
Auto power down cycles	Determines the desired number of idle controller clock cycles before the controller places the external memory device in a power-down mode. The legal range is 1 to 65,535. The auto power-down mode is disabled if you set the value to 0 clock cycles.
Enable user auto-refresh controls	Turn on to enable the controller to allow you to issue a single refresh.
Enable auto-precharge control	Turn on to enable the auto-precharge control on the controller top level. Asserting the auto-precharge control signal while requesting a read or write burst allows you to specify whether or not the controller should close (auto-precharge) the current opened page at the end of the read or write burst.
Enable reordering	Turn on to allow the controller to perform command and data reordering to achieve the highest efficiency.
Starvation limit for each command	Specifies the number of commands that can be served before a waiting command is served. The legal range is from 1 to 63.
Local-to-memory address mapping	Allows you to control the mapping between the address bits on the Avalon interface and the chip, row, bank, and column bits on the memory interface. If your application issues bursts that are greater than the column size of the memory device, choose the Chip-Row-Bank-Column option. This option allows the controller to use its look-ahead bank management feature to hide the effect of changing the currently open row when the burst reaches the end of the column. On the other hand, if your application has several masters that each use separate areas of memory, choose the Chip-Bank-Row-Column option. This option allows you to use the top address bits to allocate a physical bank in the memory to each master. The physical bank allocation avoids different masters accessing the same bank which is likely to cause inefficiency, as the controller must then open and close rows in the same bank.

Table 3–8. Controller Settings (Part 2 of 2)

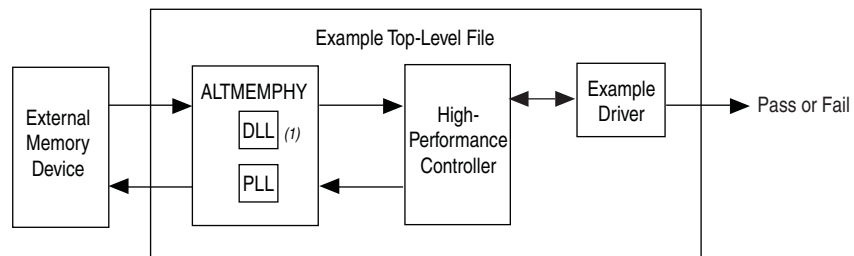
Parameter	Description
Command queue look-ahead depth	Specifies a command queue look-ahead depth value to control the number of read or write requests the look-ahead bank management logic examines.
Local maximum burst count	Specifies a burst count to configure the maximum Avalon burst count that the controller slave port accepts.
Reduce controller latency by	Specifies, in controller clock cycles, a value by which to reduce the controller latency. The default value is 0 but you have the option to choose 1 to enhance the latency performance of your design at the expense of timing closure.
Enable configuration and status register interface	Turn on to enable run-time configuration and status retrieval of the memory controller. Enabling this option adds an additional Avalon-MM slave port to the memory controller top level that allows run-time reconfiguration and status retrieving for memory timing parameters, memory address size and mode register settings, and controller features. If the Error Detection and Correction Logic option is enabled, the same slave port also allows you to control and retrieve the status of this logic. Refer to “ CSR Interface ” on page 6–4.
Enable error detection and correction logic	Turn on to enable error correction coding (ECC) for single-bit error correction and double-bit error detection.
Enable auto error correction	Turn on to allow the controller to perform auto correction when the ECC logic detects a single-bit error. Alternatively, you can turn off this option and schedule the error correction at a desired time for better system efficiency.
Multiple controller clock sharing	<p>This option is only available in SOPC Builder Flow. Turn on to allow one controller to use the Avalon clock from another controller in the system that has a compatible PLL. This option allows you to create SOPC Builder systems that have two or more memory controllers that are synchronous to your master logic.</p> <p> This option is not for use with Cyclone III or Cyclone IV family devices.</p>
Local interface protocol	<p>Specifies the local side interface between the user logic and the memory controller. The Avalon-MM interface allows you to easily connect to other Avalon-MM peripherals.</p> <p>The HPC II architecture supports only the Avalon-MM interface.</p>

After setting the parameters for the MegaCore function, you can now integrate the MegaCore function variation into your design, and compile and simulate your design. The following sections detail the steps you need to perform to compile and simulate your design.

Compiling the Design

Figure 4–1 shows the top-level view of the Altera high-performance controller design as an example of how your final design looks after you integrate the controller and the user logic.

Figure 4–1. High-Performance Controller System-Level Diagram



Note to Figure 4–1:

(1) When you choose **Instantiate DLL Externally**, DLL is instantiated outside the controller.

Before compiling a design with the ALTMEMPHY variation, you must edit some project settings, include the .sdc file, and make I/O assignments. I/O assignments include I/O standard, pin location, and other assignments, such as termination and drive strength settings. Some of these tasks are listed in the ALTMEMPHY **Generation** window. For most systems, Altera recommends that you use the **Advanced I/O Timing** feature by using the **Board Trace Model** command in the Quartus II software to set the termination and output pin loads for the device.



You cannot compile the ALTMEMPHY variation as a stand-alone top-level design because the generated .sdc timing constraints file requires the ALTMEMPHY variation be part of a larger design (with a controller and/or example driver). If you want to check whether the ALTMEMPHY variation meets your required target frequency before your memory controller is ready, create a top-level file that instantiates this ALTMEMPHY variation.

To use the Quartus II software to compile the example top-level file and perform post-compilation timing analysis, follow these steps:

1. Set up the TimeQuest timing analyzer:
 - a. On the Assignments menu, click **Timing Analysis Settings**, select **Use TimeQuest Timing Analyzer during compilation**, and click **OK**.
 - b. Add the Synopsys Design Constraints (.sdc) file, `<variation name>_phy_dds_timing.sdc`, to your project. On the Project menu, click **Add/Remove Files in Project** and browse to select the file.
 - c. Add the .sdc file for the example top-level design, `<variation name>_example_top.sdc`, to your project. This file is only required if you are using the example as the top-level design.
2. You can either use the `<variation name>_pin_assignments.tcl` or the `<variation name>.ppf` file to apply the I/O assignments generated by the MegaWizard Plug-In Manager. Using the .ppf file and the Pin Planner gives you the extra flexibility to add a prefix to your memory interface pin names. You can edit the assignments either in the Assignment Editor or Pin Planner. Use one of the following procedures to specify the I/O standard assignments for pins:
 - If you have a single SDRAM interface, and your top-level pins have default naming shown in the example top-level file, run `<variation name>_pin_assignments.tcl`.
 - or
 - If your design contains pin names that do not match the design, edit the `<variation name>_pin_assignments.tcl` file before you run the script. To edit the .tcl file, perform the following steps:
 - a. Open `<variation name>_pin_assignments.tcl` file.
 - b. Based on the flow you are using, set the `sopc_mode` value to Yes or No.
 - SOPC Builder System flow:


```
if {[info exists socp_mode]} {set socp_mode YES}
```
 - MegaWizard Plug-In Manager flow:


```
if {[info exists socp_mode]} {set socp_mode NO}
```
 - c. Type your preferred prefix in the `pin_prefix` variable. For example, to add the prefix `my_mem`, do the following:


```
if {[info exists set_prefix]}{set pin_prefix "my_mem_"}
```

After setting the prefix, the pin names are expanded as shown in the following:

 - SOPC Builder System flow:



```
my_mem_cs_n_from_the_<your instance name>
```
 - MegaWizard Plug-In Manager flow:


```
my_mem_cs_n[0]
```



If your top-level design does not use single bit bus notation for the single-bit memory interface signals (for example, `mem_dqs` rather than `mem_dqs[0]`), in the Tcl script you should change `set single_bit { [0] }` to `set single_bit {}`.

or


- Alternatively, to change the pin names that do not match the design, you can add a prefix to your pin names by performing the following steps:
 - a. On the Assignments menu, click **Pin Planner**.
 - b. On the Edit menu, click **Create/Import Megafunction**.
 - c. Select **Import an existing custom megafunction** and navigate to `<variation name>.ppf`.
 - d. Type the prefix you want to use in **Instance name**. For example, change **mem_addr** to **core1_mem_addr**.
 - 3. Set the top-level entity to the top-level design.
 - a. On the File menu, click **Open**.
 - b. Browse to your SOPC Builder system top-level design or `<variation name>_example_top` if you are using MegaWizard Plug-In Manager, and click **Open**.
 - c. On the Project menu, click **Set as Top-Level Entity**.
 - 4. Assign the DQ and DQS pin locations.
 - a. You should assign pin locations to the pins in your design, so the Quartus II software can perform fitting and timing analysis correctly.
 - b. Use either the Pin Planner or Assignment Editor to assign the clock source pin manually. Also choose which DQS pin groups should be used by assigning each DQS pin to the required pin. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group.
-  To avoid no-fit errors when you compile your design, ensure that you place the `mem_clk` pins to the same edge as the `mem_dq` and `mem_dqs` pins, and set an appropriate I/O standard for the non-memory interfaces, such as the clock source and the reset inputs, when assigning pins in your design. For example, for DDR SDRAM select **2.5 V** and for DDR2 SDRAM select **1.8 V**. Also select in which bank or side of the device you want the Quartus II software to place them.
- 5. For Stratix III and Stratix IV designs, if you are using advanced I/O timing, specify board trace models in the **Device & Pin Options** dialog box. If you are using any other device and not using advanced I/O timing, specify the output pin loading for all memory interface pins.
 - 6. Select your required I/O driver strength (derived from your board simulation) to ensure that you correctly drive each signal or ODT setting and do not suffer from overshoot or undershoot.
 - 7. To compile the design, on the Processing menu, click **Start Compilation**.

-  To attach the SignalTap[®] II logic analyzer to your design, refer to *AN 380: Test DDR or DDR2 SDRAM Interfaces on Hardware Using the Example Driver*.

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


Simulating the Design


During system generation, SOPC Builder optionally generates a simulation model and testbench for the entire system, which you can use to easily simulate your system in any of Altera's supported simulation tools. The MegaWizard also generates a set of ModelSim[®] Tcl scripts and macros that you can use to compile the testbench, IP functional simulation models, and plain-text RTL design files that describe your system in the ModelSim simulation software (refer to “Generated Files” on page 2–8).

-  For more information about simulating SOPC Builder systems, refer to **volume 4** of the *Quartus II Handbook* and *AN 351: Simulating Nios II Embedded Processor Designs*. 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 *ALTMEMPHY Design Tutorials* section in volume 5 of the *External Memory Interface Handbook*.


In ALTMEMPHY variations for DDR or DDR2 SDRAM interfaces, you have the following simulation options:

- Skip calibration—performs a static setup of the ALTMEMPHY megafunction to skip calibration and go straight into user mode.


 Skip calibration mode supports the default ALTMEMPHY parameterization with CAS latency of 3 for DDR memory, and all CAS latencies for DDR2 memory. The additive latency and registered DIMMs must be disabled for all memory types.


 Skip calibration is unavailable for DDR2 RDIMMs.

- Quick calibration—performs a calibration on a single pin and chip select.


 You may see memory model warnings about initialization times.


- Full calibration—across all pins and chip selects. This option allows for longer simulation time.

 In quick and skip calibration modes, the ALTMEMPHY megafunction ignores any delays, and assumes that all delays in the testbench and memory model are 0 ps. To successfully simulate a design with delays in the testbench and memory model, you must generate a full calibration mode model in the MegaWizard Plug-In Manager. If you are simulating your ALTMEMPHY-based design with a Denali model, Altera recommends that you use full calibration mode.

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

The ALTMEMPHY megafunction creates the datapath between the memory device and the memory controller, and user logic in various Altera devices. The ALTMEMPHY megafunction GUI helps you configure multiple variations of a memory interface. You can then connect the ALTMEMPHY megafunction variation with either a user-designed controller or with Altera's high-performance controller II (HPC II). In addition, the ALTMEMPHY megafunction and the Altera high-performance controller II are available for full-rate and half-rate DDR and DDR2 SDRAM interfaces.

 For legacy device families not supported by the ALTMEMPHY megafunction (such as Cyclone, Cyclone II, Stratix, and Stratix GX devices), use the Altera legacy integrated static datapath and controller MegaCore functions.

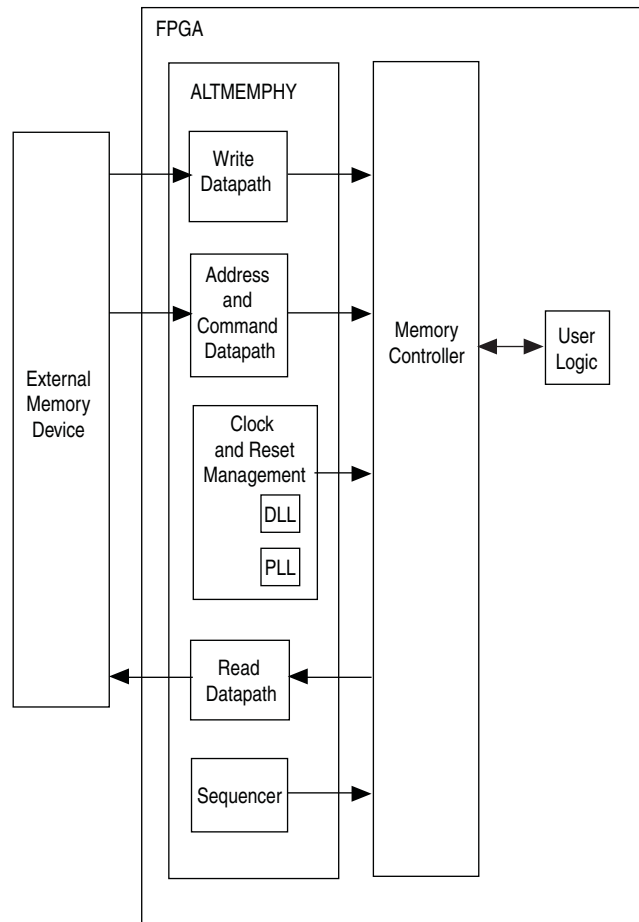
 If the ALTMEMPHY megafunction does not meet your requirements, you can also create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software. However, you are then responsible for every aspect of the interface, including timing analysis and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

This chapter describes the DDR and DDR2 SDRAM ALTMEMPHY megafunction, which uses AFI as the interface between the PHY and the controller.

Block Description

Figure 5-1 on page 5-2 shows the major blocks of the ALTMEMPHY megafunction and how it interfaces with the external memory device and the controller. The ALTPLL megafunction is instantiated inside the ALTMEMPHY megafunction, so that you do not need to generate the clock to any of the ALTMEMPHY blocks.

Figure 5-1. ALTMEMPHY Megafunction Interfacing with the Controller and the External Memory



The ALTMEMPHY megafunction comprises the following blocks:

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

Calibration

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



For information about calibration, refer to Chapter 3 of the *Debugging* section in volume 4 of the *External Memory Interface Handbook*.

Address and Command Datapath

This topic describes the address and command datapath.

Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices

The address and command datapath for full-rate designs is similar to half-rate designs, except that the address and command signals are all asserted for one memory clock cycle only (1T signaling).

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

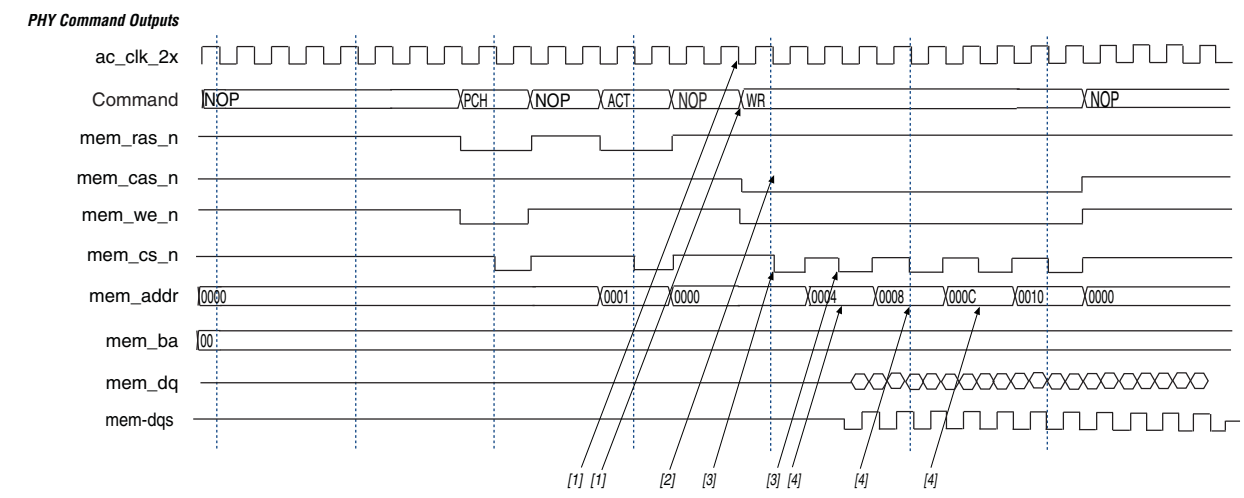
- 1T (full rate)—the duration of the address and command is a single memory clock cycle (`mem_clk_2x`, [Figure 5-2](#)). This applies to all address and command signals in full-rate designs or `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate designs.
- 2T (half rate)—the duration of the address and command is two memory clock cycles. For half-rate designs, the ALTMEMPHY megafunction supports only a burst size of four, which means the burst size on the local interface is always set to 1. The size of the data is $4n$ -bits wide on the local side and is n -bits wide on the memory side. To transfer all the $4n$ -bits at the double data rate, two memory-clock cycles are required. The new address and command can be issued to memory every two clock cycles. This scheme applies to all address and command signals, except for `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate mode.



Refer to [Table 5-4](#) in “PLL” on [page 5-5](#) to see the frequency relationship of `mem_clk_2x` with the rest of the clocks.

Figure 5-2 shows a 1T chip select signal (`mem_cs_n`), which is active low, and disables the command in the memory device. All commands are masked when the chip-select signal is inactive. The `mem_cs_n` signal is considered part of the command code.

Figure 5-2. Arria GX, Arria II GX, Cyclone II, HardCopy II, Stratix II, and Stratix II GX Address and Command Datapath



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

The waveform in Figure 5-2 shows a NOP command followed by back-to-back write commands. The following sequence corresponds with the numbered items in Figure 5-2:

1. The commands are asserted either on the rising edge of `ac_clk_2x`. The `ac_clk_2x` is derived from either `mem_clk_2x` (0°), `write_clk_2x` (270°), or the inverted variations of those two clocks (for 180° and 90° phase shifts). This depends on the setting of the address and command clock in the ALTMEMPHY parameter editor. Refer to “Address and Command Datapath” on page 5-3 for illustrations of this clock in relation to the `mem_clk_2x` or `write_clk_2x` signals.
2. All address and command signals (except for `mem_cs_n`, `mem_cke`, and `mem_odt` signals) remain asserted on the bus for two clock cycles, allowing sufficient time for the signals to settle.
3. The `mem_cs_n`, `mem_cke`, and `mem_odt` signals are asserted during the second cycle of the address/command phase. By asserting the chip-select signal in alternative cycles, back-to-back read or write commands can be issued.
4. The address is incremented every other `ac_clk_2x` cycle.



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



The address and command clock can be 0, 90, 180, or 270° from the system clock (refer to “Address and Command Datapath” on page 5-3).

Stratix III and Stratix IV Devices

The address and command clock in Stratix III and Stratix IV devices is one of the PLL dedicated clock outputs whose phase can be adjusted to meet the setup and hold requirements of the memory clock. The Stratix III address and command clock, `ac_clk_1x`, is half rate. The command and address pins use the DDIO output circuitry to launch commands from either the rising or falling edges of the clock. The chip select (`cs_n`) pins and ODT are only enabled for one memory clock cycle and can be launched from either the rising or falling edge of `ac_clk_1x` signal, while the address and other command pins are enabled for two memory clock cycles and can also be launched from either the rising or falling edge of `ac_clk_1x` signal.

The full-rate address and command datapath is the same as that of the half-rate address and command datapath, except that there is no full-rate to half-rate conversion in the IOE. The address and command signals are full-rate here.

Clock and Reset Management

This topic describes the clock and reset management for specific device types.

Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices

The clocking and reset block is responsible for clock generation, reset management, and phase shifting of clocks. It also has control of clock network types that route the clocks.

Clock Management

The clock management feature allows the ALTMEMPHY megafunction to work out the optimum resynchronization clock phase during calibration, and track the system voltage and temperature (VT) variations. Clock management is achieved by phase-shifting the clocks relative to each other.

Clock management circuitry is implemented by the following device resources:


- PLL
- PLL reconfiguration
- DLL

PLL

The ALTMEMPHY parameter editor automatically generates an ALTPLL megafunction instance. The ALTPLL megafunction is responsible for generating the different clock frequencies and relevant phases used within the ALTMEMPHY megafunction.

The minimum PHY requirement is to have 16 phases of the highest frequency clock. The PLL uses the **With No Compensation** option to minimize jitter.

You must choose a PLL and PLL input clock pin that are located on the same side of the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended for DDR/DDR2 SDRAM interfaces as jitter can accumulate with the use of cascaded PLLs causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set.

 If the design cascades PLLs, the source (upstream) PLL should have a low-bandwidth setting; the destination (downstream) PLL should have a high-bandwidth setting. Adjacent PLLs cascading is recommended to reduce clock jitters.


 For more information about the VCO frequency range and the available phase shifts, refer to the *PLLs in Stratix II and Stratix II GX Devices* chapter in the respective device family handbook.

Table 5–1 shows the clock outputs for Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.

Table 5–1. DDR/DDR2 SDRAM Clocking in Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices (Part 1 of 3)

Design Rate	Clock Name	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Half-rate	phy_clk_1x and aux_half_rate_clk	C0	0°	Half-Rate	Global	The only clocks parameterizable for the ALTMEMPHY megafunction. These clocks also feed into a divider circuit to provide the PLL scan_clk signal (for reconfiguration) that must be lower than 100 MHz.
	mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Clocks DQS and as a reference clock for the memory devices.
Full rate	aux_half_rate_clk	C0	0°	Half-Rate	Global	The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL scan_clk signal (for reconfiguration) that must be lower than 100 MHz.
	phy_clk_1x (1) and mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Clocks DQS and as a reference clock for the memory devices.

Table 5–1. DDR/DDR2 SDRAM Clocking in Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices (Part 2 of 3)

Design Rate	Clock Name	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Half-rate and full rate	write_clk_2x	C2	–90°	Full-Rate	Global	Clocks the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the mem_clk_2x by 90°.
Half-rate and full rate	mem_clk_ext_2x	C3	> 0°	Full-Rate	Dedicated	This clock is only used if the memory clock generation uses dedicated output pins. Applicable only in HardCopy II or Stratix II prototyping for HardCopy II designs.
Half-rate and full rate	resync_clk_2x	C4	Calibrated	Full-Rate	Regional	Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-clocked DDIO groups.
Half-rate and full rate	measure_clk_2x	C5	Calibrated	Full-Rate	Regional (2)	This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects.

Table 5-1. DDR/DDR2 SDRAM Clocking in Arria GX, HardCopy II, Stratix II, and Stratix II GX Devices (Part 3 of 3)

Design Rate	Clock Name	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Half-rate and full rate	ac_clk_2x	—	0, 90°, 180°, 270°	Full-Rate	Global	The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift). Refer to “Address and Command Datapath” on page 5-3 for illustrations of the address and command clock relationship with the mem_clk_2x or write_clk_2x signals.

Notes to Table 5-4:

- (1) In full-rate designs a _1x clock may run at full rate clock.
- (2) This clock should be of the same clock network clock as the resync_clk_2x clock.

For full-rate clock and reset management refer to Table 5-4. The PLL is configured exactly in the same way as in half-rate designs. The PLL information and restriction from half-rate designs also applies.



The phy_clk_1x clock is now full-rate, despite the “1x” naming convention.

You must choose a PLL and PLL input clock pin that are located on the same side of the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended for DDR/DDR2 SDRAM interfaces as jitter can accumulate with the use of cascaded PLLs causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set. The PLL restrictions in half-rate designs also applies to full-rate designs.

Table 5-2 shows the clock outputs that Arria II GX devices use.

Table 5-2. DDR/DDR2 SDRAM Clocking in Arria II GX Devices (Part 1 of 3)

Design Rate	Clock Name (1)	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type		Notes
					All Quadrants	Any 3 Quadrants (2)	
Half-rate	phy_clk_1x and aux_half_rate_clk	C0	0°	Half-Rate	Global	Global	The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL scan_clk signal (for reconfiguration) that must be lower than 100 MHz.
	mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Regional	Clocks DQS and as a reference clock for the memory devices.
Full rate	aux_half_rate_clk	C0	0°	Half-Rate	Global	Global	The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL scan_clk signal (for reconfiguration) that must be lower than 100 MHz.
	phy_clk_1x (1) and mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Regional	Clocks DQS and as a reference clock for the memory devices.
Half-rate and full rate	Unused	C2	—	—	—	—	—

Table 5–2. DDR/DDR2 SDRAM Clocking in Arria II GX Devices (Part 2 of 3)

Design Rate	Clock Name ⁽¹⁾	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type		Notes
					All Quadrants	Any 3 Quadrants ⁽²⁾	
Half-rate and full rate	write_clk_2x	C3	–90°	Full-Rate	Global	Regional	Clocks the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the mem_clk_2x by 90°.
Half-rate and full rate	ac_clk_2x	C3	90°	Full-Rate	Global	Regional	Address and command clock. The ac_clk_2x clock is derived from either mem_clk_2x (when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift). Refer to “Address and Command Datapath” on page 5–3 for illustrations of the address and command clock relationship with the mem_clk_2x or write_clk_2x signals.
Half-rate and full rate	cs_n_clk_2x	C3	90°	Full-Rate	Global	Global	Memory chip-select clock. The cs_n_clk_2x clock is derived from ac_clk_2x.
Half-rate and full rate	resync_clk_2x	C4	Calibrated	Full-Rate	Global	Regional	Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-clocked DDIO groups.

Table 5-2. DDR/DDR2 SDRAM Clocking in Arria II GX Devices (Part 3 of 3)

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

Note to Table 5-2:

- (1) In full-rate designs, a _1x clock may run at full-rate clock rate.
- (2) The default clock network type is Global, however you can specify a regional clock network to improve clock jitter if your design uses any three quadrants.

PLL Reconfiguration

The ALTMEMPHY parameter editor automatically generates the PLL reconfiguration block by instantiating an ALTPLL_RECONFIG variation for Stratix II and Stratix II GX devices to match the generated ALTPLL megafunction instance. The ALTPLL_RECONFIG megafunction varies the resynchronization clock phase and the measure clock phase.



The ALTMEMPHY parameter editor does not instantiate an ALTPLL_RECONFIG megafunction for Arria II GX devices, as this device uses the dedicated phase stepping I/O on the PLL.

DLL

A DLL instance is included in the generated ALTMEMPHY variation. When using the DQS to capture the DQ read data, the DLL center-aligns the DQS strobe to the DQ data. The DLL settings depend on the interface clock frequency.



For more information, refer to the *External Memory Interfaces* chapter in the device handbook for your target device family.

Reset Management

The reset management block is responsible for the following:

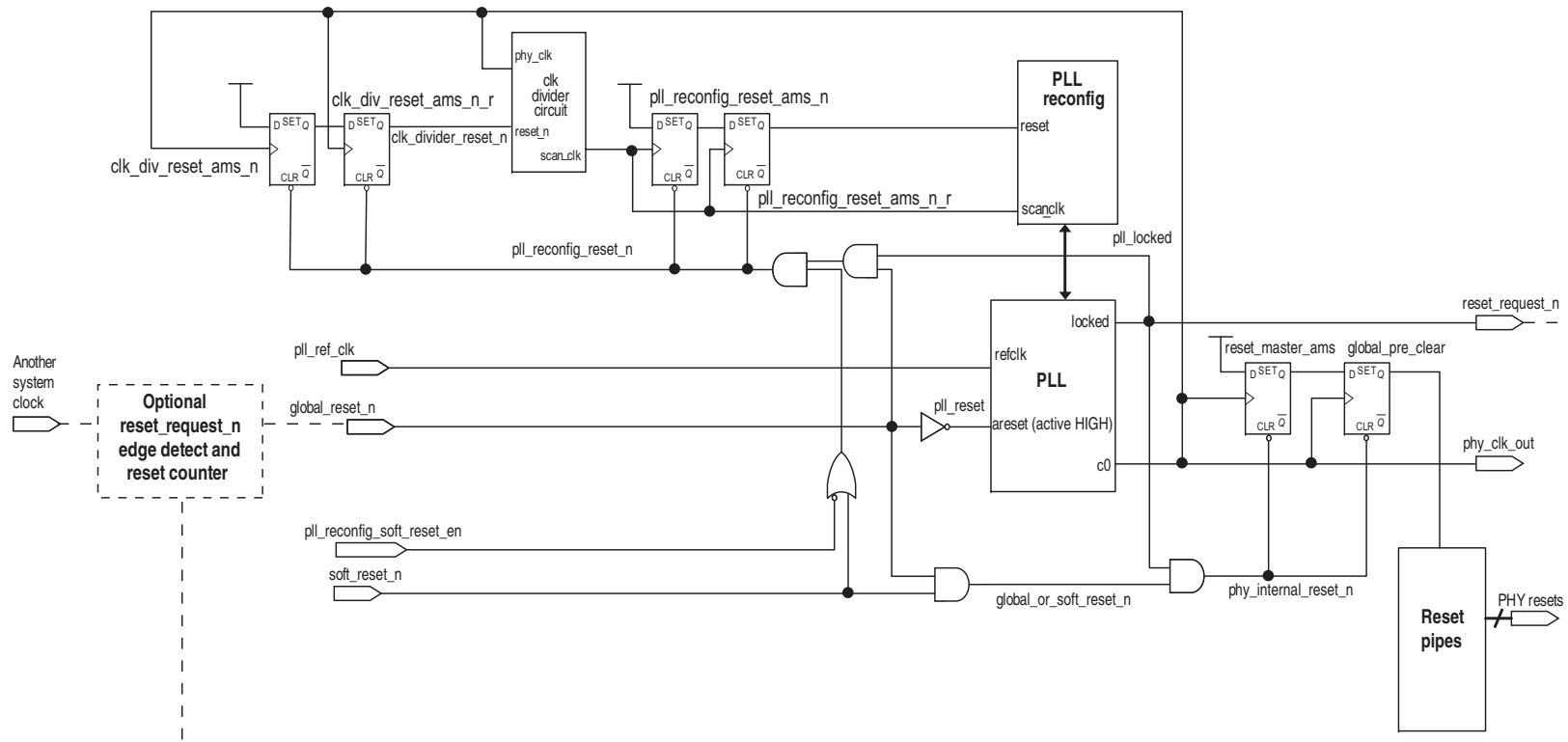
- Provides appropriately timed resets to the ALTMEMPHY megafunction datapaths and functional modules
- Performs the reset sequencing required for different clock domains
- Provides reset management of PLL and PLL reconfiguration functions

- Manages any circuit-specific reset sequencing

Each reset is an asynchronous assert and synchronous deassert on the appropriate clock domain. The reset management design uses a standard two-register synchronizer to avoid metastability. A unique reset metastability protection circuit for the clock divider circuit is required because the `phy_clk` domain reset metastability protection flipflops have fan-in from the `soft_reset_n` input, and so these registers cannot be used.

Figure 5-3 shows the ALTMEMPHY reset management block for Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX devices. The `pll_ref_clk` signal goes directly to the PLL, eliminating the need for global clock network routing. If you are using the `pll_ref_clk` signal to feed other parts of your design, you must use a global clock network for the signal. If `pll_reconfig_soft_reset_en` signal is held low, the PLL reconfig is not reset during a soft reset, which allows designs targeting HardCopy II devices to hold the PHY in reset while still accessing the PLL reconfig block. However, designs targeting Arria GX, Arria II GX, or Stratix II devices are expected to tie the `pll_reconfig_soft_en` shell to VCC to enable PLL reconfig soft resets.

Figure 5–3. ALTMEMPHY Reset Management Block for Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices (Note 1)



Note to Figure 5–3:

(1) The reset circuit for Arria II GX and Cyclone III devices have no PLL reconfig block.

Cyclone III Devices

Clock management circuitry is implemented using the ALTPLL megafunction.

The ALTPLL megafunction is instantiated within the ALTMEMPHY megafunction and is responsible for generating all the clocks used by the ALTMEMPHY megafunction and the memory controller.

The minimum PHY requirement is to have 48 phases of the highest frequency clock. The PLL uses Normal mode, unlike other device families. Cyclone III PLL in normal mode emits low jitter already such that you do not require to set the PLL in the **With No Compensation** option. Changing the PLL compensation mode may result in inaccurate timing results.

You must choose a PLL and PLL input clock pin that are located on the same side of the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended as jitter can accumulate with the use of cascaded PLLs causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set.

Table 5-3 lists the clocks generated by the ALTPLL megafunction.

Table 5-3. DDR/DDR2 SDRAM Clocking in Cyclone III Devices (Part 1 of 2) (Part 1 of 2)

Design Rate	Clock Name	Post-Scale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Half-rate	phy_clk_1x and aux_half_rate_clk	C0	0°	Half-Rate	Global	The only half-rate clock parameterizable for the ALTMEMPHY megafunction to be used by the controller. This clock is not used in full-rate controllers. This clock also feeds into a divider circuit to provide the PLL scan_clk signal for reconfiguration.
	mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Generates DQS signals and the memory clock and to clock the PHY in full-rate mode.

Table 5-3. DDR/DDR2 SDRAM Clocking in Cyclone III Devices (Part 2 of 2) (Part 2 of 2)

Design Rate	Clock Name	Post-Scale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Full rate	aux_half_rate_clk	C0	0°	Half-Rate	Global	The only half-rate clock parameterizable for the ALTMEMPHY megafunction to be used by the controller. This clock is not used in full-rate controllers. This clock also feeds into a divider circuit to provide the PLL scan_clk signal for reconfiguration.
	phy_clk_1x and mem_clk_2x and aux_full_rate_clk	C1	0°	Full-Rate	Global	Generates DQS signals and the memory clock and to clock the PHY in full-rate mode.
Half-rate and full rate	write_clk_2x	C2	-90°	Full-Rate	Global	Clocks the data (DQ) when you perform a write to the memory.
Half-rate and full rate	resynch_clk_2x	C3	Calibrated	Full-Rate	Global	A full-rate clock that captures and resynchronizes the captured read data. The capture and resynchronization clock has a variable phase that is controlled via the PLL reconfiguration logic by the control sequencer block.
Half-rate and full rate	measure_clk_2x	C4	Calibrated	Full-Rate	Global	This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, you can track VT effects on the FPGA and compensate for them.
Half-rate and full rate	ac_clk_2x	—	0°, 90°, 180°, 270°	Full-Rate	Global	This clock is derived from mem_clk_2x when you choose 0° or 180° phase shift) or write_clk_2x (when you choose 90° or 270° phase shift), refer to “Address and Command Datapath” on page 5-3 .

Reset Management

The reset management for Cyclone III devices is instantiated in the same way as it is with Stratix II devices.

Stratix III and Stratix IV Devices

The clocking and reset block is responsible for clock generation, reset management, and phase shifting of clocks. It also has control of clock network types that route the clocks.

The ability of the ALTMEMPHY megafunction to work out the optimum phase during calibration and to track voltage and temperature variation relies on phase shifting the clocks relative to each other.



Certain clocks need to be phase shifted during the ALTMEMPHY megafunction operation.

Clock management circuitry is implemented by using:

- PLL
- DLL

PLL

The ALTMEMPHY parameter editor automatically generates an ALTPLL megafunction instance. The ALTPLL megafunction is responsible for generating the different clock frequencies and relevant phases used within the ALTMEMPHY megafunction.

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

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



For more information about the VCO frequency range and the available phase shifts, refer to the *Clock Networks and PLLs in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* or the *Clock Networks and PLLs in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

For Stratix IV and Stratix III devices, the PLL reconfiguration is done using the phase-shift inputs on the PLL instead of using the PLL reconfiguration megafunction. Table 5-4 shows the Stratix IV and Stratix III PLL clock outputs.

Table 5-4. DDR2 SDRAM Clocking in Stratix IV and Stratix III Devices (Part 1 of 2)

Design Rate	Clock Name (1)	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Half-rate	phy_clk_1x and aux_half_rate_clk	C0	30	Half-Rate	Global	The only clock parameterizable for the ALTMEMPHY megafunction. It is set to 30° to ensure proper half-rate to full-rate transfer for write data and DQS. This clock also feeds into a divider circuit to provide the PLL scan_clk signal for reconfiguration.
	aux_full_rate_clk	C2	60	Full-Rate	Global	The aux_clk. The 60°-offset maintains edge alignment with the offset on phy_clk_1x.
Full-rate	aux_half_rate_clk	C0	0	Half-Rate	Global	The aux_clk.
	phy_clk_1x and aux_full_rate_clk	C2	0	Full-Rate	Global	The only clock parameterizable for the ALTMEMPHY megafunction. This clock also feeds into a divider circuit to provide the PLL scan_clk signal for reconfiguration.
Half-rate and full-rate	mem_clk_2x	C1	0	Full-Rate	Special	Generates mem_clk that provides the reference clock for the DLL. A dedicated routing resource exists from the PLL to the DLL, which you select with the regional routing resource for the mem_clk using the following attribute in the HDL: (-name global_signal dual_regional _clock; -to dll~DFFIN -name global_signal off). If you use an external DLL, apply this attribute similarly to the external DLL.

Table 5–4. DDR2 SDRAM Clocking in Stratix IV and Stratix III Devices (Part 2 of 2)

Design Rate	Clock Name (1)	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
Half-rate and full-rate	write_clk_2x	C3	–90	Full-Rate	Dual regional	Clocks the data out of the double data rate input/output (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the mem_clk_2x clock by 90°.
Half-rate and full-rate	resync_clk_2x	C4	Calibrated	Full-Rate	Dual regional	This clock feeds the I/O clock divider that then clocks the resynchronization registers after the capture registers. Its phase is adjusted in the calibration process. You can use an inverted version of this clock for postamble clocking.
Half-rate and full-rate	measure_clk_1x (2)	C5	Calibrated	Half-Rate	Dual regional	This clock is for VT tracking. This free-running clock measures relative phase shifts between the internal clock(s) and those being fed back through a mimic path. As a result, the ALTMEMPHY megafunction can track VT effects on the FPGA and compensate for the effects.
Half-rate and full-rate	ac_clk_1x	C6	Set in the GUI	Half-Rate	Dual regional	Address and command clock.

Notes to Table 5–4:

- (1) In full-rate designs a _1x clock may run at full-rate clock rate.
- (2) This clock should be of the same clock network clock as the resync_clk_2x clock.

Clock and reset management for full-rate designs is similar to half-rate support (see [Table 5–4 on page 5–17](#)). The PLL is configured exactly in the same way as for half-rate support. The mem_clk_2x output acts as the PHY full-rate clock. Also, instead of going through the I/O clock divider, the resync_clk_2x output is now directly connected to the resynchronization registers. The rest of the PLL outputs are connected in the same way as for half-rate support.

You must choose a PLL and PLL input clock pin that are located on the same side of the device as the memory interface to ensure minimal jitter. Cascaded PLLs are not recommended as jitter can accumulate, causing the memory output clock to violate the memory device jitter specification. Also, ensure that the input clock to the PLL is stable before the PLL locks. If not, you must perform a manual PLL reset (by driving the `global_reset_n` signal low) and relock the PLL to ensure that the phase relationship between all PLL outputs are properly set. The PLL restrictions in half-rate designs also applies to full-rate designs.

DLL

DLL settings are set depending on the memory clock frequency of operation.

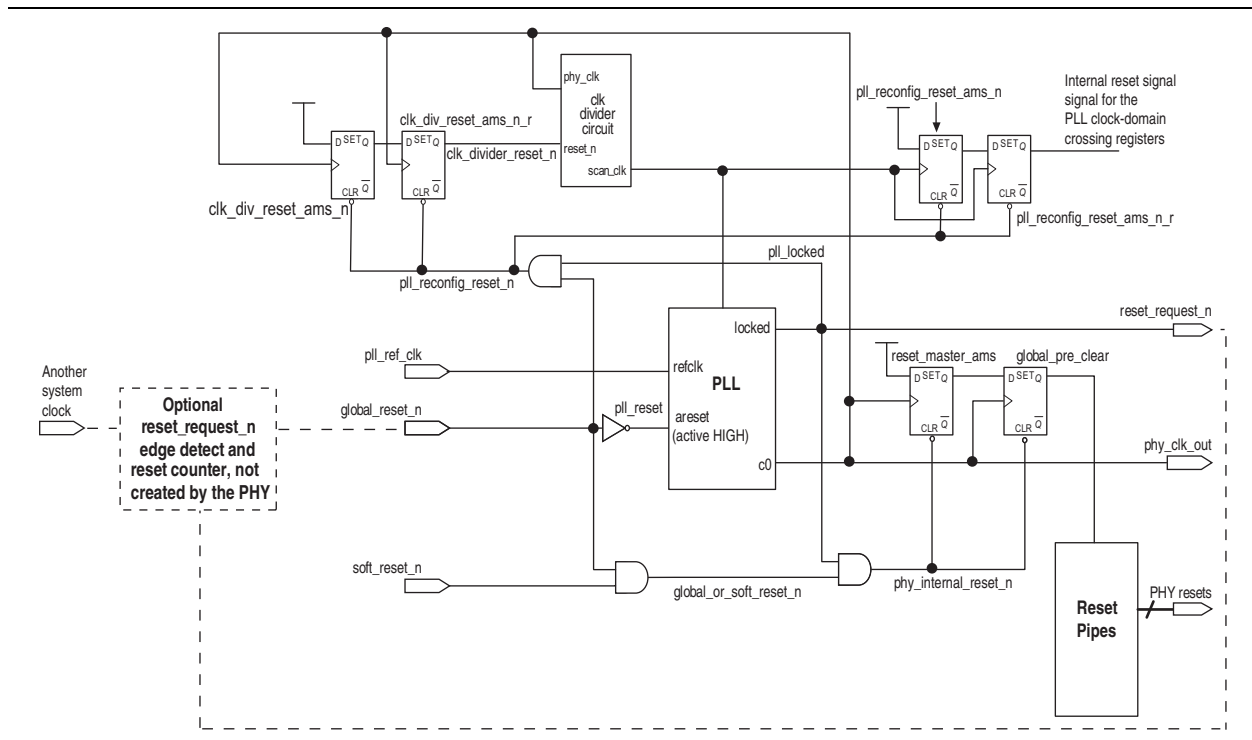
For more information on the DLL, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

Reset Management

Figure 5-4 shows the main features of the reset management block for the DDR3 SDRAM PHY. You can use the `pll_ref_clk` input to feed the optional `reset_request_n` edge detect and reset counter module. However, this requires the `pll_ref_clk` signal to use a global clock network resource.

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

Figure 5-4. ALTMEMPHY Reset Management Block for Stratix IV and Stratix III Devices



Read Datapath

This topic describes the read datapath.

Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices

The following section discusses support for DDR/DDR2 SDRAM for Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX devices.

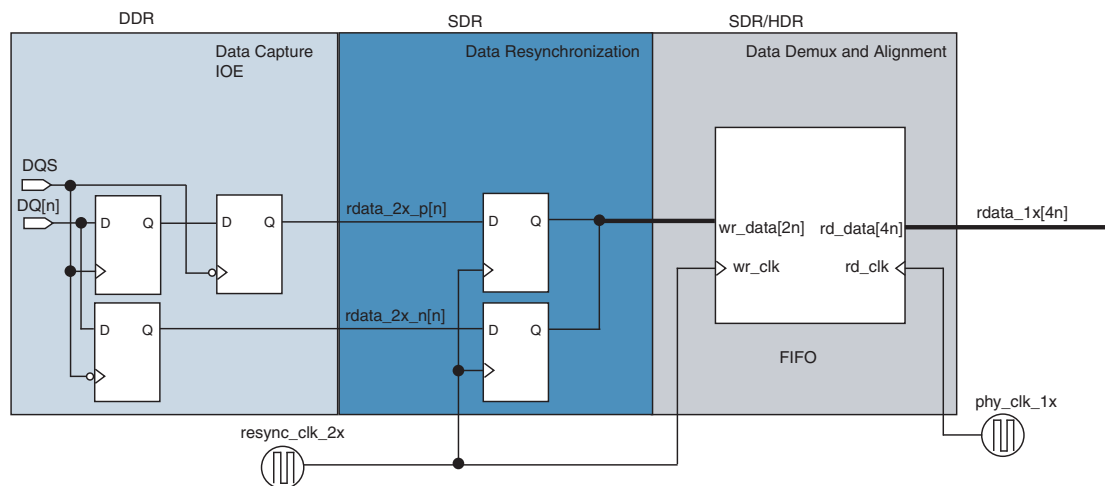
The full-rate datapath is similar to the half-rate datapath. The full-rate datapath also consists of a RAM with the same width as the data input (just like that of the half-rate), but the width on the data output of the RAM is half that of the half-rate PHY. The function of the RAM is to transfer the read data from the resynchronization clock domain to the system clock domain.

The read datapath logic is responsible for capturing data sent by the memory device and subsequently aligning the data back to the system clock domain. The following functions are performed by the read datapath:

1. Data capture and resynchronization
2. Data demultiplexing
3. Data alignment

Figure 5-5 shows the order of the functions performed by the read datapath, along with the frequency at which the read data is handled.

Figure 5-5. DDR/DDR2 SDRAM Read Datapath in Arria GX, Arria II GX, HardCopy II, Stratix II, and Stratix II GX Devices (Note 1)



Note to Figure 5-5:

(1) In Arria II GX devices the resynchronization register is implemented in IOE.

Data Capture and Resynchronization

Data capture and resynchronization is the process of capturing the read data (DQ) with the DQS strobe and re-synchronizing the captured data to an internal free-running full-rate clock supplied by the enhanced phase-locked loop (PLL).

The resynchronization clock is an intermediate clock whose phase shift is determined during the calibration stage.

Timing constraints ensure that the data resynchronization registers are placed close to the DQ pins to achieve maximum performance. Timing constraints also further limit skew across the DQ pins. The captured data (`rdata_2x_p` and `rdata_2x_n`) is synchronized to the resynchronization clock (`resync_clk_2x`), refer to [Figure 5-5](#).

Data Demultiplexing

Data demultiplexing is the process of SDR data into HDR data. Data demultiplexing is required to bring the frequency of the resynchronized data down to the frequency of the system clock, so that data from the external memory device can ultimately be brought into the FPGA DDR or DDR2 SDRAM controller clock domain. Before data capture, the data is DDR and n -bit wide. After data capture, the data is SDR and $2n$ -bit wide. After data demuxing, the data is HDR of width $4n$ -bits wide. The system clock frequency is half the frequency of the memory clock.

Demultiplexing is achieved using a dual-port memory with a $2n$ -bit wide write-port operating on the resynchronization clock (SDR) and a $4n$ -bit wide read-port operating on the PHY clock (HDR). The basic principle of operation is that data is written to the memory at the SDR rate and read from the memory at the HDR rate while incrementing the read- and write-address pointers. As the SDR and HDR clocks are generated, the read and write pointers are continuously incremented by the same PLL, and the $4n$ -bit wide read data follows the $2n$ -bit wide write data with a constant latency.

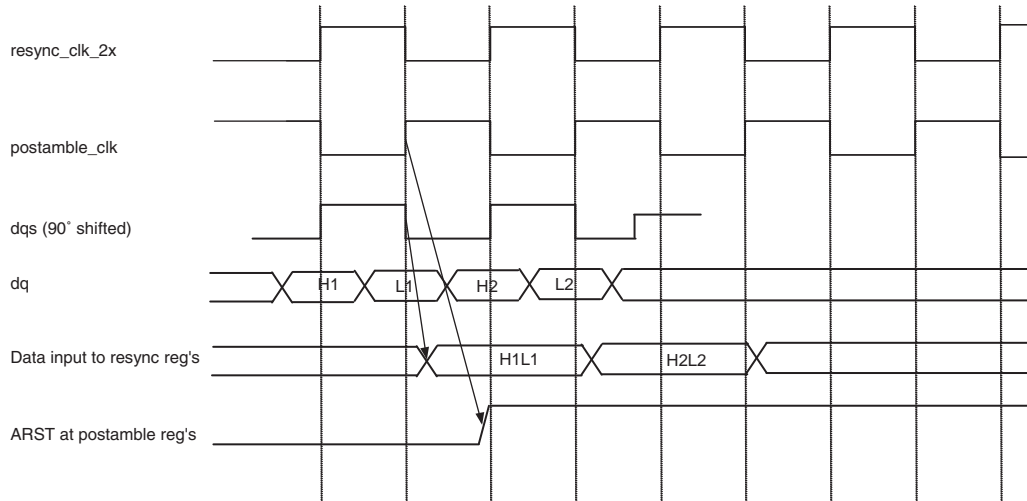
Read Data Alignment

Data alignment is the process controlled by the sequencer to ensure the correct captured read data is present in the same half-rate clock cycle at the output of the read data DPRAM. Data alignment is implemented using either M4K or M512K memory blocks. The bottom of [Figure 5-6](#) shows the concatenation of the read data into valid HDR data.

Postamble Protection

The ALTMEMPHY megafunction provides the DQS postamble logic. The postamble clock is derived from the resynchronization clock and is the negative edge of the resynchronization clock. The ALTMEMPHY megafunction calibrates the resynchronization clock such that it is in the center of the data-valid window. The clock that controls the postamble logic, the postamble clock, is the negative edge of the resynchronization clock. No additional clocks are required. Figure 5-6 shows the relationship between the postamble clock and the resynchronization clock.

Figure 5-6. Relationship Between Postamble Clock and Resynchronization Clock (Note 1)



Note to Figure 5-6:

(1) `resync_clk_2x` is delayed further to allow for the I/O element (IOE) to core transition time.

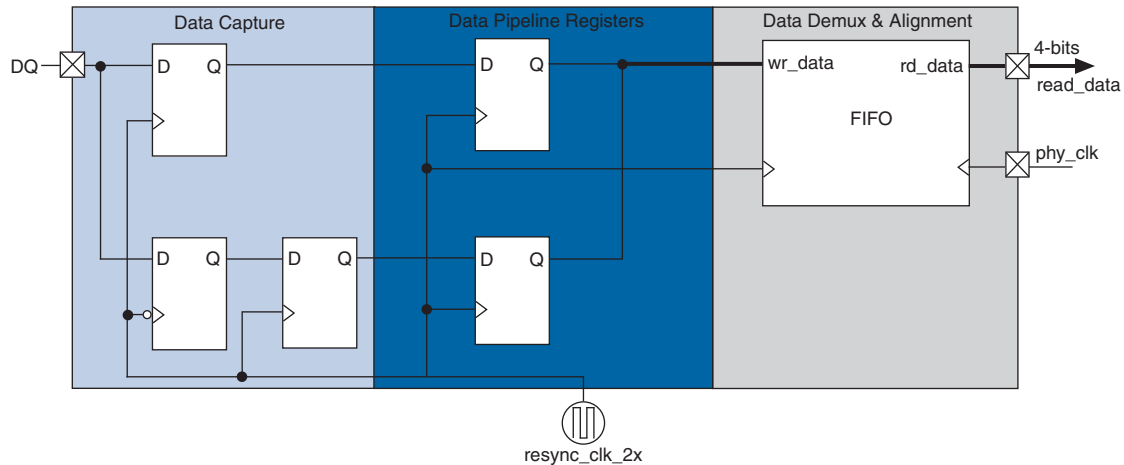


For more information about the postamble circuitry, refer to the *External Memory Interfaces* chapter in the *Stratix II Device Handbook*.

Cyclone III Devices

Figure 5-7 shows the Cyclone III read datapath for a single DQ pin. The diagram shows a half-rate read path where four data bits are produced for each DQ pin. Unlike Stratix II and Stratix III devices, data capture is entirely done in the core logic because the I/O element (IOE) does not contain DDIO capture registers (nonDQS capture).

Figure 5-7. Cyclone III Read Datapath



The full-rate read datapath for Cyclone III devices is similar to the half-rate Cyclone III implementation, except that the data is read out of the FIFO buffer with a full-rate clock instead of a half-rate clock.

Capture and Pipelining

The DDR and DDR2 SDRAM read data is captured using registers in the Cyclone III FPGA core. These capture registers are clocked using the capture clock (`resynch_clk_2x`). The captured read data generates two data bits per DQ pin; one data bit for the read data captured by the rising edge of the capture clock and one data bit for the read data captured by the falling edge of the capture clock.

After the read data has been captured, it may be necessary to insert registers in the read datapath between the capture registers and the read data FIFO buffer to help meet timing. These registers are known as pipeline registers and are clocked off the same clock used by the capture registers, the capture clock (`resynch_clk_2x`).

Data Demultiplexing

The data demultiplexing for Cyclone III devices is instantiated in the same way as it is with Stratix II devices.

Postamble Protection

Postamble protection circuitry is not required in the Cyclone III device implementation as DQS mode capture of the DQ data is not supported. The data capture is done using the clock (`resynch_clk_2x`) generated from the ALTPLL megafunction.

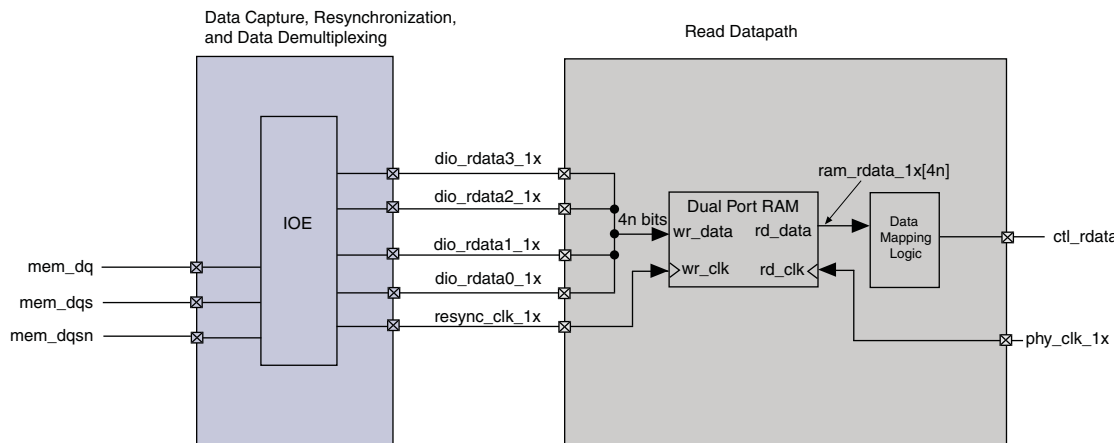
Stratix III and Stratix IV Devices

Stratix IV and Stratix III devices support half-rate or full-rate DDR/DDR2 SDRAM.

The Stratix IV and Stratix III read datapath (Figure 5-8) consists of two main blocks:

- Data capture, resynchronization, and demultiplexing
- Read datapath logic (read datapath)

Figure 5-8. DDR/DDR2 SDRAM Data Capture and Read Data Mapping in Stratix IV and Stratix III Devices



Note to Figure 5-8:

(1) This figure shows a half-rate variation. For a full-rate controller, dio_rdata2_1x and dio_rdata3_1x are unconnected.

Data Capture, Resynchronization, and Demultiplexing

In Stratix IV and Stratix III devices, the smart interface module in the IOE performs the following tasks:

- Captures the data
- Resynchronizes the captured data from the DQS domain to the resynchronization clock (resync_clk_1x) domain
- Converts the resynchronized data into half-rate data, which is performed by feeding the resynchronized data into the HDR conversion block within the IOE, which is clocked by the half-rate version of the resynchronization clock. The resync_clk_1x signal is generated from the I/O clock divider module based on the resync_clk_2x signal from the PLL.

For more information about IOE registers, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

Data Resynchronization

The read datapath block performs the following two tasks:

1. Transfers the captured read data (`rdata[n]_1x`) from the half-rate resynchronization clock (`resync_clk_1x`) domain to the half-rate system clock (`phy_clk_1x`) domain using DPRAM. Resynchronized data from the FIFO buffer is shown as `ram_data_1x`.
2. Reorders the resynchronized data (`ram_rdata_1x`) into `ctl_mem_rdata`.

The full-rate datapath is similar to the half-rate datapath, except that the resynchronization FIFO buffer converts from the full-rate resynchronization clock domain (`resync_clk_2x`) to the full-rate PHY clock domain, instead of converting it to the half-rate PHY clock domain as in half-rate designs.

Postamble Protection

A dedicated postamble register controls the gating of the shifted DQS signal that clocks the DQ input registers at the end of a read operation. This ensures that any glitches on the DQS input signals at the end of the read postamble time do not cause erroneous data to be captured as a result of postamble glitches. The postamble path is also calibrated to determine the correct clock cycle, clock phase shift, and delay chain settings. You can see the process in simulation if you choose Full calibration (long simulation time) mode in the MegaWizard Plug-In Manager.



For more information about the postamble protection circuitry, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

Write Datapath

This topic describes the write datapath.

Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices

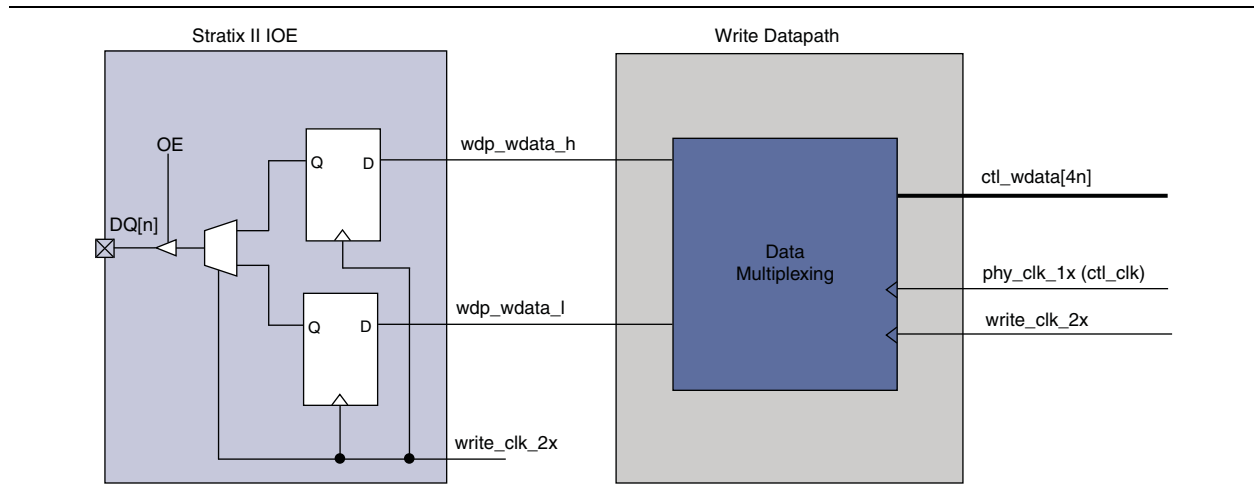
The write datapath logic efficiently transfers data from the HDR memory controller to DDR SDRAM-based memories. The write datapath logic consists of:

- DQ and DQ output-enable logic
- DQS and DQS output-enable logic
- Data mask (DM) logic

The memory controller interface outputs $4n$ -bit wide data (`ctl_wdata[4n]`) at half-rate frequency. [Figure 5-9](#) shows that the HDR write data (`ctl_wdata[4n]`) is clocked by the half-rate clock `phy_clk_1x` and is converted into SDR which is represented by `wdp_wdata_h` and `wdp_wdata_l` and clocked by the full-rate clock `write_clk_2x`.

The DQ IOEs convert 2- n SDR bits to n -DDR bits.

Figure 5–9. DDR/DDR2 SDRAM Write Datapath in Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices

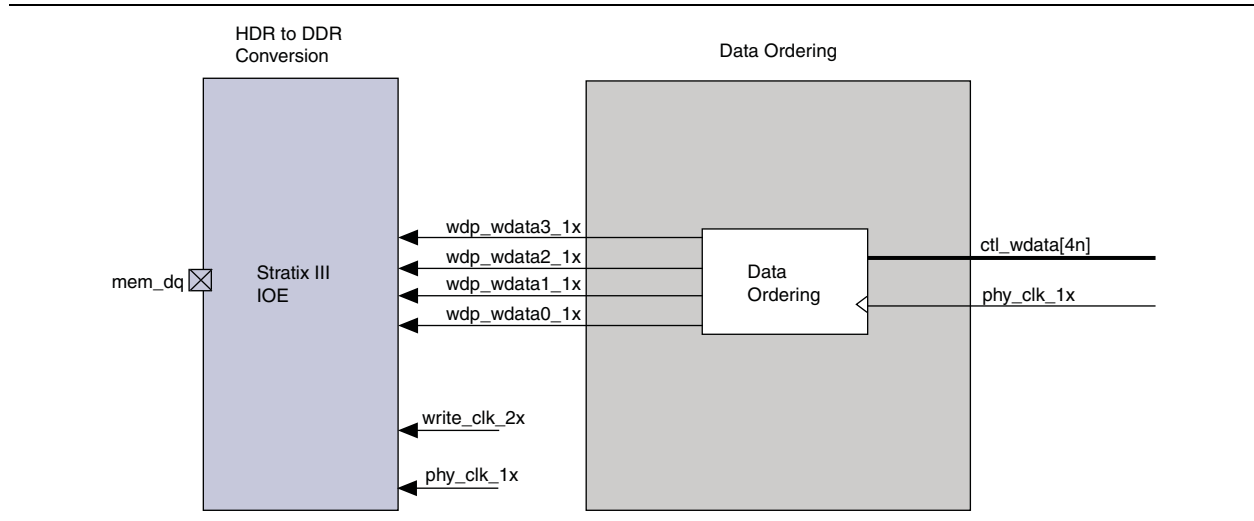


The write datapath for full-rate PHYs is similar to the half-rate PHY. The IOE block is identical to the half-rate PHY. The latency of the write datapath in the full-rate PHY is less than in the half-rate PHY because the full-rate PHY does not have the half-rate-to-full-rate conversion logic.


Stratix III and Stratix IV Devices

The memory controller interface outputs 4 n -bit wide data (ctl_wdata) at phy_clk_1x frequency. The write data is clocked by the system clock phy_clk_1x at half data rate and reordered into HDR of width 4 n -bits represented in [Figure 5–10](#) by wdp_wdata3_1x , wdp_wdata2_1x , wdp_wdata1_1x , and wdp_wdata0_1x .


Figure 5–10. DDR and DDR2 SDRAM Write Datapath in Stratix IV and Stratix III Devices



All of the write datapath registers in the Stratix IV and Stratix III devices are clocked by the half-rate clock, phy_clk_1x .

 For full-rate controllers, `phy_clk_1x` runs at full rate and there are only two bits of `wdata`.


The write datapath for full-rate PHYs is similar to the half-rate PHY. The IOE block is identical to the half-rate PHY. The latency of the write datapath in the full-rate PHY is less than in the half-rate PHY because the full-rate PHY does not have half-rate to full-rate conversion logic.

 For more information about the Stratix III I/O structure, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

ALTMEMPHY Signals

This section describes the ALMEMPHY megafunction ports for AFI variants.

Table 5-5 through Table 5-7 show the signals.

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

The signal lists include the following signal groups:

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

Table 5-5. Interface to the SDRAM Devices (Part 1 of 2) (Note 1)

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

Table 5–5. Interface to the SDRAM Devices (Part 2 of 2) (Note 1)

Signal Name	Type	Width (2)	Description
mem_dqs	Bidirectional	MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS	The memory bidirectional data strobe bus.
mem_dqsn	Bidirectional	MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS	The memory bidirectional data strobe bus.
mem_odt	Output	MEM_IF_CS_WIDTH	The memory on-die termination control signal.
mem_ras_n	Output	1	The memory row address strobe.
mem_reset_n	Output	1	The memory reset signal. This signal is derived from the PHY's internal reset signal, which is generated by gating the global reset, soft reset, and the PLL locked signal.
mem_we_n	Output	1	The memory write enable signal.

Notes to Table 5–5:

- (1) Connected to I/O pads.
- (2) Refer to Table 5–8 for parameter description.
- (3) Output is for memory device, and input path is fed back to ALTMEMPHY megafunction for VT tracking.

Table 5–6. AFI Signals (Part 1 of 4)

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

Table 5-6. AFI Signals (Part 2 of 4)

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

Table 5-6. AFI Signals (Part 3 of 4)

Signal Name	Type	Width (1)	Description
Read Data Interface			
ctl_doing_rd	Input	$\text{MEM_IF_DQS_WIDTH} \times \text{DWIDTH_RATIO} / 2$	Doing read input. Indicates that the DDR or DDR2 SDRAM controller is currently performing a read operation. The controller generates <code>ctl_doing_rd</code> to the ALTMEMPHY megafunction. The <code>ctl_doing_rd</code> signal is asserted for one <code>phy_clk</code> cycle for every read command it issues. If there are two read commands, <code>ctl_doing_rd</code> is asserted for two <code>phy_clk</code> cycles. The <code>ctl_doing_rd</code> signal also enables the capture registers and generates the <code>ctl_mem_rdata_valid</code> signal. The <code>ctl_doing_rd</code> signal should be issued at the same time the read command is sent to the ALTMEMPHY megafunction.
ctl_rdata	Output	$\text{DWIDTH_RATIO} \times \text{MEM_IF_DWIDTH}$	Read data from the PHY to the controller.
ctl_rdata_valid	Output	$\text{DWIDTH_RATIO} / 2$	Read data valid indicating valid read data on <code>ctl_rdata</code> . This signal is two-bits wide (as only half-rate or $\text{DWIDTH_RATIO} = 4$ is supported) to allow controllers to issue reads and writes that are aligned to either the half-cycle of the half-rate clock.
ctl_rlat	Output	READ_LAT_WIDTH	Contains the number of clock cycles between the assertion of <code>ctl_doing_rd</code> and the return of valid read data (<code>ctl_rdata</code>). This is unused by the Altera high-performance controllers do not use <code>ctl_rlat</code> .
Address and Command Interface			
ctl_addr	Input	$\text{MEM_IF_ROWADDR_WIDTH} \times \text{DWIDTH_RATIO} / 2$	Row address from the controller.
ctl_ba	Input	$\text{MEM_IF_BANKADDR_WIDTH} \times \text{DWIDTH_RATIO} / 2$	Bank address from the controller.
ctl_cke	Input	$\text{MEM_IF_CS_WIDTH} \times \text{DWIDTH_RATIO} / 2$	Clock enable from the controller.
ctl_cs_n	Input	$\text{MEM_IF_CS_WIDTH} \times \text{DWIDTH_RATIO} / 2$	Chip select from the controller.
ctl_odt	Input	$\text{MEM_IF_CS_WIDTH} \times \text{DWIDTH_RATIO} / 2$	On-die-termination control from the controller.
ctl_ras_n	Input	$\text{DWIDTH_RATIO} / 2$	Row address strobe signal from the controller.
ctl_we_n	Input	$\text{DWIDTH_RATIO} / 2$	Write enable.
ctl_cas_n	Input	$\text{DWIDTH_RATIO} / 2$	Column address strobe signal from the controller.
ctl_rst_n	Input	$\text{DWIDTH_RATIO} / 2$	Reset from the controller.
Calibration Control and Status Interface			
ctl_mem_clk_disable	Input	$\text{MEM_IF_CLK_PAIR_COUNT}$	When asserted, <code>mem_clk</code> and <code>mem_clk_n</code> are disabled. Not supported for Cyclone III devices.
ctl_cal_success	Output	1	A 1 indicates that calibration was successful.
ctl_cal_fail	Output	1	A 1 indicates that calibration has failed.

Table 5-6. AFI Signals (Part 4 of 4)

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

Note to Table 5-5:

(1) Refer to Table 5-8 for parameter descriptions.

Table 5-7. Other Interface Signals (Part 1 of 3)

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

Table 5-7. Other Interface Signals (Part 2 of 3)

Signal Name	Type	Width	Description
PLL Reconfiguration Signals—Stratix III and Stratix IV Devices			
pll_reconfig_enable	Input	1	This signal enables the PLL reconfiguration I/O, and is used if the user requires some custom PLL phase reconfiguration. It should otherwise be tied low.
pll_phasecounterselect	Input	4	When <code>pll_reconfig_enable</code> is asserted, this input is directly connected to the PLL's <code>phasecounterselect</code> input. Otherwise this input has no effect.
pll_phaseupdown	Input	1	When <code>pll_reconfig_enable</code> is asserted, this input is directly connected to the PLL's <code>phaseupdown</code> input. Otherwise this input has no effect.
pll_phasestep	Input	1	When <code>pll_reconfig_enable</code> is asserted, this input is directly connected to the PLL's <code>phasestep</code> input. Otherwise this input has no effect.
pll_phase_done	Output	1	Directly connected to the PLL's <code>phase_done</code> output.
PLL Reconfiguration Signals—Stratix II Devices			
pll_reconfig_enable	Input	1	Allows access to the PLL reconfiguration block. This signal should be held low in normal operation. While the PHY is held in reset (with <code>soft_reset_n</code>), and <code>reset_request_n</code> is 1, it is safe to reconfigure the PLL. To reconfigure the PLL, set this signal to 1 and use the other <code>pll_reconfig</code> signals to access the PLL. When finished reconfiguring set this signal to 0, and then set the <code>soft_reset_n</code> signal to 1 to bring the PHY out of reset. For this signal to work, the <code>PLL_RECONFIG_PORTS_EN</code> GUI parameter must be set to TRUE.
pll_reconfig_write_param	Input	9	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig_read_param	Input	9	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig	Input	1	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig_counter_type	Input	4	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig_counter_param	Input	3	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig_data_in	Input	9	Refer to the ALTPLL_RECONFIG User Guide for more information.
pll_reconfig_busy	Output	1	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig_data_out	Output	9	Refer to the ALTPLL_RECONFIG User Guide , for more information.
pll_reconfig_clk	Output	1	Synchronous clock to use for any logic accessing the <code>pll_reconfig</code> interface. The same as <code>aux_scan_clk</code> .
pll_reconfig_reset	Output	1	Resynchronised reset to use for any logic accessing the <code>pll_reconfig</code> interface.

Table 5-7. Other Interface Signals (Part 3 of 3)

Signal Name	Type	Width	Description
Calibration Interface Signals—without leveling only			
rsu_codvw_phase	Output	—	The sequencer sweeps the phase of a resynchronization clock across 360° or 720° of a memory clock cycle. Data reads from the DIMM are performed for each phase position, and a data valid window is located, which is the set of resynchronization clock phase positions where data is successfully read. The final resynchronization clock phase is set at the center of this range: the center of the data valid window or CODVW. This output is set to the current calculated value for the CODVW, and represents how many phase steps were performed by the PLL to offset the resynchronization clock from the memory clock.
rsu_codvw_size	Output	—	The final centre of data valid window size (<i>rsu_codvw_size</i>) is the number of phases where data was successfully read in the calculation of the resynchronization clock centre of data valid window phase (<i>rsu_codvw_phase</i>).
rsu_read_latency	Output	—	The <i>rsu_read_latency</i> output is then set to the read latency (in <i>phy_clk</i> cycles) using the <i>rsu_codvw_phase</i> resynchronization clock phase. If calibration is unsuccessful then this signal is undefined.
rsu_no_dvw_err	Output	—	If the sequencer sweeps the resynchronization clock across every phase and does not see any valid data at any phase position, then calibration fails and this output is set to 1.
rsu_grt_one_dvw_err	Output	—	If the sequencer sweeps the resynchronization clock across every phase and sees multiple data valid windows, this is indicative of unexpected read data (random bit errors) or an incorrectly configured PLL that must be resolved. Calibration has failed and this output is set to 1.
Unused Signals			
hc_scan_enable_access	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_enable_dq	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_enable_dm	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_enable_dqs	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_enable_dqs_config	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_din	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_update	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_ck	Input	—	This signal is unused; you should connect this signal to logic level 0.
hc_scan_dout	Output	—	This signal is unused; you should connect this signal to logic level 0.
mem_err_out_n	Input	—	This signal is unused; you should connect this signal to logic level 1.

Notes to Table 5-7:

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

Table 5–8 shows the parameters that Table 5–5 through Table 5–7 refer to.

Table 5–8. Parameters

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

PHY-to-Controller Interfaces

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

The AFI standardizes and simplifies the interface between controller and PHY for all Altera memory designs, thus allowing you to easily interchange your own controller code with Altera's high-performance controllers. The AFI includes an administration block that configures the memory for calibration and performs necessary mode registers accesses to configure the memory as required (these calibration processes are different). Figure 5–11 shows an overview of the connections between the PHY, the controller, and the memory device.


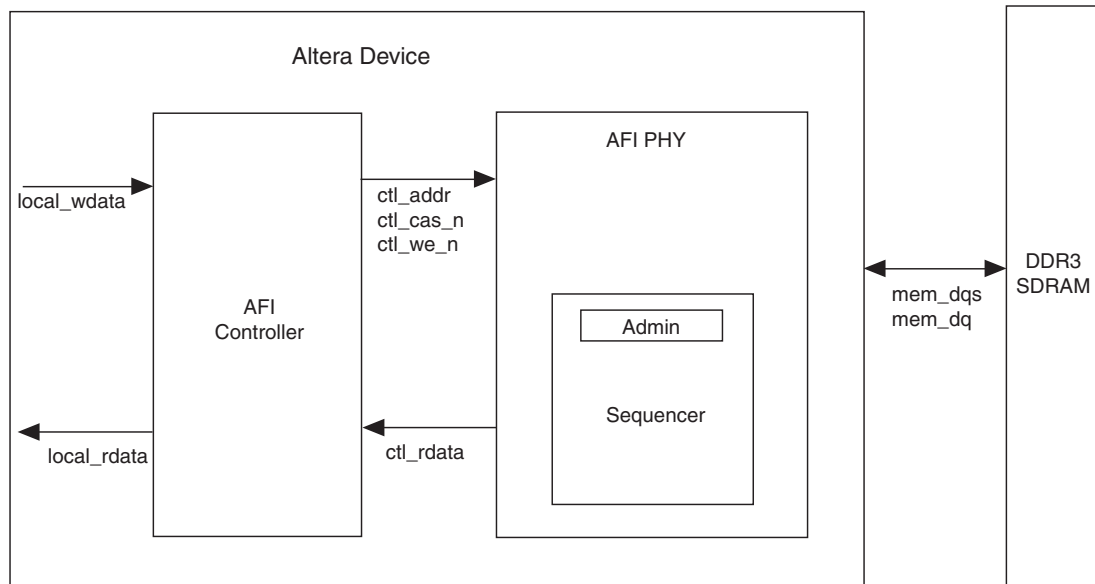
 Altera recommends that you use the AFI for new designs.

Figure 5–11. AFI PHY Connections



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

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

For half-rate designs, the AFI allows the controller to issue reads and writes that are aligned to either half-cycle of the half-rate `phy_clk`, which means that the datapaths can support multiple data alignments—word-unaligned and word-aligned writes and reads. Figure 5-12 and Figure 5-13 display the half-rate write operation.

Figure 5-12. Half-Rate Write with Word-Unaligned Data

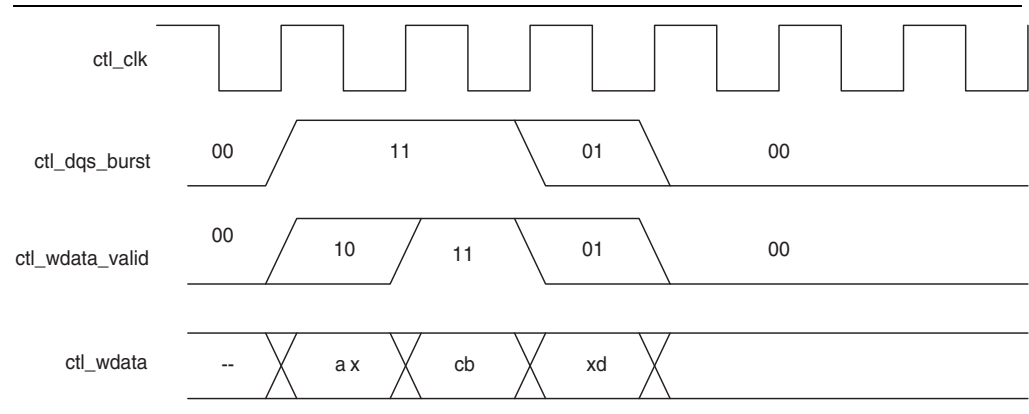


Figure 5-13. Half-Rate Write with Word-Aligned Data

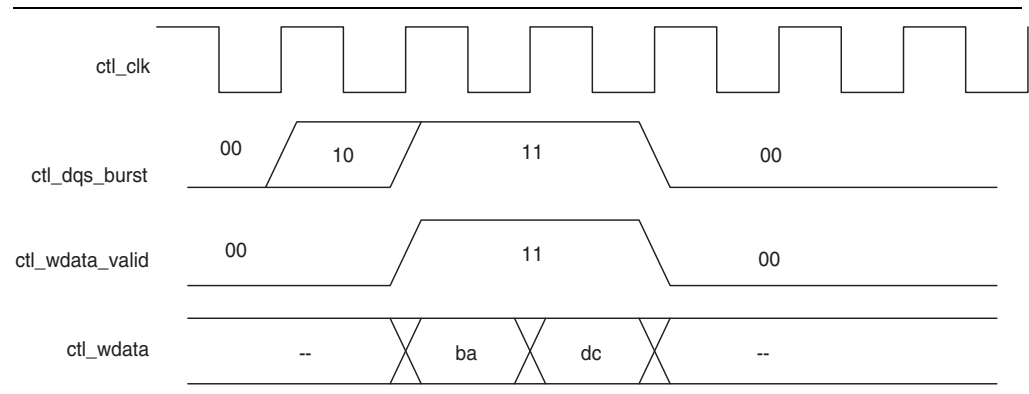
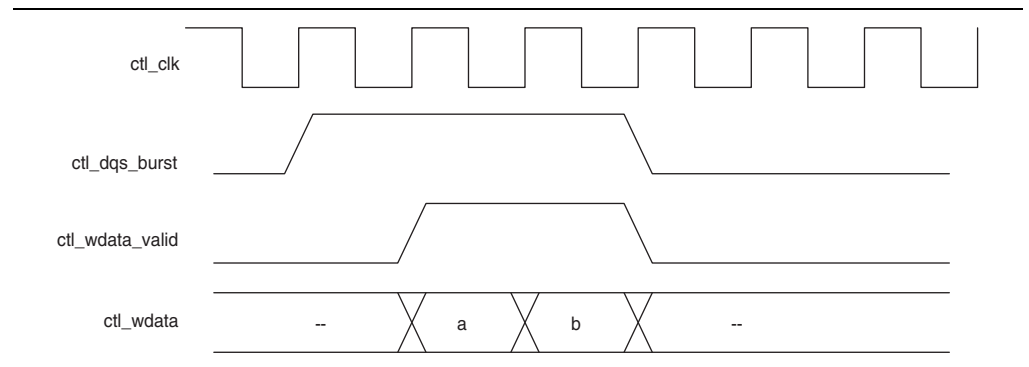


Figure 5-14 shows a full-rate write.

Figure 5-14. Full-Rate Write



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

Figure 5-15 shows full-rate reads; Figure 5-16 shows half-rate reads.

Figure 5-15. Full-Rate Reads

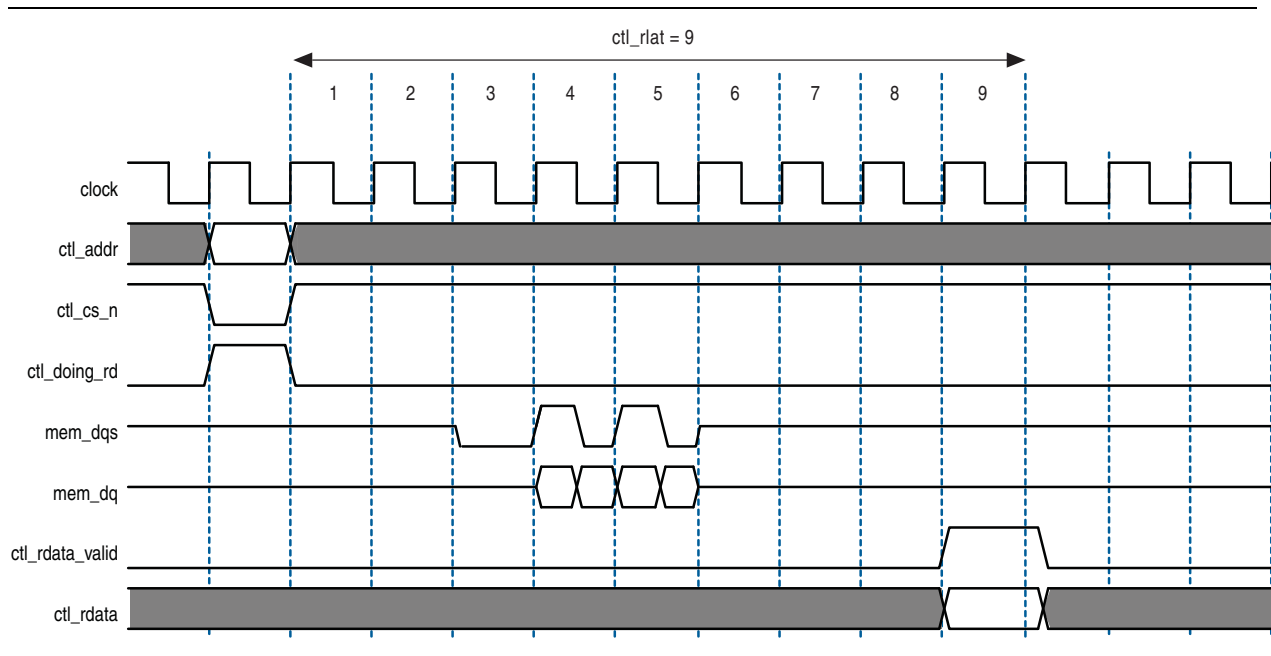


Figure 5-16. Half-Rate Reads

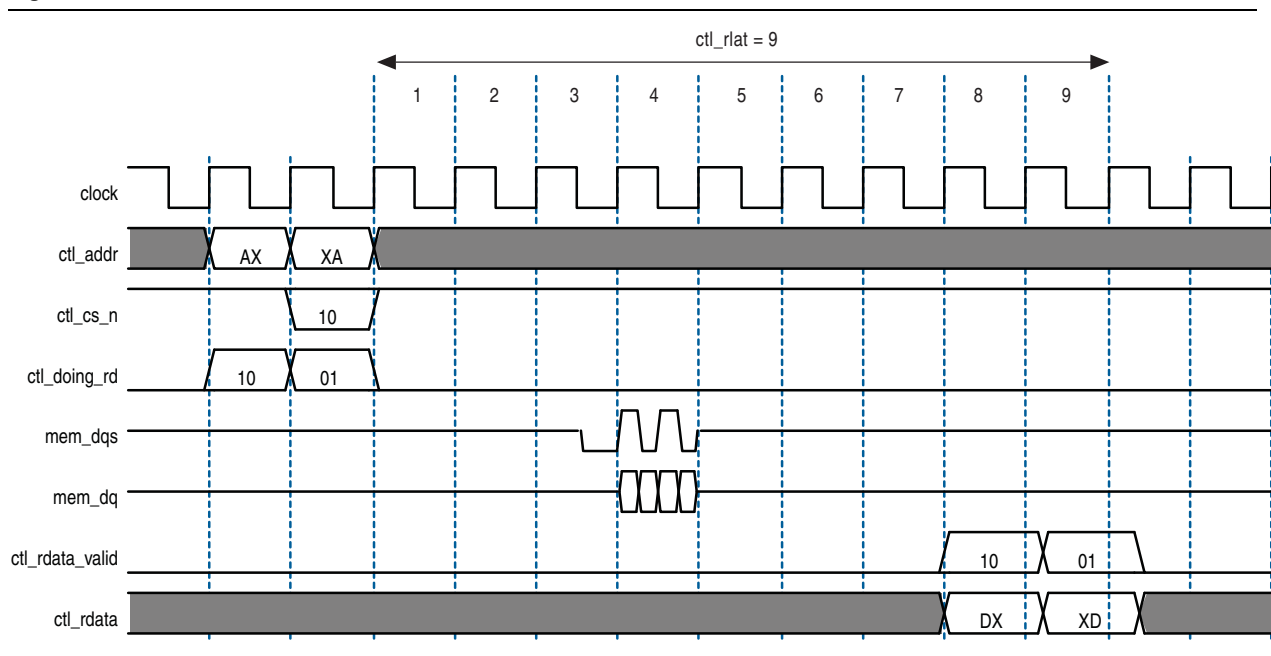


Figure 5-17 and Figure 5-18 show word-aligned writes and reads. In the following read and write examples the data is written to and read from the same address. In each example, **ctl_rdata** and **ctl_wdata** are aligned with controller clock (**ctl_clk**) cycles. All the data in the bit vector is valid at once. For comparison, refer Figure 5-19 and Figure 5-20 that show the word-unaligned writes and reads.



The `ctl_doing_rd` is represented as a half-rate signal when passed into the PHY. Therefore, the lower half of this bit vector represents one memory clock cycle and the upper half the next memory clock cycle. [Figure 5-20 on page 5-42](#) shows separated word-unaligned reads as an example of two `ctl_doing_rd` bits are different. Therefore, for each x16 device, at least two `ctl_doing_rd` bits need to be driven, and two `ctl_rdata_valid` bits need to be interpreted.

The AFI has the following conventions:

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



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

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



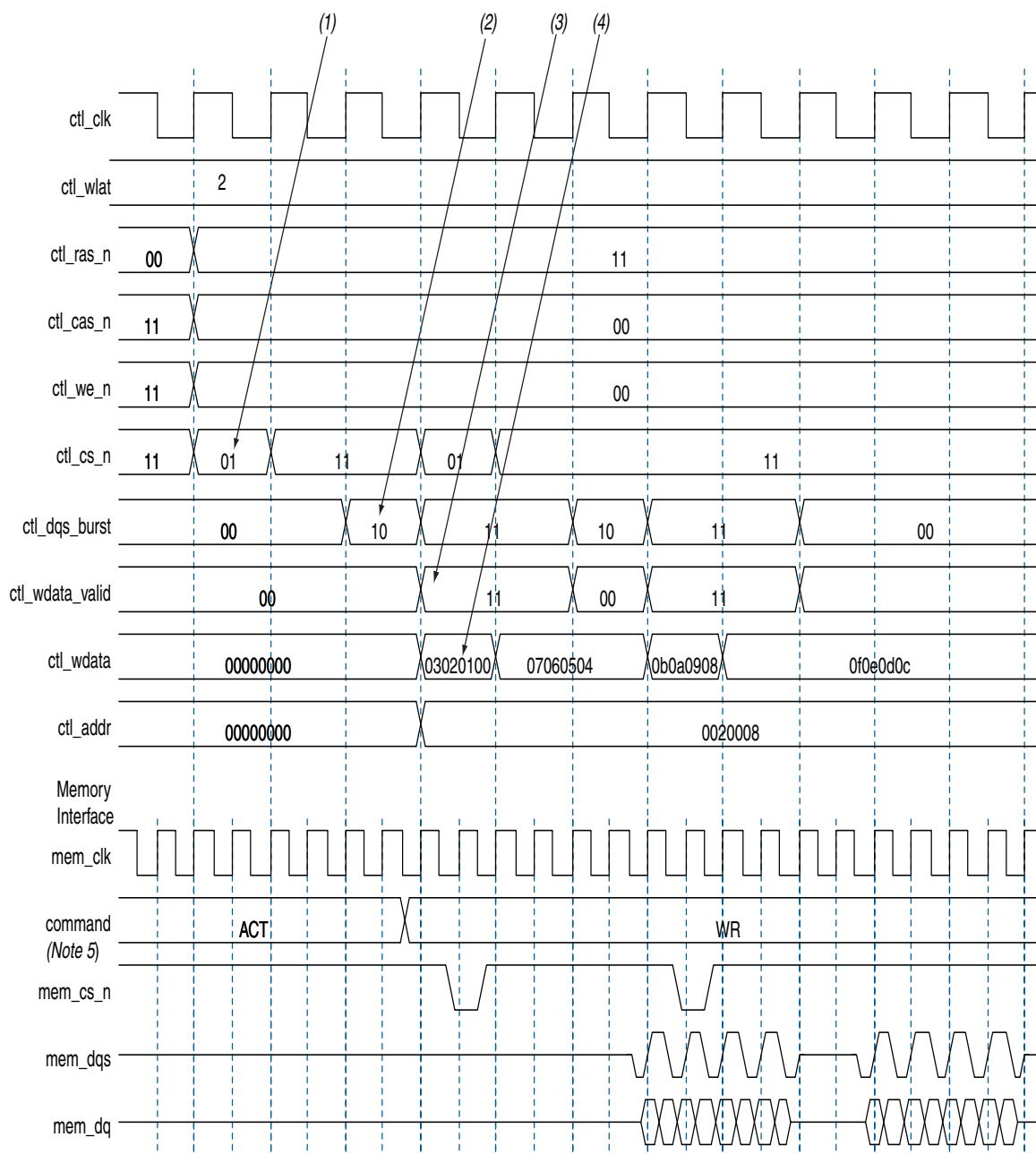
The high-performance controller II uses word-aligned data only. The timing analysis script does not support word-unaligned reads and writes. Word-unaligned reads and writes are only supported on Stratix III and Stratix IV devices.

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

[Figure 5-17](#) through [Figure 5-20](#) assume the following general points:

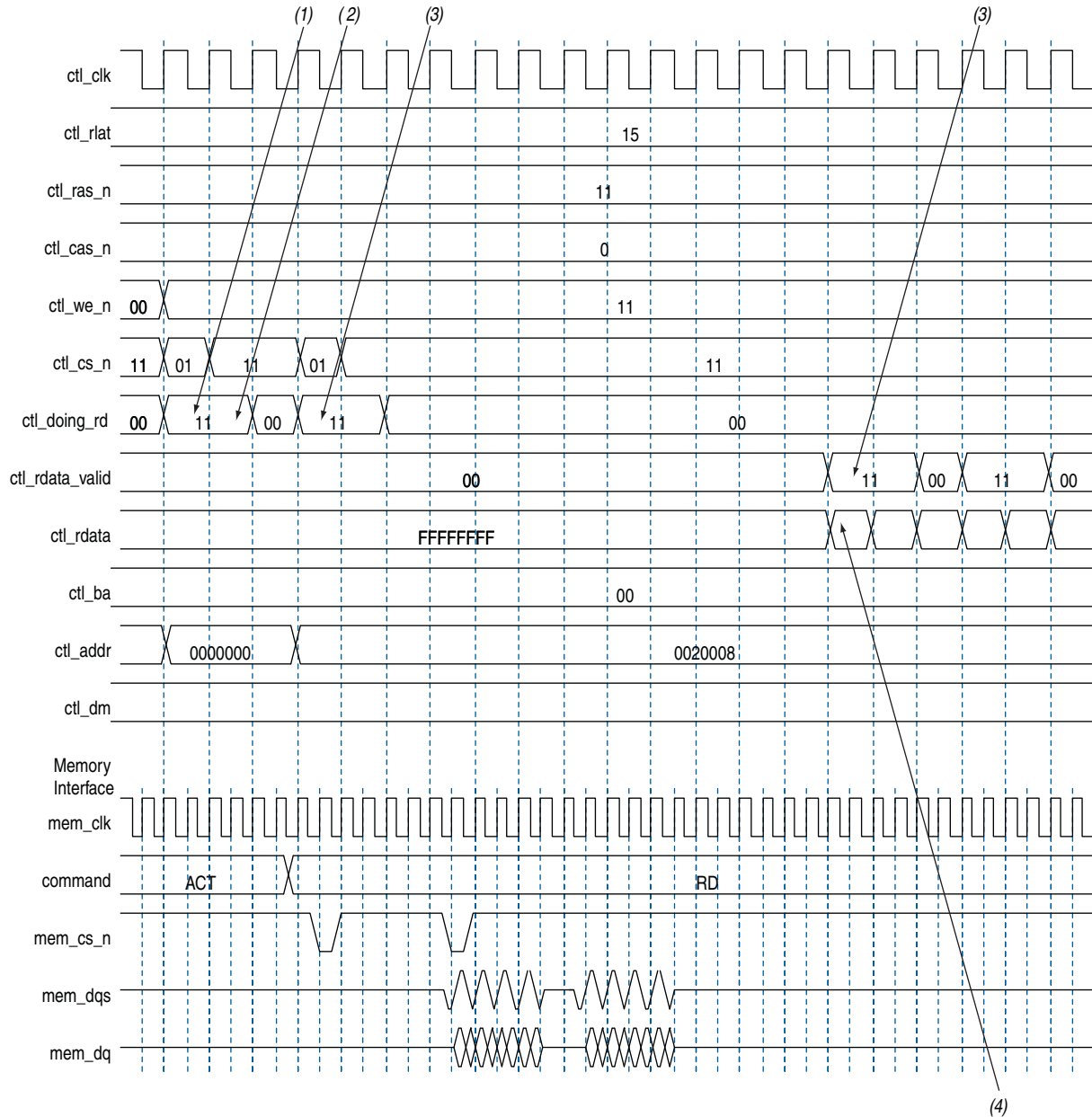
- The burst length is four. A DDR2 SDRAM is used—the interface timing is identical for DDR3 devices.
- An 8-bit interface with one chip select.
- The data for one controller clock (`ctl_clk`) cycle represents data for two memory clock (`mem_clk`) cycles (half-rate interface).

Figure 5-17. Word-Aligned Writes



Notes to Figure 5-17:

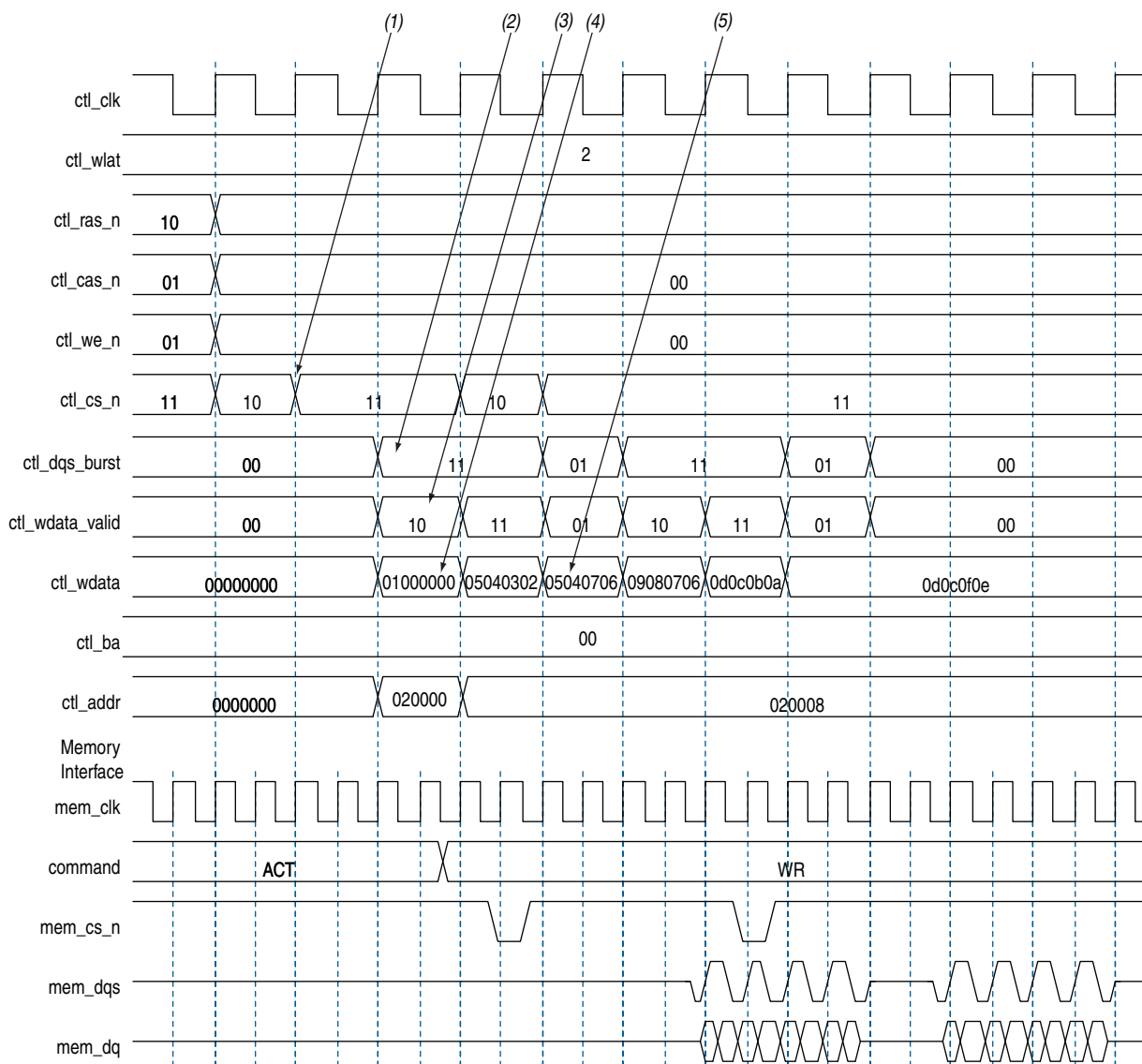
- (1) To show the even alignment of **ctl_cs_n**, expand the signal (this convention applies for all other signals).
- (2) The **ctl_dqs_burst** must go high one memory clock cycle before **ctl_wdata_valid**. Compare with the word-unaligned case.
- (3) The **ctl_wdata_valid** is asserted two **ctl_wlat**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

Figure 5-18. Word-Aligned Reads**Notes to Figure 5-18:**

- (1) For AFI, **ctl_doing_rd** is required to be asserted one memory clock cycle before chip select (**ctl_cs_n**) is asserted. In the half-rate **ctl_clk** domain, this requirement manifests as the controller driving 11 (as opposed to the 01) on **ctl_doing_rd**.
- (2) AFI requires that **ctl_doing_rd** is driven for the duration of the read. In this example, it is driven to 11 for two half-rate **ctl_clks**, which equates to driving to 1, for the four memory clock cycles of this four-beat burst.
- (3) The **ctl_rdata_valid** returns 15 (**ctl_rlat**) controller clock (**ctl_clk**) cycles after **ctl_doing_rd** is asserted. Returned is when the **ctl_rdata_valid** signal is observed at the output of a register within the controller. A controller can use the **ctl_rlat** value to determine when to register to returned data, but this is unnecessary as the **ctl_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.

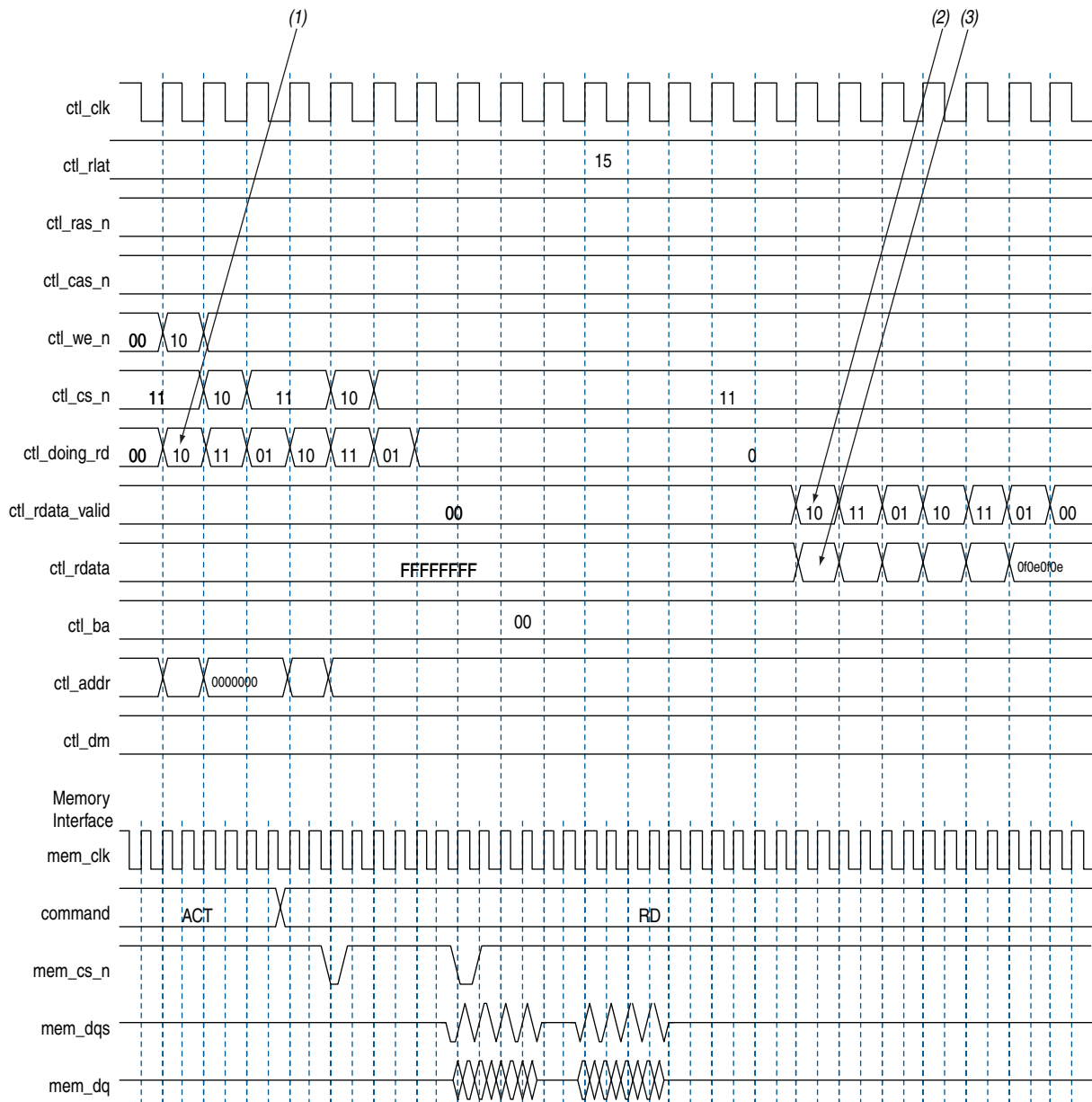
Figure 5-19 and Figure 5-20 show spaced word-unaligned writes and reads.

Figure 5-19. Word-Unaligned Writes



Notes to Figure 5-19:

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

Figure 5–20. Word-Unaligned Reads**Notes to Figure 5–20:**

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

Using a Custom Controller

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

Preliminary Steps

Perform the following steps to generate the ALTMEMPHY megafunction:

1. If you are creating a custom DDR or DDR2 SDRAM controller, generate the Altera DDR or DDR2 SDRAM Controller with ALTMEMPHY IP targeting your chosen Altera memory devices.
2. Compile and verify the timing. This step is optional; refer to [“Compiling and Simulating” on page 4-1](#).
3. If targeting a DDR or DDR2 SDRAM device, simulate the high-performance controller design.
4. Integrate the top-level ALTMEMPHY design with your controller. If you started with the high-performance controller, the PHY variation name is `<controller_name>_phy.v/.vhd`. Details about integrating your controller with Altera’s ALTMEMPHY megafunction are described in the following sections.
5. Compile and simulate the whole interface to ensure that you are driving the PHY properly and that your commands are recognized by the memory device.

Design Considerations

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



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

Clocks and Resets

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

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

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

Calibration Process Requirements

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



You can ignore the following two warning and critical warning messages:

Warning: Timing Analysis for multiple chip select DDR/DDR2/DDR3-SDRAM configurations is preliminary (memory interface has a chip select width of 4)

Critical Warning: Read Capture and Write timing analyses may not be valid due to violated timing model assumptions

Other Local Interface Requirements

The memory burst length can be two, four, or eight for DDR SDRAM devices, and four or eight for DDR2 SDRAM devices. For a half-rate controller, the memory clock runs twice as fast as the clock provided to the local interface, so data buses on the local interface are four times as wide as the memory data bus. For a full-rate controller, the memory clock runs at the same speed as the clock provided to the local interface, so the data buses on the local interface are two times as wide as the memory data bus.

This section describes the DDR or DDR2 SDRAM high-performance controller II with the AFI.

Address and Command Interfacing

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



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

Handshake Mechanism Between Read Commands and Read Data

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

- Control of the postamble circuit

- Generation of `ctl_rdata_valid`
- Dynamic termination (R_t) control timing

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

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

Figure 5-21 and Figure 5-22 show these relationships.

Figure 5-21. Address and Command and Read-Path Timing—Full-Rate Design

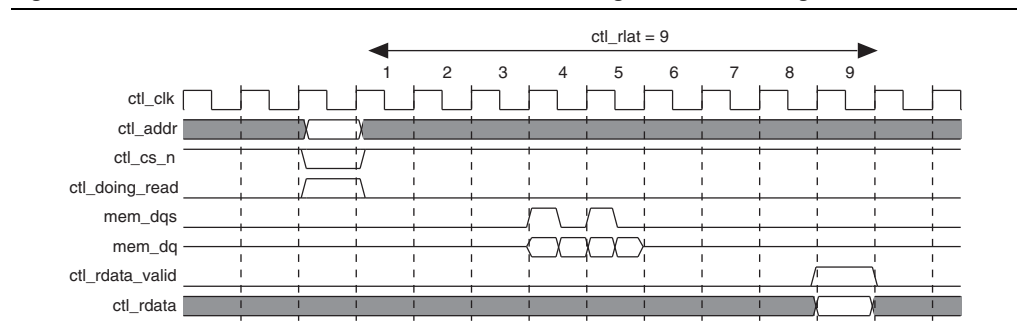
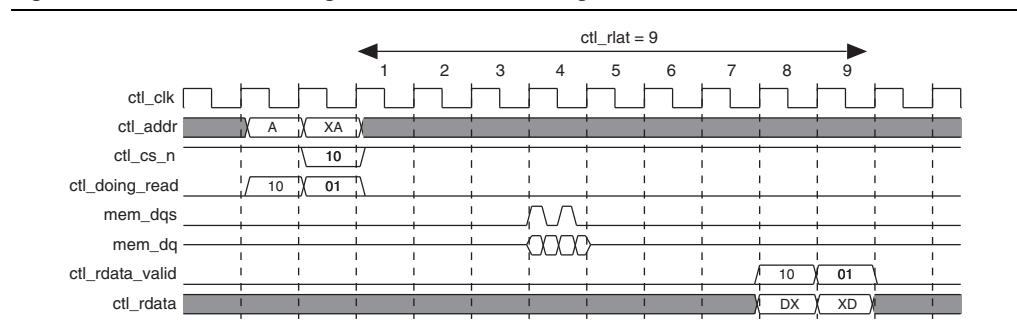


Figure 5-22. Second Read Alignment—Half-Rate Design




Handshake Mechanism Between Write Commands and Write Data

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

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

3. In the **Compilation Process Settings** dialog box, click **More Settings**.
4. In the **More Compilation Process Settings** dialog box, under **Existing option settings**, set **Display entity name for node name** to **On**.

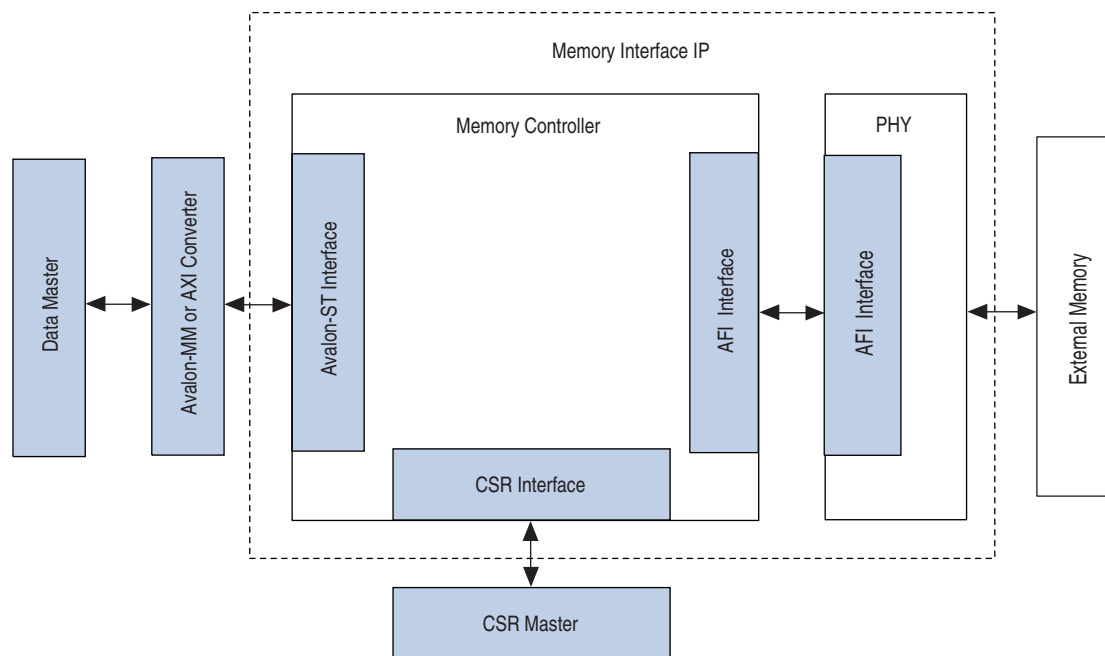
 This chapter describes the High Performance Controller II (HPC II) with advanced features introduced in version 11.0 for designs generated in version 11.0. Designs created in earlier versions and regenerated in version 11.0 do not inherit the new advanced features; for information on HPC II without the 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 6–1 shows a high-level block diagram of the overall memory interface architecture.

Figure 6–1. High-Level Diagram of Memory Interface Architecture

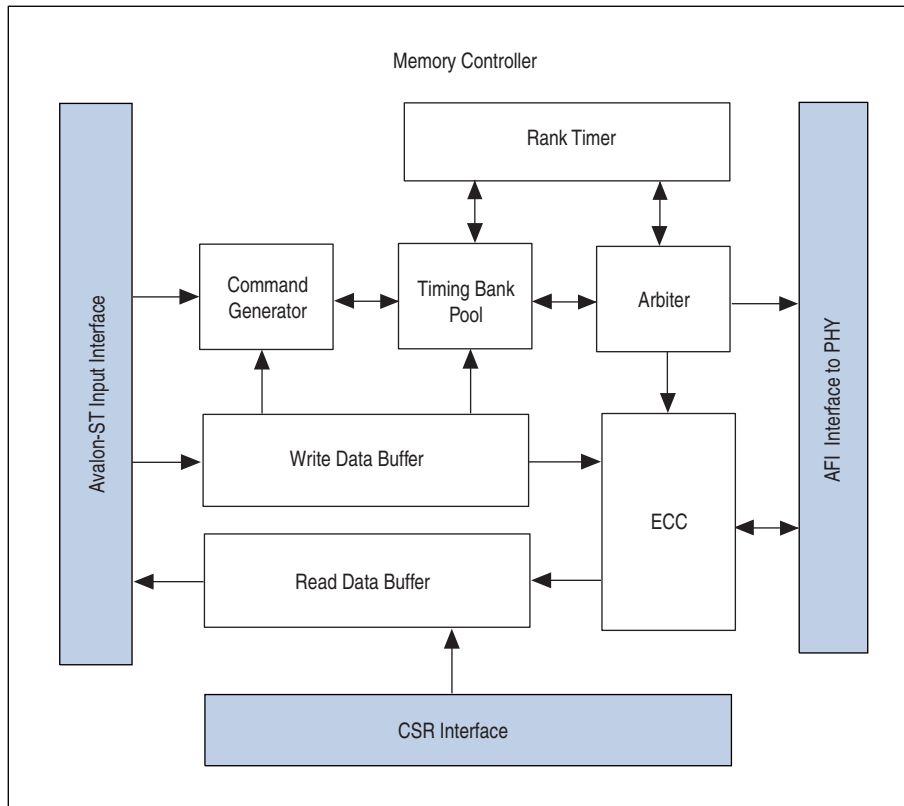


The memory interface consists of the memory controller logic block, the physical (PHY) logic layer, and their associated interfaces.

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). The controller supports an Avalon Memory Mapped (Avalon-MM) bus protocol.

Figure 6–2 shows a block diagram of the memory controller architecture.

Figure 6–2. Memory Controller Architecture Block Diagram



The following sections describe the blocks in Figure 6–2.

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 with the memory device, through the physical layer logic (PHY).

For more information about AFI signals, refer to [“AFI Signals” on page 5–28](#).

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_powerdown_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 6–1 shows which ODT signal is enabled for single-slot single chip-select per DIMM.



There is no ODT for reads.

Table 6–1. ODT—DDR2 SDRAM Single Slot Single Chip-select Per DIMM (Write)

Write On	ODT Enabled
mem_cs [0]	mem_odt [0]

Table 6–2 shows which ODT signal is enabled for single-slot dual chip-select per DIMM.



There is no ODT for reads.

Table 6–2. 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 6–3 shows which ODT signal is enabled for dual-slot single chip-select per DIMM.

Table 6–3. 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 6–4 shows which ODT signal is enabled for dual-slot dual chip-select per DIMM.

Table 6–4. 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]

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 `local_rdata_error` signal to indicate that the data is incorrect. The `local_rdata_error` signal follows the same timing as the `local_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 6-3 and Figure 6-4 show partial write operations for the controller, for full and half rate configurations, respectively.

Figure 6-3. Partial Write for the Controller—Full Rate

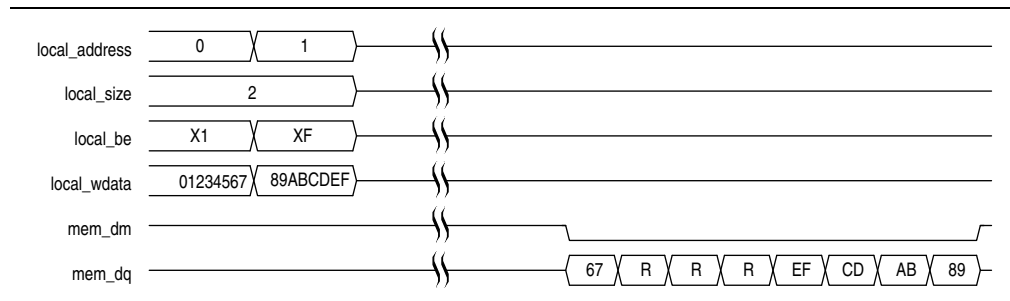
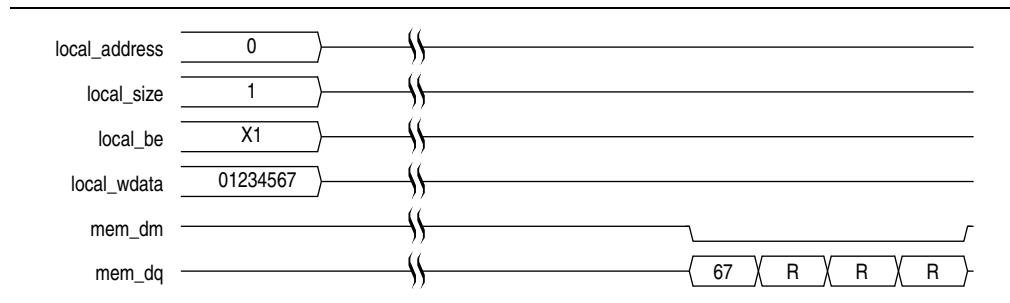


Figure 6-4. Partial Write for the Controller—Half Rate

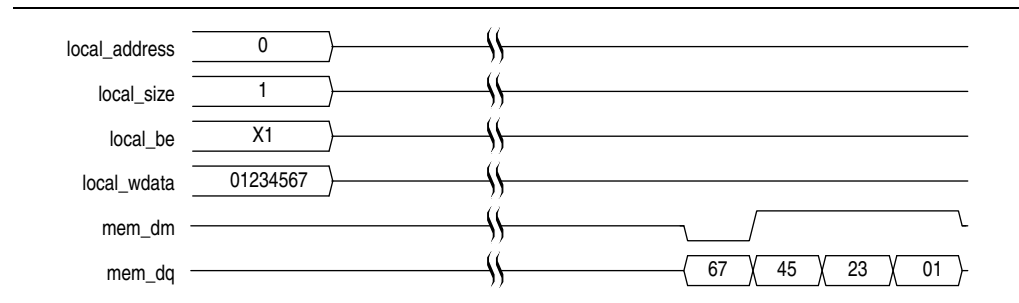


Partial Bursts

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 6-5 shows a partial burst operation for the controller.

Figure 6-5. 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.

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.
- The write response channel, which is an AXI protocol.



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 Signals” on page 5-28](#).

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-Controller 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 6-5 summarizes the controller's external interfaces.

Table 6-5. 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.

Table 6-5. Summary of Controller External Interfaces (Part 2 of 2)

Interface Name	Display Name	Type	Description
Write Data Channel	Avalon-ST Data Slave Interface	Avalon-ST (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.
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 Signals . (2) For information about Avalon signals, refer to Avalon Interface Specifications .			

Top-Level Signals Description

Table 6-6 shows the clock and reset signals.



The suffix `_n` denotes active low signals.

Table 6-6. 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.

Table 6–6. Clock and Reset Signals (Part 2 of 2)

Name	Direction	Description
aux_half_rate_clk	Output	An alternative clock that the PHY provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate designs, this clock is half the frequency of the <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.
dll_reference_clk	Output	Reference clock to feed to an externally instantiated DLL.
reset_request_n	Output	Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using Altera advises you detect a reset request on a falling edge rather than by level detection.
soft_reset_n	Input	Edge detect reset input for SOPC Builder or for control by other system reset logic. Assert to cause a complete reset to the PHY, but not to the PLL that the PHY uses.
seriesterminationcontrol	Input (for OCT slave)	Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (<code>alt_oct</code>) 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 (<code>alt_oct</code>) 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 <code>export</code> port on the PHY instance with a DLL to the <code>import</code> port on the other PHY instance.

Table 6–7 on page 6–13 shows the controller local interface signals.

Table 6-7. Local Interface Signals (Part 1 of 4)

Signal Name	Direction	Description
local_address[]	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>local_address is 24 bits wide</p> <p>local_address[23:11] = row address[12:0]</p> <p>local_address[10:9] = bank address[1:0]</p> <p>local_address[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>local_address is 23 bits wide</p> <p>local_address[22:10] = row address[12:0]</p> <p>local_address[9:8] = bank address[1:0]</p> <p>local_address[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>
local_be[]	Input	<p>Byte enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low.</p> <p>To map local_wdata and local_be to mem_dq and mem_dm, consider a full-rate design with 32-bit local_wdata and 16-bit mem_dq.</p> <p>Local_wdata = < 22334455 > < 667788AA > < BBCCDDEE ></p> <p>Local_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 6–7. Local Interface Signals (Part 2 of 4)

Signal Name	Direction	Description
local_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 local_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 local_ready. The IP core samples this signal at the rising edge of phy_clk when local_write_req is asserted. After the slave deasserts the local_ready signal, the master keeps all the write request signals asserted until local_ready signal becomes high again.</p> <p>For read transactions, assert this signal for one clock cycle when read request is asserted and local_address from which the data should be read is given to the memory. After the slave deasserts local_ready (waitrequest_n in Avalon interface), the master keeps all the read request signals asserted until local_ready becomes high again.</p>
local_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>
local_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.
local_wdata[]	Input	Write data bus. The width of local_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.
local_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 local_write_req.

Table 6-7. 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>
local_rdata[]	Output	<p>Read data bus. The width of local_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>
local_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 local_rdata_valid signal.</p> <p>If the controller encounters double-bit errors, no correction is made and the controller asserts this signal.</p>
local_rdata_valid	Output	<p>Read data valid signal. The local_rdata_valid signal indicates that valid data is present on the read data bus.</p>

Table 6-7. Local Interface Signals (Part 4 of 4)

Signal Name	Direction	Description
local_ready	Output	The local_ready signal indicates that the controller is ready to accept request signals. If controller asserts the local_ready signal in the clock cycle that it asserts a read or write request, the controller accepts that request. The controller deasserts the local_ready signal to indicate that it cannot accept any more requests. The controller can buffer eight read or write requests, after which the local_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 6-8 shows the controller interface signals.

Table 6-8. 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_clk	Bidirectional	Clock for the memory device.
mem_clk_n	Bidirectional	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 6-8. 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 6-8:

(1) This signal is for registered DIMMs only.

Table 6-9 shows the CSR interface signals.

Table 6-9. 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 then issues 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.

Example Top-Level File

The MegaWizard Plug-In Manager helps you create an example top-level file that shows you how to instantiate and connect the DDR or DDR2 SDRAM HPC II. The example top-level file consists of the DDR or DDR2 SDRAM HPC II, some driver logic to issue read and write requests to the controller, a PLL to create the necessary clocks, and a DLL (Stratix series only). The example top-level file is a working system that you can compile and use for both static timing checks and board tests.

Figure 6-6 shows the testbench and the example top-level file.

Figure 6-6. Testbench and Example Top-Level File

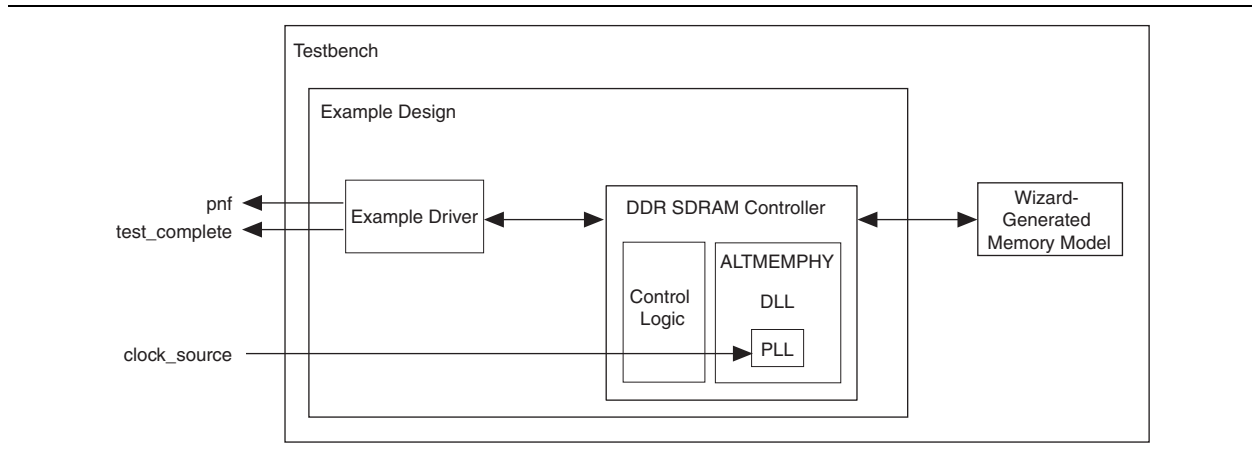


Table 6-10 describes the files that are associated with the example top-level file and the testbench.

Table 6-10. Example Top-Level File and Testbench Files

Filename	Description
<variation name>_example_top_tb.v or .vhd	Testbench for the example top-level file.
<variation name>_example_top.v or .vhd	Example top-level file.
<variation name>_mem_model.v or .vhd	Associative-array memory model.
<variation name>_full_mem_model.v or .vhd	Full-array memory model.
<variation name>_example_driver.v or .vhd	Example driver.
<variation name>.v or .vhd	Top-level description of the custom MegaCore function.
<variation name>.qip	Contains Quartus II project information for your MegaCore function variations.

There are two Altera-generated memory models available—associative-array memory model and full-array memory model.

The associative-array memory model (<variation name>_mem_model.v) allocates reduced set of memory addresses with a default depth of 2,048 or 2K address spaces. This allocation allows for a larger memory array compilation and simulation which enables you to easily reconfigure the depth of the associate array.

The full-array memory model (<variation name>_mem_model_full.v) allocates memory for all addresses accessible by the DDR cores. This allocation makes it impossible to simulate large memory designs.

Both the memory models display similar behaviors and have the same calibration time.



The memory model, *<variation name>_test_component.v/vhd*, used in SOPC Builder designs, is actually a variation of the full-array memory model. To ensure your simulation works in SOPC Builder, use memory model with less than 512-Mbit capacity.

Example Driver

The example driver is a self-checking test pattern generator for the memory interface. It uses a state machine to write and read from the memory to verify that the interface is operating correctly.

The example driver performs the following tests and loops back the tests indefinitely:

- Sequential addressing writes and reads

The state machine writes pseudo-random data generated by a linear feedback shift register (LFSR) to a set of incrementing row, bank, and column addresses. The state machine then resets the LFSR, reads back the same set of addresses, and compares the data it receives against the expected data. You can adjust the length and pattern of the bursts that are written by changing the `MAX_ROW`, `MAX_BANK`, and `MAX_COL` constants in the example driver source code, and the entire memory space can be tested by adjusting these values. You can skip this test by setting the `test_seq_addr_on` signal to logic zero.

- Incomplete write operation

The state machine issues a series of write requests that are less than the maximum burst size supported by your controller variation. The addresses are then read back to ensure that the controller has issued the correct signals to the memory. This test is only applicable in full-rate mode, when the local burst size is two. You can skip this test by setting the `test_incomplete_writes_on` signal to logic zero.

- Byte enable/data mask pin operation

The state machine issues two sets of write commands, the first of which clears a range of addresses. The second set of write commands has only one byte enable bit asserted. The state machine then issues a read request to the same addresses and the data is verified. This test checks if the data mask pins are operating correctly. You can skip this test by setting the `test_dm_pin_on` signal to logic zero.

- Address pin operation

The example driver generates a series of write and read requests starting with an all-zeros pattern, a walking-one pattern, a walking-zero pattern, and ending with an all-zeros pattern. This test checks to make sure that all the individual address bits are operating correctly. You can skip this test by setting the `test_addr_pin_on` signal to logic zero.

■ Low-power mode operation

The example driver requests the controller to place the memory into power-down and self-refresh states, and hold it in those states for the amount of time specified by the COUNTER_VALUE signal. You can vary this value to adjust the duration the memory is kept in the low-power states. This test is only available if your controller variation enables the low-power mode option.

The example driver has four outputs that allow you to observe which tests are currently running and if the tests are passing. The pass not fail (pnf) signal goes low once one or more errors occur and remains low. The pass not fail per byte (pnf_per_byte) signal goes low when there is incorrect data in a byte but goes back high again once correct data is observed in the following byte. The test_status signal indicates the test that is currently running, allowing you to determine which test has failed. The test_complete signal goes high for a single clock cycle at the end of the set of tests.

Table 6-11 shows the bit mapping for each test status.

Table 6-11. Test Status[] Bit Mapping

Bit	Test
0	Sequential address test
1	Incomplete write test
2	Data mask pin test
3	Address pin test
4	Power-down test
5	Self-refresh test
6	Auto-precharge test

Table 6-12 shows the ALTMEMPHY Debug interface signals, which are located in *<variation_name>_phy.v/vhd* file.

Table 6-12. ALTMEMPHY Debug Interface Signals

Signal Name	Direction	Description
dbg_clk	Input	Debug interface clock
dbg_addr	Input	Debug interface address
dbg_cs	Input	Debug interface chip select
dbg_wr	Input	Debug interface write request
dbg_wr_data	Input	Debug interface write data
dbg_rd	Input	Debug interface read request
dbg_rd_data	Input	Debug interface read data
dbg_waitrequest	Output	Debug interface wait request

Register Maps

Table 6-13 shows the overall register mapping for the DDR and DDR2 SDRAM Controllers with ALTMEMPHY.

Table 6-13. Register Map

Address	Description
ALTMEMPHY Register Map	
0x005	Mode register 0-1.
0x006	Mode register 2-3.
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.

ALTMEMPHY Register Map

The ALYMEMPHY register map allows you to control the memory components' mode register settings. Table 6-14 shows the register map for ALYMEMPHY.

To access the ALTMEMPHY register map, connect the ALTMEMPHY debug interface signals using the Avalon-MM protocol. After configuring the ALTMEMPHY register map, initialize a calibration request by setting bit 2 in the CSR register map address 0x100 for the mode register settings to take effect.

Table 6–14. ALTMEMPHY Register Map (Part 1 of 2)

Address	Bit	Name	Default	Access	Description
0x005	2:0	Burst length	8	Read only	This value is set to 8.
	3	BT	0	Read only	This value is set to 0.
	6:4	CAS latency	—	Read write	CAS latency setting. The default value for these bits is set by the MegaWizard CAS Latency setting for your controller instance. You must set this value in the CSR interface register map 0x126 as well.
	7	Reserved	0	—	Reserved for future use.
	8	DLL	0	Read write	Not used by the controller, but you can set and program into the memory device mode register.
	11:9	Write recovery	—	Read write	Write recovery (t_{WR}) setting. The default value for these bits is set by the MegaWizard Write Recovery setting for your controller instance. You must set this value in CSR interface register map 0x126 as well.
	12	PD	0/1	Read only	This value is set to 0.
	15:13	Reserved	0	C	Reserved for future use.
	16	DLL	0	Read write	Not used by the controller, but you can set and program into the memory device mode register.
	17	ODS	0	Read write	
	18	RTT	0	Read write	
	21:19	AL	—	Read write	Additive latency setting. The default value for these bits is set by the MegaWizard Additive Latency setting for your controller instance. You must set this value in CSR interface register map 0x126 as well.
	22	RTT	0	Read write	Not used by the controller, but you can set and program into the memory device mode register.
	25:23	RTT/WL/OCD	0	Read write	
	26	DQS#	0	Read write	
	27	TDQS/RDQS	0	Read write	
	28	QOFF	0	Read write	
	31:29	Reserved	0	—	Reserved for future use.

Table 6–14. ALTMEMPHY Register Map (Part 2 of 2)

Address	Bit	Name	Default	Access	Description
0x006	2:0	Reserved	0	—	Reserved for future use.
	5:3	CWL	—	Read write	CAS write latency setting. The default value for these bits is set by the MegaWizard CAS Write Latency setting for your controller instance. You must set this value in the CSR interface register map 0x126 as well.
	6	ASR	0	Read write	Not used by the controller, but you can set and program into the memory device mode register.
	7	SRT/ET	0	Read write	
	8	Reserved	0	—	Reserved for future use.
	10:9	RTT_WR	0	Read write	Not used by the controller, but you can set and program into the memory device mode register.
	15:11	Reserved	0	—	Reserved for future use.
	17:16	MPR_RF	0	Read write	Not used by the controller, but you can set and program into the memory device mode register.
	18	MPR	0	Read write	
	31:19	Reserved	0	—	Reserved for future use.

Controller Register Map

The controller register map allows you to control the memory controller settings. To access the controller register map, connect the CSR interface signals using the Avalon-MM protocol. [Table 6–15](#) shows the register map for the controller.

Table 6–15. Controller Register Map (Part 1 of 5)

Address	Bit	Name	Default	Access	Description
0x100	0	CAL_SUCCESS	—	Read only	This bit reports the value of the ALTMEMPHY <code>ctl_cal_success</code> output. Writing to this bit has no effect.
	1	CAL_FAIL	—	Read only	This bit reports the value of the ALTMEMPHY <code>ctl_cal_fail</code> output. Writing to this bit has no effect.
	2	CAL_REQ	0	Read write	Writing a 1 to this bit asserts the <code>ctl_cal_req</code> signal to the ALTMEMPHY megafunction. Writing a 0 to this bit deasserts the signal, and the ALTMEMPHY megafunction will then initiate its calibration sequence. You must not use this register during the ALTMEMPHY megafunction calibration. You must wait until the CAL_SUCCESS or CAL_FAIL register shows a value of 1.
	7:3	Reserved.	0	—	Reserved for future use.
	13:8	Reserved.	0	—	Reserved for future use.
	30:14	Reserved.	0	—	Reserved for future use.

Table 6-15. Controller Register Map (Part 2 of 5)

Address	Bit	Name	Default	Access	Description
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	REGDIMM	0	Read write	Setting this bit to 1 enables REGDIMM support in the controller.
	24:23	Reserved.	0	—	Reserved for future use.
	30:24	Reserved	0	—	Reserved for future use.
0x120	7:0	Column address width	—	Read write	The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12.
	15:8	Row address width	—	Read write	The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16.
	19:16	Bank address width	—	Read write	The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3.
	23:20	Chip select address width	—	Read write	The number of chip select address bits for the memory devices in your memory interface. The range of legal values is 0-2. If there is only one single chip select in the memory interface, set this bit to 0.
	31:24	Reserved.	0	—	Reserved for future use.
0x121	31:0	Data width representation (word)	—	Read only	The number of DQS bits in the memory interface. This bit can be used to derive the width of the memory interface by multiplying this value by the number of DQ pins per DQS pin (typically 8).
0x122	7:0	Chip select representation	—	Read only	The number of chip select in binary representation. For example, a design with 2 chip selects has the value of 00000011.
	31:8	Reserved.	0	—	Reserved for future use.

Table 6–15. Controller Register Map (Part 3 of 5)

Address	Bit	Name	Default	Access	Description
0x123	3:0	t_{RCD}	—	Read write	The activate to read or write a timing parameter. The range of legal values is 2-11 cycles.
	7:4	t_{RRD}	—	Read write	The activate to activate a timing parameter. The range of legal values is 2-8 cycles.
	11:8	t_{RP}	—	Read write	The precharge to activate a timing parameter. The range of legal values is 2-11 cycles.
	15:12	t_{MRD}	—	Read write	The mode register load time parameter. This value is not used by the controller, as the controller derives the correct value from the memory type setting.
	23:16	t_{RAS}	—	Read write	The activate to precharge a timing parameter. The range of legal values is 4-29 cycles.
	31:24	t_{RC}	—	Read write	The activate to activate a timing parameter. The range of legal values is 8-40 cycles.
0x124	3:0	t_{WTR}	—	Read write	The write to read a timing parameter. The range of legal values is 1-10 cycles.
	7:4	t_{RTP}	—	Read write	The read to precharge a timing parameter. The range of legal values is 2-8 cycles.
	15:8	t_{FAW}	—	Read write	The four-activate window timing parameter. The range of legal values is 6-32 cycles.
	31:16	Reserved.	0	—	Reserved for future use.
0x125	15:0	t_{REFI}	—	Read write	The refresh interval timing parameter. The range of legal values is 780-6240 cycles.
	23:16	t_{RFC}	—	Read write	The refresh cycle timing parameter. The range of legal values is 12-88 cycles.
	31:24	Reserved.	0	—	Reserved for future use.
0x126	3:0	CAS latency, t_{CL}	—	Read write	This value must be set to match the memory CAS latency. You must set this value in the 0x04 register map as well.
	7:4	Additive latency, AL			Additive latency setting. The default value for these bits is set in the Memory additive CAS latency setting in the Preset Editor dialog box. You must set this value in the 0x05 register map as well.
	11:8	CAS write latency, CWL			CAS write latency setting. You must set this value in the 0x06 register map as well.
	15:12	Write recovery, t_{WR}			This value must be set to match the memory write recovery time (t_{WR}). You must set this value in the 0x04 register map as well.
	19:16	Burst Length	—	Read write	Value must match memory burst length.
	31:20	Reserved.	0	—	Reserved for future use.

Table 6–15. Controller Register Map (Part 4 of 5)

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 6–15. Controller Register Map (Part 5 of 5)

Address	Bit	Name	Default	Access	Description
0x133	31:0	CORDROP_ADDR	0	Read only	The address of the most recent autocorrection that was dropped. This is a memory burst-aligned local address.
0x134	0	REORDER_DATA	—	Read write	
	15:1	Reserved.	0	—	Reserved for future use.
	23:16	STARVE_LIMIT	0	Read write	Number of commands that can be served before a starved command.
	31:24	Reserved.	0	—	Reserved for future use.

Latency is defined using the local (user) side frequency and absolute time (ns). There are two types of latencies that exist while designing with memory controllers—read and write latencies, which have the following definitions:

- Read latency—the amount of time it takes for the read data to appear at the local interface after initiating the read request.
- Write latency—the amount of time it takes for the write data to appear at the memory interface after initiating the write request.



For a half-rate controller, the local side frequency is half of the memory interface frequency. For a full-rate controller, the local side frequency is equal to the memory interface frequency.

Altera defines read and write latencies in terms of the local interface clock frequency and by the absolute time for the memory controllers. These latencies apply to supported device families ([Table 1–1 on page 1–2](#)) with the following memory controllers:

- Half-rate HPC II
- Full-rate HPC II

The latency defined in this section uses the following assumptions:

- The row is already open, there is no extra bank management needed.
- The controller is idle, there is no queued transaction pending, indicated by the `local_ready` signal asserted high.
- No refresh cycles occur before the transaction.

The latency for the high-performance controller II comprises many different stages of the memory interface. Figure 7-1 on page 7-2 shows a typical memory interface read latency path showing read latency from the time a `local_read_req` signal assertion is detected by the controller up to data available to be read from the dual-port RAM (DPRAM) module.

Figure 7-1. Typical Latency Path

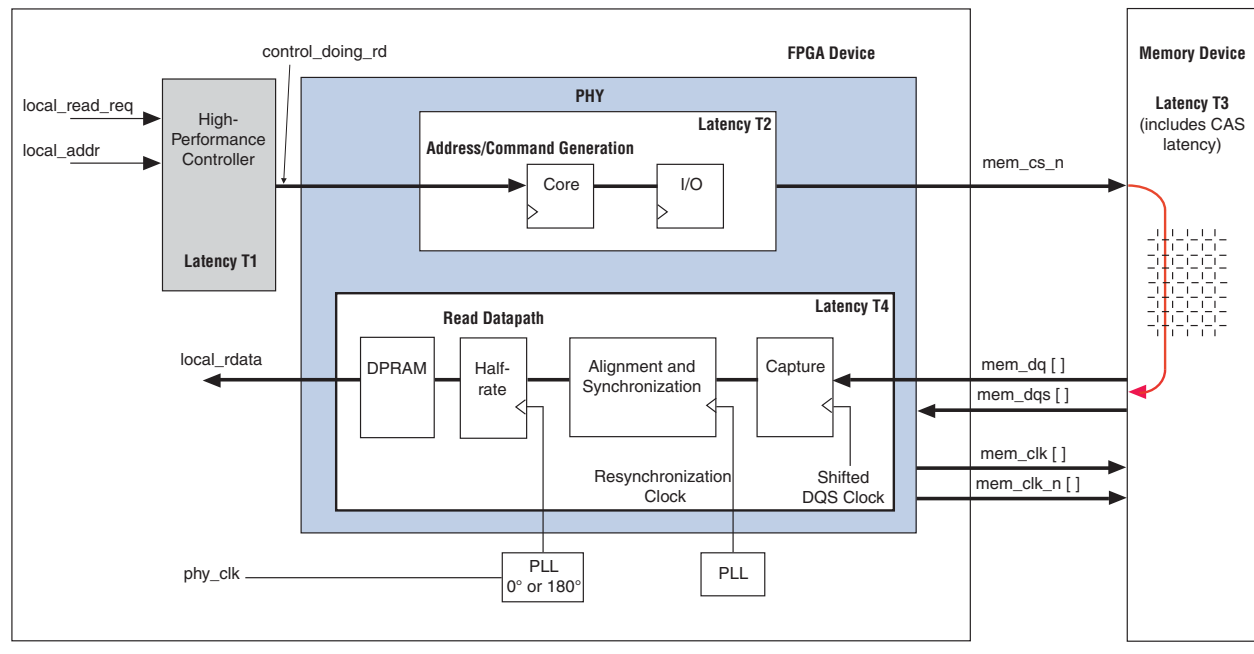


Table 7-1 shows the different stages that make up the whole read and write latency that Figure 7-1 shows.

Table 7-1. High-Performance Controller Latency Stages and Descriptions

Latency Number	Latency Stage	Description
T1	Controller	<code>local_read_req</code> or <code>local_write_req</code> signal assertion to <code>ddr_cs_n</code> signal assertion.
T2	Command Output	<code>ddr_cs_n</code> signal assertion to <code>mem_cs_n</code> signal assertion.
T3	CAS or WL	Read command to DQ data from the memory or write command to DQ data to the memory.
T4	ALTMEMPHY read data input	Read data appearing on the local interface.
T2 + T3	Write data latency	Write data appearing on the memory interface.

From Figure 7-1, the read latency in the high-performance controllers is made up of four components:

$$\text{read latency} = \text{controller latency (T1)} + \text{command output latency (T2)} + \text{CAS latency (T3)} + \text{PHY read data input latency (T4)}$$

Similarly, the write latency in the high-performance controller II is made up of three components:

$$\text{write latency} = \text{controller latency (T1)} + \text{write data latency (T2+T3)}$$

You can separate the controller and ALTMEMPHY read data input latency into latency that occurred in the I/O element (IOE) and latency that occurred in the FPGA fabric.

Table 7-2 shows the minimum and maximum supported CAS latency for the DDR and DDR2 SDRAM high-performance controller II.

Table 7-2. Supported CAS Latency (Note 1)

Device Family	Minimum Supported CAS Latency		Maximum Supported CAS Latency	
	DDR	DDR2	DDR	DDR2
Arria GX	3.0	3.0	3.0	6.0
Arria II GX	3.0	3.0	3.0	6.0
Cyclone III	2.0	3.0	3.0	6.0
Cyclone IV	2.0	3.0	3.0	6.0
HardCopy III	3.0	3.0	3.0	6.0
HardCopy IV	3.0	3.0	3.0	6.0
Stratix II	3.0	3.0	3.0	6.0
Stratix III	3.0	3.0	3.0	6.0
Stratix IV	3.0	3.0	3.0	6.0

Note to Table 7-2:

- (1) The registered DIMMs, where supported, effectively introduce one extra cycle of CAS latency. For the registered DIMMs, you need to subtract 1.0 from the CAS figures to determine the minimum supported CAS latency, and add 1.0 to the CAS figures to determine the maximum supported CAS latency.

Table 7-3 and Table 7-4 show a typical latency that can be achieved in Arria GX, Arria II GX, Cyclone III, Cyclone IV, Stratix IV, Stratix III, Stratix II, and Stratix II GX devices. The exact latency for your memory controller depends on your precise configuration. You can obtain precise latency from simulation, but this figure can vary slightly in hardware because of the automatic calibration process.

The actual memory CAS and write latencies shown are halved in half-rate designs as the latency calculation is based on the local clock.

The read latency also depends on your board trace delay. The latency found in simulation can be different from that found in board testing as functional simulation does not take into account the board trace delays. For a given design on a given board, the latency may change by one clock cycle (for full-rate designs) or two clock cycles (for half-rate designs) upon resetting the board. Different boards could also show different latencies even with the same design.

The CAS and write latencies are different between DDR and DDR2 SDRAM interfaces. To calculate latencies for DDR SDRAM interfaces, use the numbers from DDR2 SDRAM listed below and replace the CAS and write latency with the DDR SDRAM values.

Table 7-3. Typical Read Latency in HPC II (1), (2)

Device	Frequency (MHz)	Interface	Controller Latency (3)	Address and Command Latency		CAS Latency (4)	Read Data Latency		Total Read Latency (5)	
				FPGA	I/O		FPGA	I/O	Local Clock Cycles	Time (ns)
Arria GX	233	Half-rate	5	3	1	2	4.5	1	18	154
	167	Full-rate	5	2	1	4	5	1	19	114
Arria II GX	233	Half-rate	5	3	1	2.5	5.5	1	18	154
	167	Full-rate	5	2	1	4	6	1	20	120
Cyclone III and Cyclone IV	200	Half-rate	5	3	1	2	4.5	1	18	180
	167	Full-rate	5	2	1	4	5	1	19	114
Stratix II and Stratix II GX	333	Half-rate	5	3	1	2	4.5	1	18	108
	267	Half-rate	5	3	1	2	4.5	1	18	135
	200	Full-rate	5	2	1	4	5	1	19	95
Stratix III and Stratix IV	400	Half-rate	5	3	1	2.5	7.125	1.5	20	100
	267	Full-rate	4	2	1.5	4	7	1	20	75

Notes to Table 7-3:

- (1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.
- (2) Numbers shown may have been rounded up to the nearest higher integer.
- (3) The controller latency value is from the Altera high-performance controller.
- (4) CAS latency is per memory device specification and is programmable in the MegaWizard Plug-In Manager.
- (5) Total read latency is the sum of controller, address and command, CAS, and read data latencies.

Table 7-4. Typical Write Latency in HPC II (1), (2) (Part 1 of 2)

Device	Frequency (MHz)	Interface	Controller Latency (3)	Address and Command Latency		Memory Write Latency (4)	Total Write Latency (5)	
				FPGA	I/O		Local Clock Cycles	Time (ns)
Arria GX	233	Half-rate	5	3	1	1.5	12	103
	167	Full-rate	5	2	1	3	12	72
Arria II GX	233	Half-rate	5	3	1	2.5	12	103
	167	Full-rate	5	2	1	4	12	72
Cyclone III and Cyclone IV	200	Half-rate	5	3	1	1.5	12	120
	167	Full-rate	5	2	1	3	12	72

Table 7–4. Typical Write Latency in HPC II (1), (2) (Part 2 of 2)

Device	Frequency (MHz)	Interface	Controller Latency (3)	Address and Command Latency		Memory Write Latency (4)	Total Write Latency (5)	
				FPGA	I/O		Local Clock Cycles	Time (ns)
Stratix II and Stratix II GX	333	Half-rate	5	3	1	1.5	12	72
	267	Half-rate	5	3	1	1.5	12	90
	200	Full-rate	5	2	1	3	12	60
Stratix III and Stratix IV	400	Half-rate	5	3	1	2	12	60
	267	Full-rate	5	2	1.5	3	13	49

Notes to Table 7–4:

- (1) These are typical latency values using the assumptions listed in the beginning of the section. Your actual latency may be different than shown. Perform your own simulation for your actual latency.
- (2) Numbers shown may have been rounded up to the nearest higher integer.
- (3) The controller latency value is from the Altera high-performance controller.
- (4) Memory write latency is per memory device specification. The latency from when you provide the command to write to when you need to provide data at the memory device.
- (5) Total write latency is the sum of controller, address and command, and memory write latencies.



To see the latency incurred in the IOE for both read and write paths for ALTMEMPHY variations in Stratix IV and Stratix III devices refer to the IOE figures in the *External Memory Interfaces in Stratix III Devices* chapter of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter of the *Stratix IV Device Handbook*.

This chapter shows timing diagrams for the DDR and DDR2 SDRAM high-performance controllers II (HPC II).

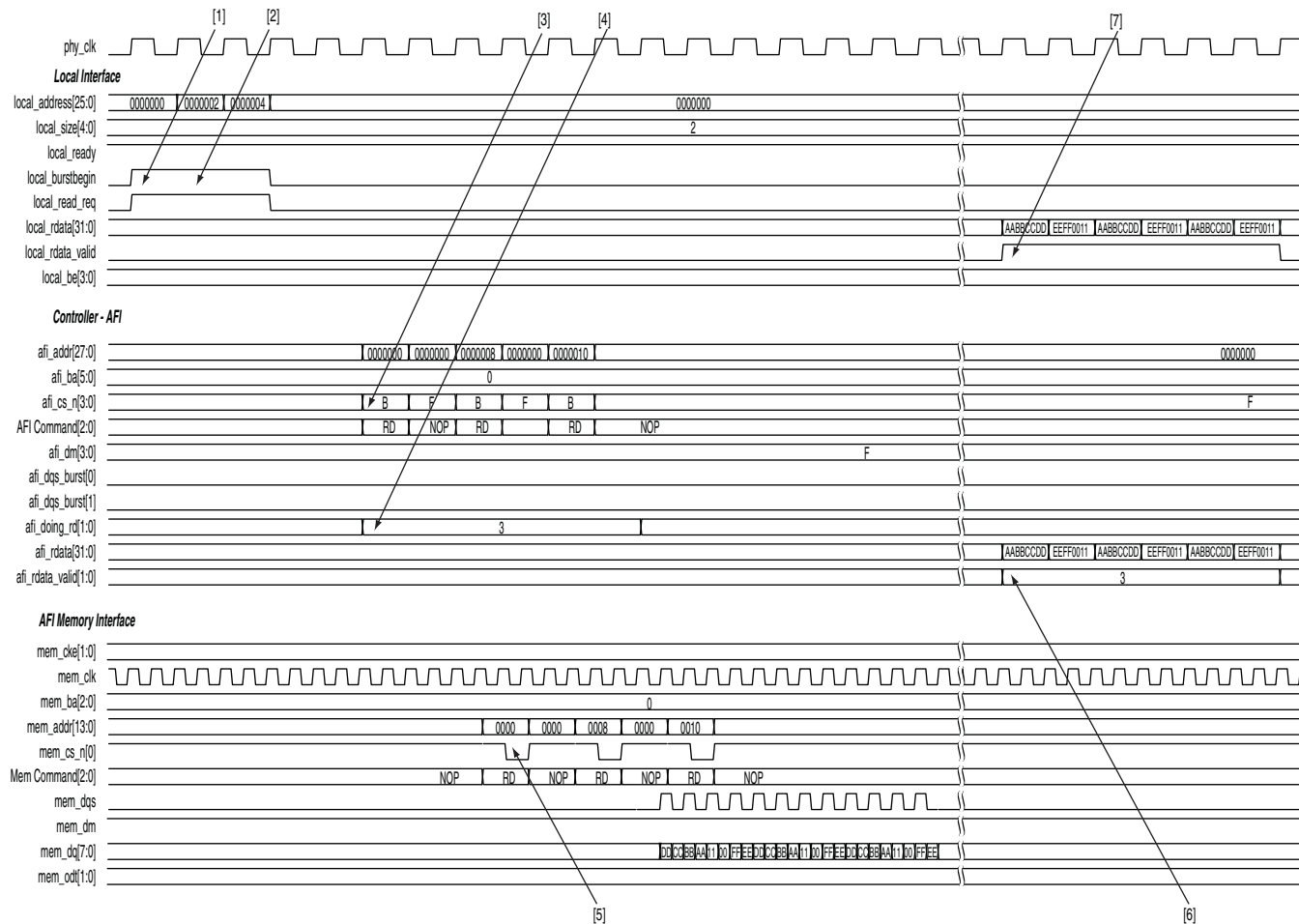
DDR and DDR2 High-Performance Controllers II

This section discusses the following timing diagrams for HPC II:

- “Half-Rate Read”
- “Half-Rate Write”
- “Full-Rate Read”
- “Full-Rate Write”

Half-Rate Read

Figure 8–1. Half-Rate Read Operation for HPC II



The following sequence corresponds with the numbered items in Figure 8–1:

1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0x000000`. This local address is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x000000
```

```
mem_col_address = 0x0000
```

```
mem_bank_address = 0x00
```

2. The user logic initiates a second read to a different memory column within the same row. The request for the second write is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the `local_ready` signal. The starting local address `0x000002` is mapped to the following memory address in half-rate mode:

```
mem_row_address = 0x0000
```

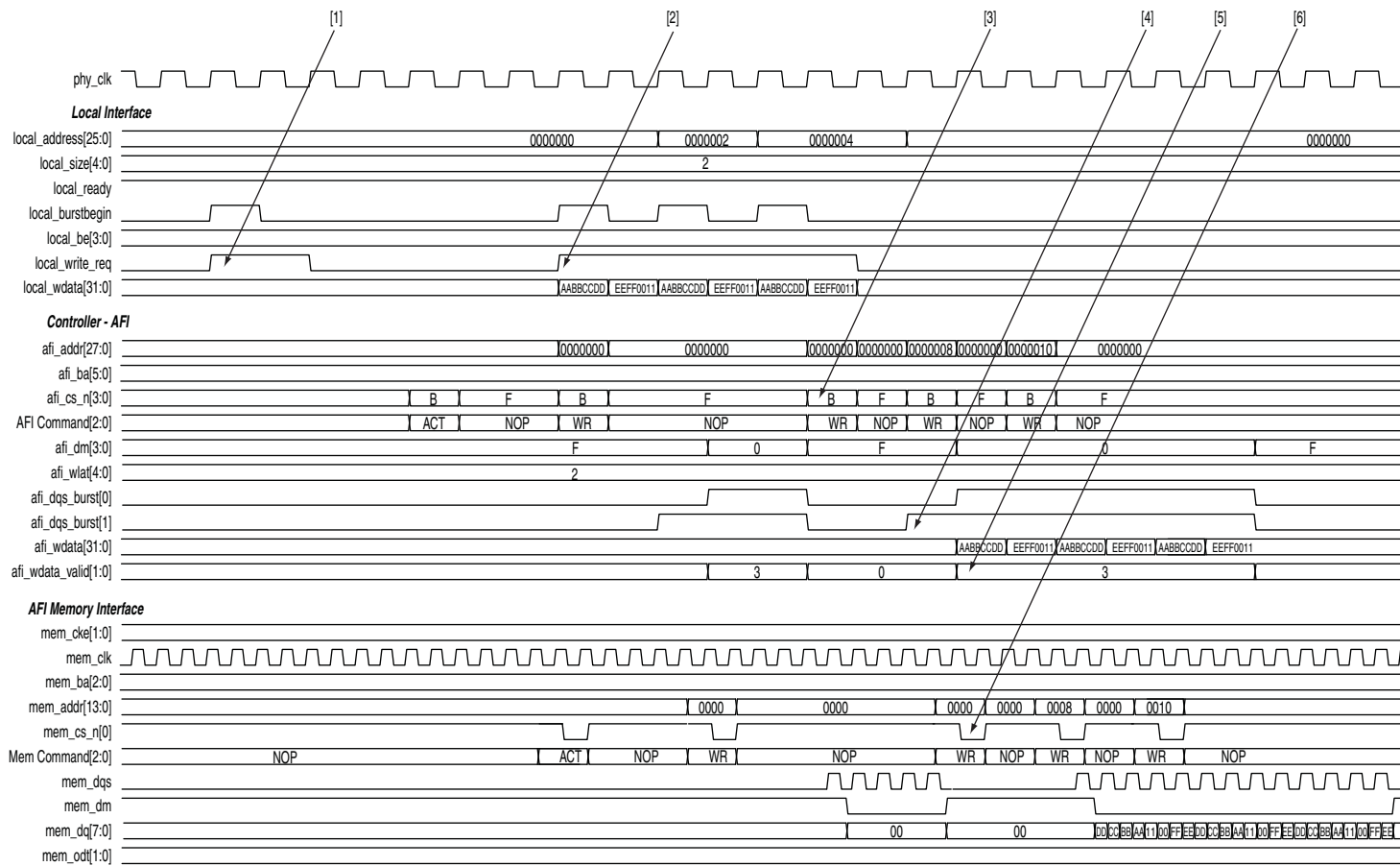
```
mem_col_address = 0x0002<<2 = 0x0008
```

```
mem_bank_address = 0x00
```

3. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
5. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
6. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.
7. The controller returns the first read data to the user by asserting the `local_rdata_valid` signal when there is valid read data on the `local_rdata` bus. If the ECC logic is disabled, there is no delay between the `afi_rdata` and the `local_rdata` buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the `afi_rdata` and `local_rdata` buses.

Half-Rate Write

Figure 8–2. Half-Rate Write Operation for HPC II

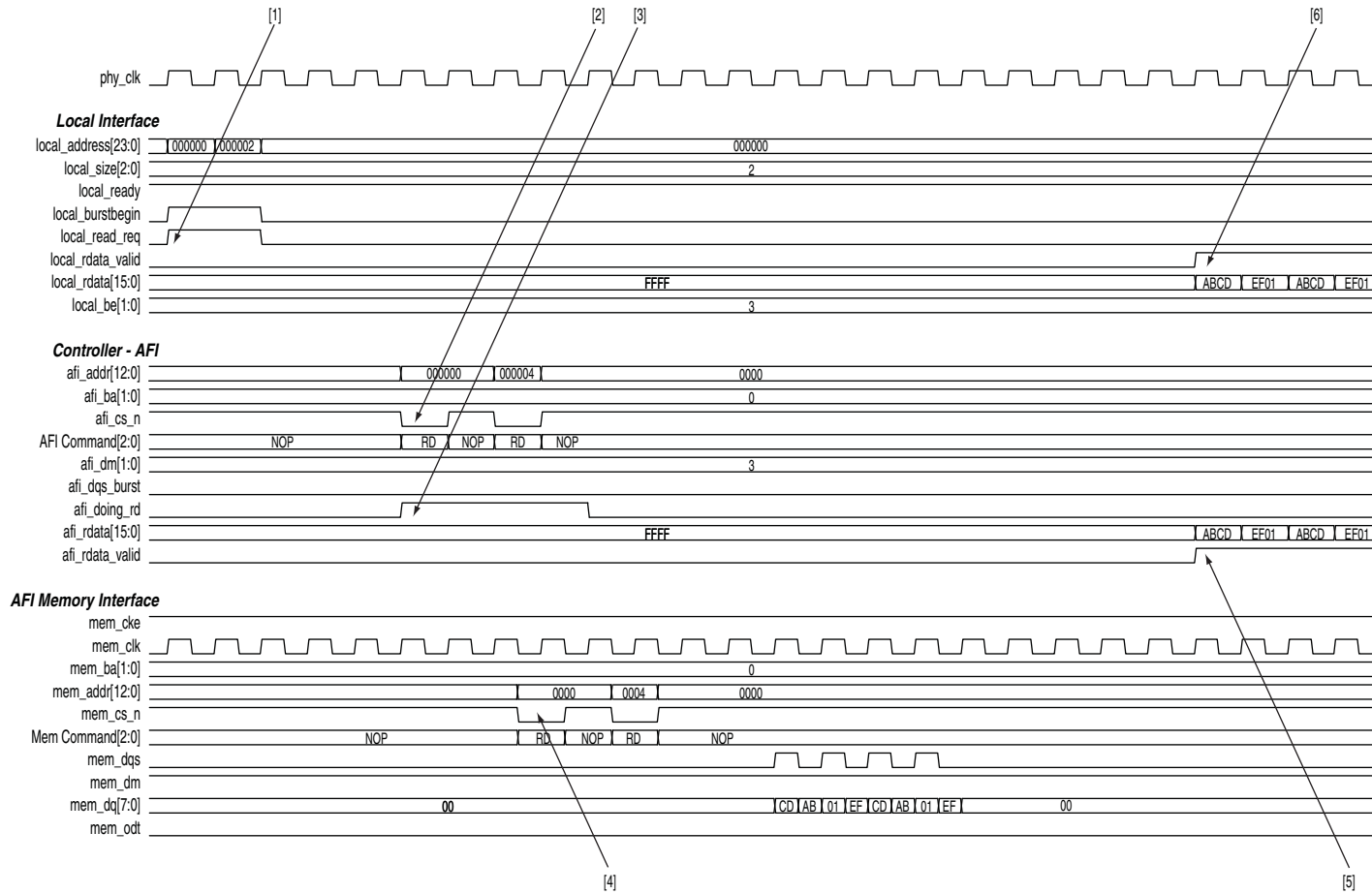


The following sequence corresponds with the numbered items in [Figure 8-2](#):

1. The user logic asserts the first write request to row 0 so that row 0 is open before the next transaction.
2. The user logic asserts a second `local_write_req` signal with size of 2 and address of 0 (`col = 0`, `row = 0`, `bank = 0`, `chip = 0`). The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

Full-Rate Read

Figure 8–3. Full-Rate Read Operation for HPC II



The following sequence corresponds with the numbered items in [Figure 8-3](#):

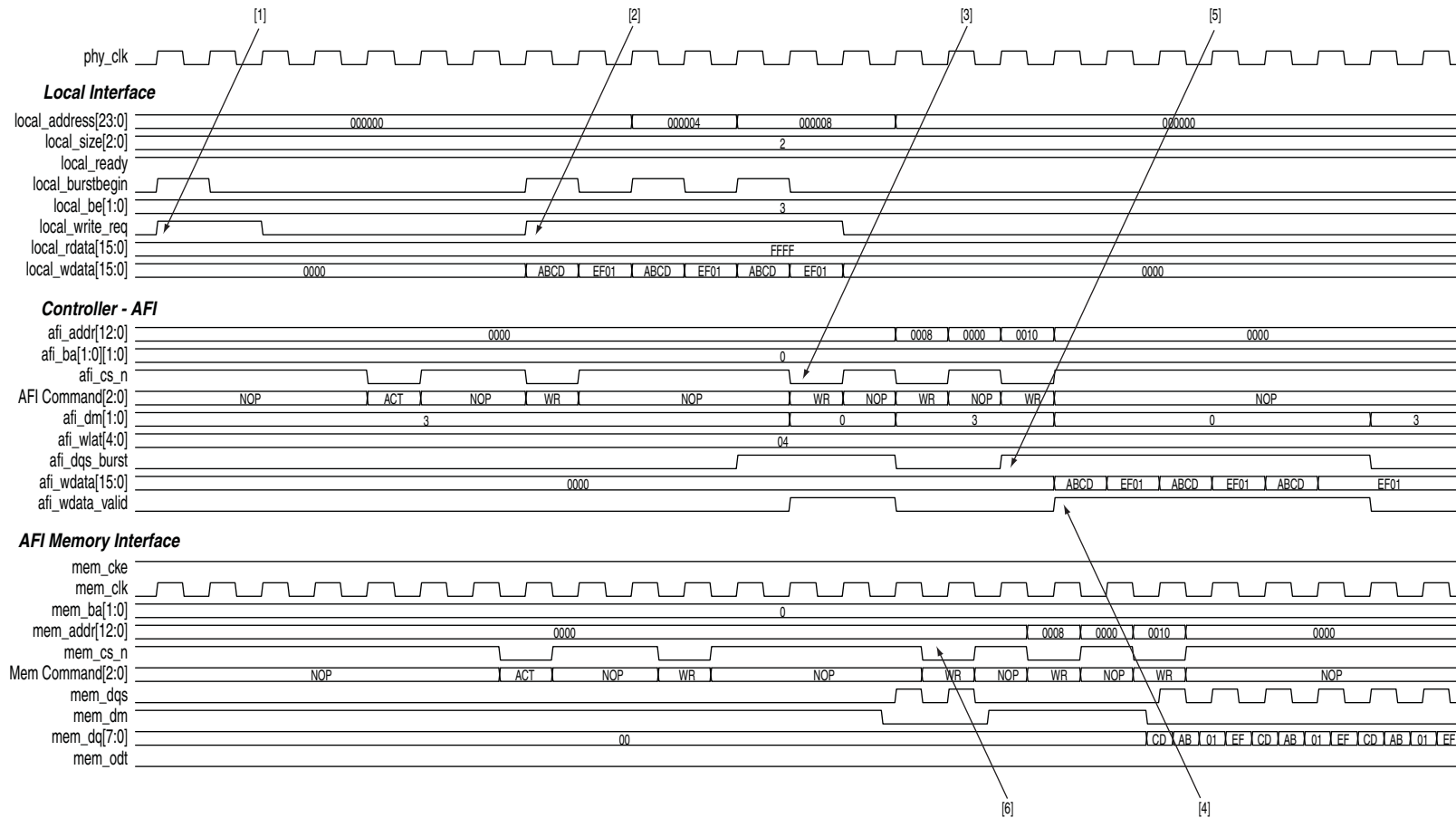
1. The user logic requests the first read by asserting `local_read_req` signal, and the size and address for this read. In this example, the request is a burst length of 2 to a local address `0x000000`. This local address is mapped to the following memory address in full-rate mode:

```
mem_row_address = 0x0000  
mem_col_address = 0x0000<<2 = 0x0000  
mem_bank_address = 0x00
```

2. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
4. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
5. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.
6. The controller returns the first read data to the user by asserting the `local_rdata_valid` signal when there is valid read data on the `local_rdata` bus. If the ECC logic is disabled, there is no delay between the `afi_rdata` and the `local_rdata` buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the `afi_rdata` and `local_rdata` buses.

Full-Rate Write

Figure 8–4. Full-Rate Write Operation for HPC II



The following sequence corresponds with the numbered items in [Figure 8-4](#):

1. The user logic asserts the first write request to row 0 so that row 0 is open before the next transaction.
2. The user logic asserts a second `local_write_req` signal with a size of 2 and address of 0 (`col = 0`, `row = 0`, `bank = 0`, `chip = 0`). The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

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	3.0	<ul style="list-style-type: none"> Removed references to High-Performance Controller. Updated High-Performance Controller II information. Removed HardCopy III, HardCopy IV E, HardCopy IV GX, Stratix III, and Stratix IV support.
December 2010	2.1	Updated for 10.1.
July 2010	2.0	<ul style="list-style-type: none"> Added information for new GUI parameters: Controller latency, Enable reduced bank tracking for area optimization, and Number of banks to track. Removed information about IP Advisor. This feature is removed from the DDR/DDR2 SDRAM IP support for version 10.0.
February 2010	1.3	Corrected typos.
February 2010	1.2	<ul style="list-style-type: none"> Full support for Stratix IV devices. Added timing diagrams for initialization and calibration stages for HPC.
November 2009	1.1	Minor corrections.
November 2009	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) (Software Licensing)	Email	nacomp@altera.com
	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.