Section I. DDR and DDR2 SDRAM High-Performance Controllers and ALTMEMPHY IP User Guide

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

EMI_DDR_UG-1.3

Document Version: 1.3
Document Date: February 2010

Contents

Chapter 6. Functional Description—High-Performance Controller

Additional Information

Revision History

The following table shows the revision history for this section.

Date	Version	Changes Made
February 2010	1.3	Corrected typos.
February 2010	1.2	■ 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.

1. About This IP

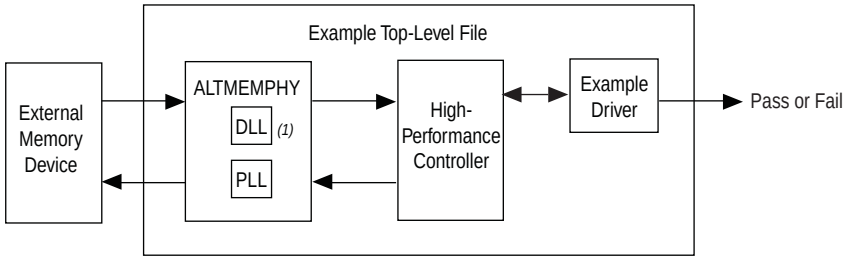
The Altera® DDR and DDR2 SDRAM High-Performance Controller MegaCore® functions provide simplified interfaces to industry-standard DDR SDRAM and DDR2 SDRAM. The ALTMEMPHY megafunction is an interface between a memory controller and the memory devices, and performs read and write operations to the memory.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

The DDR and DDR2 SDRAM High-Performance Controller MegaCore functions and ALTMEMPHY megafunction offer full-rate or half-rate DDR and DDR2 SDRAM interfaces. The DDR and DDR2 SDRAM High-Performance Controller MegaCore functions offer two controller architectures: high-performance controller (HPC) and high-performance controller II (HPC II). HPC II provides higher efficiency and more advanced features.

☞ The DDR and DDR2 SDRAM high-performance controllers denote both HPC and HPC II unless indicated otherwise.

Figure 1–1 shows a system-level diagram including the example top-level file that the DDR or DDR2 SDRAM High-Performance Controller MegaCore functions create for you.

Figure 1–1. System-Level Diagram



Note to Figure 1–1:

(1) When you choose **Instantiate DLL Externally**, 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 PLL and DLL. You can optionally 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. As a stand-alone product, use the ALTMEMPHY megafunction 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 controllers and ALTMEMPHY IP.

Table 1–1. Release Information

Item	Description
Version	9.1 SP1
Release Date	February 2010
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) 00CO (ALTMEMPHY Megafunction)
Vendor ID	6AF7

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release. For information about issues on the DDR 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 full or preliminary support for target Altera device families:

- Full support means the megafunction meets all functional and timing requirements for the device family and can be used in production designs.
- Preliminary support means the megafunction meets all functional requirements, but can still be undergoing timing analysis for the device family.

Table 1–2 shows the level of support offered by the DDR and DDR2 SDRAM high-performance controller to each of the Altera device families.

Table 1–2. Device Family Support

Device Family	Support
Arria® GX	Full
Arria II GX	Preliminary
Cyclone® III	Full
Cyclone III LS	Preliminary
Cyclone IV	Preliminary
HardCopy® II	Full
HardCopy III	Preliminary
HardCopy IV E	Preliminary
Stratix® II	Full
Stratix II GX	Full
Stratix III	Full
Stratix IV	Full
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
- 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 MegaWizard interface

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

Table 1–3. DDR and DDR2 SDRAM HPC and HPC II Features

Features	Controller Architecture	
	HPC	HPC II
Half-rate controller	✓	✓
Support for AFI ALTMEMPHY	✓	✓
Support for Avalon®Memory Mapped (MM) local interface	✓	✓
Support for Native local interface	✓	—
Configurable command look-ahead bank management with in-order reads and writes	—	✓
Additive latency	—	✓ *(1)*
Optional support for multi-cast write for t_{RC} mitigation	—	✓
Support for arbitrary Avalon burst length	—	✓
Memory burst length of 4	✓	✓ *(2)*
Memory burst length of 8	—	✓ *(3)*
Built-in flexible memory burst adapter	—	✓
Configurable Local-to-Memory address mappings	—	✓
Integrated half-rate bridge for low latency option	—	✓
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 SOPC Builder Flow	✓	✓
Integrated error correction coding (ECC) function 72-bit	✓	✓
Integrated ECC function 40-bit	—	✓
Support for partial-word write with optional automatic error correction	—	✓
SOPC Builder ready	✓	✓
Support for OpenCore Plus evaluation	✓	—
Support for the Quartus II IP Advisor	✓	—
IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulator	✓	✓

Notes to Table 1–3:

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

(2) HPC II only supports memory burst length of 4 in full-rate mode.

(3) HPC II only supports memory burst length of 8 in half-rate mode.

Unsupported 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

MegaCore verification involves simulation testing. Altera has carried out extensive random, directed tests with functional test coverage using industry-standard Denali models to ensure the functionality of the DDR and DDR2 SDRAM high-performance controllers.

Resource Utilization

The following sections show the resource utilization data for the ALTMEMPHY megafunction, and the DDR and DDR2 high-performance controllers (HPC and HPC II).

ALTMEMPHY Megafunction

Table 1–4 through Table 1–7 show the typical size of the ALTMEMPHY megafunction with the AFI in the Quartus II software version 9.1 for the following devices:

- Arria II GX (EP2AGX260FF35C4) devices
- Cyclone III (EP3C16F484C6) devices
- Stratix II (EP2S60F1020C3) devices
- Stratix III (EP3SL110F1152C2) devices
- Stratix IV (EP4SGX230HF35C2) devices

The resource utilization for Arria and Stratix GX devices is similar to Stratix II devices.

Table 1–4. Resource Utilization in Arria II GX Devices (Part 1 of 2) *(Note 1)*

PHY Rate	Memory Width (Bits)	Combinational ALUTS	Logic Registers	M9K Blocks	Memory ALUTs
Half	8	1,428	1,179	2	18
	16	1,480	1,254	4	2
	64	1,787	1,960	12	22
	72	1,867	2,027	13	2

Table 1–4. Resource Utilization in Arria II GX Devices (Part 2 of 2) *(Note 1)*

PHY Rate	Memory Width (Bits)	Combinational ALUTS	Logic Registers	M9K Blocks	Memory ALUTs
Full	8	1,232	975	0	35
	16	1,240	915	3	1
	64	1,287	1,138	7	41
	72	1,303	1,072	9	1

Note to Table 1–4:

(1) The listed resource utilization refers to resources used by the ALTMEMPHY megafunction with AFI only. Memory controller overhead is additional.

Table 1–5. Resource Utilization in Cyclone III Devices *(Note 1)*

PHY Rate	Memory Width (Bits)	Combinational LUTS	Logic Registers	M9K Blocks
Half	8	1,995	1,199	2
	16	2,210	1,396	3
	64	3,523	2,574	9
	72	3,770	2,771	9
Full	8	1,627	870	2
	16	1,762	981	2
	64	2,479	1,631	5
	72	2,608	1,740	5

Note to Table 1–5:

(1) The listed resource utilization refers to resources used by the ALTMEMPHY megafunction with AFI only. Memory controller overhead is additional.

Table 1–6. Resource Utilization in Stratix II Devices *(Note 1)* and *(2)*

PHY Rate	Memory Width (Bits)	Combinational LUTS	Logic Registers	M512K Blocks	M4K Blocks
Half	8	1,444	1,201	4	1
	16	1,494	1,375	4	2
	64	1,795	2,421	5	7
	72	1,870	2,597	4	8

Notes to Table 1–6:

(1) The listed resource utilization refers to resources used by the ALTMEMPHY megafunction with AFI only. Memory controller overhead is additional.

(2) The resource utilization for Arria and Stratix GX devices is similar to Stratix II devices.

Table 1–7. Resource Utilization in Stratix III and Stratix IV Devices *(Note 1)*

PHY Rate	Memory Width (Bits)	Combinational ALUTS	Logic Registers	M9K Blocks	Memory ALUTs
Half	8	1,356	1,040	1	40
	16	1,423	1,189	1	80
	64	1,805	2,072	1	320
	72	1,902	2,220	1	360
Full	8	1,216	918	1	20
	16	1,229	998	1	40
	64	1,319	1,462	1	160
	72	1,337	1,540	1	180

Note to Table 1–7:

(1) The listed resource utilization refers to resources used by the ALTMEMPHY megafunction with AFI only. Memory controller overhead is additional.

High-Performance Controller (HPC)

Table 1–8 through Table 1–13 show the typical sizes for the DDR or DDR2 SDRAM HPC with the AFI (including ALTMEMPHY) for Arria GX, Arria II GX, Cyclone III, Stratix II, Stratix II GX, Stratix III, and Stratix IV devices.

Table 1–8. Resource Utilization in Arria GX Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory	
					M512	M4K
Half	32	8	1,851	1,562	4	2
	64	16	1,904	1,738	4	4
	256	64	2,208	2,783	5	15
	288	72	2,289	2,958	4	17
Full	16	8	1,662	1,332	6	0
	32	16	1,666	1,421	3	3
	128	64	1738	1,939	3	9
	144	72	1,758	2,026	4	9

Table 1–9. Resource Utilization in Arria II GX Devices (Part 1 of 2)

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory
					(M9K)
Half	32	8	1,837	1,553	3
	64	16	1,894	1,628	6
	256	64	2,201	2,334	20
	288	72	2,279	2,401	22

Table 1–9. Resource Utilization in Arria II GX Devices (Part 2 of 2)

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Full	16	8	1,671	1400	1
	32	16	1,684	1,340	4
	128	64	1725	1,562	11
	144	72	1,738	2,497	14

Table 1–10. Resource Utilization in Cyclone III Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Half	32	8	2,683	1,563	3
	64	16	2,905	1,760	5
	256	64	4,224	2,938	17
	288	72	4,478	3,135	18
Full	16	8	2,386	1,276	3
	32	16	2,526	1,387	3
	128	64	3,257	2,037	9
	144	72	3,385	2,146	10

Table 1–11. Resource Utilization in Stratix II and Stratix II GX Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory	
					M512	M4K
Half	32	8	1,853	1,581	4	2
	64	16	1,901	1,757	4	4
	256	64	2,206	2,802	5	15
	288	72	2,281	2,978	4	17
Full	16	8	1,675	1,371	6	0
	32	16	1,675	1,456	3	3
	128	64	1740	1,976	3	9
	144	72	1,743	2,062	4	9

Table 1–12. Resource Utilization in Stratix III Devices (Part 1 of 2)

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Half	32	8	1,752	1,432	2
	64	16	1,824	1,581	3
	256	64	2,210	2,465	9
	288	72	2,321	2,613	10

Table 1–12. Resource Utilization in Stratix III Devices (Part 2 of 2)

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Full	16	8	1,622	1,351	2
	32	16	1,630	1,431	2
	128	64	1736	1,897	5
	144	72	1,749	1,975	6

Table 1–13. Resource Utilization in Stratix IV Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Half	32	8	1,755	1,452	1
	64	16	1,820	1,597	2
	256	64	2,202	2,457	8
	288	72	2,289	2,601	9
Full	16	8	1,631	1,369	1
	32	16	1,630	1,448	1
	128	64	1731	1,906	4
	144	72	1,743	1,983	5

High-Performance Controller II (HPC II)

Table 1–14 through Table 1–18 show the typical sizes for the DDR or DDR2 SDRAM HPC II (including ALTMEMPHY) for Arria II GX, Cyclone III, Stratix II, Stratix II GX, Stratix III, and Stratix IV devices.

Table 1–14. Resource Utilization in Arria II GX Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Half	32	8	3,038	2,041	3
	64	16	3,156	2,197	5
	256	64	3,649	3,115	17
	288	72	3,716	3,269	18
Full	16	8	2,860	1,856	1
	32	16	2,900	1,872	2
	128	64	3,138	2,246	7
	144	72	3,187	2,251	9

Table 1–15. Resource Utilization in Cyclone III Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Half	32	8	4,229	1,979	3
	64	16	4.409	2,155	5
	256	64	5,632	3,207	17
	288	72	5,811	3,382	18
Full	16	8	4,003	1,684	3
	32	16	4,090	1,763	3
	128	64	4,680	2,221	9
	144	72	4,776	2,298	10

Table 1–16. Resource Utilization in Stratix II and Stratix II GX Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory	
					M512	M4K
Half	32	8	3,063	1,991	4	3
	64	16	3,122	2,145	4	6
	256	64	3,433	3,065	5	23
	288	72	3,517	3,219	4	26
Full	16	8	2,818	1,756	4	2
	32	16	2,833	1,817	3	4
	128	64	2,869	2,137	3	13
	144	72	2,906	2,193	3	14

Table 1–17. Resource Utilization in Stratix III Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	Memory (M9K)
Half	32	8	2,907	1,935	2
	64	16	2,997	2,084	3
	256	64	3,392	2,968	9
	288	72	3,464	3,116	10
Full	16	8	2,859	1,758	2
	32	16	2,872	1,838	2
	128	64	2,948	2,302	5
	144	72	2,914	2,378	6

Table 1–18. Resource Utilization in Stratix IV Devices

Controller Rate	Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Dedicated Logic Registers	M9K
Half	32	8	2,935	1,966	2
	64	16	3,018	2,111	3
	256	64	3,405	2,971	9
	288	72	3,475	3,115	10
Full	16	8	2,856	1,792	2
	32	16	2,872	1,871	2
	128	64	2,938	2,329	5
	144	72	2,962	2,404	6

System Requirements

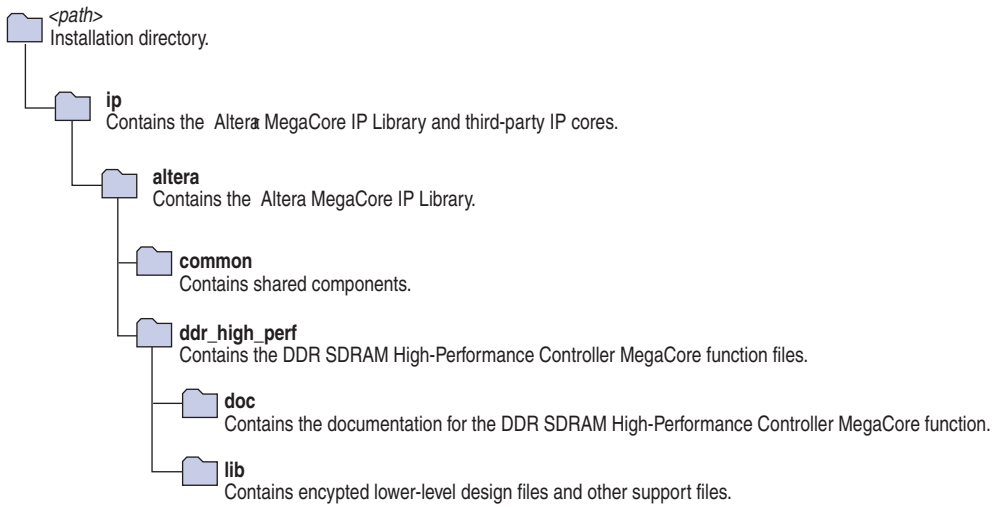
The DDR and DDR2 SDRAM High-Performance Controller MegaCore functions are part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

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 High-Performance Controller MegaCore functions, 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 MegaCore function 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 AMPP^{SM} 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

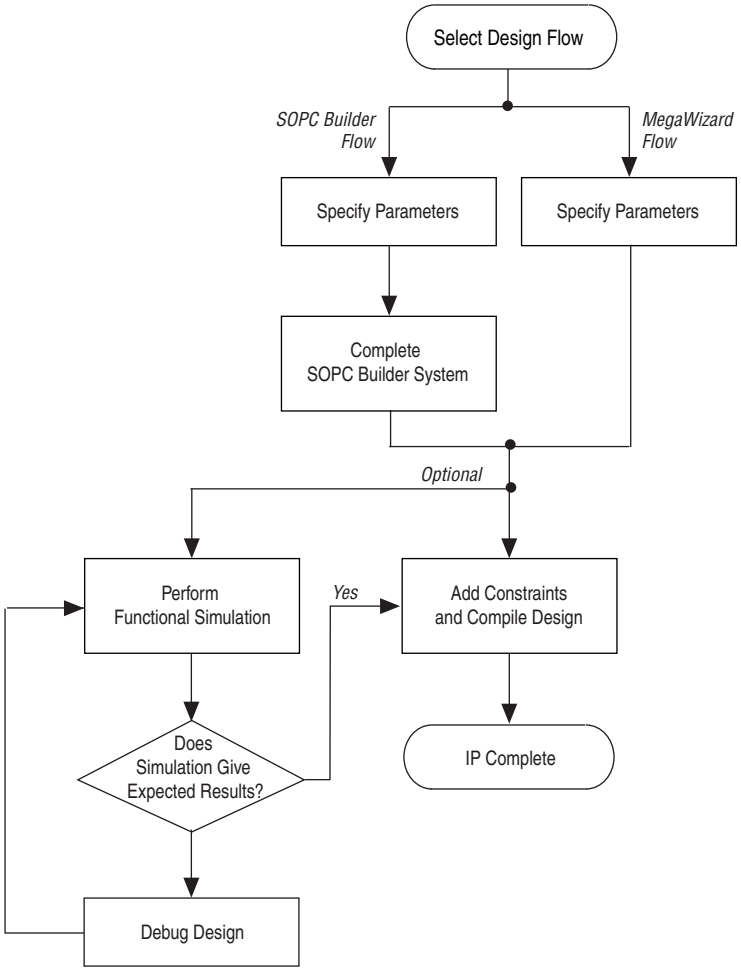
You can implement a DDR or DDR2 SDRAM High-Performance Controller MegaCore functions using either one of the following flows:

- SOPC Builder 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 either one of the 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 high-performance controller, such as the Nios II processor and scatter-gather direct memory access (DMA) 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 *DDR, DDR2, and DDR3 SDRAM 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.

Specify Parameters

To specify the parameters for the DDR and DDR2 SDRAM high-performance controllers 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 High-Performance Controller** to your system from the **System Contents** tab.

   The **DDR or DDR2 SDRAM High-Performance Controller** 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.

   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 high-performance controller and add it to the system.

Complete the SOPC Builder System

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

1. In the **System Contents** 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.

   CAUTION: 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.

   If you are upgrading your Nios system design from version 8.1 or previous, ensure that you change the **Reset Vector Offset** and the **Exception Vector Offset** to **AFI** mode.

   To calculate the Avalon-MM address equivalent of the memory address range `0×0` to `0×1f`, 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	0×40	0×60
16	0×80	0×A0
32	0×100	0×120
64	0×200	0×220

4. Click **Finish**.
5. On the **System Contents** 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**.

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 "Compile and Simulate" on page 4–1.

MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize the DDR and DDR2 SDRAM high-performance controllers or ALTMEMPHY megafunction, and manually integrate the function into your design.

You can alternatively use the IP Advisor to help you start your DDR or DDR2 SDRAM high-performance controller design. On the Quartus II Tools menu, point to **Advisors**, and then click **IP Advisor**. The IP Advisor guides you through a series of recommendations for selecting, parameterizing, evaluating, and instantiating a DDR2 SDRAM high-performance controller into your design. It then guides you through a complete Quartus II compilation of your project.

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

Specify 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 high-performance controller 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 MegaWizard interface, 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 MegaWizard interface 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**.

Generated Files

Table 2–2 shows the ALTMEMPHY generated files.

Table 2–2. 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 2–2. ALTMEMPHY Generated Files (Part 2 of 2)

File Name	Description
<*variation_name*>**_alt_mem_phy_pll.qip**	Quartus II IP file for the PLL that your ALTMEMPHY variation uses that contains the files associated with this megafunction.
<*variation_name*>**_alt_mem_phy_pll_bb.v/.cmp**	Black box file for the PLL used in your ALTMEMPHY variation. Typically unused.
<*variation_name*>**_alt_mem_phy_reconfig . qip**	Quartus II IP file for the PLL reconfiguration block. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.
<*variation_name*>**_alt_mem_phy_reconfig.v/.vhd**	PLL reconfiguration block module. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.
<*variation_name*>**_alt_mem_phy_reconfig_bb.v/cmp**	Black box file for the PLL reconfiguration block. Only generated when targeting Arria GX, HardCopy II, Stratix II, and Stratix II GX devices.
<*variation_name*>**_bb.v/.cmp**	Black box file for your ALTMEMPHY variation, depending whether you are using Verilog HDL or VHDL language.
<*variation_name*>**_ddr_pins.tcl**	Contains procedures used in the <*variation_name*>**_ddr_timing.sdc** and <*variation_name*>**_report_timing.tcl** files.
<*variation_name*>**_pin_assignments.tcl**	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.
<*variation_name*>**_ddr_timing.sdc**	Contains timing constraints for your ALTMEMPHY variation.
<*variation_name*>**_report_timing.tcl**	Script that reports timing for your ALTMEMPHY variation during compilation.

Table 2–3 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 2–3. Modules in <*variation_name*>_alt_mem_phy.v File (Part 1 of 2)

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

Table 2–3. 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 resyncronization 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_rdata_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 2–4 through Table 2–6 show the additional files generated by the high-performance controllers, that may be in your project directory. The names and types of files specified in the MegaWizard Plug-In Manager report vary based on whether you created your design with VHDL or Verilog HDL.

In addition to the files in Table 2–4 through Table 2–6, the MegaWizard also generates the ALTMEMPHY files in Table 2–2, but with a **_phy** prefix. For example, <*variation_name*>**_alt_mem_phy_delay.vhd** becomes <*variation_name*>**_phy_alt_mem_phy_delay.vhd**.

Table 2–4. Controller Generated Files—All High Performance Controllers (Part 1 of 2)

Filename	Description
<*variation name*>**.bsf**	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<*variation name*>**.html**	MegaCore function report file.
<*variation name*>.**v** or **.vhd**	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
<*variation name*>**.qip**	Contains Quartus II project information for your MegaCore function variations.

Table 2–4. Controller Generated Files—All High Performance Controllers (Part 2 of 2)

Filename	Description
<variation name>.**ppf**	XML file that describes the MegaCore pin attributes to the Quartus II Pin Planner. MegaCore pin attributes include pin direction, location, I/O standard assignments, and drive strength. If you launch IP Toolbench outside of the Pin Planner application, you must explicitly load this file to use Pin Planner.
*<variation name>*_**example_driver.v** or **.vhd**	Example self-checking test generator that matches your variation.
*<variation name>*_**example_top.v** or **.vhd**	Example top-level design file that you should set as your Quartus II project top level. Instantiates the example driver and the controller.

Table 2–5. Controller Generated Files—DDR and DDR2 High-Performance Controllers (HPC)

Filename	Description
*<variation name>*_**auk_ddr_hp_controller_wrapper.vo** or **.vho**	VHDL or Verilog HDL IP functional simulation model.
*<variation_name>*_**auk_ddr_hp_controller_ecc_wrapper.vo** or **.vho**	ECC functional simulation model.

Table 2–6. Controller Generated Files—DDR and DDR2 High-Performance Controllers II (HPC II) (Part 1 of 2)

Filename	Description
*<variation name>*_**alt_ddrx_controller_wrapper.v** or **.vho**	A controller wrapper that instantiates the **alt_ddrx_controller.v** file and configures the controller accordingly by the wizard.
alt_ddrx_addr_cmd.v	Decodes the state machine outputs into the memory address and command signals.
alt_ddrx_afi_block.v	Generates the read and write control signals for the AFI.
alt_ddrx_bank_tracking.v	Tracks which row is open in which memory bank.
alt_ddrx_clock_and_reset.v	Contains the clock and reset logic.
alt_ddrx_cmd_queue.v	Contains the command queue logic.
alt_ddrx_controller.v	The controller top-level file that instantiates all the sub-blocks.
alt_ddrx_csr.v	Contains the control and status register interface logic.
alt_ddrx_ddr2_odt_gen.v	Generates the on-die termination (ODT) control signal for DDR2 memory interfaces.
alt_ddrx_avalon_if.v	Communicates with the Avalon-MM interface.
alt_ddrx_decoder_40.v	Contains the 40 bit version of the ECC decoder logic.
alt_ddrx_decoder_72.v	Contains the 72 bit version of the ECC decoder logic.
alt_ddrx_decoder.v	Instantiates the appropriate width ECC decoder logic.
alt_ddrx_encoder_40.v	Contains the 40 bit version of the ECC encoder logic.
alt_ddrx_encoder_72.v	Contains the 72 bit version of the ECC encoder logic.
alt_ddrx_encoder.v	Instantiates the appropriate width ECC encoder logic.
alt_ddrx_input_if.v	The input interface block. It instantiates the **alt_ddrx_cmd_queue.v**, **alt_ddrx_wdata_fifo.v**, and **alt_ddrx_avalon_if.v** files.
alt_ddrx_odt_gen.v	Instantiates the **alt_ddrx_ddr2_odt_gen.v** file selectively. It also controls the ODT addressing scheme.

Table 2–6. Controller Generated Files—DDR and DDR2 High-Performance Controllers II (HPC II) (Part 2 of 2)

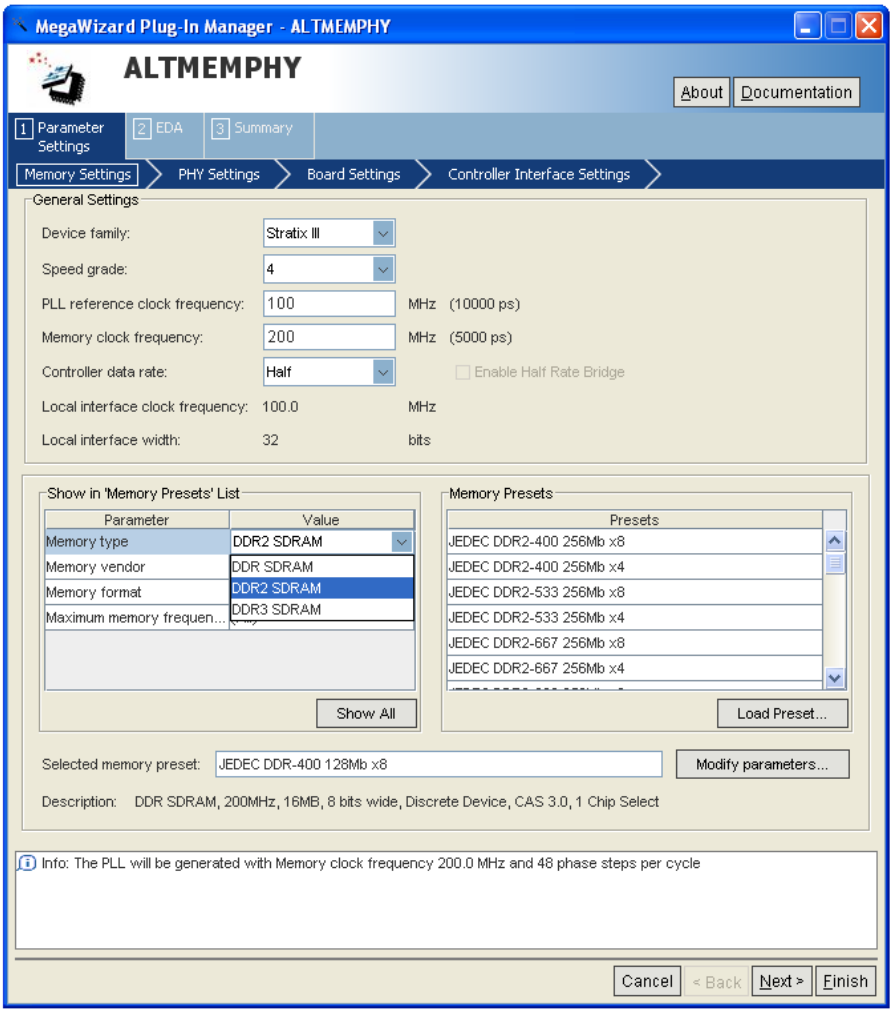
Filename	Description
alt_ddrx_state_machine.v	The main state machine of the controller.
alt_ddrx_timers_fsm.v	The state machine that tracks the per-bank timing parameters.
alt_ddrx_timers.v	Instantiates **alt_ddrx_timers_fsm.v** and contains the rank specific timing tracking logic.
alt_ddrx_wdata_fifo.v	The write data FIFO logic. This logic buffers the write data and byte-enables from the Avalon interface.
alt_avalon_half_rate_bridge_constraints.sdc	Contains timing constraints if your design has the **Enable Half Rate Bridge** option turned on.
alt_avalon_half_rate_bridge.v	The integrated half-rate bridge logic block.

ALTMEMPHY Parameter Settings

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

- Memory Settings
- PHY Settings
- Board Settings
- Controller Interface 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 MegaWizard interface.

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 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 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.

Use 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.

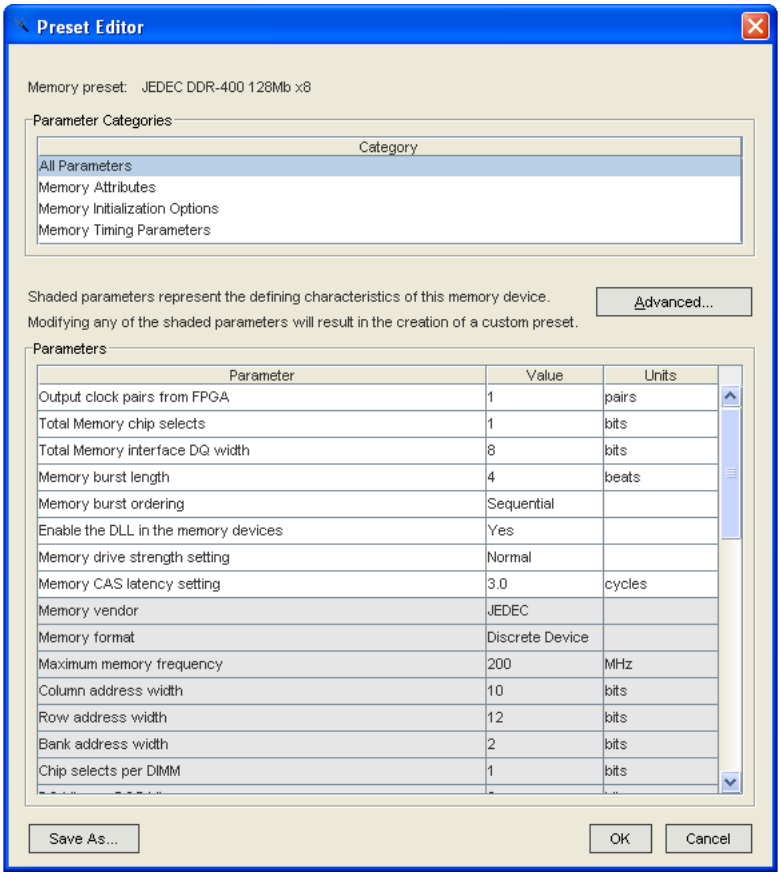
You can change the parameters with a white background to reflect your system. You can also change the parameters with a gray background so the device parameters match the device you are using. These parameters in gray background are characteristics of the chosen memory device and changing them creates 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 IV and Stratix III devices for differential signaling.
Memory chip selects	1, 2, 4, or 8	bits	Sets the number of chip selects in your memory interface. The depth of your memory in terms of number of chips. 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, `mem_cs_n` 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 MegaWizard interface 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			

Notes 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	Ohms	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. You need to 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_{DQSCK}	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.
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 "Derate Memory Setup and Hold Timing" on page 3–8 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 "Derate Memory Setup and Hold Timing" on page 3–8 for more information about how to derate this specification.
t_{DSH}	0.1–0.5	t_{CK}	DQS falling edge hold time from CK.

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

Parameter Name	Range	Units	Description
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 "Derate Memory Setup and Hold Timing" on page 3–8 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 "Derate Memory Setup and Hold Timing" on page 3–8 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 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).

Derate 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
- $V_{IH}(ac) = V_{REF} + 0.2$ V
- $V_{IH}(dc) = V_{REF} + 0.125$V
- $V_{IL}(ac) = V_{REF} – 0.2$ V
- $V_{IL}(dc) = V_{REF} – 0.125$ 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:

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

$t_{DH}(V_{REF})$ = Base t_{DH} + delta t_{DH} + $(V_{IH}(dc) – V_{REF})$/slew_rate = 125 + 0 + 67.5 = 192.5 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:

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

$t_{DH}(V_{REF})$ = Base t_{DH} + delta t_{DH} + $(V_{IH}(dc) – V_{REF})$/slew_rate = 100 + 45 + 33.75 = 178.75 ps

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

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

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

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 and Stratix II (prototyping for HardCopy II)	When enabling this option for Stratix II and HardCopy II 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 (`mem_clk_2x`) 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.
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.

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

Parameter Name	Applicable Device Families	Description
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 *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* or the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.
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 `phy_clk` and `write_clk` 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 **.sdc** 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 *Simulation* section in volume 4 of the *External Memory Interface Handbook*.

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 multirank 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 groups	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 group	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.

Controller Interface Settings

The **Controller Interface Settings** tab allows you to specify the native interface or the default Avalon-MM interface for your local interface as required by the ALTMEMPHY megafunction for DDR and DDR2 SDRAM. The options are disabled if you select **AFI** for the controller-to-PHY interface protocol. The Avalon-MM interface is the only local interface supported in these variations.

Altera recommends that you use the AFI for new designs; only use the non-AFI for existing designs.

Native interface is a superset of the Avalon-MM interface, containing the following additional signals in addition to the Avalon-MM interface signals:

- `local_init_done`
- `local_refresh_req`

- `local_refresh_ack`
- `local_wdata_req`

These signals provide extra information and control that is not possible in the Avalon-MM bus protocol.

The other difference between the native and the Avalon-MM local interface is in the write transaction. In an Avalon-MM interface, the write data is presented along with the write request. In native local interfaces, the write data (and byte enables) are presented in the clock cycle after the `local_wdata_req` signal is asserted. Avalon-MM interfaces do not use the `local_wdata_req` signal.

There is no difference in latency between the native and Avalon-MM interfaces.

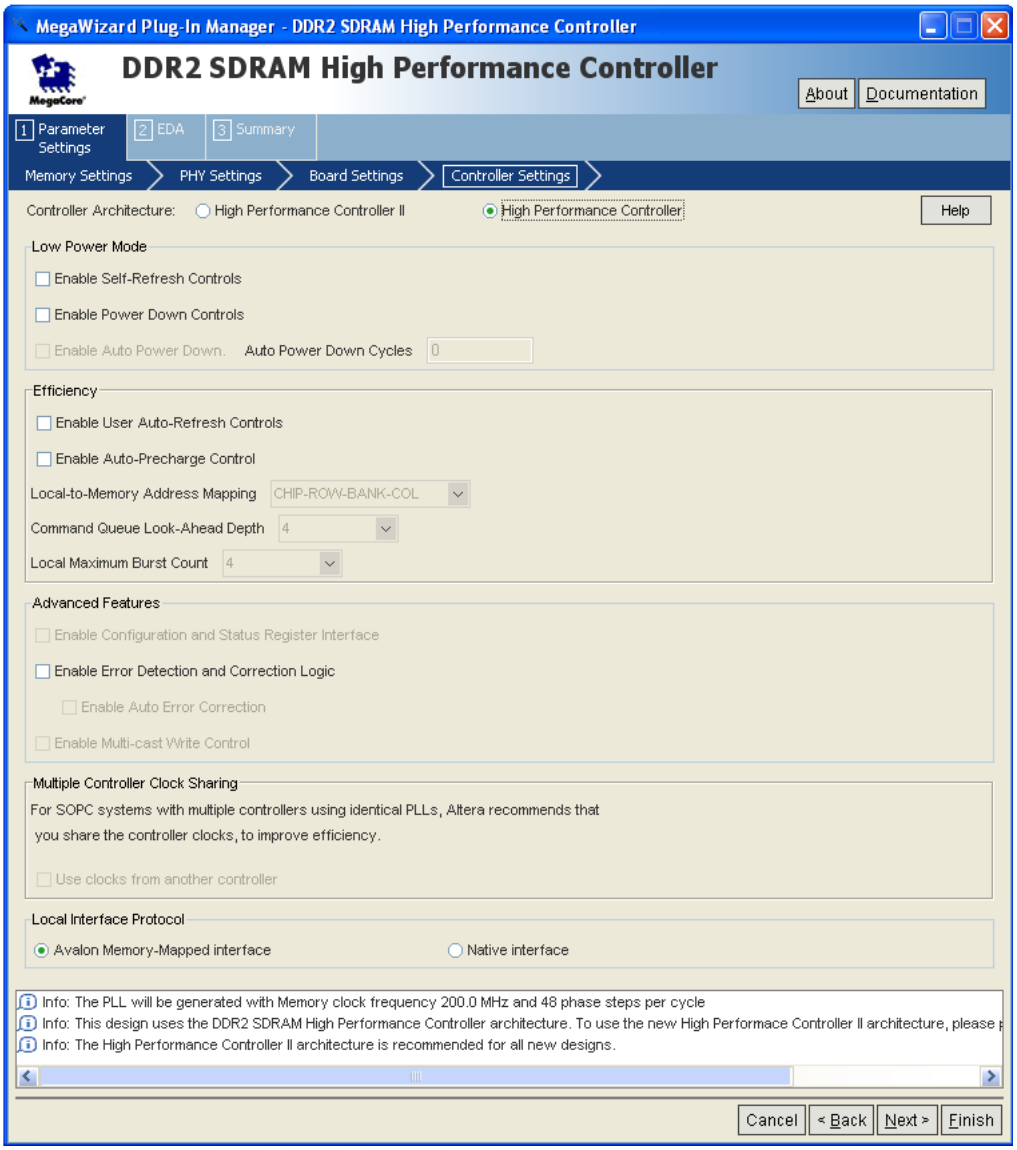
DDR or DDR2 SDRAM High-Performance Controller Parameter Settings

The **DDR or DDR2 SDRAM High-Performance Controller Parameter Settings** page in the DDR or DDR2 SDRAM High-Performance Controller MegaWizard interface (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 High-Performance Controller Settings



Controller Settings

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

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

Parameter	Controller Architecture	Description
Controller architecture	—	Specifies the controller architecture.
Enable self-refresh controls	Both	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 Logic" on page 7–7.
Enable power down controls	HPC	Turn on to enable the controller to allow you to have control on when to place the external memory device in power-down mode.

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

Parameter	Controller Architecture	Description
Enable auto power down	HPC II	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 7–7.
Auto power down cycles	HPC II	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	Both	Turn on to enable the controller to allow you to have control on when to place the external memory device in refresh mode.
Enable auto-precharge control	Both	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.
Local-to-memory address mapping	HPC II	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.
Command queue look-ahead depth	HPC II	This option allows you to select a command queue look-ahead depth value to control the number of read or write requests the look-ahead bank management logic examines, refer to "Command Queue" on page 7–4.
Local maximum burst count	HPC II	Specifies a burst count to configure the maximum Avalon burst count that the controller slave port accepts.

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

Parameter	Controller Architecture	Description
Enable configuration and status register interface	HPC II	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 "Configuration and Status Register (CSR) Interface" on page 7–7.
Enable error detection and correction logic	Both	Turn on to enable error correction coding (ECC) for single-bit error correction and double-bit error detection. Refer to "Error Correction Coding (ECC)" on page 6–5 for HPC, and "Error Correction Coding (ECC)" on page 7–7 for HPC II.
Enable auto error correction	HPC II	Turn on to allow the controller to perform auto correction when 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. Refer to "Error Correction Coding (ECC)" on page 7–7.
Enable multi-cast write control	HPC II	Turn on to enable the multi-cast write control on the controller top level. Asserting the multi-cast write control when requesting a write burst causes the write data to be written to all the chip selects in the memory system. Multi-cast write is not supported for registered DIMM interfaces or if the ECC logic is enabled.
Multiple controller clock sharing	Both	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. Refer to
Local interface protocol	HPC	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. The HPC II architecture supports only the Avalon-MM interface.

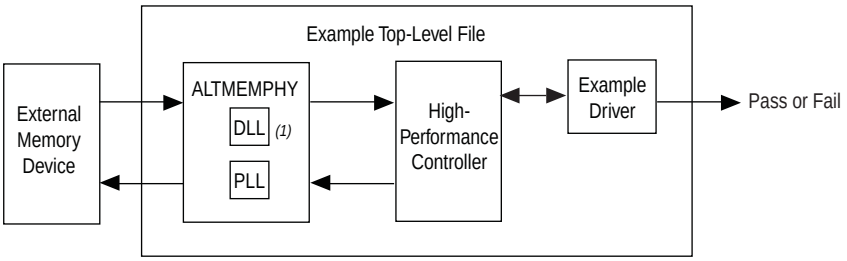
After setting the parameters to the MegaCore function, you can now integrate the MegaCore function variation into your design, and compile and simulate. The following sections detail the steps you need to perform to compile and simulate your design:

- Compile the Design
- Simulate the Design

Compile the Design

Figure 4–1 shows the top-level view of the Altera high-performance controller design as an example on 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 at 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_ddr_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. Follow these 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 sopc_mode]} {set sopc_mode YES}
     ```

     - MegaWizard Plug-In Manager flow:

     ```
     if {![info exists sopc_mode]} {set sopc_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 doing the following:
  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.

Simulate 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 2–6).

For more information about simulating SOPC Builder systems, refer to volume 4 of the *Quartus II Handbook* and *AN 351: Simulating Nios II Systems*. For more information about simulation, refer to the *Simulating an External Memory Interface Design* section in volume 4 of the *External Memory Interfaces Handbook*. 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 *DDR, DDR2, and DDR3 Tutorials* in volume 6 of the *External Memory Interfaces 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 must be disabled for all memory types.

- 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 is not able to cope with any delays, and it simply assumes that all delays in the testbench and memory model are 0 ps. In order 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.

Simulating Using NativeLink

To set up simulation using NativeLink for the DDR or DDR2 high-performance controllers (HPC and HPC II), follow these steps:

1. Create a custom variation with an IP functional simulation model, refer to step 4 in the "Specify Parameters" section on page 2–4.
2. Set the top-level entity to the example project.
   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**.
3. Set up the Quartus II NativeLink.
   a. On the Assignments menu, click **Settings**. In the **Category** list, expand **EDA Tool Settings** and click **Simulation**.
   b. From the **Tool name** list, click on your preferred simulator.

   Check that the absolute path to your third-party simulator executable is set. On the Tools menu, click **Options** and select **EDA Tools Options**.

   c. In **NativeLink settings**, select **Compile test bench** and click **Test Benches**.
   d. Click **New** at the **Test Benches** page to create a testbench.
4. On the **New Test Bench Settings** dialog box, do the following:
   a. Type a name for the **Test bench name**.
   b. In **Top level module in test bench**, type the name of the automatically generated testbench, <*variation name*>`_example_top_tb`.

   If you modified the <*variation name*>`_example_top_tb` to have a different port name, you need to change the testbench file with the new port names as well.

   c. In **Design instance in test bench**, type the name of the top-level instance, `dut`.
   d. Under **Simulation period**, set **Run simulation until all vector simuli are used**.
   e. Add the testbench files and automatically-generated memory model files. In the **File name** field, browse to the location of the memory model and the testbench, click **Open** and then click **Add**. The testbench is <*variation name*>**_example_top_tb.v**; memory model is <*variation name*>**_mem_model.v**.

   The auto generated generic SDRAM model may be used as a placeholder for a specific memory vendor supplied model.

   f. Select the files and click **OK**.
5. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration** to start analysis.

6. On the Tools menu, point to **Run EDA Simulation Tool** and click **EDA RTL Simulation**.

   Ensure that the Quartus II **EDA Tool Options** are configured correctly for your simulation environment. On the Tools menu, click **Options**. In the **Category** list, click **EDA Tool Options** and verify the locations of the executable files.

If your Quartus II project appears to be configured correctly but the example testbench still fails, check the known issues on the *Knowledge Database* page before filing a service request.

IP Functional Simulations

For VHDL simulations with IP functional simulation models, perform the following steps:

1. Create a directory in the <*project directory*>**testbench** directory.
2. Launch your simulation tool from this directory and create the following libraries:
   - altera_mf
   - lpm
   - sgate
   - <*device name*>
   - altera
   - ALTGXB
   - <*device name*>_hssi
   - auk_ddr_hp_user_lib
3. Compile the files into the appropriate library (AFI mode) as shown in Table 4–1. The files are in VHDL93 format.

Table 4–1. Files to Compile—VHDL IP Functional Simulation Models (Part 1 of 2)

Library	File Name
altera_mf	<*QUARTUS ROOTDIR*>**/eda/sim_lib/altera_mf_components.vhd** <*QUARTUS ROOTDIR*>**/eda/sim_lib/altera_mf.vhd**
lpm	**/eda/sim_lib/220pack.vhd** **/eda/sim_lib/220model.vhd**
sgate	**eda/sim_lib/sgate_pack.vhd** **eda/sim_lib/sgate.vhd**
<*device name*>	**eda/sim_lib/**<*device name*>**_atoms.vhd** **eda/sim_lib/**<*device name*>**_ components.vhd** **eda/sim_lib/**<*device name*>**_hssi_atoms.vhd** *(1)*
altera	**eda/sim_lib/altera_primitives_components.vhd** **eda/sim_lib/altera_syn_attributes.vhd** **eda/sim_lib/altera_primitives.vhd**

Table 4–1. Files to Compile—VHDL IP Functional Simulation Models (Part 2 of 2)

Library	File Name
ALTGXB *(1)*	*<device name>*_**mf.vhd** *<device name>*_**mf_components.vhd**
*<device name>*_hssi *(1)*	*<device name>*_**hssi_components.vhd** *<device name>*_**hssi_atoms.vhd**
auk_ddr_hp_user_lib	*<QUARTUS ROOTDIR>/* **libraries/vhdl/altera/altera_europa_support_lib.vhd**
	*<project directory>/<variation name>*_**phy_alt_mem_phy_seq_wrapper.vho**
	*<project directory>/<variation name>*_**phy.vho**
	<project directory>/<variation name>.**vhd**
	*<project directory>/<variation name>*_**example_top.vhd**
	*<project directory>/<variation name>*_**controller_phy.vhd**
	*<project directory>/<variation name>*_**phy_alt_mem_phy_reconfig.vhd** *(2)*
	*<project directory>/<variation name>*_**phy_alt_mem_phy_pll.vhd**
	*<project directory>/<variation name>*_**phy_alt_mem_phy_seq.vhd**
	*<project directory>/<variation name>*_**example_driver.vhd**
	*<project directory>/<variation name>*_**ex_lfsr8.vhd**
	testbench/*<variation name>*_**example_top_tb.vhd**
	testbench/*<variation name>*_**mem_model.vhd**
	*<project directory>/<variation name>*_**auk_ddr_hp_controller_wrapper.vho** (HPC)
	*<project directory>/<variation name>*_**alt_ddrx_controller_wrapper.vho** (HPC II)

Note for Table 4–1:

(1) Applicable only for Arria GX, Arria II GX, Stratix GX, Stratix II GX and Stratix IV devices.

(2) Applicable only for Arria GX, Hardcopy II, Stratix II and Stratix II GX devices.

If you are targeting Stratix IV devices, you need both the Stratix IV and Stratix III files (**stratixiv_atoms** and **stratixiii_atoms**) to simulate in your simulator, unless you are using NativeLink.

4. Load the testbench in your simulator with the timestep set to picoseconds.

For Verilog HDL simulations with IP functional simulation models, follow these steps:

1. Create a directory in the *<project directory>***testbench** directory.

2. Launch your simulation tool from this directory and create the following libraries:
   - altera_mf_ver
   - lpm_ver
   - sgate_ver
   - <*device name*>_ver
   - altera_ver
   - ALTGXB_ver
   - <*device name*>_hssi_ver
   - auk_ddr_hp_user_lib
3. Compile the files into the appropriate library as shown in Table 4–2.

Table 4–2. Files to Compile—Verilog HDL IP Functional Simulation Models (Part 1 of 3)

Library	File Name
altera_mf_ver	<*QUARTUS ROOTDIR*>**/eda/sim_lib/altera_mf.v**
lpm_ver	**/eda/sim_lib/220model.v**
sgate_ver	**eda/sim_lib/sgate.v**
<*device name*>_ver	**eda/sim_lib/**<*device name*>**_atoms.v** **eda/sim_lib/**<*device name*>**_hssi_atoms.v** *(1)*
altera_ver	**eda/sim_lib/altera_primitives.v**
ALTGXB_ver *(1)*	<*device name*>**_mf.v**
<*device name*>_hssi_ver *(1)*	<*device name*>**_hssi_atoms.v**

Table 4–2. Files to Compile—Verilog HDL IP Functional Simulation Models (Part 2 of 3)

Library	File Name
auk_ddr_hp_user_lib	*<QUARTUS ROOTDIR>/* **libraries/vhdl/altera/altera_europa_support_lib.v**
	alt_mem_phy_defines.v
	*<project directory>/<variation name>*_**phy_alt_mem_phy_seq_wrapper.vo**
	<project directory>/<variation name>.**v**
	*<project directory>/<variation name>*_**example_top.v**
	*<project directory>/<variation name>*_**phy.v**
	*<project directory>/<variation name>*_**controller_phy.v**
	*<project directory>/<variation name>*_**phy_alt_mem_phy_reconfig.v** *(2)*
	*<project directory>/<variation name>*_**phy_alt_mem_phy_pll.v**
	*<project directory>/<variation name>*_**phy_alt_mem_phy.v**
	*<project directory>/<variation name>*_**example_driver.v**
	*<project directory>/<variation name>*_**ex_lfsr8.v**
	testbench/*<variation name>*_**example_top_tb.v**
	testbench/*<variation name>*_**mem_model.v**
	*<project directory>/<variation name>*_**auk_ddr_hp_controller_wrapper.vo** (HPC)
	*<project directory>/<variation name>*_**alt_ddrx_controller_wrapper.v** (HPC II)
	*<project directory>/***alt_ddrx_addr_cmd.v** (HPC II)
	*<project directory>/***alt_ddrx_afi_block.v** (HPC II)
	*<project directory>/***alt_ddrx_bank_tracking.v** (HPC II)
	*<project directory>/***alt_ddrx_clock_and_reset.v** (HPC II)
	*<project directory>/***alt_ddrx_cmd_queue.v** (HPC II)
	*<project directory>/***alt_ddrx_controller.v** (HPC II)
	*<project directory>/***alt_ddrx_csr.v** (HPC II)
	*<project directory>/***alt_ddrx_ddr2_odt_gen.v** (HPC II)

Table 4–2. Files to Compile—Verilog HDL IP Functional Simulation Models (Part 3 of 3)

Library	File Name
	<project directory>/**alt_ddrx_avalon_if.v** (HPC II)
	<project directory>/**alt_ddrx_decoder_40.v** (HPC II)
	<project directory>/**alt_ddrx_decoder_72.v** (HPC II)
	<project directory>/**alt_ddrx_decoder.v** (HPC II)
	<project directory>/**alt_ddrx_encoder_40.v** (HPC II)
	<project directory>/**alt_ddrx_encoder_72.v** (HPC II)
	<project directory>/**alt_ddrx_encoder.v** (HPC II)
	<project directory>/**alt_ddrx_input_if.v** (HPC II)
	<project directory>/**alt_ddrx_odt_gen.v** (HPC II)
	<project directory>/**alt_ddrx_state_machine.v** (HPC II)
	<project directory>/**alt_ddrx_timers_fsm.v** (HPC II)
	<project directory>/**alt_ddrx_timers.v** (HPC II)
	<project directory>/**alt_ddrx_wdata_fifo.v** (HPC II)
	<project directory>/**alt_avalon_half_rate_bridge.v** (HPC II)

Notes for Table 4–2:

(1) Applicable only for Arria GX, Arria II GX, Stratix GX, Stratix II GX and Stratix IV devices.

(2) Applicable only for Arria GX, Hardcopy II, Stratix II and Stratix II GX devices.

If you are targeting Stratix IV devices, you need both the Stratix IV and Stratix III files (**stratixiv_atoms** and **stratixiii_atoms**) to simulate in your simulator, unless you are using NativeLink.

4. Configure your simulator to use transport delays, a timestep of picoseconds, and to include all the libraries in Table 4–2.

5. Functional Description—ALTMEMPHY

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 an Altera high-performance controller. In addition, the ALTMEMPHY megafunction and the Altera high-performance controllers 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.

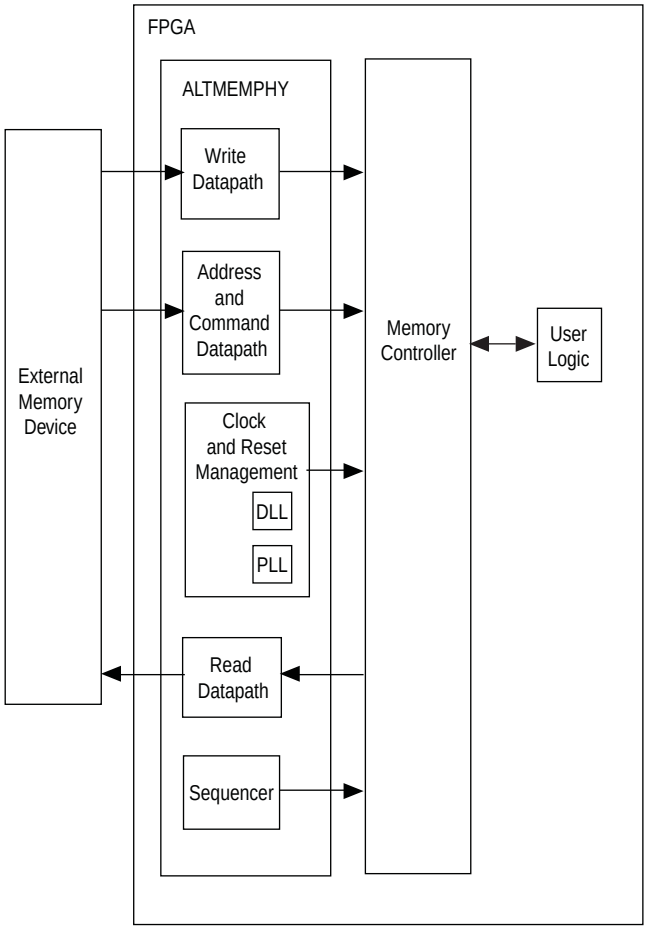
If the ALTMEMPHY megafunction does not meet your requirements, you can also create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software. However, you are then responsible for every aspect of the interface, including timing analysis and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

This section describes the calibration that the sequencer performs, to find the optimal clock phase for the memory interface.

The ALTMEMPHY variation for DDR/DDR2 SDRAM interfaces has a similar calibration process for the AFI and non-AFI.

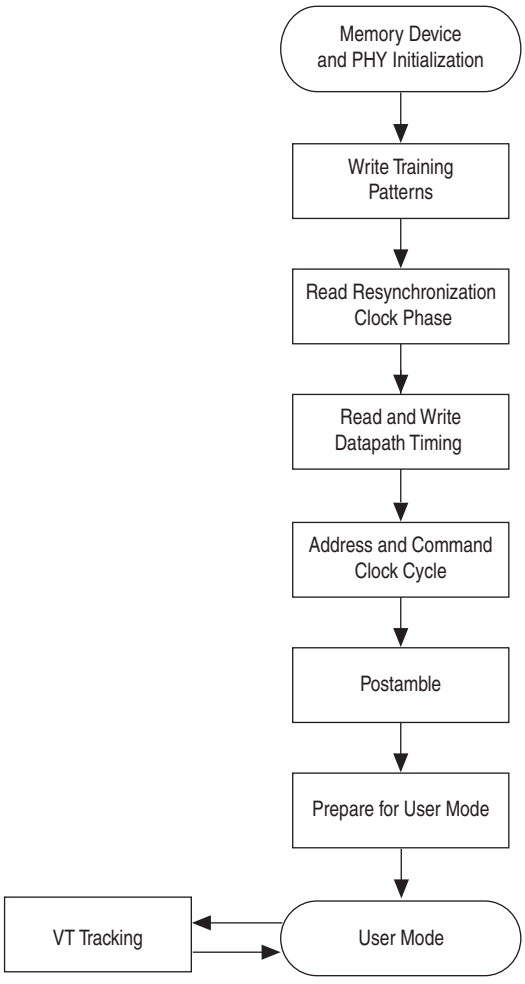
The calibration process for the DDR/DDR2 SDRAM PHY includes the following steps:

- "Step 1: Memory Device Initialization"
- "Step 2: Write Training Patterns"
- "Step 3: Read Resynchronization (Capture) Clock Phase"
- "Step 4: Read and Write Datapath Timing"
- "Step 5: Address and Command Clock Cycle"
- "Step 6: Postamble"
- "Step 7: Prepare for User Mode"

For more detailed information about each calibration step, refer to the *Hardware Debugging* section in volume 4 of the *External Memory Interfaces Handbook*.

Figure 5–2 shows the calibration flow.

Figure 5–2. Calibration Flow—DDR/DDR2 SDRAM



Step 1: Memory Device Initialization

This step initializes the memory device according to the DDR and DDR2 SDRAM specification. The initialization procedure includes specifying the mode registers and memory device ODT setting (DDR2 only), and initializing the memory device DLL. Calibration requires overriding some of the user-specified mode register settings, which are reverted in "Step 7: Prepare for User Mode".

Step 2: Write Training Patterns

In this step, a pattern is written to the memory to be read in later calibration stages. The matched trace lengths to DDR SDRAM devices mean that after memory initialization, write capture functions. The pattern is `0x30F5` and comprises the following separately written patterns:

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

This pattern is required to match the characterization behavior for non-DQS capture-based schemes, for example, the Cyclone III devices.

Loading a mixed pattern is complex, because the write latency is unknown at this time. Two sets of write and read operations (single pin resynchronization (capture) clock phase sweeps, ("Step 3: Read Resynchronization (Capture) Clock Phase") are required to accurately write the mixed pattern to memory.

Memory bank 0, row 0, and column addresses 0 to 55 store calibration data.

Step 3: Read Resynchronization (Capture) Clock Phase

This step adjusts the phase of the resynchronization (or capture) clock to determine the optimal phase that gives the greatest margin. For DQS-based capture schemes, the resynchronization clock captures the outputs of DQS capture registers (DQS is the capture clock). In a non-DQS capture-based scheme, the capture clock captures the input DQ pin data (the DQS signal is unused, and there is no resynchronization clock).

To correctly calibrate resynchronization (or capture) clock phase, based on a data valid window, requires the following degrees of phase sweep:

- 720° for all half-rate interfaces and full-rate DQS-based capture PHY
- 360° for a full-rate non-DQS capture PHY

Step 4: Read and Write Datapath Timing

In this step, the sequencer calculates the calibrated write latency (the `ctl_wlat` signal) between write commands and write data. The sequencer also calculates the calibrated read latency (the `ctl_rlat` signal) between the issue of a read command and valid read data. Both read and write latencies are output to a controller. In addition to advertising the read latency, the sequencer calibrates a read data valid signal to the delay between a controller issuing a read command and read data returning. The controller can use the read data valid signal in place of the advertised read latency, to determine when the read data is valid.

Step 5: Address and Command Clock Cycle

For half-rate interfaces, this step also optionally adds an additional memory clock cycle of delay from the address and command path. This delay aligns write data to memory commands given in the controller clock domain. If you require this additional delay, this step reruns the calibration ("Step 2: Write Training Patterns" to "Step 4: Read and Write Datapath Timing") to calibrate to the new setting.

Step 6: Postamble

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

Postamble is only required for DQS-based capture schemes.

Step 7: Prepare for User Mode

In this step, the PHY applies user mode register settings and performs periodic VT tracking.

VT Tracking

VT tracking is a background process that tracks the voltage and temperature variations to maintain the relationship between the resynchronization or capture clock and the data valid window that are achieved at calibration.

When the data calibration phase is completed, the sequencer issues the mimic calibration sequence every 128 ms.

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

In user mode, the sequencer periodically performs a tracking operation as defined in the tracking calibration description. At the end of the tracking calibration operation, the sequencer compares the most recent optimum tracking phase against the reference sampling phase. If the sampling phases do not match, the mimic path delays have changed due to voltage and temperature variations.

When the sequencer detects that the mimic path reference and most recent sampling phases do not match, the sequencer uses the PLL reconfiguration logic to change the phase of the resynchronization clock by the VCO taps in the same direction. This allows the tracking process to maintain the near-optimum capture clock phase setup during data tracking calibration as voltage and temperature vary over time.

The relationship between the resynchronization or capture clock and the data valid window is maintained by measuring the mimic path variations due to the VT variations and applying the same variation to the resynchronization clock.

Mimic Path

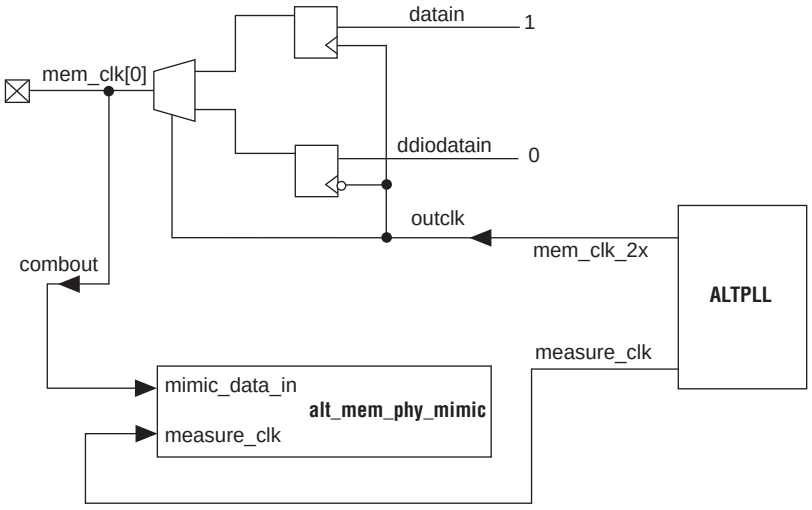
The mimic path mimics the FPGA elements of the round-trip delay, which enables the calibration sequencer to track delay variation due to VT changes during the memory read and write transactions without interrupting the operation of the ALTMEMPHY megafunction.

The assumption made about the mimic path is that the VT variation on the round trip delay path that resides outside of the FPGA is accounted for in the board skew and memory parameters entered in the MegaWizard Plug-In Manager. For the write direction, any VT variation in the memory devices is accounted for by timing analysis.

Figure 5–3 shows the mimic path in Arria GX, Cyclone III, Stratix II, and Stratix II GX devices, which mimics the delay of the clock outputs to the memory as far as the pads of the FPGA and the delay from the input DQS pads to a register in the FPGA core. During the tracking operation, the sequencer measures the delay of the mimic path by varying the phase of the measure clock. Any change in the delay of the mimic path indicates a corresponding change in the round-trip delay, and a corresponding adjustment is made to the phase of the resynchronization or capture clock.

The mimic path in Arria II GX, Stratix III and Stratix IV devices is similar to Figure 5–3. The only difference is that the `mem_clk[0]` pin is generated by DDIO register; `mem_clk_n[0]` is generated by signal splitter.

Figure 5–3. Mimic Path in Arria GX, Arria II GX, Cyclone III, Stratix II, and Stratix II GX Devices



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).

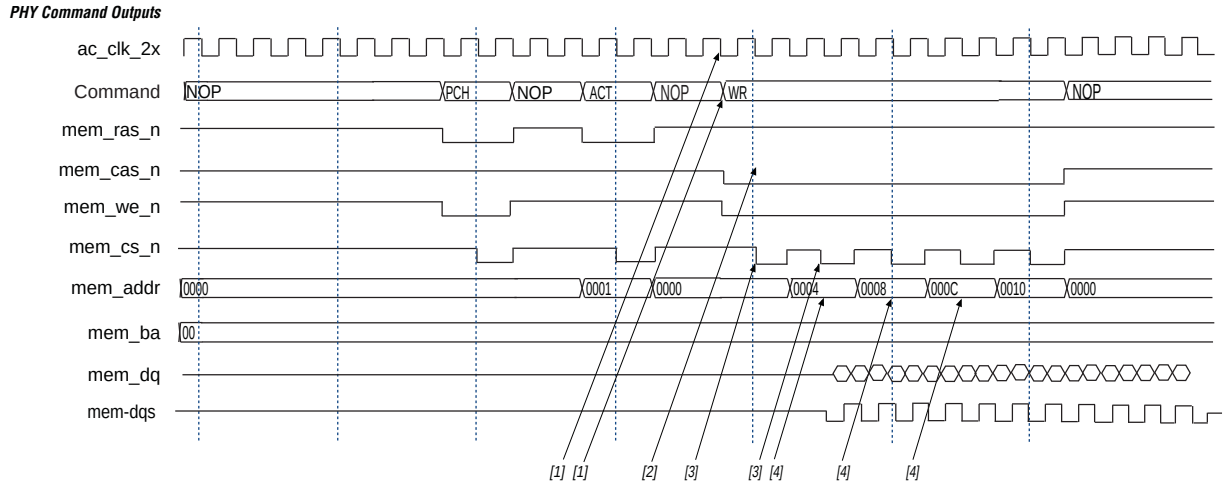
The address and command datapath is responsible for taking the address and command outputs from the controller and converting them from half-rate clock to full-rate clock. Two types of addressing are possible:

- 1T (full rate)—The duration of the address and command is a single memory clock cycle (`mem_clk_2x`, Figure 5–4). 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 4*n*-bits wide on the local side and is *n*-bits wide on the memory side. To transfer all the 4*n*-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 5–9 to see the frequency relationship of `mem_clk_2x` with the rest of the clocks.

Figure 5–4 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 5–4. Arria GX, Arria II GX, Cyclone II, HardCopy III, 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 5–4 shows a NOP command followed by back-to-back write commands. The following sequence corresponds with the numbered items in Figure 5–4:

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 MegaWizard interface. Refer to "Address and Command Datapath" on page 5–7 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_ns`, `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 5–7).

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 MegaWizard interface 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 5–4 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 5–7 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 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	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.
	`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.
Half-rate and full rate	Unused	C2	—	—	—	—
Half-rate and full rate	`write_clk_2x`	C3	–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°.

Table 5–2. DDR/DDR2 SDRAM Clocking in Arria II GX 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	ac_clk_2x	C3	90°	Full-Rate	Global	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 tp "Address and Command Datapath" on page 5–7 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	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	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	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, 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.

PLL Reconfiguration

The ALTMEMPHY MegaWizard interface 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 MegaWizard interface 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

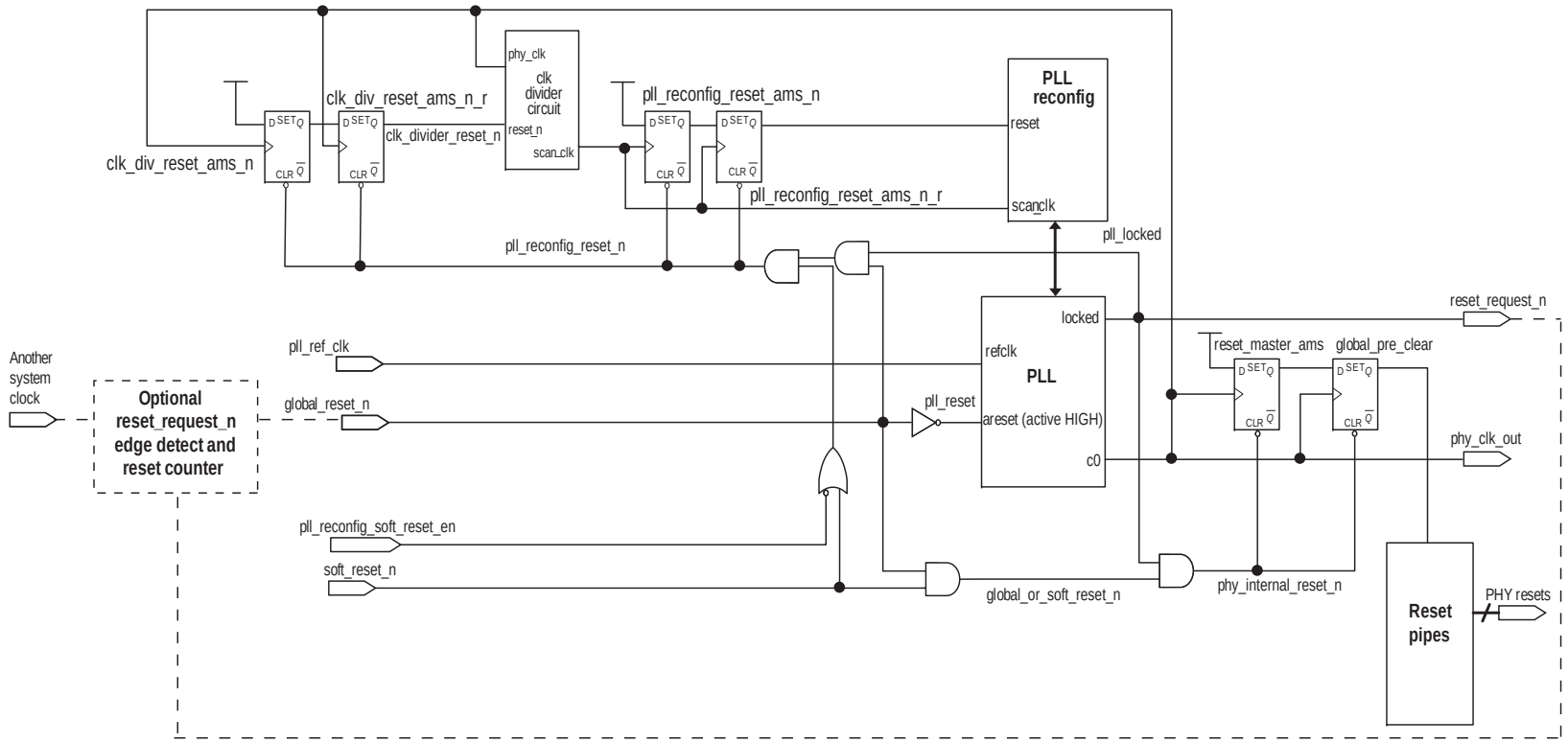
The reset management block is responsible for the following:

- 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 5–5 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 5–5. 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 5–5:

(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.
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.

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
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 5–7.

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 MegaWizard interface 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 3)

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`.

Table 5–4. DDR2 SDRAM Clocking in Stratix IV and Stratix III Devices (Part 2 of 3)

Design Rate	Clock Name *(1)*	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
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.
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°.

Table 5–4. DDR2 SDRAM Clocking in Stratix IV and Stratix III Devices (Part 3 of 3)

Design Rate	Clock Name *(1)*	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type	Notes
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 5–19). 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.

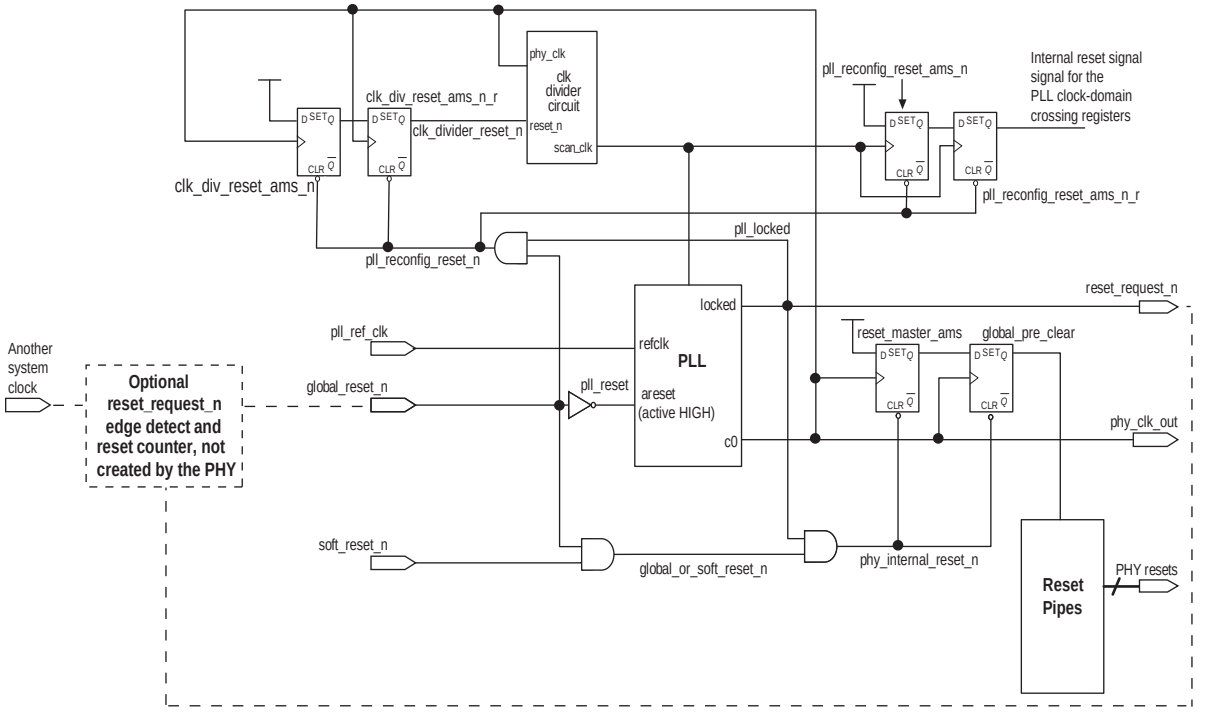
For more information on the DLL, refer to the *External Memory Interfaces in Stratix III Devices* chapter in volume 1 of the *Stratix III Device Handbook* and the *External Memory Interfaces in Stratix IV Devices* chapter in volume 1 of the *Stratix IV Device Handbook*.

Reset Management

Figure 5–6 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 5–6. 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.

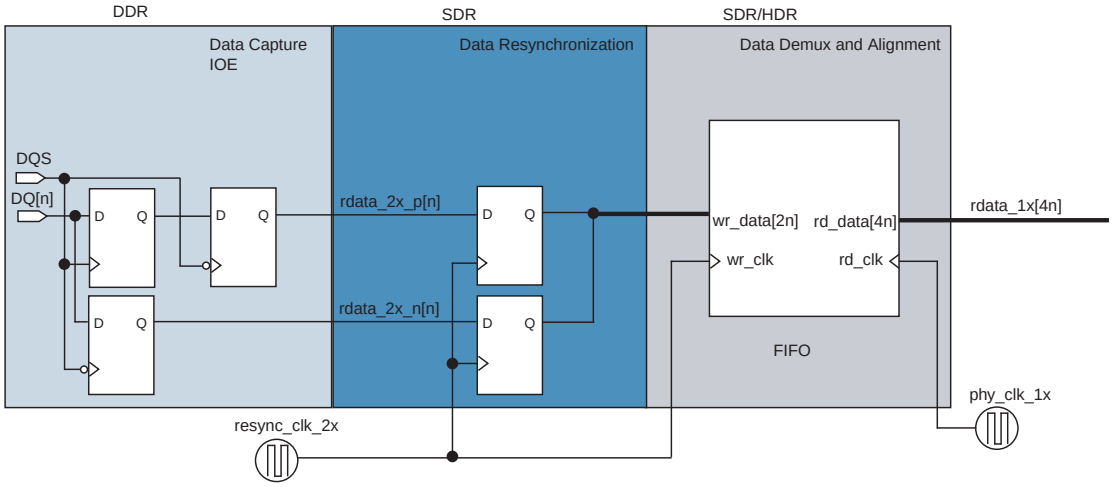
The full-rate datapath is similar to the half-rate datapath. The full-rate datapath also consists of a RAM with the same width as the data input (just like that of the half-rate), but the width on the data output of the RAM is half that of the half-rate PHY. The function of the RAM is to transfer the 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 5–7 shows the order of the functions performed by the read datapath, along with the frequency at which the read data is handled.

Figure 5–7. 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 5–7:

(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 5–7.

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 2*n*-bit wide. After data demuxing, the data is HDR of width 4*n*-bits wide. The system clock frequency is half the frequency of the memory clock.

Demultiplexing is achieved using a dual-port memory with a 2*n*-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 5–8 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 5–8 shows the relationship between the postamble clock and the resynchronization clock.

Figure 5–8. Relationship Between Postamble Clock and Resynchronization Clock *(Note 1)*

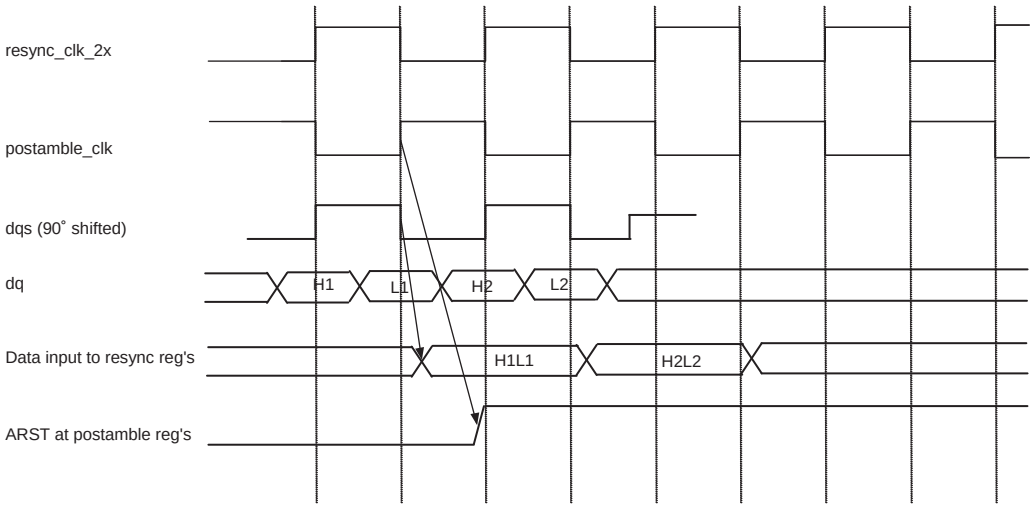


Figure 5–8. Relationship Between Postamble Clock and Resynchronization Clock *(Note 1)*

Note to Figure 5–8:

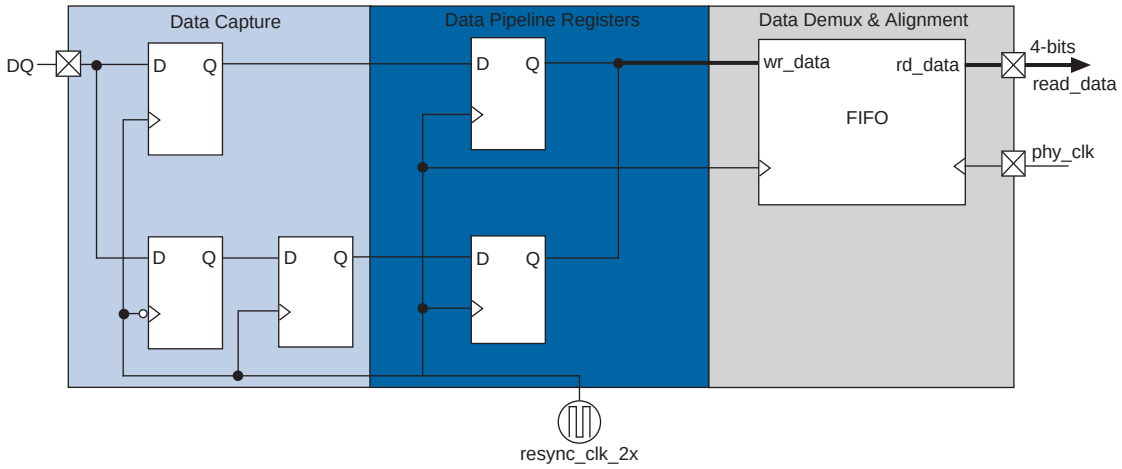
(1) `resync_clk_2x` is delayed further to allow for the I/O element (IOE) to core transition time.

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

Cyclone III Devices

Figure 5–9 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 5–9. 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 (`resync_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 (`resync_clk_2x`) generated from the ALTPLL megafunction.

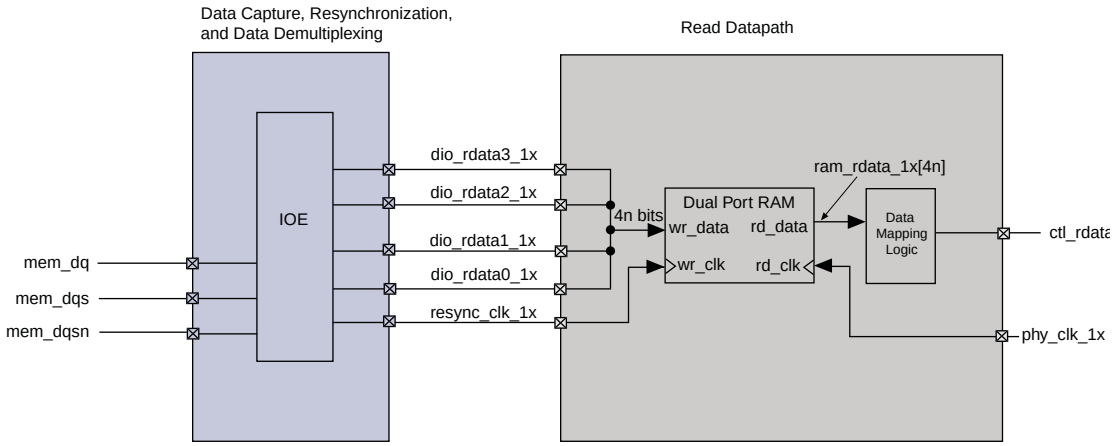
Stratix III and Stratix IV Devices

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

The Stratix IV and Stratix III read datapath (Figure 5–10) consists of two main blocks:

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

Figure 5–10. DDR/DDR2 SDRAM Data Capture and Read Data Mapping in Stratix IV and Stratix III Devices



Note to Figure 5–10:

(1) This figure shows a half-rate variation. For a full-rate controller, `dio_radata2_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

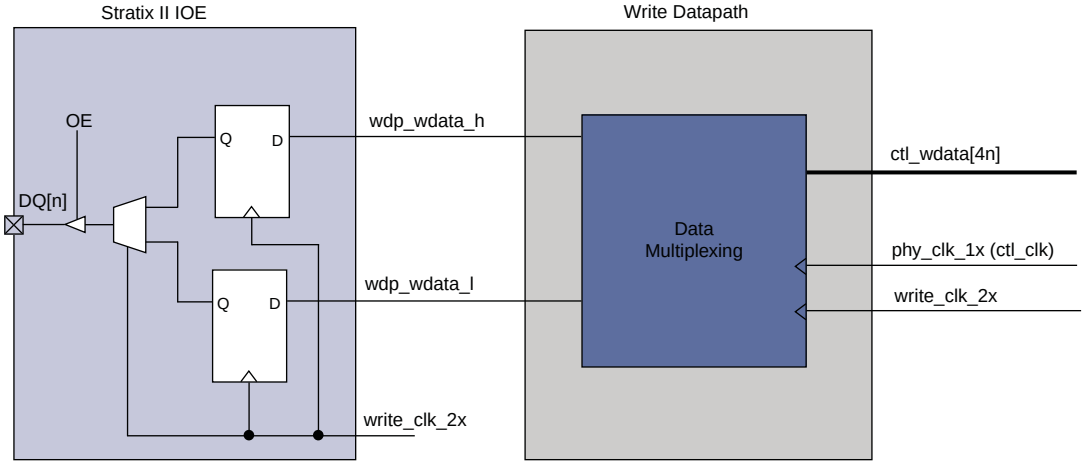
The write datapath logic efficiently transfers data from the HDR memory controller to DDR SDRAM-based memories. The write datapath logic consists of:

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

The memory controller interface outputs 4*n*-bit wide data (`ctl_wdata[4n]`) at half-rate frequency. Figure 5–11 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 5–11. DDR/DDR2 SDRAM Write Datapath in Arria GX, Arria II GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX Devices

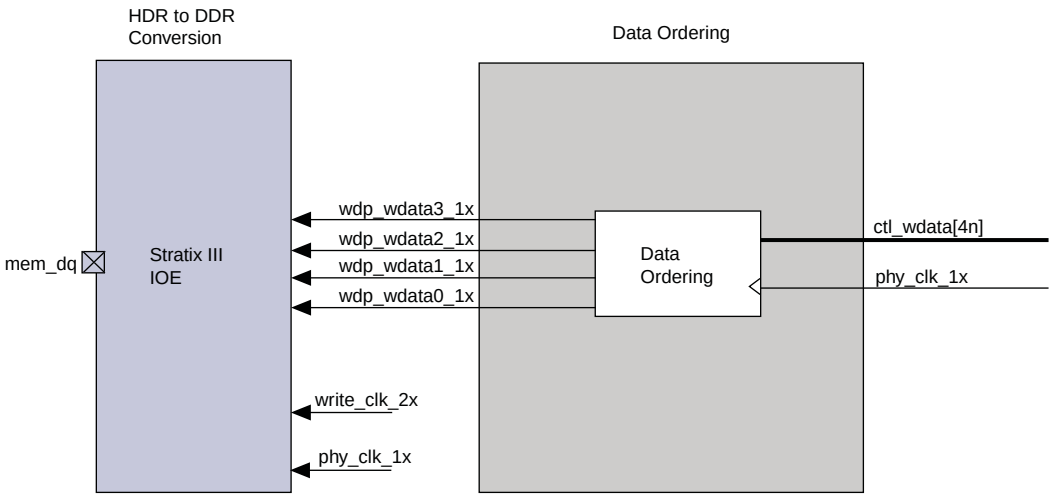


The write datapath for full-rate PHYs is similar to the half-rate PHY. The IOE block is identical to the half-rate PHY. The latency of the write datapath in the full-rate PHY is less than in the half-rate PHY because the full-rate PHY does not have the half-rate-to-full-rate conversion logic.

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 5–12 by `wdp_wdata3_1x, wdp_wdata2_1x, wdp_wdata1_1x,` and `wdp_wdata0_1x`.

Figure 5–12. 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 *(Note 1)*

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

Table 5–5. Interface to the SDRAM Devices *(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.
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 3)

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.
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.

Table 5–6. AFI Signals (Part 2 of 3)

Signal Name	Type	Width *(1)*	Description
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	MEM_IF_DQS_WIDTH × 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	MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2	Write data valid. Generates ctl_wdata and ctl_dm output enables.
ctl_wdata	Input	MEM_IF_DWIDTH × DWIDTH_RATIO	Write data input from the controller to the PHY to generate mem_dq.
ctl_dm	Input	MEM_IF_DM_WIDTH × DWIDTH_RATIO	DM input from the controller to the PHY.
ctl_wlat	Output	5	Required write latency between address/command and write data that is issued to ALTMEMPHY controller local interface. This signal is only valid when the ALTMEMPHY sequencer successfully completes calibration, and does not change at any point during normal operation. The legal range of values for this signal is 0 to 31; and the typical values are between 0 and ten, 0 mostly for low CAS latency DDR memory types.
Read Data Interface			
ctl_doing_rd	Input	MEM_IF_DQS_WIDTH × DWIDTH_RATIO / 2	Doing read input. Indicates that the DDR or DDR2 SDRAM controller is currently performing a read operation. The controller generates ctl_doing_rd to the ALTMEMPHY megafunction. The ctl_doing_rd signal is asserted for one phy_clk cycle for every read command it issues. If there are two read commands, ctl_doing_rd is asserted for two phy_clk cycles. The ctl_doing_rd signal also enables the capture registers and generates the ctl_mem_rdata_valid signal. The ctl_doing_rd signal should be issued at the same time the read command is sent to the ALTMEMPHY megafunction.

Table 5–6. AFI Signals (Part 3 of 3)

Signal Name	Type	Width *(1)*	Description
ctl_rdata	Output	DWIDTH_RATIO × MEM_IF_DWIDTH	Read data from the PHY to the controller.
ctl_rdata_valid	Output	DWIDTH_RATIO/2	Read data valid indicating valid read data on ctl_rdata. This signal is two-bits wide (as only half-rate or 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 ctl_doing_rd and the return of valid read data (ctl_rdata). This is unused by the Altera high-performance controllers do not use ctl_rlat.
Address and Command Interface			
ctl_addr	Input	MEM_IF_ROWADDR_WIDTH × DWIDTH_RATIO / 2	Row address from the controller.
ctl_ba	Input	MEM_IF_BANKADDR_WIDTH × DWIDTH_RATIO / 2	Bank address from the controller.
ctl_cke	Input	MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2	Clock enable from the controller.
ctl_cs_n	Input	MEM_IF_CS_WIDTH ×DWIDTH_RATIO / 2	Chip select from the controller.
ctl_odt	Input	MEM_IF_CS_WIDTH × DWIDTH_RATIO / 2	On-die-termination control from the controller.
ctl_ras_n	Input	DWIDTH_RATIO / 2	Row address strobe signal from the controller.
ctl_we_n	Input	DWIDTH_RATIO / 2	Write enable.
ctl_cas_n	Input	DWIDTH_RATIO / 2	Column address strobe signal from the controller.
ctl_rst_n	Input	DWIDTH_RATIO / 2	Reset from the controller.
Calibration Control and Status Interface			
ctl_mem_clk_disable	Input	MEM_IF_CLK_PAIR_COUNT	When asserted, mem_clk and mem_clk_n are disabled. Unsupported 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.
ctl_cal_req	Input	1	When asserted, a new calibration sequence is started. Currently not supported.
ctl_cal_byte_lane_sel_n	Input	MEM_IF_DQS_WIDTH × MEM_CS_WIDTH	Indicates which DQS groups should be calibrated. Not supported.

Note to Table 5–5:

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

Table 5–7. Other Interface Signals

Signal Name	Type	Width	Description
External DLL Signals			
dqs_delay_ctrl_export	Output	DQS_DELAY_CTL_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_CTL_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_CTL_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.
dgb_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.
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 pll_reconfig_enable is asserted, this input is directly connected to the PLL's phasecounterselect input. Otherwise this input has no effect.
pll_phaseupdown	Input	1	When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phaseupdown input. Otherwise this input has no effect.

Table 5–7. Other Interface Signals

Signal Name	Type	Width	Description
pll_phasestep	Input	1	When pll_reconfig_enable is asserted, this input is directly connected to the PLL's phasestep input. Otherwise this input has no effect.
pll_phase_done	Output	1	Directly connected to the PLL's phase_done 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 soft_reset_n), and reset_request_n is 1, it is safe to reconfigure the PLL. To reconfigure the PLL, set this signal to 1 and use the other pll_reconfig signals to access the PLL. When finished reconfiguring set this signal to 0, and then set the soft_reset_n signal to 1 to bring the PHY out of reset. For this signal to work, the PLL_RECONFIG_PORTS_EN 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 pll_reconfig interface. The same as aux_scan_clk.
pll_reconfig_reset	Output	1	Resynchronised reset to use for any logic accessing the pll_reconfig interface.
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.

Table 5–7. Other Interface Signals

Signal Name	Type	Width	Description
rsu_codvw_size	Output	—	The final centre of data valid window size (rsu_codvw_size) is the number of phases where data was successfully read in the calculation of the resynchronization clock centre of data valid window phase (rsu_codvw_phase).
rsu_read_latency	Output	—	The rsu_read_latency output is then set to the read latency (in phy_clk cycles) using the rsu_codvw_phase resynchronization clock phase. If calibration is unsuccessful then this signal is undefined.
rsu_no_dvw_err	Output	—	If the sequencer sweeps the resynchronization clock across every phase and does not see any valid data at any phase position, then calibration fails and this output is set to 1.
rsu_grt_one_dvw_err	Output	—	If the sequencer sweeps the resynchronization clock across every phase and sees multiple data valid windows, this is indicative of unexpected read data (random bit errors) or an incorrectly configured PLL that must be resolved. Calibration has failed and this output is set to 1.

Notes to Table 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.

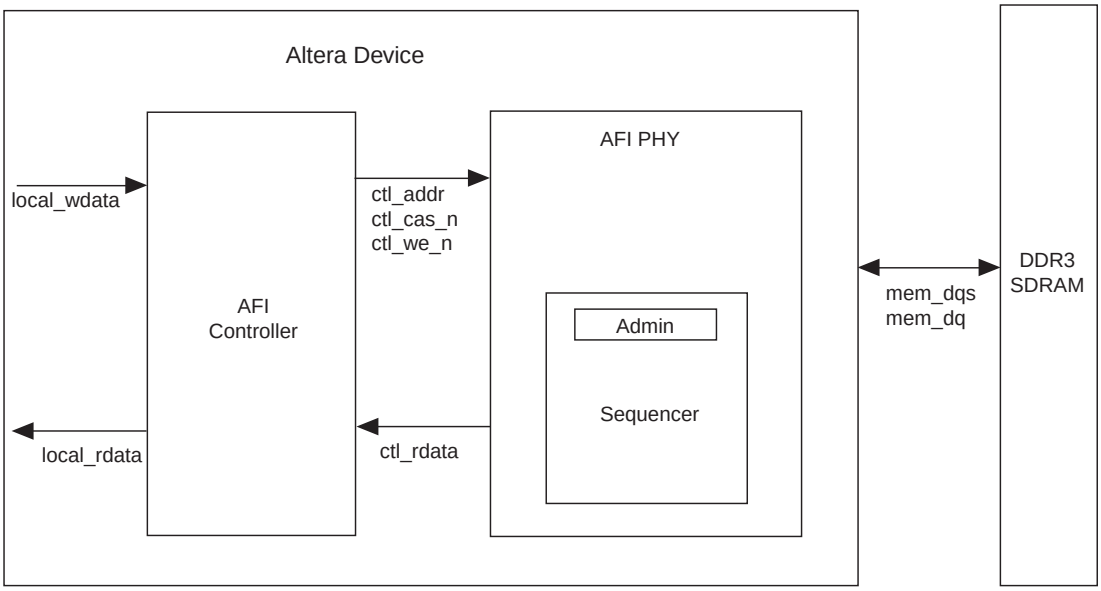
PHY-to-Controller Interfaces

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 5–13 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 5–13. 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 5–14 and Figure 5–15 display the half-rate write operation.

Figure 5–14. Half-Rate Write with Word-Unaligned Data

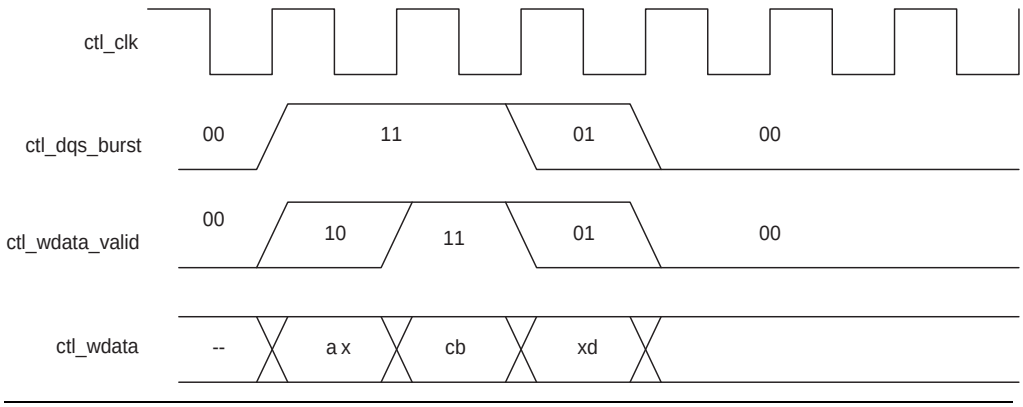


Figure 5–15. Half-Rate Write with Word-Aligned Data

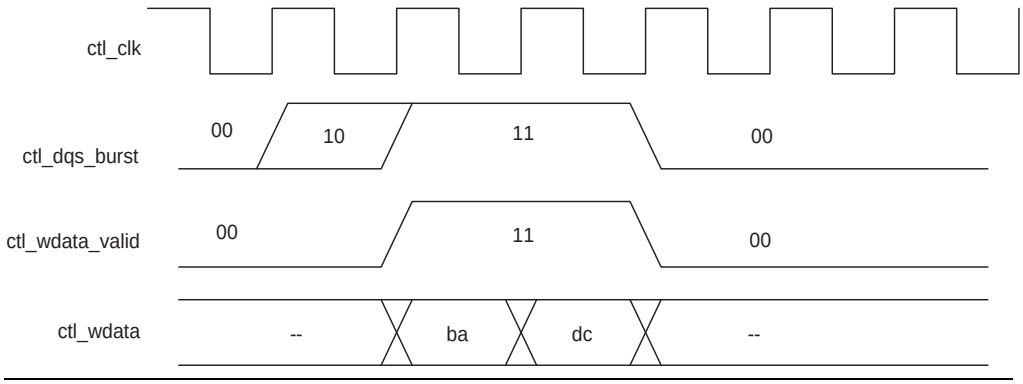
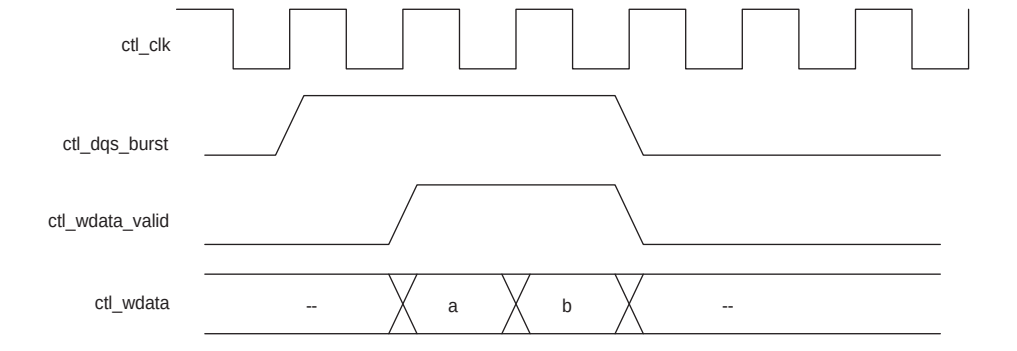


Figure 5–16 shows a full-rate write.

Figure 5–16. Full-Rate Write



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

Figure 5–17 shows full-rate reads; Figure 5–18 shows half-rate reads.

Figure 5–17. Full-Rate Reads

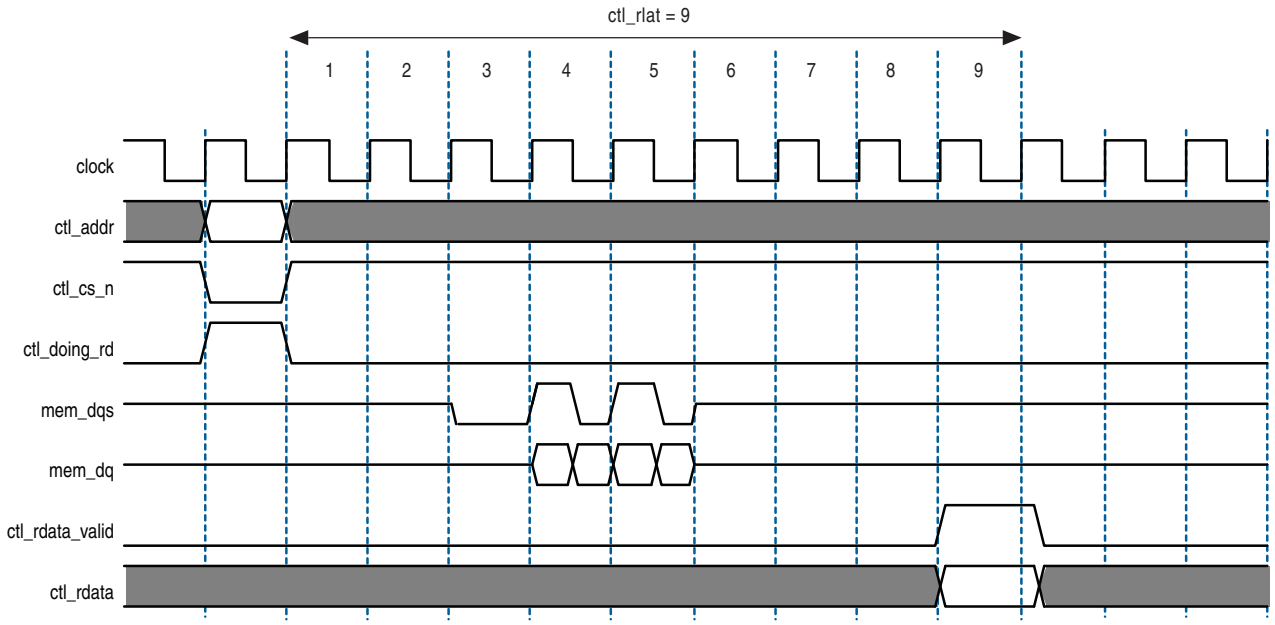


Figure 5–18. Half-Rate Reads

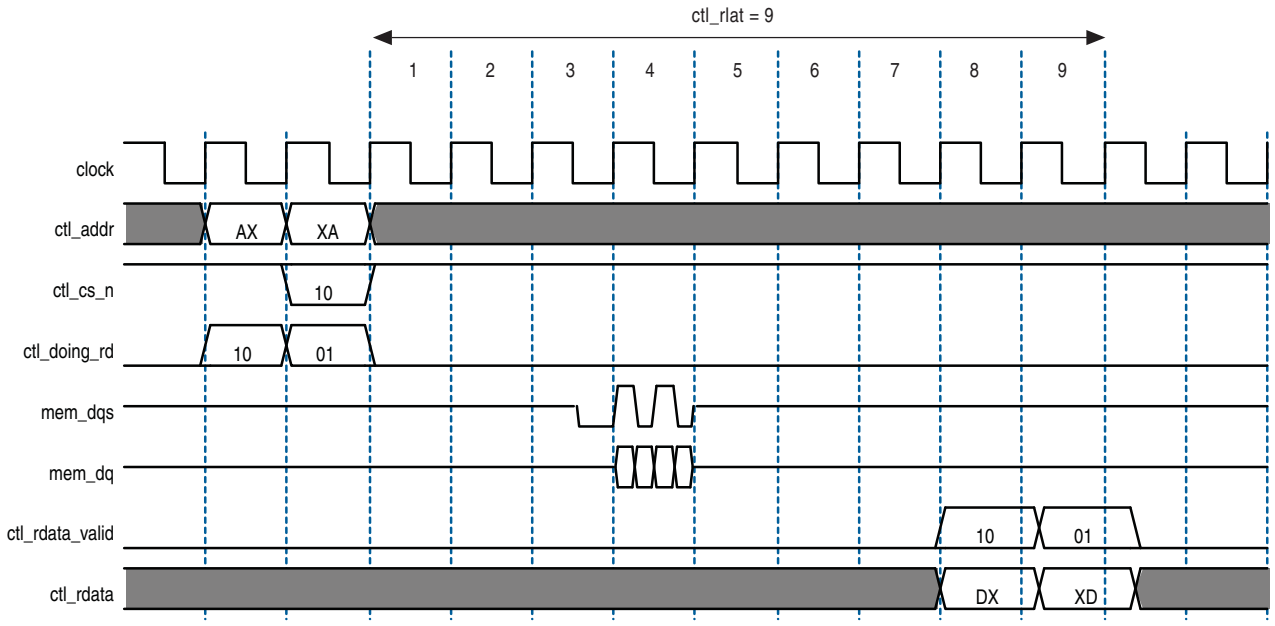


Figure 5–19 and Figure 5–20 show word-aligned writes and reads. In the following read and write examples the data is written to and read from the same address. In each example, `ctl_rdata` and `ctl_wdata` are aligned with controller clock (`ctl_clk`) cycles. All the data in the bit vector is valid at once. For comparison, refer Figure 5–21 and Figure 5–22 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 5–22 on page 5–44 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 (`_l`). `ctl_cs_n[1:0]` = `01` when a write occurs. This alignment is the easiest alignment to design with.
  - Word-unaligned is the opposite, so `ctl_cs_n[1:0]` = `10` when a read or write occurs and the other control and data signals are distributed across consecutive `ctl_clk` cycles.

  The Altera high-performance controllers use word-aligned data only.

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

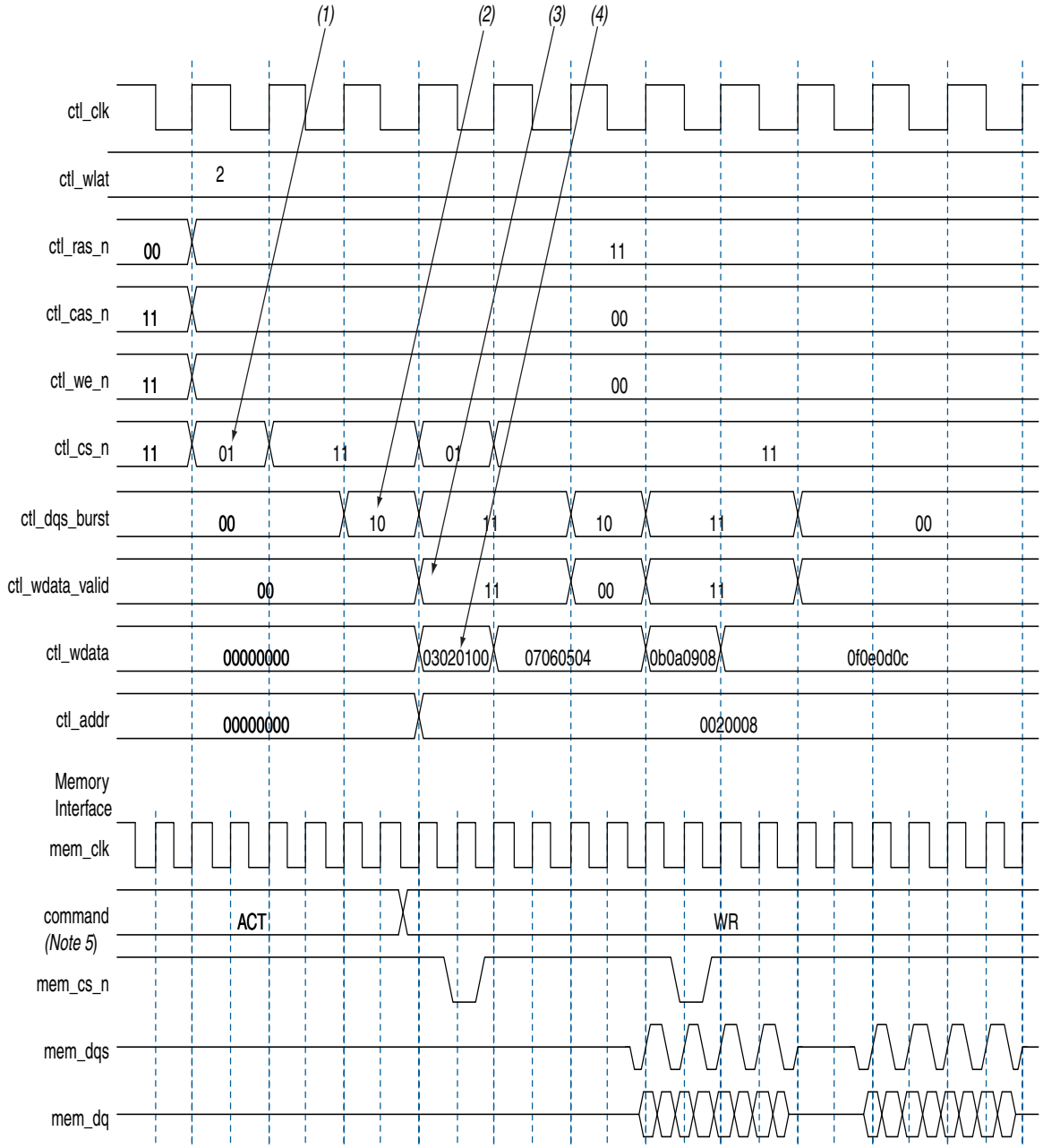
  Word-unaligned reads and writes are only supported on Stratix III and Stratix IV devices.

- 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 5–19 through Figure 5–22 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 5–19. Word-Aligned Writes



Notes to Figure 5–19:

(1) To show the even alignment of `ctl_cs_n`, expand the signal (this convention applies for all other signals).

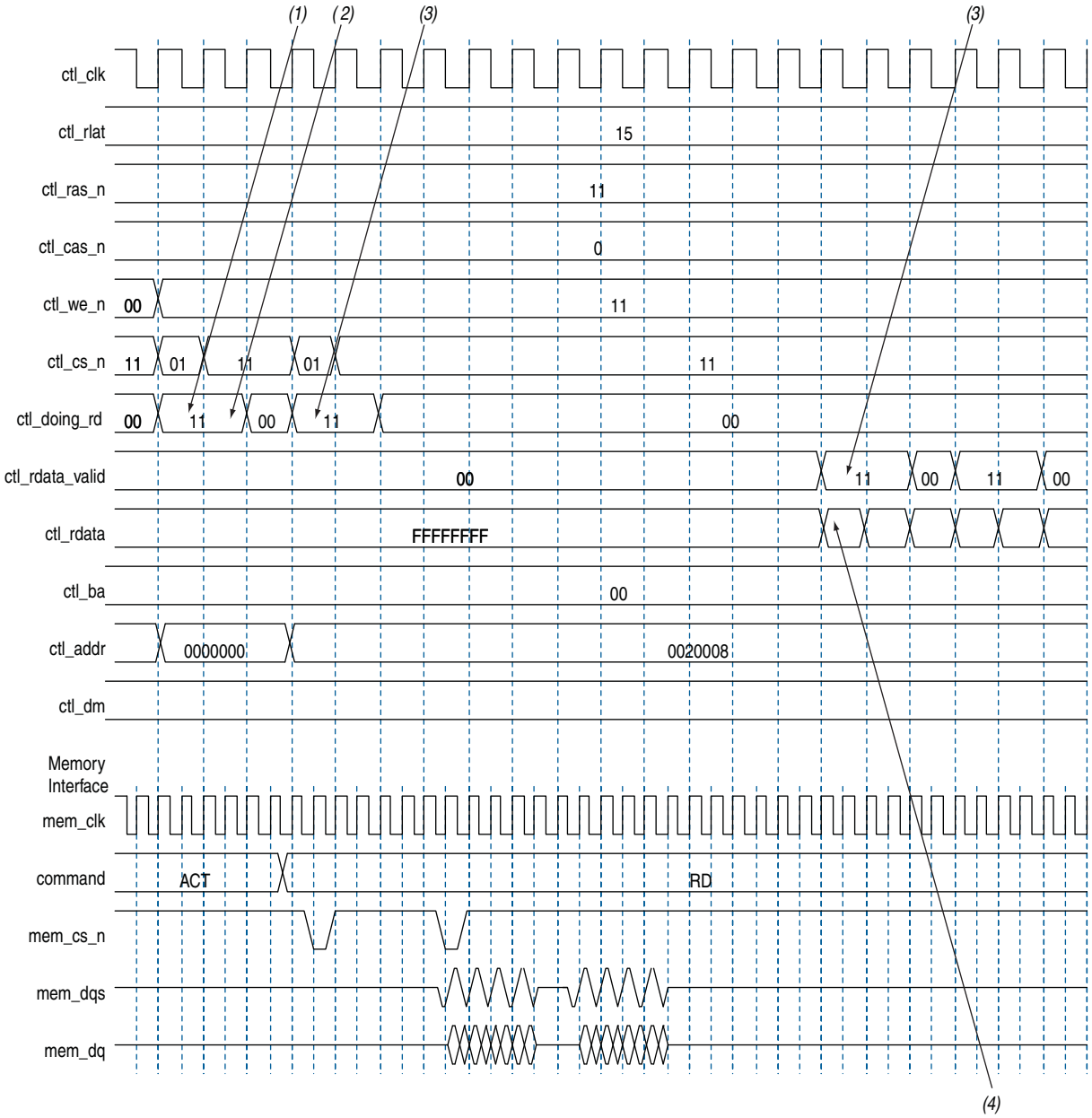
(2) The `ctl_dqs_burst` must go high one memory clock cycle before `ctl_wdata_valid`. Compare with the word-unaligned case.

(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 5–20. Word-Aligned Reads



Notes to Figure 5–20:

(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`.

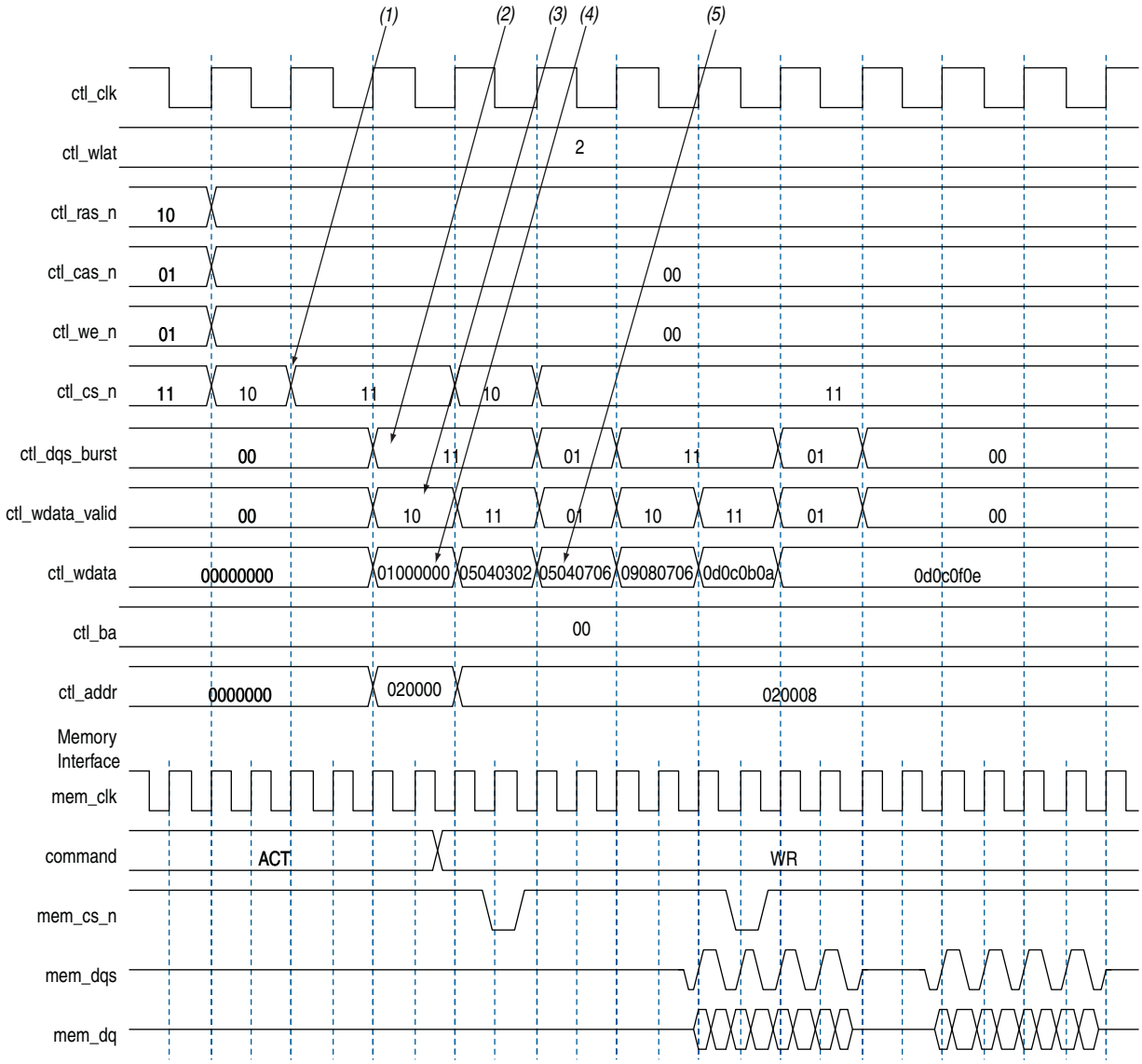
(2) AFI requires that `ctl_doing_rd` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `ctl_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(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 5–21 and Figure 5–22 show spaced word-unaligned writes and reads.

Figure 5–21. Word-Unaligned Writes



Notes to Figure 5–21:

(1) Alternative word-unaligned chip select (`ctl_cs_n`).

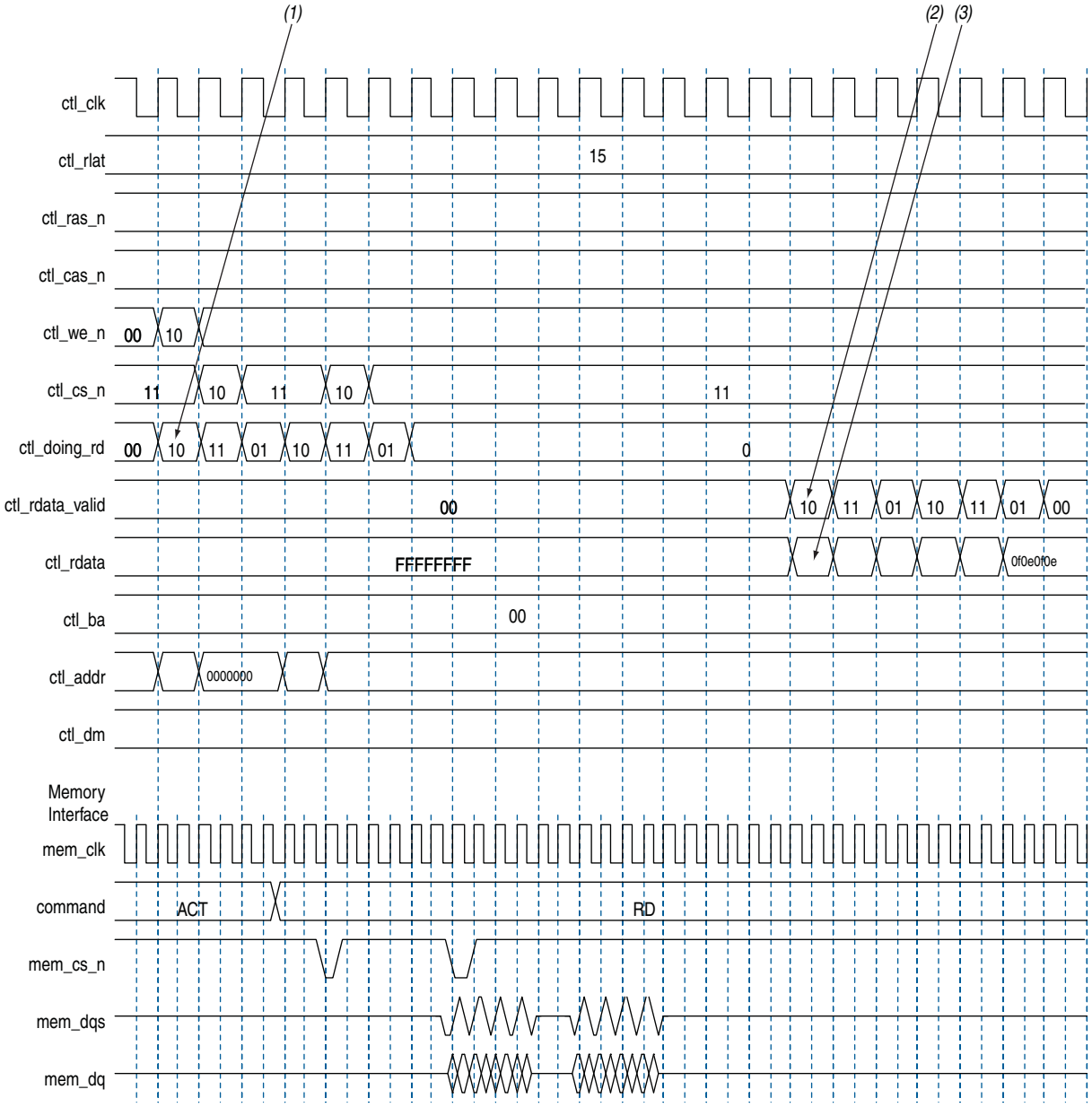
(2) As with word- aligned writes, `ctl_dqs_burst` is asserted one memory clock cycle before `ctl_wdata_valid`. You can see `ctl_dqs_burst` is 11 in the same cycle where `ctl_wdata_valid` is 10. The LSB of these two becomes the first value the signal takes in the `mem_clk` domain. You can see that `ctl_dqs_burst` 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 5–22. Word-Unaligned Reads



Notes to Figure 5–22:

(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 High-Performance Controller MegaCore function targeting your chosen Altera memory devices.
2. Compile and verify the timing. This step is optional; refer to "Compile and Simulate" 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 discuss the important considerations for implementing your own controller with the ALTMEMPHY megafunction. This section describes the design considerations for AFI variants.

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

Clocks and Resets

The ALTMEMPHY megafunction automatically generates a PLL instance, but you must still provide the reference clock input (`pll_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 controllers 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 controllers support only 2T addressing, so PHY timing analysis is performed assuming 2T address and command signals.

Handshake Mechanism Between Read Commands and Read Data

When performing a read, a high-performance controller with the AFI asserts 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 (Rt) control timing

The read latency, `ctl_rlat`, is advertised back to the controller. This signal indicates how long it takes in `ctl_clk` clock cycles from assertion of 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 5–23 and Figure 5–24 show these relationships.

Figure 5–23. Address and Command and Read-Path Timing—Full-Rate Design

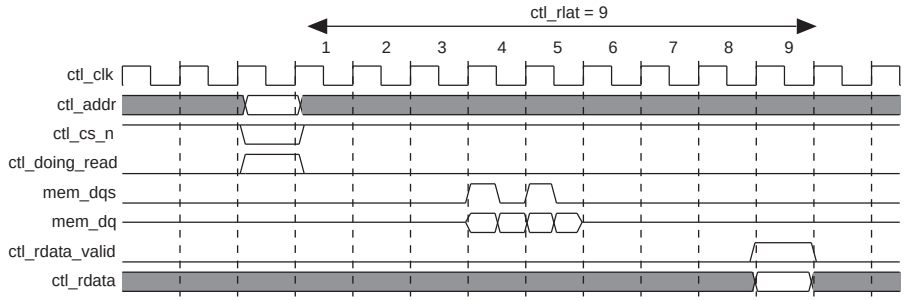
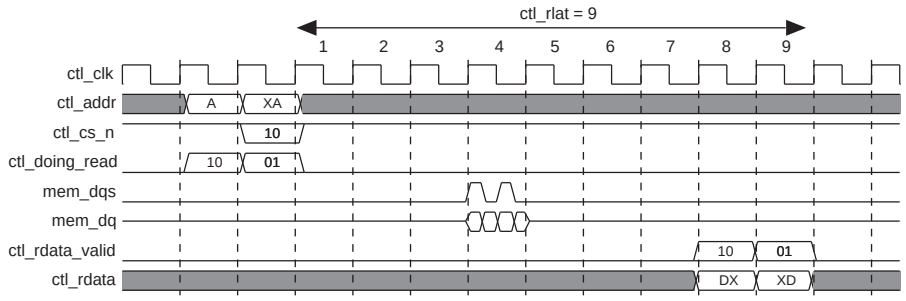


Figure 5–24. Second Read Alignment—Half-Rate Design



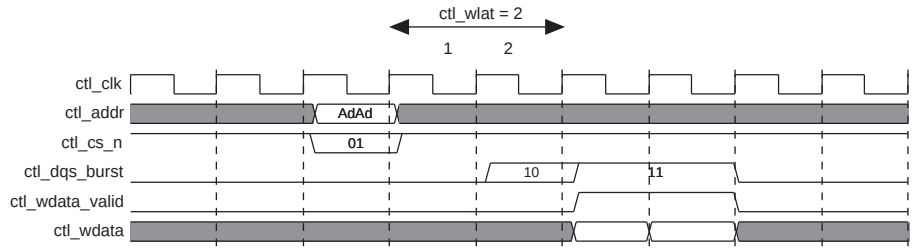
Handshake Mechanism Between Write Commands and Write Data

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 setup to take into account any leveling effects

The ctl_wlat signal is only valid when the calibration has been successfully completed by the ALTMEMPHY sequencer and does not change at any point during normal user mode operation. Figure 5–25 shows the operation of ctl_wlat port.

Figure 5–25. Timing for ctl_dqs_burst, ctl_wdata_valid, Address, and Command—Half-Rate Design



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

Partial Write Operations

As part of the DDR and DDR2 SDRAM memory specifications, you have the option for partial write operations by asserting the DM pins for part of the write signal.

For designs targeting the Stratix III device families, deassert the ctl_wdata_valid signal during partial writes, when the write data is invalid, to save power by not driving the DQ outputs.

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

The I/O difference between Stratix III device families and other device families makes it only possible to use the ctl_dqs_burst signal for the DQS enable in Stratix III devices.

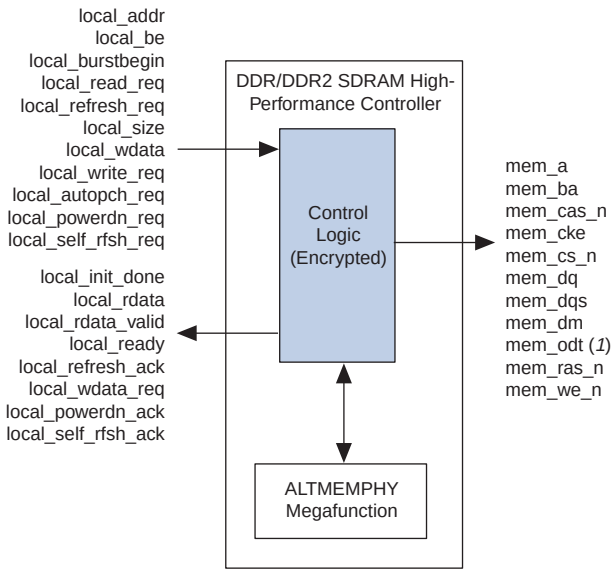
The high-performance controller (HPC) architecture instantiates encrypted control logic and the ALTMEMPHY megafunction. The controller accepts read and write requests from the user on its local interface, using either the Avalon-MM interface protocol or the native interface protocol. It converts these requests into the necessary SDRAM commands, including any required bank management commands. Each read or write request on the Avalon-MM or native interface maps to one SDRAM read or write command. Since the controller uses a memory burst length of 4, read and write requests are always of length 1 on the local interface if the controller is in half-rate mode. In full-rate mode, the controller accepts requests of size 1 or 2 on the local interface. Requests of size 2 on the local interface produce better throughput as whole memory burst is used.

The bank management logic in the controller keeps a row open in every bank in the memory system. For example, a controller configured for a double-sided, 4-bank DDR or DDR2 SDRAM DIMM keeps an open row in each of the 8 banks. The controller allows you to request an auto-precharge read or auto-precharge write, allowing control over whether to keep that row open after the request. You can achieve maximum efficiency when you issue reads and writes to the same bank, with the last access to that bank being an auto-precharge read or write. The controller does not do any access reordering.

Block Description

Figure 6–1 shows the top-level block diagram of the DDR or DDR2 SDRAM HPC.

Figure 6–1. DDR and DDR2 SDRAM HPC Block Diagram

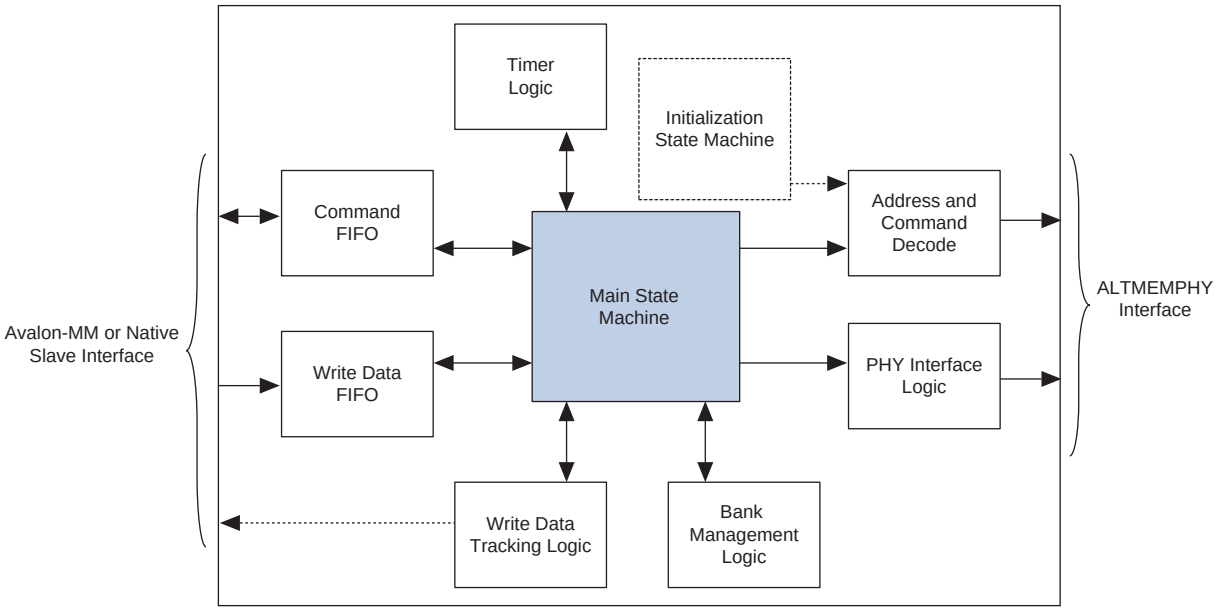


Note to Figure 6–1:

(1) For DDR2 SDRAM HPC only.

Figure 6–2 shows a block diagram of the DDR or DDR2 SDRAM high-performance controller architecture.

Figure 6–2. DDR and DDR2 SDRAM High-Performance Controller Architecture Block Diagram



The blocks in Figure 6–2 on page 6–2 are described in the following sections.

Command FIFO Buffer

This FIFO buffer allows the controller to buffer up to four consecutive read or write commands. It is built from logic elements, and stores the address, read or write flag, and burst count information. If this FIFO buffer fills up, the `local_ready` signal to the user is deasserted until the main state machine takes a command from the FIFO buffer.

Write Data FIFO Buffer

The write data FIFO buffer holds the write data from the user until the main state machine can send it to the ALTMEMPHY megafunction, which does not have a write data buffer. In the Avalon-MM interface mode, the user logic presents a write request, address, burst count, and one or more beats of data at the same time. The write data beats are placed into the FIFO buffer until they are needed. In the native interface mode, the user logic presents a write request, address, and burst count. The controller then requests the correct number of write data beats from the user via the `local_wdata_req` signal, and the user logic must return the write data in the clock cycle after the write data request signal.

This FIFO buffer is sized to be deeper than the command FIFO buffer to prevent it from filling up and interrupting streaming writes.

Write Data Tracking Logic

The write data tracking logic keeps track of the number of write data beats in the FIFO buffer. In the native interface mode, this logic manages how much more data to request from the user logic and issues the `local_wdata_req` signal.

Main State Machine

The main state machine decides what DDR commands to issue based on inputs from the command FIFO buffer, the bank management logic, and the timer logic.

Bank Management Logic

The bank management logic keeps track the current state of each bank. It can keep a row open in every bank in your memory system. The state machine uses the information provided by this logic to decide whether it needs to issue bank management commands before it reads or writes to the bank. The controller always leaves the bank open unless the user requests an auto-precharge read or write. The periodic refresh process also causes all the banks to be closed.

Timer Logic

The timer logic tracks whether the required minimum number of clock cycles has passed since the last relevant command was issued. For example, the timer logic records how many cycles have elapsed since the last activate command so that the state machine knows it is safe to issue a read or write command (t_{RCD}). The timer logic also counts the number of clock cycles since the last periodic refresh command and sends a high priority alert to the state machine if the number of clock cycles has expired.

Initialization State Machine

The initialization state machine issues the appropriate sequence of command to initialize the memory devices. It is specific to DDR and DDR2 as each memory type requires a different sequence of initialization commands.

With the AFI, the ALTMEMPHY megafunction initializes the memory, otherwise the controller is responsible for initializing the memory.

Address and Command Decode

When the state machine wants to issue a command to the memory, it asserts a set of internal signals. The address and command decode logic turns these into the DDR-specific RAS, CAS, and WE commands.

PHY Interface Logic

When the main state machine issues a write command to the memory, the write data for that write burst has to be fetched from the write data FIFO buffer. The relationship between write command and write data depends on the memory type, ALTMEMPHY megafunction interface type, CAS latency, and the full-rate or half-rate setting. The PHY interface logic adjusts the timing of the write data FIFO read request signal so that the data arrives on the external memory interface DQ pins at the correct time.

ODT Generation Logic

The ODT generation logic (not shown) calculates when and for how long to enable the ODT outputs. It also decides which ODT bit to enable, based on the number of chip selects in the system.

- 1 DIMM (1 or 2 Chip Selects)

  In the case of a single DIMM, the ODT signal is only asserted during writes. The ODT signal on the DIMM at `mem_cs[0]` is always used, even if the write command on the bus is to `mem_cs[1]`. In other words, `mem_odt[0]` is always asserted even if there are two ODT signals.

- 2 or more DIMMs

  Table 6–1 shows which ODT signal on the adjacent DIMM is enabled.

Table 6–1. ODT

Write or Read On	ODT Enabled
`mem_cs[0]`or `cs[1]`	`mem_odt[2]`
`mem_cs[2]` or `cs[3]`	`mem_odt[0]`
`mem_cs[4]` or `cs[5]`	`mem_odt[6]`
`mem_cs[6]` or `cs[7]`	`mem_odt[4]`

Low-Power Mode Logic

The low-power mode logic (not shown) monitors the `local_powerdn_req` and `local_self_rfsh_req` request signals. This logic also informs the user of the current low-power state via the `local_powerdn_ack` and `local_self_rfsh_ack` acknowledge signals.

Control Logic

Bus commands control SDRAM devices using combinations of the `mem_ras_n`, `mem_cas_n`, and `mem_we_n` signals. For example, on a clock cycle where all three signals are high, the associated command is a no operation (NOP). A NOP command is also indicated when the chip select signal is not asserted. Table 6–2 shows the standard SDRAM bus commands.

Table 6–2. Bus Commands

Command	Acronym	ras_n	cas_n	we_n
No operation	NOP	High	High	High
Active	ACT	Low	High	High
Read	RD	High	Low	High
Write	WR	High	Low	Low
Burst terminate	BT	High	High	Low
Precharge	PCH	Low	High	Low
Auto refresh	ARF	Low	Low	High
Load mode register	LMR	Low	Low	Low

The DDR or DDR2 SDRAM HPC must open SDRAM banks before they access the addresses in that bank. The row and bank to be opened are registered at the same time as the active (ACT) command. The HPC closes the bank and opens it again if it needs to access a different row. The precharge (PCH) command closes only a bank.

The primary commands used to access SDRAM are read (RD) and write (WR). When the WR command is issued, the initial column address and data word is registered. When a RD command is issued, the initial address is registered. The initial data appears on the data bus 2 to 3 clock cycles later (3 to 5 for DDR2 SDRAM). This delay is the column address strobe (CAS) latency and is due to the time required to read the internal DRAM core and register the data on the bus. The CAS latency depends on the speed of the SDRAM and the frequency of the memory clock. In general, the faster the clock, the more cycles of CAS latency are required. After the initial RD or WR command, sequential reads and writes continue until the burst length is reached or a burst terminate (BT) command is issued. DDR and DDR2 SDRAM devices support burst lengths of 2, 4, or 8 data cycles. The auto-refresh command (ARF) is issued periodically to ensure data retention. This function is performed by the DDR or DDR2 SDRAM high-performance controller.

The load mode register command (LMR) configures the SDRAM mode register. This register stores the CAS latency, burst length, and burst type.

For more information, refer to the specification of the SDRAM that you are using.

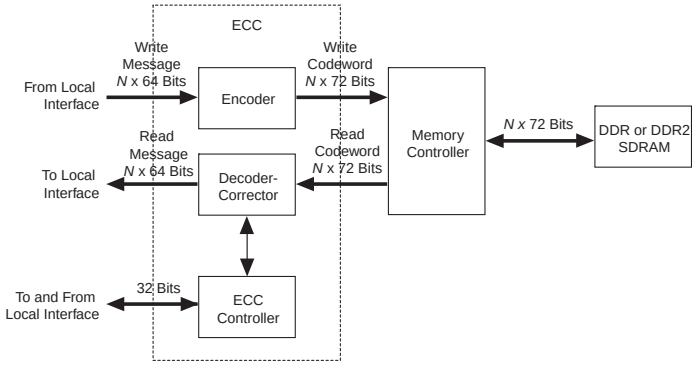
Error Correction Coding (ECC)

The optional ECC comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors and detect double-bit errors. The ECC uses an 8-bit ECC for each 64-bit message. The ECC has the following features:

- Hamming code ECC that encodes every 64-bits of data into 72-bits of codeword with 8-bits of Hamming code parity bits
- Latency:
  - Maximum of 1 or 2 clock delay during writes
  - Minimum 1 or 3 clock delay during reads
- Detects and corrects all single-bit errors. Also the ECC sends an interrupt when the user-defined threshold for a single-bit error is reached.
- Detects all double-bit errors. Also, the ECC counts the number of double-bit errors and sends an interrupt when the user-define threshold for double-bit error is reached.
- Accepts partial writes
- Creates forced errors to check the functioning of the ECC
- Powers up to a ready state

Figure 6–3 shows the ECC block diagram.

Figure 6–3. ECC Block Diagram



The ECC comprises the following blocks:

- The encoder—encodes the 64-bit message to a 72-bit codeword
- The decoder-corrector—decodes and corrects the 72-bit codeword if possible
- The ECC logic—controls multiple encoder and decoder-correctors, so that the ECC can handle different bus widths. Also, it controls the following functions of the encoder and decoder-corrector:
  - Interrupts:
    - Detected and corrected single-bit error
    - Detected double-bit error
    - Single-bit error counter threshold exceeded
    - Double-bit error counter threshold exceeded
  - Configuration registers:
    - Single-bit error detection counter threshold
    - Double-bit error detection counter threshold
    - Capture status for first encountered error or most recent error
    - Enable deliberate corruption of ECC for test purposes
  - Status registers:
    - Error address
    - Error type: single-bit error or double-bit error
    - Respective byte error ECC syndrome
  - Error signal—an error signal corresponding to the data word is provided with the data and goes high if a double-bit error that cannot be corrected occurs in the return data word.

- Counters:
  - Detected and/or corrected single-bit errors
  - Detected double-bit errors

The ECC can instantiate multiple encoders, each running in parallel, to encode any width of data words assuming they are integer multiples of 64.

The ECC operates between the local (native or Avalon-MM interface) and the memory controller.

The ECC has an $N \times 64$-bit (where N is an integer) wide interface, between the local interface and the ECC, for receiving and returning data from the local interface. This interface can be a native interface or an Avalon-MM slave interface, you select the type of interface in the MegaWizard interface.

The ECC has a second interface between the local interface and the ECC, which is a 32-bit wide Avalon-MM slave to control and report the status of the operation of the ECC logic.

The encoded data from the ECC is sent to the memory controller using a $N \times 72$-bit wide Avalon-MM master interface, which is between the ECC and the memory controller.

When testing the DDR SDRAM high-performance controller, you can turn off the ECC.

Interrupts

The ECC issues an interrupt signal when one of the following scenarios occurs:

- The single-bit error counter reaches the set maximum single-bit error threshold value.
- The double-bit error counter reaches the set maximum double-bit error threshold value.

The error counters increment every time the respective event occurs for all N parts of the return data word. This incremented value is compared with the maximum threshold and an interrupt signal is sent when the value is equal to the maximum threshold. The ECC clears the interrupts when you write a `1` to the respective status register. You can mask the interrupts from either of the counters using the control word.

Partial Writes

The ECC supports partial writes. Along with the address, data, and burst signals, the Avalon-MM interface also supports a signal vector that is responsible for byte-enable. Every bit of this signal vector represents a byte on the data-bus. Thus, a `0` on any of these bits is a signal for the controller not to write to that particular location—a partial write.

For partial writes, the ECC performs the following steps:

1. The ECC logic stalls further read or write commands from the Avalon-MM interface when it receives a partial write condition.
2. It simultaneously sends a self-generated read command, for the partial write address, to the memory controller.

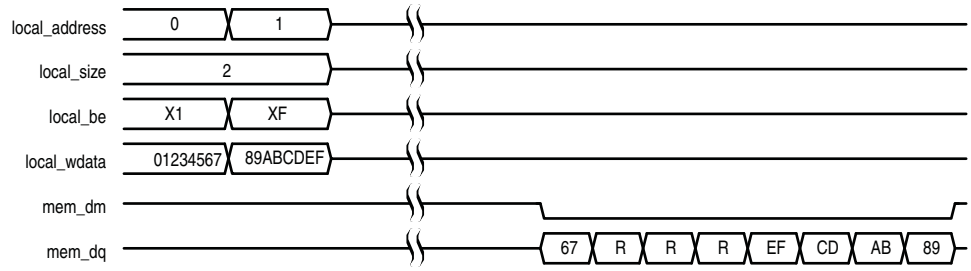
3. Upon receiving the returned read data from the memory controller for the particular address, the decoder decodes the data, checks for errors, and then sends it to the ECC logic.
4. The ECC logic merges the corrected or correct dataword with the incoming information.
5. The ECC logic sends the updated dataword to the encoder for encoding, and then sends updated dataword to the memory controller with a write command.
6. The ECC logic stops stalling the commands from the Avalon-MM interface so that the logic can receive new commands.

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the single-bit error is corrected first, the single-bit error counter is incremented and then a partial write is performed to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the double-bit error counter is incremented and an interrupt is sent through the Avalon-MM interface. The new write word is not written to its location. A separate field in the interrupt status register highlights this condition.

Figure 6–4 and Figure 6–5 show the partial write operation for HPC in full-rate and half-rate mode. The full-rate HPC supports a local size of 1 and 2, and the half-rate HPC supports a local size of 1 only.

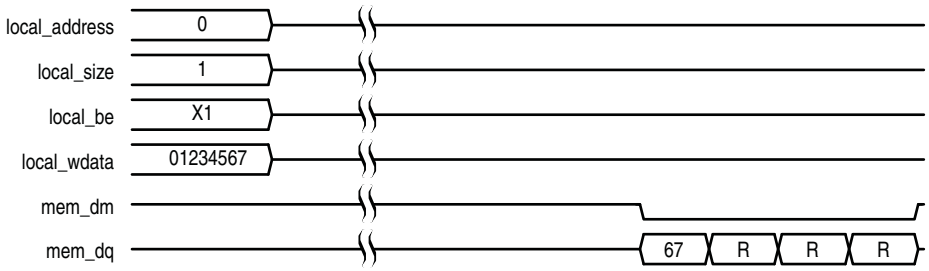
Figure 6–4. Partial Write for HPC—Full Rate



Note to Figure 6–4:

(1) R represents the internal read-back memory data during the read-modify-write process.

Figure 6–5. Partial Write for HPC—Half Rate



Note to Figure 6–5:

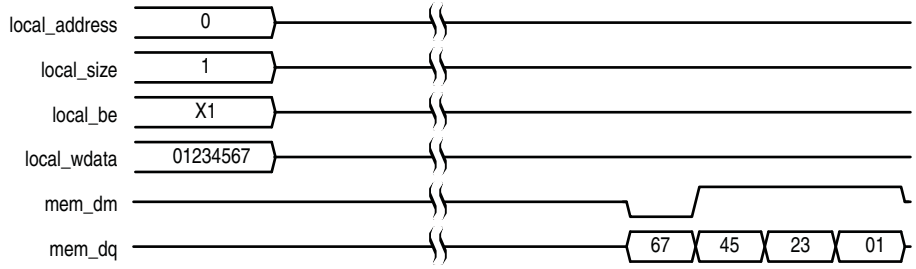
(1) R represents the internal read-back memory data during the read-modify-write process.

Partial Bursts

DIMMs that do not have the DM pins do not support partial bursts. A minimum of four words must be written to the memory at the same time.

Figure 6–6 shows the partial burst operation for HPC.

Figure 6–6. Partial Burst for HPC



ECC Latency

Using the ECC results in the following latency changes:

- Local Burst Length 1
- Local Burst Length 2

Local Burst Length 1

For a local burst length of 1, the write latency increases by one clock cycle; the read latency increases by one clock cycle (including checking and correction).

A partial write results in a read followed by write in the ECC logic, so latency depends on the time the controller takes to fetch the data from the particular address.

Table 6–3 shows the relationship between burst lengths and rate.

Table 6–3. Burst Lengths and Rates

Local Burst Length	Rate	Memory Burst Length
1	Half	4
2	Full	4

Local Burst Length 2

For a local burst length of 2, the write latency increases by two clock cycles; the read latency increases by one clock cycle (including checking and correction).

A partial write results in a read followed by write in the ECC logic, so latency depends on the time the controller takes to fetch the data from the particular address.

For a single-bit error, the automatic correction of memory takes place without stalling the read cycle (if enabled), which stalls further commands to the ECC logic, while the correction takes place.

ECC Registers

Table 6–4 shows the ECC registers.

Table 6–4. ECC Registers (Part 1 of 2)

Name	Address	Size (Bits)	Attribute	Default	Description
Control word specifications	00	32	R/W	0000000F	This register contains all commands for the ECC functioning.
Maximum single-bit error counter threshold	01	32	R/W	00000001	The single-bit error counter increments (when a single-bit error occurs) until the maximum threshold, as defined by this register. When this threshold is crossed, the ECC generates an interrupt.
Maximum double-bit error counter threshold	02	32	R/W	00000001	The double-bit error counter increments (when a double-bit error occurs) until the maximum threshold, as defined by this register. When this threshold is crossed, the ECC generates an interrupt.
Current single-bit error count	03	32	RO	00000000	The single-bit error counter increments (when a single-bit error occurs) until the maximum threshold. You can find the value of the count by reading this status register.
Current double-bit error count	04	32	RO	00000000	The double-bit error counter increments (when a double-bit error occurs) until the maximum threshold. You can find the value of the count by reading this status register.
Last or first single-bit error error address	05	32	RO	00000000	This status register stores the last single-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored.
Last or first double-bit error error address	06	32	RO	00000000	This status register stores the last double-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored.
Last single-bit error error data	07	32	RO	00000000	This status register stores the last single-bit error error data word. As the data word is an *M*th multiple of 64, the data word is stored in a 2*N*-deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear.

Table 6–4. ECC Registers (Part 2 of 2)

Name	Address	Size (Bits)	Attribute	Default	Description
Last single-bit error syndrome	08	32	RO	00000000	This status register stores the last single-bit error syndrome, which specifies the location of the error bit on a 64-bit data word. As the data word is an *M*th multiple of 64, the syndrome is stored in a *N* deep, 8-bit wide FIFO buffer where each syndrome represents errors in every 64-bit part of the data word. The register gets updated with the correct syndrome depending on which part of the data word is shown on the last single-bit error error data register. It can be cleared individually by using the control word clear.
Last double-bit error error data	09	32	RO	00000000	This status register stores the last double-bit error error data word. As the data word is an *M*th multiple of 64, the data word is stored in a 2*N* deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear.
Interrupt status register	0A	5	RO	00000000	This status register stores the interrupt status in four fields (refer to Table 6–6). These status bits can be cleared by writing a `1` in the respective locations.
Interrupt mask register	0B	5	WO	00000001	This register stores the interrupt mask in four fields (refer to Table 6–7).
Single-bit error location status register	0C	32	R/W	00000000	This status register stores the occurrence of single-bit error for each 64-bit part of the data word in every bit (refer to Table 6–8). These status bits can be cleared by writing a `1` in the respective locations.
Double-bit error location status register	0D	32	R/W	00000000	This status register stores the occurrence of double-bit error for each 64-bit part of the data word in every bit (refer to Table 6–9). These status bits can be cleared by writing a `1` in the respective locations.

ECC Register Bits

Table 6–5 shows the control word specification register.

Table 6–5. Control Word Specification Register (Part 1 of 2)

Bit	Name	Direction	Description
0	Count single-bit error	Decoder-corrector	When 1, count single-bit errors.
1	Correct single-bit error	Decoder-corrector	When 1, correct single-bit errors.

Table 6–5. Control Word Specification Register (Part 2 of 2)

Bit	Name	Direction	Description
2	Double-bit error enable	Decoder-corrector	When 1, detect all double-bit errors and increment double-bit error counter.
3	Reserved	N/A	Reserved for future use.
4	Clear all status registers	Controller	When 1, clear counters single-bit error and double-bit error status registers for first and last error address.
5	Reserved	N/A	Reserved for future use.
6	Reserved	N/A	Reserved for future use.
7	Counter clear on read	Controller	When 1, enables counters to clear on read feature.
8	Corrupt ECC enable	Controller	When 1, enables deliberate ECC corruption during encoding, to test the ECC.
9	ECC corruption type	Controller	When 0, creates single-bit errors in all ECC codewords; when 1, creates double-bit errors in all ECC codewords.
10	First or last error	Controller	When 1, stores the first error address rather than the last error address of single-bit error or double-bit error.
11	Clear interrupt	Controller	When 1, clears the interrupt.

Table 6–6 shows the interrupt status register.

Table 6–6. Interrupt Status Register

Bit	Name	Description
0	Single-bit error	When 1, single-bit error occurred.
1	Double-bit error	When 1, double-bit error occurred.
2	Maximum single-bit error	When 1, single-bit error maximum threshold exceeded.
3	Maximum double-bit error	When 1, double-bit error maximum threshold exceeded.
4	Double-bit error during read-modify-write	When 1, double-bit error occurred during a read modify write condition. (partial write).
Others	Reserved	Reserved.

Table 6–7 shows the interrupt mask register.

Table 6–7. Interrupt Mask Register (Part 1 of 2)

Bit	Name	Description
0	Single-bit error	When 1, masks single-bit error.
1	Double-bit error	When 1, masks double-bit error.
2	Maximum single-bit error	When 1, masks single-bit error maximum threshold exceeding condition.
3	Maximum double-bit error	When 1, masks double-bit error maximum threshold exceeding condition.

Table 6–7. Interrupt Mask Register (Part 2 of 2)

Bit	Name	Description
4	Double-bit error during read-modify-write	When 1, masks interrupt when double-bit error occurs during a read-modify-write condition. (partial write).
Others	Reserved	Reserved.

Table 6–8 shows the single-bit error location status register.

Table 6–8. Single-Bit Error Location Status Register

Bit	Name	Description
Bits *N* – 1 down to 0	Interrupt	When 0, no single-bit error; when 1, single-bit error occurred in this 64-bit part.
Others	Reserved	Reserved.

Table 6–9 shows the double-bit error location status register.

Table 6–9. Double-Bit Error Location Status Register

Bit	Name	Description
Bits N-1 down to 0	Cause of Interrupt	When 0, no double-bit error; when 1, double-bit error occurred in this 64-bit part.
Others	Reserved	Reserved.

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. The example top-level file consists of the DDR or DDR2 SDRAM HPC, 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 6–7 shows the testbench and the example top-level file.

Figure 6–7. Testbench and Example Top-Level File

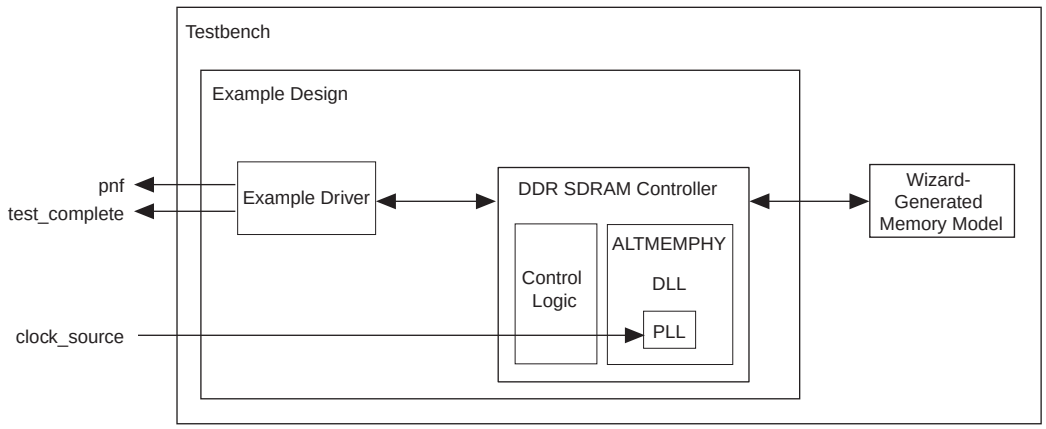


Table 6–10 describes the files that are associated with the example top-level file and the testbench.

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 (more than 2K address spaces) designs, because simulators need more memory than what is available on a typical system.

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.

It 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 that the controller 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

Top-level Signals Description

Table 6–12 shows the clock and reset signals.

Table 6–12. Clock and Reset Signals

Name	Direction	Description
global_reset_n	Input	The asynchronous reset input to the controller. All other reset signals are derived from resynchronized versions of this signal. This signal holds the complete ALTMEMPHY megafunction, including the PLL, in reset while low.
pll_ref_clk	Input	The reference clock input to PLL.
soft_reset_n	Input	Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. It is asserted to cause a complete reset to the PHY, but not to the PLL used in the PHY.
oct_ctl_rs_value	Input	ALTMEMPHY signal that specifies the serial termination value. Should be connected to the ALT_OCT megafunction output seriesterminationcontrol.
oct_ctl_rt_value	Input	ALTMEMPHY signal that specifies the parallel termination value. Should be connected to the ALT_OCT megafunction output parallelterminationcontrol.
dqs_delay_ctrl_import	Input	Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the export port on the ALTMEMPHY instance with a DLL to the import port on the other ALTMEMPHY instance.

Table 6–13 shows the DDR and DDR2 SDRAM HPC local interface signals.

Table 6–13. Local Interface Signals (Part 1 of 4)

Signal Name	Direction	Description

`local_address[]`	Input	Memory address at which the burst should start.
		■ **Full rate controllers**
		The width of this bus is sized using the following equation:
		For one chip select:
		width = bank bits + row bits + column bits – 1
		For multiple chip selects:
		width = chip bits + bank bits + row bits + column bits – 1
		If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 24 bits wide. To map `local_address` to bank, row and column address:
		`local_address[23:22]` = bank address `[1:0]`
		`local_address[21:9]` = row address `[13:0]`
		`local_address [8:0]` = `col_address[9:1]`
		The least significant bit (LSB) of the column address (multiples of four) on the memory side is ignored, because the local data width is twice that of the memory data bus width.
		■ **Half rate controllers**
		The width of this bus is sized using the following equation:
		For one chip select:
		width = bank bits + row bits + column bits – 2
		For multiple chip selects:
		width = chip bits + bank bits + row bits + column bits – 2
		If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 23 bits wide. To map `local_address` to bank, row and column address:
		`local_address` is 23 bits wide
		`local_address[22:21]` = bank address
		`local_address[20:8]` = row address `[13:0]`
		`local_address [7:0]` = `col_address[9:2]`
		Two LSBs of the column address on the memory side are ignored, because the local data width is four times that of the memory data bus width.
		☞ You can get the information on address mapping from the *<variation_name>*_**example_top.v** or **vhd** file.

Table 6–13. Local Interface Signals (Part 2 of 4)

Signal Name	Direction	Description
local_be[]	Input	Byte-enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low. 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. Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE > Local_be = < 1100 >< 0110 >< 1010 > These values map to: Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC> Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 >
local_burstbegin	Input	The Avalon burst begin strobe, which indicates the beginning of an Avalon burst. This signal is only available when the local interface is an Avalon-MM interface and the memory burst length is greater than 2. Unlike all other Avalon-MM signals, the burst begin signal does not stay asserted if local_ready is deasserted. 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 has deasserted the local_ready signal. This signal is sampled at the rising edge of phy_clk when the local_write_req signal 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. For read transactions, assert this signal for one clock cycle when read request is asserted and the 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 the local_ready signal becomes high again.
local_read_req	Input	Read request signal. You cannot assert read request and write request signals at the same time.
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 ganging together multiple refresh commands. Refresh requests take priority over read and write requests unless they are already being processed.
local_size[]	Input	Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The range of values depend on the memory burst length and whether you select full or half rate in the wizard. If you select a memory burst length 4 and half rate, the local burst length is 1 and so local_size should always be driven with 1. If you select a memory burst length 4 and full rate, the local burst length is 2 and you should set the local_size to either 1 or 2 for each read or write request.
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 signals at the same time.

Table 6–13. Local Interface Signals (Part 3 of 4)

Signal Name	Direction	Description
local_autopch_req	Input	User control of precharge. If **Enable Auto-Precharge Control** is turned on, local_autopch_req becomes available and you can request the controller to issue an auto-precharge write or auto-precharge read command. 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 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.
local_powerdn_req	Input	User control of the power-down feature. If **Enable Power Down Controls** option is enabled, you can request that the controller place the memory devices into a power-down state as soon as it can without violating the relevant timing parameters and responds by asserting the local_powerdn_ack signal. You can hold the memory in the power-down state by keeping this signal asserted. The controller brings the memory out of the power-down state to issue periodic auto-refresh commands to the memory at the appropriate interval if you hold it in the power-down state. You can release the memory from the power-down state at any time by deasserting the local_powerdn_ack signal once it has successfully brought the memory out of the power-down state.
local_self_rfsh_req	Input	User control of the self-refresh feature. If **Enable Self-Refresh Controls** option is enabled, 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 the local_self_rfsh_ack signal. 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 the local_self_rfsh_req signal and the controller responds by deasserting the local__self_rfsh_ack signal once it has successfully brought the memory out of the self-refresh state.
phy_clk	Output	The system clock that the ALTMEMPHY megafunction provides to the user. All user inputs to and outputs from the DDR high-performance controller must be synchronous to this clock.
reset_phy_clk_n	Output	The reset signal that the ALTMEMPHY megafunction provides to the user. It is asserted asynchronously and deasserted synchronously to phy_clk clock domain.
aux_full_rate_clk	Output	An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate mode, this clock is twice the frequency of the phy_clk and can be used whenever a 2x clock is required. In full-rate mode, this clock is driven by the same PLL output as the phy_clk signal.
aux_half_rate_clk	Output	An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate mode, this clock is half the frequency of the phy_clk and can be used, for example to clock the user side of a half-rate bridge. In half-rate mode, this clock is driven by the same PLL output as the phy_clk signal.
dll_reference_clk	Output	Reference clock to feed to an externally instantiated DLL.

Table 6–13. Local Interface Signals (Part 4 of 4)

Signal Name	Direction	Description
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 it is advised to detect a reset request on a falling edge rather than by level detection.
local_init_done	Output	When the memory initialization, training, and calibration are complete, the ALTMEMPHY sequencer asserts the ctrl_usr_mode_rdy signal to the memory controller, which then asserts this signal to indicate that the memory interface is ready to be used. Read and write requests are still accepted before local_init_done is asserted, however they are not issued to the memory until it is safe to do so. This signal does not indicate that the calibration is successful. To find out if the calibration is successful, look for the calibration signal, resynchronization_successful, or postamble_successful for Stratix IV devices.
local_rdata[]	Output	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.
local_rdata_error	Output	Asserted if the current read data has an error. This signal is only available if the **Enable Error Detection and Correction Logic** option is turned on. This signal is asserted together with the local_rdata_valid signal. If the controller encounters double-bit errors, no correction is made and the controller asserts this signal.
local_rdata_valid	Output	Read data valid signal. The local_rdata_valid signal indicates that valid data is present on the read data bus.
local_ready	Output	The local_ready signal indicates that the DDR or DDR2 SDRAM high-performance controller is ready to accept request signals. If local_ready is asserted in the clock cycle that a read or write request is asserted, that request has been accepted. The local_ready signal is deasserted to indicate that the DDR or DDR2 SDRAM high-performance controller cannot accept any more requests. The controller is able to buffer four read or write requests.
local_refresh_ack	Output	Refresh request acknowledge, which is asserted for one clock cycle every time a refresh is issued. Even if the **Enable User Auto-Refresh Controls** option is not selected, local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command.
local_wdata_req	Output	Write data request signal, which indicates to the local interface that it should present valid write data on the next clock edge. This signal is only required when the controller is operating in **Native interface** mode.
local_powerdn_ack	Output	Power-down request acknowledge signal. This signal is asserted and deasserted in response to the local_powerdn_req signal from the user.
local_self_rfsh_ack	Output	Self-refresh request acknowledge signal. This signal is asserted and deasserted in response to the local_self_rfsh_req signal from the user.

Table 6–14 shows the DDR and DDR2 SDRAM interface signals.

Table 6–14. DDR and DDR2 SDRAM Interface Signals

Signal Name	Direction	Description
mem_dq[]	Bidirectional	Memory data bus. This bus is half the width of the local read and write data busses.
mem_dqs[]	Bidirectional	Memory data strobe signal, which writes data into the DDR or DDR2 SDRAM and captures read data into the Altera device.
mem_clk *(1)*	Bidirectional	Clock for the memory device.
mem_clk_n *(1)*	Bidirectional	Inverted clock for the memory device.
mem_a[]	Output	Memory address bus.
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, for DDR2 SDRAM only.
mem_ras_n	Output	Memory row address strobe signal.
mem_we_n	Output	Memory write enable signal.

Note to Table 6–14:

(1) The mem_clk signals are output only signals from the FPGA. However, in the Quartus II software they must be defined as bidirectional (INOUT) I/Os to support the mimic path structure that the ALTMEMPHY megafunction uses.

Table 6–15 shows the ECC logic signals.

Table 6–15. ECC Logic Signals

Signal Name	Direction	Description
ecc_addr[]	Input	Address for ECC logic.
ecc_be[]	Input	ECC logic byte enable.
ecc_read_req	Input	Read request for ECC logic.
ecc_wdata[]	Input	ECC logic write data.
ecc_write_req	Input	Write request for ECC logic.
ecc_interrupt	Output	Interrupt from ECC logic.
ecc_rdata[]	Output	Return data from ECC logic.

The high-performance controller II (HPC II) architecture is an upgraded controller with higher efficiency and more features than the HPC. HPC II is recommended for all new designs.

HPC II is pin-out compatible with your existing DDR high-performance designs. HPC II has the following additional features:

- Higher efficiency with in-order read and write commands, and out-of-order bank management command.
- Run-time programmability to configure the behavior of the controller.
- Half-rate bridge option to reduce memory access latency.
- Integrated burst adapter supporting a range of burst sizes on the local interface.
- Integrated ECC, supporting 40-bit and 72-bit interfaces with partial word writes and optional write back on error.
- Support for multi-rank UDIMMs and RDIMMs.

Upgrading from HPC to HPC II

If you want to migrate your designs from the existing HPC to the more efficient HPC II, you have to ensure that you do the following:

- In the **Preset Editor** dialog box, assign the following HPC II timing parameters to match your memory specification. Set these parameters according to the memory datasheet:
  - t_{FAW}
  - t_{RRD}
  - t_{RTP}

  For example, for Micron DDR3-800 datasheet, t_{FAW}=40 ns, t_{RRD}=10 ns, t_{RTP}=10 ns.
- If you are using the Avalon-MM interface, HPC II replaces the port interface level for the AFI and Avalon interface without requiring any top-level change.
- The side-band signals differ slightly for HPC II. If you use these signals, you need to perform the following steps.
  - `local_refresh_req`

    You need to drive an additional active high signal, `local_refresh_chip`, to control which chip to issue the user-refresh to.
  - `local_powerdn_req`

    The user-manual power signal is no longer supported in HPC II. Instead, you can select auto power-down on the **Controller Settings** tab in the MegaWizard Plug-In Manager, and specify the desired time-out (*n* cyles) after which the controller automatically powers down the memory.

- Because HPC II only supports a specific memory burst length, you must update the memory burst length to match the controller settings in Table 7–1.

Table 7–1. Burst Length Support

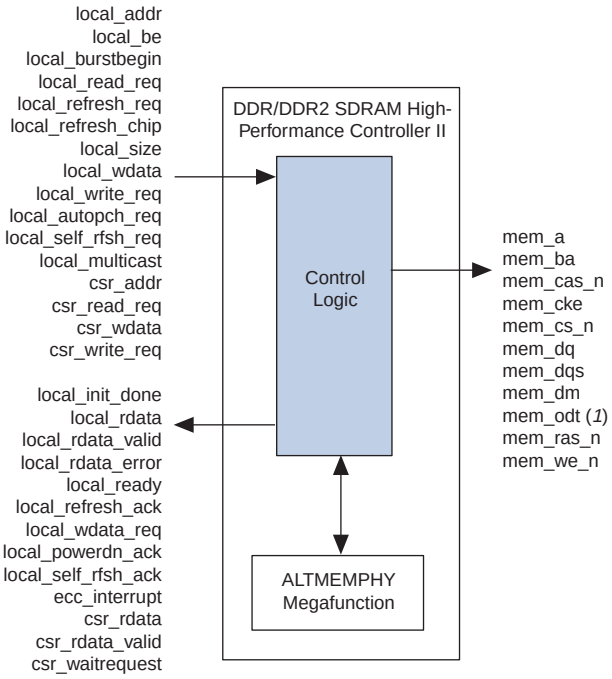
Controller	HPC	HPC II
DDR	Burst length of 2 and 4	Burst length of 4 in full-rate mode, and burst length of 8 in half-rate mode.
DDR2	Burst length of 4	

- Because HPC II supports arbitrary user burst length ranging from of 1 to 64, you can adjust the `max_local_size` value in HPC II. Adjusting the maximum local size value changes the width of the `local_size` signal. The maximum `local_size` signal value is 2^{n-1}, where n is the width of the `local_size` signal. HP has a fixed `local_size` signal width of either 1 or 2.

Block Description

Figure 7–1 shows the top-level block diagram of the DDR or DDR2 SDRAM HPC II.

Figure 7–1. DDR and DDR2 SDRAM HPC II Block Diagram

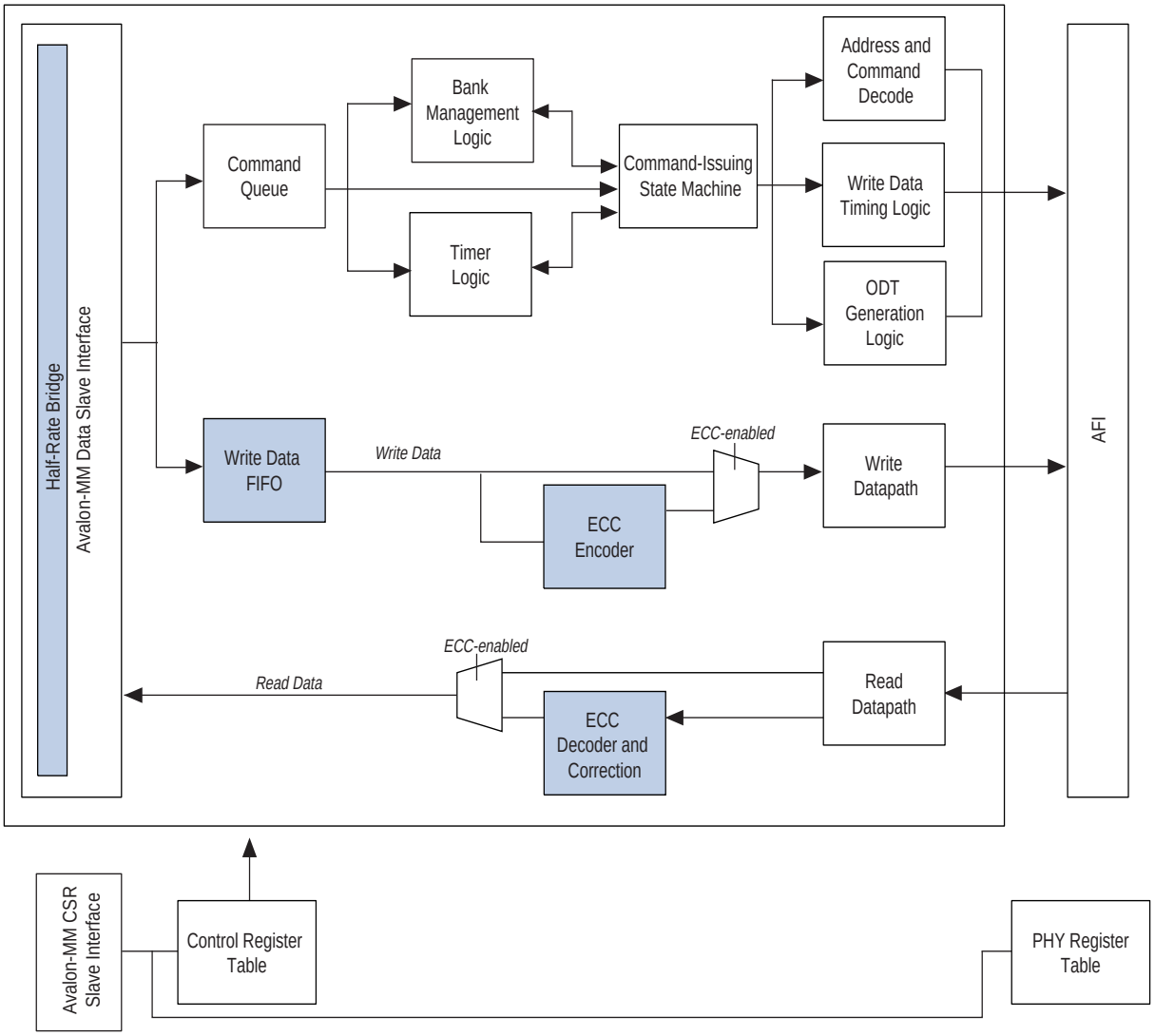


Note to Figure 7–1:

(1) For DDR2 SDRAM HPC II only.

Figure 7–2 shows a block diagram of the DDR or DDR2 SDRAM HPC II architecture.

Figure 7–2. DDR and DDR2 SDRAM HPC II Architecture Block Diagram



The blocks in Figure 7–2 are described in the following sections.

Avalon-MM Data Slave Interface

The Avalon-MM data slave interface accepts read and write requests from the Avalon-MM master. The width of the data busses, `local_wdata` and `local_rdata`, is twice or four times the width of the external memory interface, depending on whether you choose full or half rate.

The local address width is sized based on the memory chip, row, bank, and column address widths. For example:

- For multiple chip selects:

  width = chip bits + row bits + bank bits + column – *N*

- For a single chip select:

  width = row bits + bank bits + column – *N*

Where *N* = 1 for full-rate controller and 2 for half-rate controller.

For every Avalon transaction, the number of read or write requests can go up to the maximum local burst count of 64. Altera recommends that you set this maximum burst count to match your system master's supported burst count.

Write Data FIFO Buffer

The write data FIFO buffer holds the write data and byte-enable from the user logic until the data is needed by the main state machine. The `local_ready` signal is deasserted when either the command queue or write data FIFO buffer is full. The write data FIFO buffer is wide enough to store the write data and the byte-enable signals.

Command Queue

The command queue allows the controller to buffer up to eight consecutive reads or writes. The command queue presents the next 4, 6, or 6 accesses to the internal logic for the look-ahead bank management. The bank management is more efficient if the look-ahead is deeper, but a deeper queue consumes more resources, and may cause maximum frequency degradation.

Other than storing incoming commands, the command queue also maps the local address to memory address based on the address mapping option selected. By default, the command queue leverages bank interleaving scheme, where the address increment goes to the next bank instead of the next row to increase chances of page hit.

Bank Management Logic

The bank management logic keeps track of the current state in each bank across multiple chips. It can keep a row open in every bank in your memory system. When a command is issued by the state machine, the bank management logic is updated with the latest bank status. With the look-ahead capability, the main state machine is able to issue early bank management commands. With the auto-precharge feature, the controller supports an open page policy, where the last accessed row in each bank is kept open and a close page policy, where a bank is closed after it is used.

Timer Logic

The timer logic models the state of each bank in the memory interface. The timer logic models the internal behavior of each bank and provides status output signals to the state machine. The state machine then decides whether to issue the look-ahead bank management command based on the timer status signals.

Command-Issuing State Machine

The command-issuing state machine decides what DDR commands to issue based on the inputs from the command queue, the bank management logic, and the timer logic. The command-issuing state machine operates in two modes: full-rate or half-rate. The full-rate state machine supports 1T address and command, and always issues memory burst length of 4. The half-rate state machine supports 2T address and command, and always issues memory burst length of 8.

A longer memory burst length, in this case 8 beats, increases the command bandwidth by allowing more data cycles for the same amount of command cycles. A longer memory burst length also provides more command cycles that ensures a more effective look-ahead bank management. However, longer memory burst lengths are less efficient if the bursts you issue do not provide enough data to fill the burst.

This state machine accepts any local burst count of 1 to 64. The built-in burst adapter in this state machine maps the local burst count to the most efficient memory burst. The state machine also supports reads and writes that start on non-aligned memory burst boundary addresses. For effective command bus bandwidth, this state machine supports additive latency which issues reads and writes immediately after the ACT command. This state machine accepts additive latency values greater or equal to t_{RCD} – 1, in clock cycle unit (t_{CK}).

Address and Command Decode Logic

When the main state machine issues a command to the memory, it asserts a set of internal signals. The address and command decode 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`

Write and Read Datapath, and Write Data Timing Logic

The write and read datapath, and the write data timing logic generate the AFI read and write control signals.

When the state machine issues a write command to the memory, the write data for that write burst has to be fetched from the write data FIFO buffer. The relationship between the write command and write data depends on the `afi_wlat` signal. This logic presents the write data FIFO read request signal so that the data arrives on the external memory interface DQ pins at the correct time.

During write, the following AFI signals are generated based on the state machine outputs and the `afi_wlat` signal:

- `afi_dqs_burst`
- `afi_wdata_valid`
- `afi_wdata`
- `afi_dm`

During read, the afi_doing_read signal generates the afi_rdata_valid signal and controls the ALTMEMPHY postamble circuit.

ODT Generation Logic

The ODT generation logic generates the necessary ODT signals for DDR2 memory devices, based on the scheme recommended by Altera.

Figure 7–2 shows which ODT signal on the adjacent DIMM is enabled for DDR2 SDRAM.

Table 7–2. ODT

Write or Read On	ODT Enabled
mem_cs[0]	mem_odt[2]
mem_cs[1]	mem_odt[3]
mem_cs[2]	mem_odt[0]
mem_cs[3]	mem_odt[1]

User-Controlled Side-Band Signals

The user-controlled side-band signals consists of the following signals.

User Auto-Precharge Commands

The auto-precharge read and auto-precharge 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 auto-precharges 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. You can request an auto-precharge by asserting the local_autopch signal during a read or write request.

User-Refresh Commands

The user-refresh command enables the request to place the memory into refresh mode. The user-refresh control takes precedence over a read or write request. You can issue up to nine consecutive refresh commands to the memory.

Multi-Cast Write

The multi-cast write request signal allows you to ask the controller to send the current write requests to all the chip-selects. This means that the write data is written to all the ranks in the system. The multi-cast write feature is useful for t_{RC} mitigation where you can cycle through chips to continuously read data without hitting t_{RC}. The multi-cast write is not supported in ECC and RDIMM modes.

Low-Power Mode Logic

There are two types of low-power mode logic: user-controlled self-refresh logic and automatic power-down with programmable time-out logic.

User-Controlled Self-Refresh Logic

When you assert the `local_self_rfsh_req` signal, the controller completes all pending reads and writes before it places the memory into self-refresh mode. Once the controller places the memory into self-refresh mode, it responds by asserting the 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.

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`.

Configuration and Status Register (CSR) Interface

The configuration and status register interface is a 32-bit wide interface that uses the Avalon-MM interface standard. The CSR interface allows you to configure the timing parameters, address widths, and the behavior of the controller. If you do not need this feature, you can disable it and all the programmable settings are fixed to the values configured during the generation process. This interface is synchronous to the controller clock.

Refer to Table 7–9 through Table 7–23 on page 7–20 for detailed information about the register maps.

Error Correction Coding (ECC)

The optional 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 two widths: 64/72 bit and 32/40 bit. The ECC logic has the following features:

- Hamming code ECC that encodes every 64 or 32 bits of data into 72 or 40 bits of codeword.
- A latency increase of one clock for both writes and reads.
- 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.
- Is able to 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 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 issues an internal read to the error address, and performs an internal write to write 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 auto-correction allows the ECC logic to hold off all controller pending activities until the correction is complete. You can choose to disable auto-correction and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, do the following:

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 doing one of the following:
   - Issue 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.

   or

   - Enable the ENABLE_AUTO_CORR register using the CSR interface and issue a read request to the memory address stored in the ERR_ADDR register. The read request triggers auto-error correction to the memory address stored in the ERR_ADDR register.

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 ECC code must be recalculated and the new code must be written back to the memory.

For partial writes, the ECC logic performs the following steps:

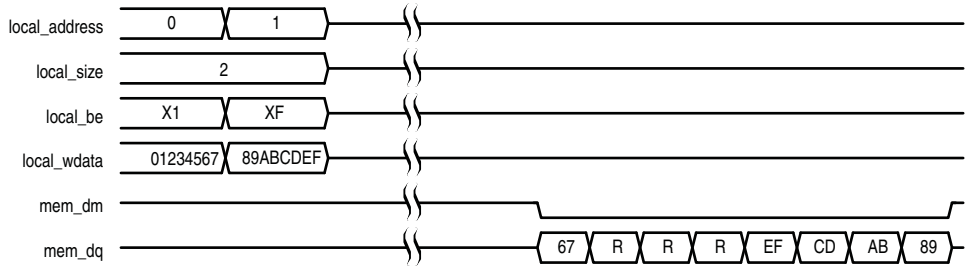
1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a 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 single-bit error is corrected first, the single-bit error counter is incremented and then a partial write is performed to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the double-bit error counter is incremented and an interrupt is issued. A new write word is written back to the memory location. The ECC status register keeps track of the error information.

Figure 7–3 and Figure 7–4 show the partial write operation for HPC II.

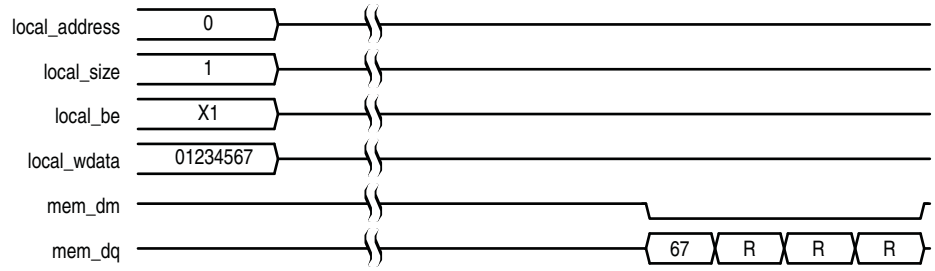
Figure 7–3. Partial Write for HPC II—Full Rate



Note to Figure 7–3:

(1) R represents the internal read-back memory data during the read-modify-write process.

Figure 7–4. Partial Write for HPC II—Half Rate



Note to Figure 7–4:

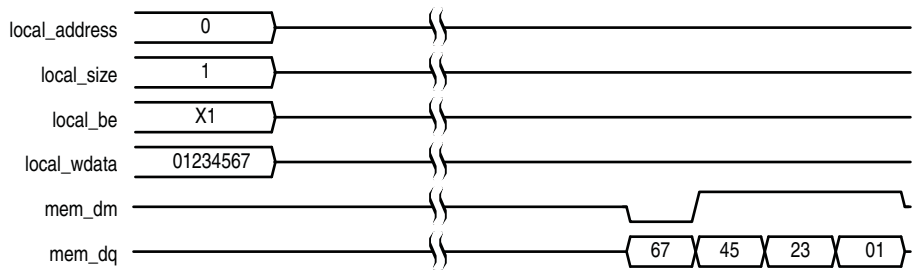
(1) R represents the internal read-back memory data during the read-modify-write process.

Partial Bursts

DIMMs that do not have the DM pins do not support partial bursts. A minimum of four (half rate) or eight words (full rate) must be written to the memory at the same time.

Figure 7–5 shows the partial burst operation for HPC II.

Figure 7–5. Partial Burst for HPC II



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 7–6 shows the testbench and the example top-level file.

Figure 7–6. Testbench and Example Top-Level File

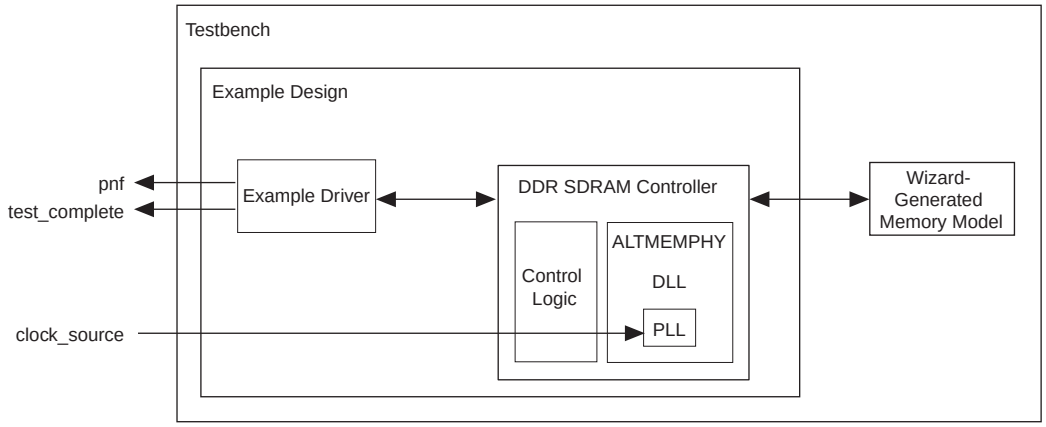


Table 7–3 describes the files that are associated with the example top-level file and the testbench.

Table 7–3. 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 7–4 shows the bit mapping for each test status.

Table 7–4. 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

Top-level Signals Description

Table 7–5 shows the clock and reset signals.

Table 7–5. Clock and Reset Signals (Part 1 of 2)

Name	Direction	Description
`global_reset_n`	Input	The asynchronous reset input to the controller. All other reset signals are derived from resynchronized versions of this signal. This signal holds the complete ALTMEMPHY megafunction, including the PLL, in reset while low.
`pll_ref_clk`	Input	The reference clock input to PLL.
`phy_clk`	Output	The system clock that the ALTMEMPHY megafunction provides to the user. All user inputs to and outputs from the DDR HPC II must be synchronous to this clock.
`reset_phy_clk_n`	Output	The reset signal that the ALTMEMPHY megafunction provides to the user. It is asserted asynchronously and deasserted synchronously to `phy_clk` clock domain.
`aux_full_rate_clk`	Output	An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate mode, this clock is twice the frequency of the `phy_clk` and can be used whenever a 2x clock is required. In full-rate mode, this clock is driven by the same PLL output as the `phy_clk` signal.

Table 7–5. Clock and Reset Signals (Part 2 of 2)

Name	Direction	Description
aux_half_rate_clk	Output	An alternative clock that the ALTMEMPHY megafunction provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate mode, this clock is half the frequency of the phy_clk and can be used, for example to clock the user side of a half-rate bridge. In half-rate mode, or if the **Enable Half Rate Bridge** option is turned on, this clock is driven by the same PLL output that drives the phy_clk signal.
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 it is advised to detect a reset request on a falling edge rather than by level detection.
soft_reset_n	Input	Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. It is asserted to cause a complete reset to the PHY, but not to the PLL used in the PHY.
oct_ctl_rs_value	Input	ALTMEMPHY signal that specifies the serial termination value. Should be connected to the ALT_OCT megafunction output seriesterminationcontrol.
oct_ctl_rt_value	Input	ALTMEMPHY signal that specifies the parallel termination value. Should be connected to the ALT_OCT megafunction output parallelterminationcontrol.
dqs_delay_ctrl_import	Input	Allows the use of DLL in another ALTMEMPHY instance in this ALTMEMPHY instance. Connect the export port on the ALTMEMPHY instance with a DLL to the import port on the other ALTMEMPHY instance.

Table 7–6 shows the DDR and DDR2 SDRAM HPC II local interface signals.

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

Signal Name	Direction	Description

`local_address[]`	Input	Memory address at which the burst should start.
		By default, the local address is mapped to the bank interleaving scheme. You can change the ordering via the **Local-to-Memory Address Mapping** option in the **Controller Settings** page.
		The width of this bus is sized using the following equation:
		■ **Full rate controllers**
		The width of this bus is sized using the following equation:
		For one chip select:
		width = row bits + bank bits + column bits – 1
		For multiple chip selects:
		width = chip bits + row bits + bank bits + column bits – 1
		If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 24 bits wide. To map `local_address` to bank, row and column address:
		`local_address` is 24 bits wide
		`local_address[23:11]` = row address`[12:0]`
		`local_address[10:9]` = bank address `[1:0]`
		`local_address [8:0]` = column address`[9:1]`
		The least significant bit (LSB) of the column address (multiples of four) on the memory side is ignored, because the local data width is twice that of the memory data bus width.
		■ **Half rate controllers**
		The width of this bus is sized using the following equation:
		For one chip select:
		width = row bits + bank bits + column bits – 2
		For multiple chip selects:
		width = chip bits + row bits + bank bits + column bits – 2
		If the bank address is 2 bits wide, row is 13 bits wide and column is 10 bits wide, then the local address is 23 bits wide. To map `local_address` to bank, row and column address:
		`local_address` is 23 bits wide
		`local_address[22:10]` = row address`[12:0]`
		`local_address[9:8]` = bank address `[1:0]`
		`local_address [7:0]` = column address`[9:2]`
		Two LSBs of the column address on the memory side are ignored, because the local data width is four times that of the memory data bus width.

Table 7–6. Local Interface Signals (Part 2 of 4)

Signal Name	Direction	Description
local_be[]	Input	Byte enable signal, which you use to mask off individual bytes during writes. local_be is active high; mem_dm is active low. 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. Local_wdata = < 22334455 >< 667788AA >< BBCCDDEE > Local_be = < 1100 >< 0110 >< 1010 > These values map to: Mem_dq = <4455><2233><88AA><6677><DDEE><BBCC> Mem_dm = <1 1 ><0 0 ><0 1 ><1 0 ><0 1 ><0 1 >
local_burstbegin	Input	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. 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 has deasserted the local_ready signal. This signal is sampled at the rising edge of phy_clk when the local_write_req signal 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. For read transactions, assert this signal for one clock cycle when read request is asserted and the 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 the local_ready signal becomes high again.
local_read_req	Input	Read request signal. You cannot assert read request and write request signal at the same time.
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 requests are already being processed.
local_refresh_chip	Input	Controls which chip to issue the user refresh to. This active high signal is used together with the local_refresh_req signal. 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. For example: If the 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.
local_size[]	Input	Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The range of supported Avalon burst lengths is 1 to 64. The width of this signal is derived based on the burst count specified in the **Local Maximum Burst Count** option. With the derived width, choose a value ranging from 1 to the local maximum burst count specified.

Table 7–6. Local Interface Signals (Part 3 of 4)

Signal Name	Direction	Description
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.
local_multicast	Input	In-band multi-cast write request signal. This active high signal is used together with the local_write_req signal. When this signal is asserted high, data is written to all the memory chips available.
local_autopch_req	Input	User control of auto-precharge. If **Enable Auto-Precharge Control** option is turned on, the local_autopch_req signal becomes available, and you can request the controller to issue an auto-precharge write or auto-precharge read command. 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 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. If you issue a local burst longer than the memory burst with the local_autopch_req signal asserted, the controller only issues auto-precharge with the last read or write command.
local_self_rfsh_req	Input	User control of the self-refresh feature. If **Enable Self-Refresh Controls** option is enabled, 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 the local_self_rfsh_ack signal. 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 the local_self_rfsh_req signal and the controller responds by deasserting the local__self_rfsh_ack signal once it has successfully brought the memory out of the self-refresh state.
local_init_done	Output	When the memory initialization, training, and calibration are complete, the ALTMEMPHY sequencer asserts the ctrl_usr_mode_rdy signal to the memory controller, which then asserts this signal to indicate that the memory interface is ready to be used. Read and write requests are still accepted before local_init_done is asserted, however they are not issued to the memory until it is safe to do so. This signal does not indicate that the calibration is successful.
local_rdata[]	Output	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.
local_rdata_error	Output	Asserted if the current read data has an error. This signal is only available if the **Enable Error Detection and Correction Logic** option is turned on. This signal is asserted together with the local_rdata_valid signal. If the controller encounters double-bit errors, no correction is made and the controller asserts this signal.
local_rdata_valid	Output	Read data valid signal. The local_rdata_valid signal indicates that valid data is present on the read data bus.

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

Signal Name	Direction	Description
local_ready	Output	The local_ready signal indicates that the DDR or DDR2 SDRAM HPC II is ready to accept request signals. If local_ready is asserted in the clock cycle that a read or write request is asserted, that request has been accepted. The local_ready signal is deasserted to indicate that the DDR or DDR2 SDRAM HPC II cannot accept any more requests. The DDR or DDR2 SDRAM HPC II is able to buffer eight read or write requests.
local_refresh_ack	Output	Refresh request acknowledge, which is asserted for one clock cycle every time a refresh is issued. Even if the **Enable User Auto-Refresh Controls** option is not selected, 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 feature. This signal is asserted and deasserted in response to the local_self_rfsh_req signal from the user.
local_power_down_ack	Output	Auto power-down acknowledge signal. This signal is asserted for one clock cycle every time auto power-down is issued.
ecc_interrupt	Output	Interrupt signal from the ECC logic. This signal is asserted when the ECC feature is turned on, and an error is detected. This signal remains asserted until the user logic clears the error through the CSR interface.

Table 7–7 shows the DDR and DDR2 SDRAM HPC II CSR interface signals.

Table 7–7. CSR Interface Signals

Signal Name	Direction	Description
csr_addr[]	Input	Register map address.The width of csr_addr is 16 bits.
csr_be[]	Input	Byte-enable signal, which you use to mask off individual bytes during writes. csr_be is active high.
csr_wdata[]	Input	Write data bus. The width of csr_wdata is 32 bits.
csr_write_req	Input	Write request signal. You cannot assert csr_write_req and csr_read_req signals at the same time.
csr_read_req	Input	Read request signal. You cannot assert csr_read_req and csr_write_req signals at the same time.
csr_rdata[]	Output	Read data bus. The width of csr_rdata is 32 bits.
csr_rdata_valid	Output	Read data valid signal. The csr_rdata_valid signal indicates that valid data is present on the read data bus.
csr_waitrequest	Output	The csr_waitrequest signal indicates that the HPC II is busy and not ready to accept request signals. If the csr_waitrequest signal goes high in the clock cycle when a read or write request is asserted, that request is not accepted. If the csr_waitrequest signal goes low, the HPC II is then ready to accept more requests.

Table 7–8 shows the DDR and DDR2 SDRAM interface signals.

Table 7–8. DDR and DDR2 SDRAM Interface Signals

Signal Name	Direction	Description
mem_dq[]	Bidirectional	Memory data bus. This bus is half the width of the local read and write data busses.
mem_dqs[]	Bidirectional	Memory data strobe signal, which writes data into the DDR or DDR2 SDRAM and captures read data into the Altera device.
mem_dqs_n[]	Bidirectional	Inverted memory data strobe signal, which is used together with the mem_dqs signal to improve signal integrity.
mem_clk *(1)*	Bidirectional	Clock for the memory device.
mem_clk_n *(1)*	Bidirectional	Inverted clock for the memory device.
mem_addr[]	Output	Memory address bus.
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, for DDR2 SDRAM only.
mem_ras_n	Output	Memory row address strobe signal.
mem_we_n	Output	Memory write enable signal.

Note to Table 7–8:

(1) The mem_clk signals are output only signals from the FPGA. However, in the Quartus II software they must be defined as bidirectional (INOUT) I/Os to support the mimic path structure that the ALTMEMPHY megafunction uses.

Register Maps Description

Table 7–9 through Table 7–23 show the register maps for the DDR and DDR2 SDRAM HPC II.

Table 7–9. Register Map

Address	Contents
0x005	Mode register 0-1
0x006	Mode register 2-3
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

Table 7–10. Address 0x005 Mode Register 0-1 (Part 1 of 2)

Bit	Name	Default	Access	Description
0-2	Burst length	8	RO	This value is set to 4 for full-rate and 8 in half-rate for DDR or DDR2 SDRAM HPC II
3	BT	0	RO	This value is set to 0 because the DDR or DDR2 SDRAM HPC II only supports sequential bursts.
4–6	CAS latency	—	RW	This value must be set to match the memory CAS latency. You must set this value in CSR interface register map as well.
7	Reserved	0	—	Reserved for future use.
8	DLL	0	RW	Not used by the controller, but you can set and programm into the memory device mode register.
9–11	Write recovery	—	RW	This value must be set to match the memory write recovery time (t_{WR}). You must set this value in CSR interface register map as well.
12	PD	0/1	RO	This value is set to 0 because the DDR or DDR2 SDRAM HPC II only supports power-down fast exit mode.
13–15	Reserved	0	—	Reserved for future use.
16	DLL	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
17	ODS	0	RW	Not used by the controller, but you can set and program into the memory device mode register.

Table 7–10. Address 0x005 Mode Register 0-1 (Part 2 of 2)

Bit	Name	Default	Access	Description
18	RTT	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
19–21	AL	—	RW	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 as well.
22	RTT	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
23–25	RTT/WL/OCD	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
26	DQS#	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
27	TDQS/RDQS	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
28	QOFF	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
29–31	Reserved	0	—	Reserved for future use.

Table 7–11. Address 0x006 Mode Register 2-3

Bit	Name	Default	Access	Description
0-2	Reserved	0	—	Reserved for future use.
3-5	CWL	—	RW	CAS write latency setting. You must set this value in CSR interface register map as well.
6	ASR	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
7	SRT/ET	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
8	Reserved	0	—	Reserved for future use.
9–10	RTT_WR	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
11–15	Reserved	0	—	Reserved for future use.
16–17	MPR_RF	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
18	MPR	0	RW	Not used by the controller, but you can set and program into the memory device mode register.
19–31	Reserved	0	—	Reserved for future use.

Table 7–12. Address 0x100 ALTMEMPHY Status and Control Register

Bit	Name	Default	Access	Description
0	CAL_SUCCESS	—	RO	This bit reports the value of the ALTMEMPHY `ctl_cal_success` output. Writing to this bit has no effect.
1	CAL_FAIL	—	RO	This bit reports the value of the ALTMEMPHY `ctl_cal_fail` output. Writing to this bit has no effect.
2	CAL_REQ	0	RW	Writing a 1 to this bit asserts the `ctl_cal_req` signal to the ALTMEMPHY megafunction. Writing a 0 to this bit deaaserts the signal, and the ALTMEMPHY megafunction will then initiate its calibration sequence. CAUTION 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.
3–7	Reserved	0	—	Reserved for future use.
8–13	CLOCK_OFF	0	RW	Writing a 1 to any of the bits in this register causes the appropriate `ctl_mem_clk_disable` signal to the ALTMEMPHY megafunction to be asserted, which will disable the memory clock outputs. Writing a 0 to this register causes the signal to be deasserted and the memory clocks to be reenabled. ALTMEMPHY can support up to 6 individual memory clocks, each bit will represent each individual clock.
14-30	Reserved	0	—	Reserved for future use.

Table 7–13. Address 0x110 Controller Status and Configuration Register (Part 1 of 2)

Bit	Name	Default	Access	Description
0–15	AUTO_PD_CYCLES	0x0	RW	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	AUTO_PD_ACK	1	RO	This bit indicates that the memory is in power-down state.
17	SELF_RFSH	0	RW	Setting this bit, or asserting the `local_self_rfsh` signal, causes the memory to go into self-refresh state.
18	SELF_RFSH-ACK	0	RO	This bit indicates that the memory is in self-refresh state.
19	Reserved	0	—	Reserved for future use.

Table 7–13. Address 0x110 Controller Status and Configuration Register (Part 2 of 2)

Bit	Name	Default	Access	Description
20–21	ADDR_ORDER	00	RW	00 - Chip, row, bank, column. 01 - Chip, bank, row, column. 10 - Reserved for future use. 11 - Reserved for future use.
22	REGDIMM	0	RW	Setting this bit to 1 enables REGDIMM support in controller.
23–24	CTRL_DRATE	00	RO	These bits represent controller date rate: 00 - Full rate. 01 - Half rate. 10 - Reserved for future use. 11 - Reserved for future use.
24–30	Reserved	0	—	Reserved for future use.

Table 7–14. Address 0x120 Memory Address Size Register 0

Bit	Name	Default	Access	Description
0–7	Column address width	—	RW	The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12.
8–15	Row address width	—	RW	The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16.
16–19	Bank address width	—	RW	The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3.
20–23	Chip select address width	—	RW	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.
24–31	Reserved	0	—	Reserved for future use.

Table 7–15. Address 0x121 Memory Address Size Register 1

Bit	Name	Default	Access	Description
0–31	Data width representation (word)	—	RW	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).

Table 7–16. Address 0x122 Memory Address Size Register 2

Bit	Name	Default	Access	Description
0–7	Chip select representation	—	RW	The number of chip select in binary representation. For example, a design with 2 chip selects has the value of 00000011.
8–31	Reserved	0	—	Reserved for future use.

Table 7–17. Address 0x123 Memory Timing Parameters Register 0

Bit	Name	Default	Access	Description
0–3	t_{RCD}	—	RW	The activate to read or write the timing parameter. The range of legal values is 2-11 cycles.
4–7	t_{RRD}	—	RW	The activate to activate timing parameter. The range of legal values is 2-8 cycles.
8–11	t_{RP}	—	RW	The precharge to activate timing parameter. The range of legal values is 2-11 cycles.
11–15	t_{MRD}	—	RW	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.
16–23	t_{RAS}	—	RW	The activate to precharge timing parameter. The range of legal values is 4-29 cycles.
24–31	t_{RC}	—	RW	The activate to activate timing parameter. The range of legal values is 8-40 cycles.

Table 7–18. Address 0x124 Memory Timing Parameters Register 1

Bit	Name	Default	Access	Description
0–3	t_{WTR}	—	RW	The write to read timing parameter. The range of legal values is 1-10 cycles.
4–7	t_{RTP}	—	RW	The read to precharge timing parameter. The range of legal values is 2-8 cycles.
8–15	t_{FAW}	—	RW	The four-activate window timing parameter. The range of legal values is 6-32 cycles.
16–31	Reserved	0	—	Reserved for future use.

Table 7–19. Address 0x125 Memory Timing Parameters Register 2

Bit	Name	Default	Access	Description
0–15	t_{REFI}	—	RW	The refresh interval timing parameter. The range of legal values is 780-6240 cycles.
16–23	t_{RFC}	—	RW	The refresh cycle timing parameter. The range of legal values is 12-88 cycles.
24–31	Reserved	0	—	Reserved for future use.

Table 7–20. Address 0x126 Memory Timing Parameters Register 3

Bit	Name	Default	Access	Description
0–3	CAS latency, t_{CL}	—	RW	This value must be set to match the memory CAS latency. You must set this value in the 0x04 register map as well.
4–7	Additive latency, AL	—	RW	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.
8–11	CAS write latency, CWL	—	RW	CAS write latency setting. You must set this value in the 0x06 register map as well.
12–15	Write recovery, t_{WR}	—	RW	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.
16–31	Reserved	0	—	Reserved for future use.

Table 7–21. Address 0x130 ECC Control Register

Bit	Name	Default	Access	Description
0	ENABLE_ECC	1	RW	When 1, enables the generation and checking of ECC.
1	ENABLE_AUTO_CORR	1	RW	When 1, enables auto-correction when a single-bit error is detected.
2	GEN_SBE	0	RW	When 0, enables the deliberate insertion of single-bit errors, bit 0, in the data written to memory. This bit is only used for testing purposes.
3	GEN_DBE	0	RW	When 0, enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data written to memory. This bit is only used for testing purposes.
4	ENABLE_INTR	0	RW	When 0, enables the interrupt output.
5	MASK_SBE_INTR	0	RW	When 0, masks the single-bit error interrupt.
6	MASK_DBE_INTR	0	RW	When 0, masks the double-bit error interrupt
7	CLEAR	0	RW	When 0, writing to this self-clearing bit clears the interrupt signal, and the error status and error address registers.
9	Reserved	0	—	Reserved for future use.

Table 7–22. Address 0x131 ECC Status Register (Part 1 of 2)

Bit	Name	Default	Access	Description
0	SBE_ERROR	1	RO	Set to 1 when any single-bit errors occur.
1	DBE_ERROR	1	RO	Set to 1 when any double-bit errors occur.
2–7	Reserved	0	—	Reserved for future use.

Table 7–22. Address 0x131 ECC Status Register (Part 2 of 2)

Bit	Name	Default	Access	Description
8–15	SBE_COUNT	0	RO	Reports the number of single-bit errors that have occurred since the status register counters were last cleared.
16–23	DBE_COUNT	0	RO	Reports the number of double-bit errors that have occurred since the status register counters were last cleared.
24–31	Reserved	0	—	Reserved for future use.

Table 7–23. Address 0x132 ECC Error Address Register

Bit	Name	Default	Access	Description
0–31	ERR_ADDR	0	RO	The address of the most recent ECC error. This address contains concatenation of chip, bank, row, and column addresses.

8. Latency

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

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

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

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

- Legacy DDR and DDR2 SDRAM controllers
- Half-rate HPC and HPC II
- Full-rate HPC and 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 controllers comprises many different stages of the memory interface. Figure 8–1 on page 8–2 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 8–1. Typical Latency Path

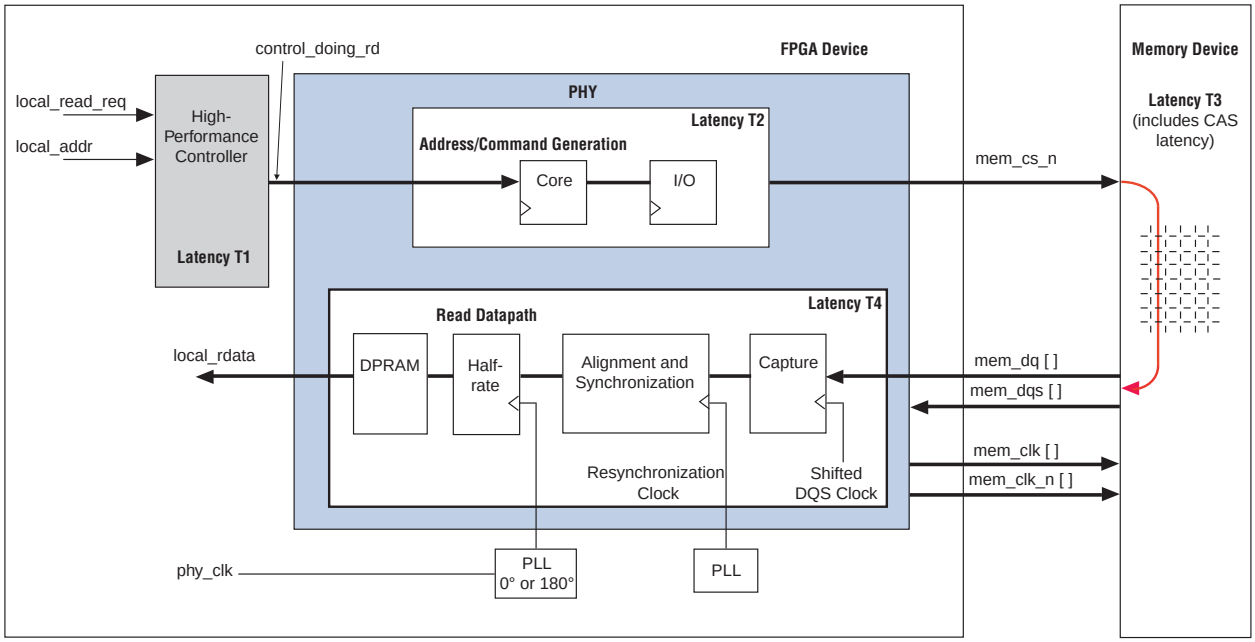


Table 8–1 shows the different stages that make up the whole read and write latency that Figure 8–1 shows.

Table 8–1. High Performance Controller Latency Stages and Descriptions

Latency Number	Latency Stage	Description
T1	Controller	`local_read_req` or `local_write_req` signal assertion to `ddr_cs_n` signal assertion.
T2	Command Output	`ddr_cs_n` signal assertion to `mem_cs_n` 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 8–1, the read latency in the high-performance controllers is made up of four components:

Read latency = controller latency + command output latency + CAS latency + PHY read data input latency = T1 + T2 + T3 + T4

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

Write latency = controller latency + write data latency = T1 + 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 8–2 shows the minimum and maximum supported CAS latency for the DDR and DDR2 SDRAM high-performance controllers (HPC and HPC II).

Table 8–2. 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 8–2:

(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 8–3 through Table 8–6 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 8–3. Typical Read Latency in HPC *(Note 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	17	154
	167	Full-rate	4	2	1	4	5	1	17	108
Arria II GX	233	Half-rate	5	3	1	2.5	5.5	1	18	154
	167	Full-rate	4	2	1	4	6	1	18	114
Cyclone III and Cyclone IV	200	Half-rate	5	3	1	2	4.5	1	17	180
	167	Full-rate	4	2	1	4	5	1	17	108
Stratix II and Stratix II GX	333	Half-rate	5	3	1	2	4.5	1	17	108
	267	Half-rate	5	3	1	2	4.5	1	17	135
	200	Full-rate	4	2	1	4	5	1	17	90
Stratix III and Stratix IV	400	Half-rate	5	3	1	2.5	7.125	1.5	21	100
	267	Full-rate	4	2	1.5	4	7	1	20	71

Notes to Table 8–3:

(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 8–4. Typical Read Latency in HPC II *(Note 1)*, *(2)* (Part 1 of 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

Table 8–4. Typical Read Latency in HPC II *(Note 1)*, *(2)* (Part 2 of 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)
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 8–3:

(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 8–5. Typical Write Latency in HPC *(Note 1)*, *(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	4	2	1	3	11	66
Arria II GX	233	Half-rate	5	3	1	2.5	12	103
	167	Full-rate	4	2	1	4	11	66
Cyclone III and Cyclone IV	200	Half-rate	5	3	1	1.5	12	120
	167	Full-rate	4	2	1	3	11	66
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	4	2	1	3	11	55
Stratix III and Stratix IV	400	Half-rate	5	3	1	2	12	60
	267	Full-rate	4	2	1.5	3	12	44

Notes to Table 8–5:

(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.

Table 8–6. Typical Write Latency in HPC II *(Note 1)*, *(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
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 8–5:

(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*.

9. Timing Diagrams

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

DDR and DDR2 High-Performance Controllers

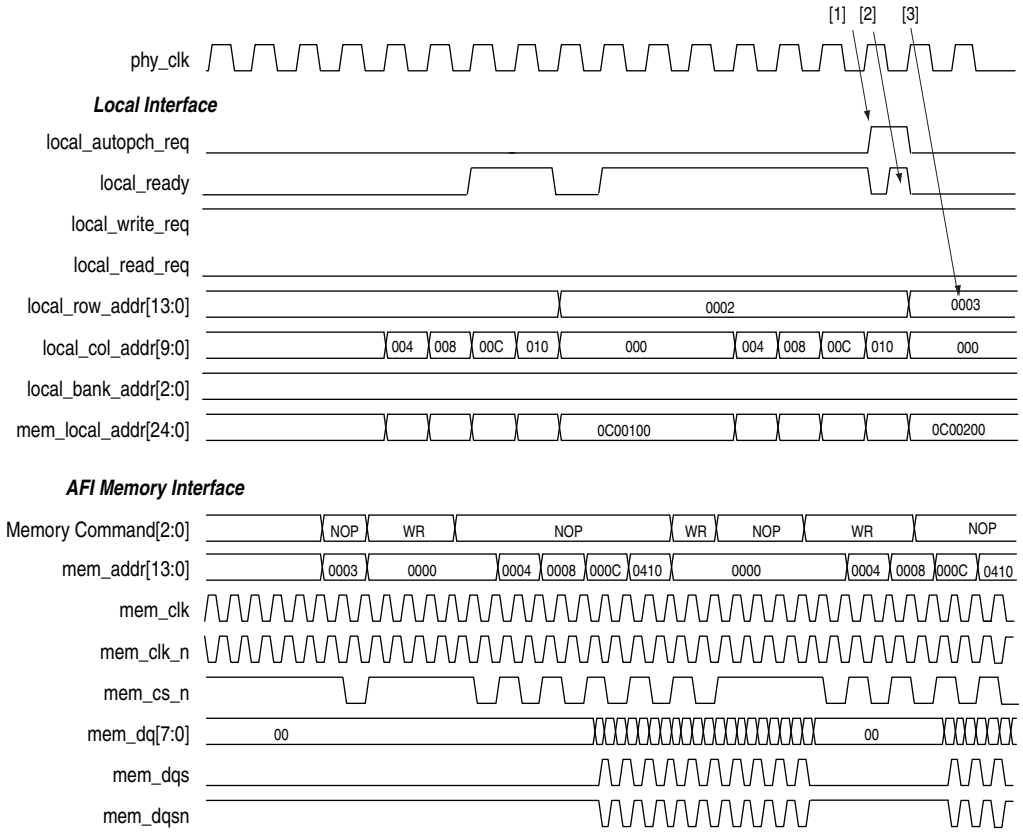
This section discusses the following timing diagrams for HPC in AFI mode:

- "Auto-Precharge"
- "User Refresh"
- "Full-Rate Read"
- "Half-Rate Read"
- "Full-Rate Write"
- "Half Rate Write"
- "Initialization Timing"
- "Calibration Timing"

Auto-Precharge

The auto-precharge read and auto-precharge 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 (auto-precharges) the page it is currently accessing so that the next access to the same bank is quicker. This command is particularly useful for applications that require fast random accesses.

Figure 9–1. Auto-Precharge Operation for HPC



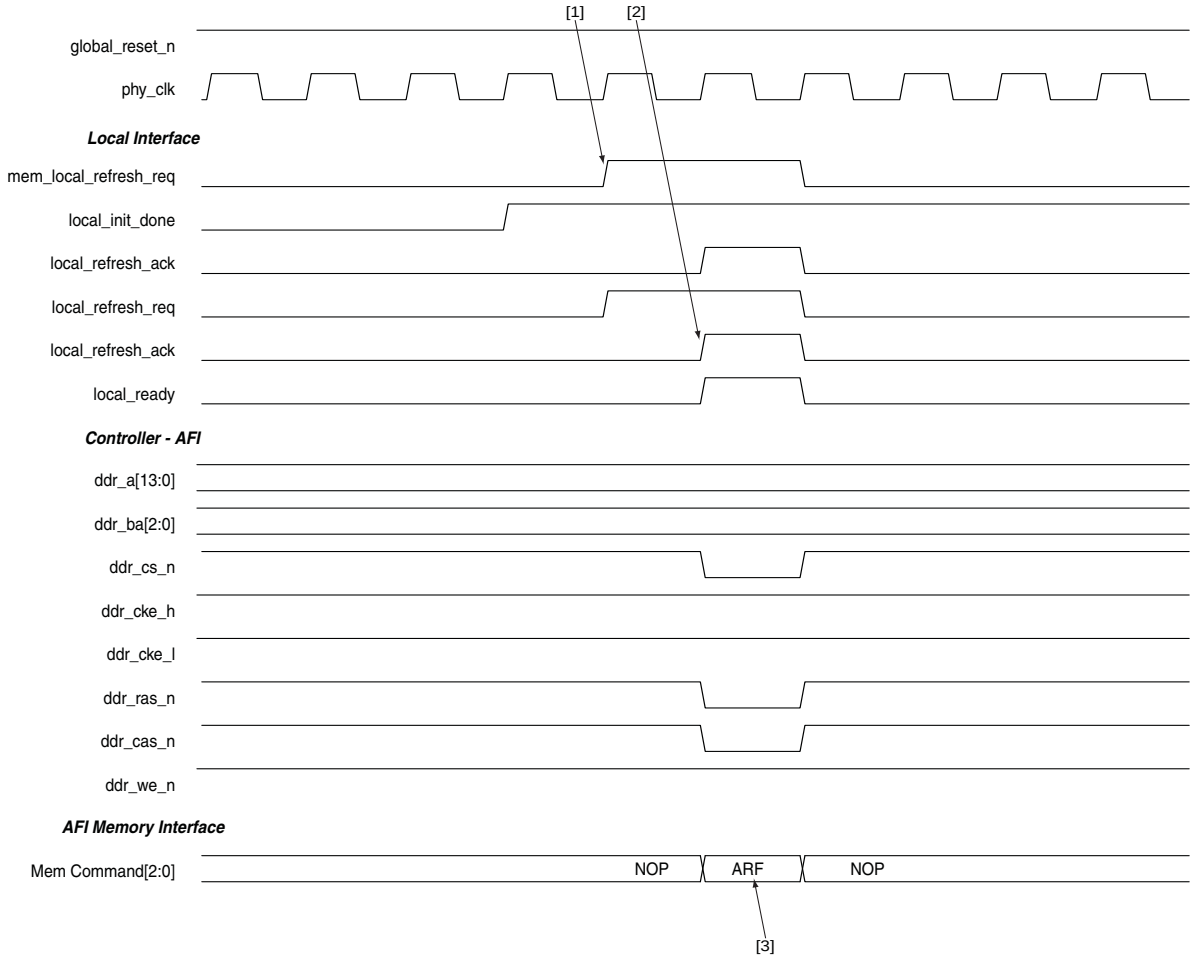
Notes to Figure 9–1:

(1) The auto-precharge request goes high.
(2) The `local_ready` signal is asserted and remains high until the auto-precharge request goes low.
(3) A new row address begins.

User Refresh

Figure 9–2 shows the user refresh control interface. This feature allows you to control when the controller issues refreshes to the memory. This feature allows better control of worst case latency and allows refreshes to be issued in bursts to take advantage of idle periods.

Figure 9–2. User-Refresh Operation for HPC

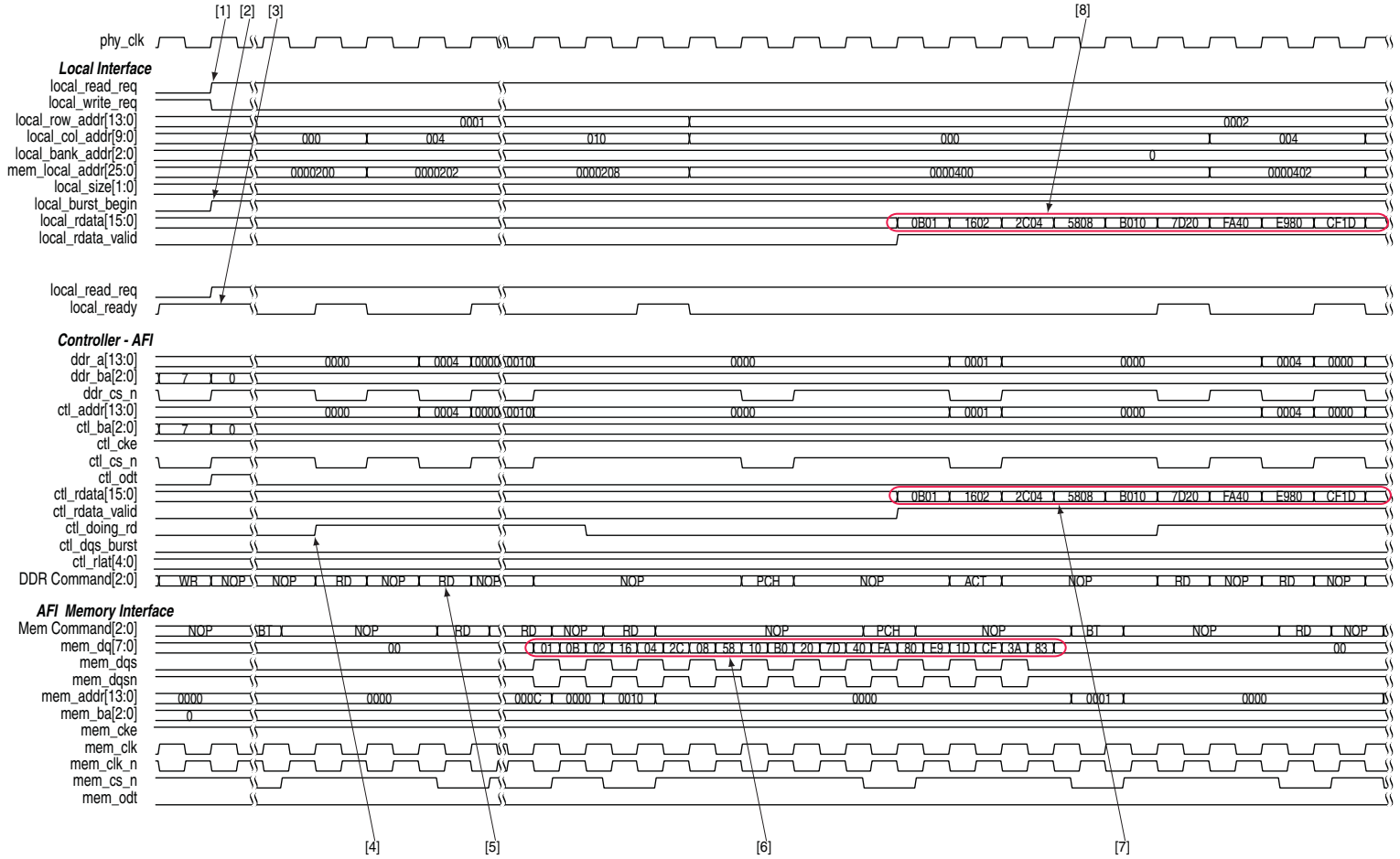


Notes to Figure 9–2:

(1) The local refresh request signal is asserted.
(2) The controller asserts the `local_refresh_ack` signal.
(3) The auto-refresh (ARF) command on the command bus.

Full-Rate Read

Figure 9–3. Full-Rate Read Operation for HPC Using Native and Avalon-MM Interfaces

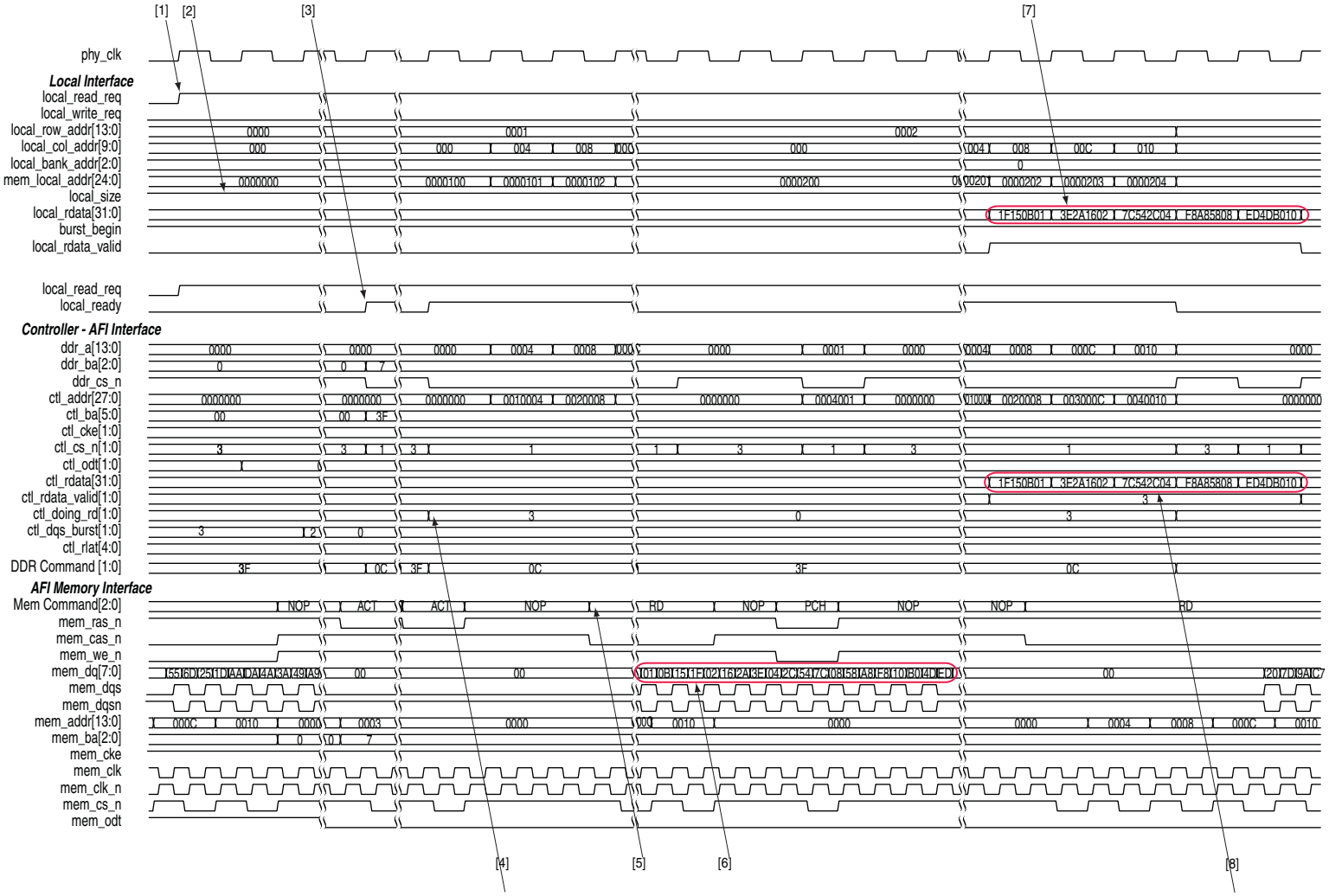


The following sequence corresponds with the numbered items in Figure 9–3:

1. The local read request signal is asserted.
2. The controller accepts the request, the `local_ready` signal is asserted.
3. The controller asserts the `ctl_doing_rd` to tell the PHY how many clock cycles of read data to expect.
4. The read command (RD) on the bus.
5. The `mem_dqs` signal has the read data.
6. These are the data to the controller with the valid signal.
7. The controller returns the valid read data to the user logic by asserting the `local_rdata_valid` signal when there is valid local read data.

Half-Rate Read

Figure 9–4. Half-Rate Read Operation for HPC Using Native and Avalon-MM Interfaces

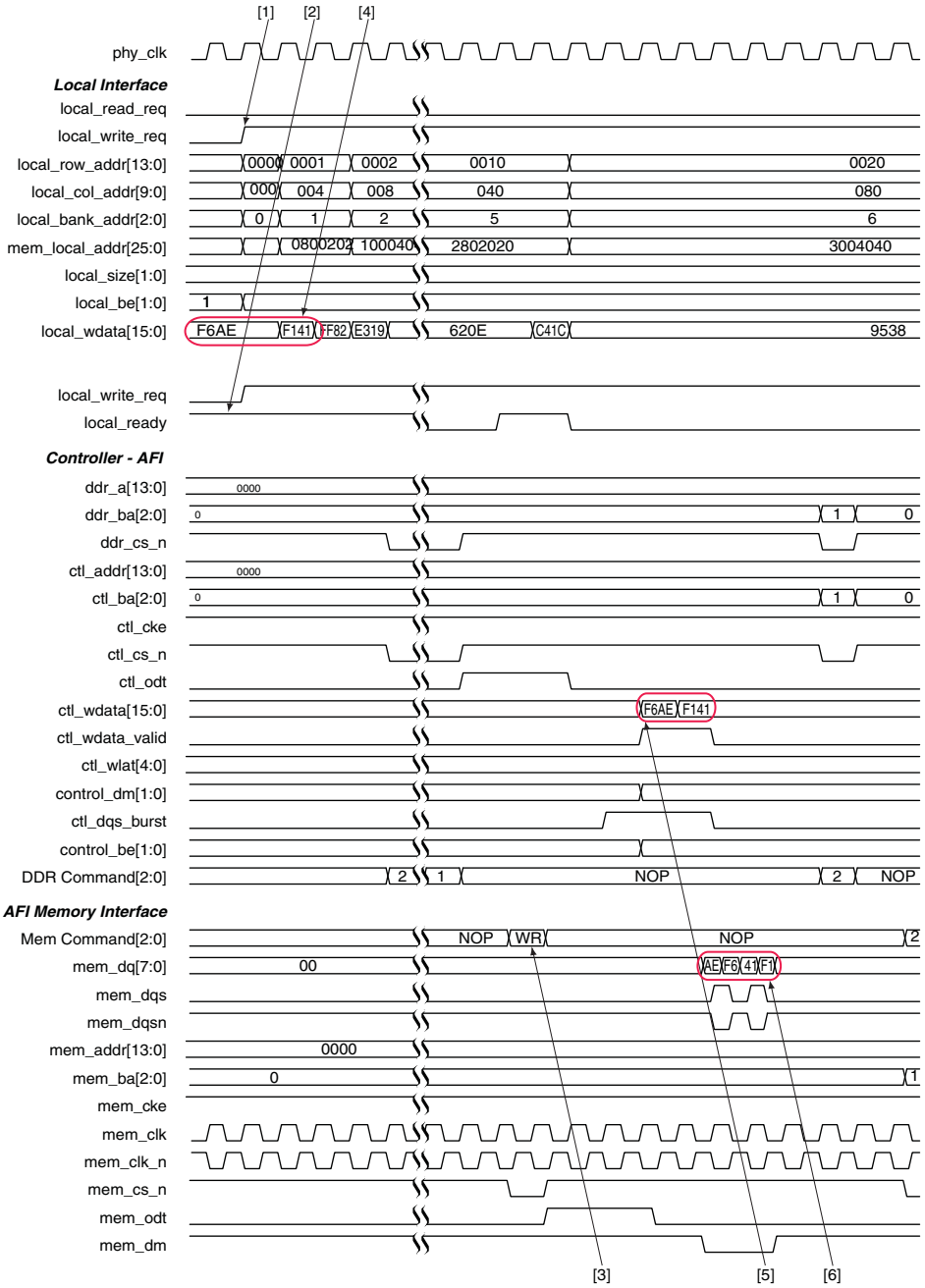


The following sequence corresponds with the numbered items in Figure 9–4:

1. The local read request signal is asserted.
2. The controller accepts the request, the `local_ready` signal is asserted.
3. The controller asserts the `ctl_doing_rd` to tell the PHY how many clock cycles of read data to expect.
4. The read command (RD) on the bus.
5. The `mem_dqs` signal has the read data.
6. These are the data to the controller with the valid signal.
7. The controller returns the valid read data to the user logic by asserting the `local_rdata_valid` signal when there is valid local read data.

Full-Rate Write

Figure 9–5. Full-Rate Write Operation for HPC Using Native and Avalon-MM Interfaces



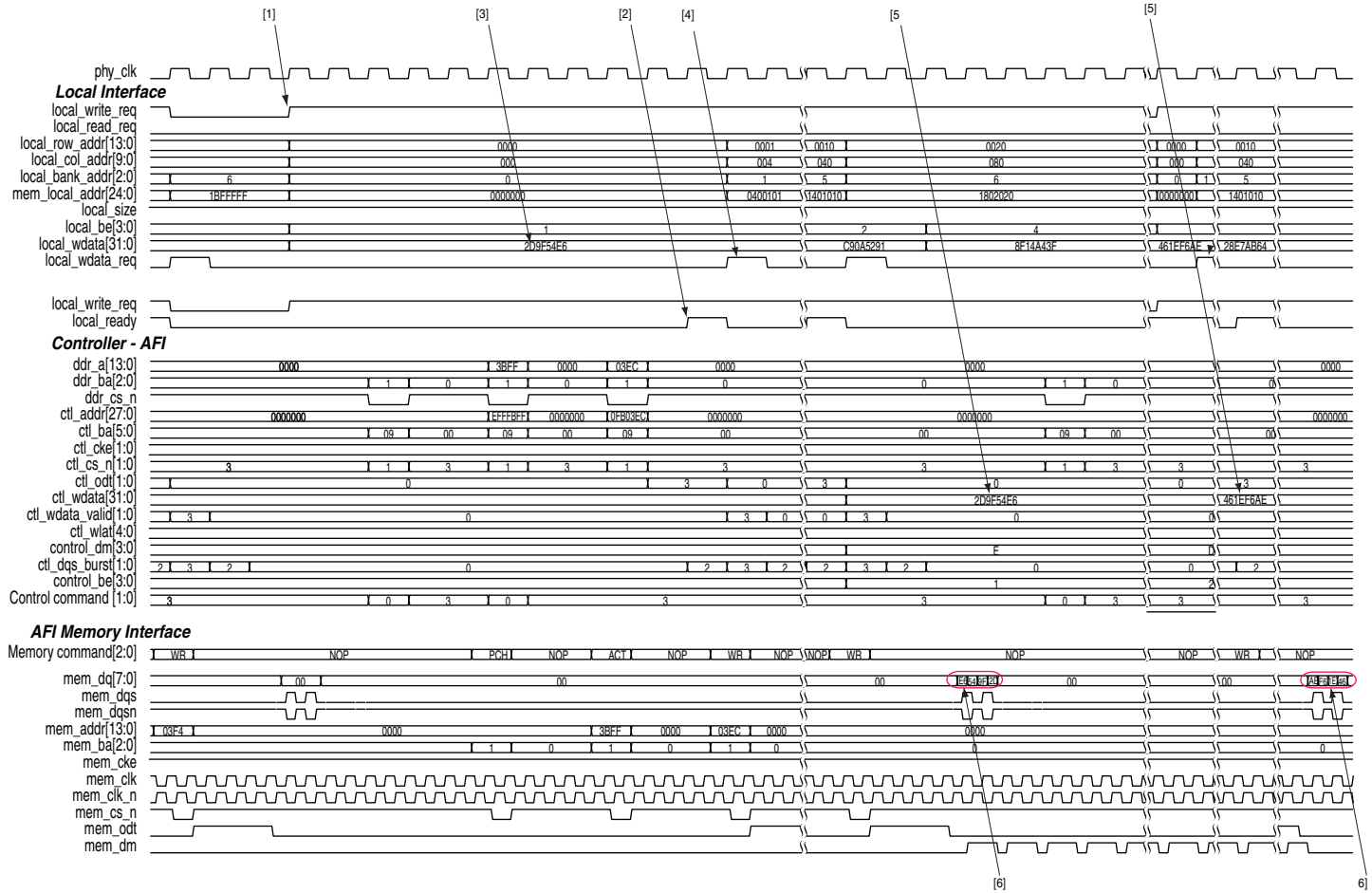
The following sequence corresponds with the numbered items in Figure 9–5:

1. The local write request signal is asserted.
2. The `local_ready` signal is high at the time of the write request.
3. The date is written to the memory for this write command.
4. The write command (WR) on the command bus.

5. The valid write data on the `ctl_wdata` signal. The `ctl_wdata_valid` is 1.
6. Data on the `mem_dqs` signal goes to the controller.

Half Rate Write

Figure 9–6. Half-Rate Write Operation for HPC Using Native and Avalon-MM Interfaces

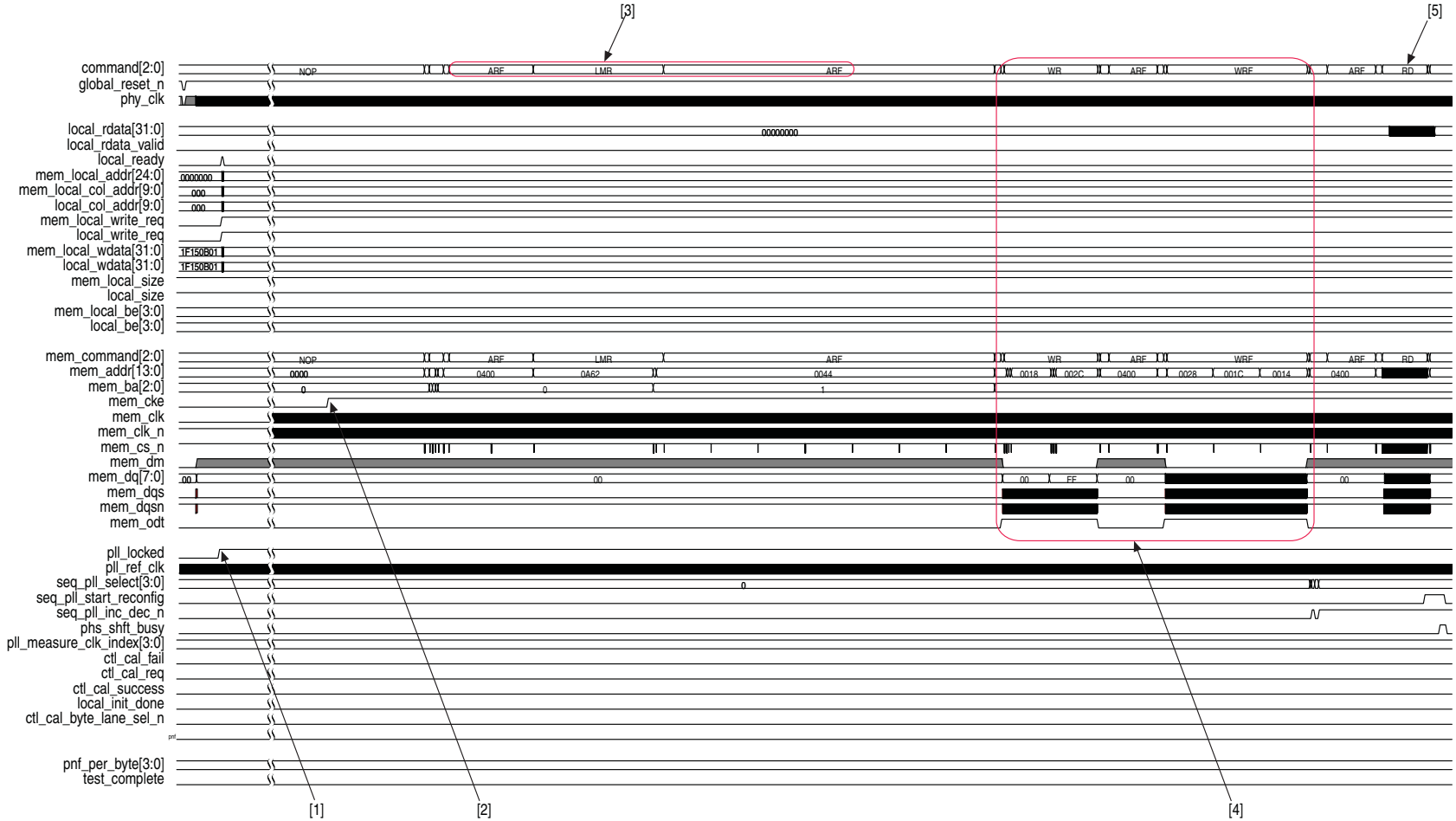


The following sequence corresponds with the numbered items in Figure 9–6:

1. The user logic requests write by asserting the `local_write_req` signal.
2. The `local_ready` signal is asserted, indicating that the controller has accepted the request.
3. The data written to the memory for the write command.
4. The controller requests the user logic for the write data and byte-enables for the write by asserting the `local_wdata_req` signal, (only for native interface).
5. The valid write data on the `ctl_wdata` signal.
6. The valid data on the `mem_dq` signal goes to the controller.

Initialization Timing

Figure 9–7. Initialization Timing for HPC

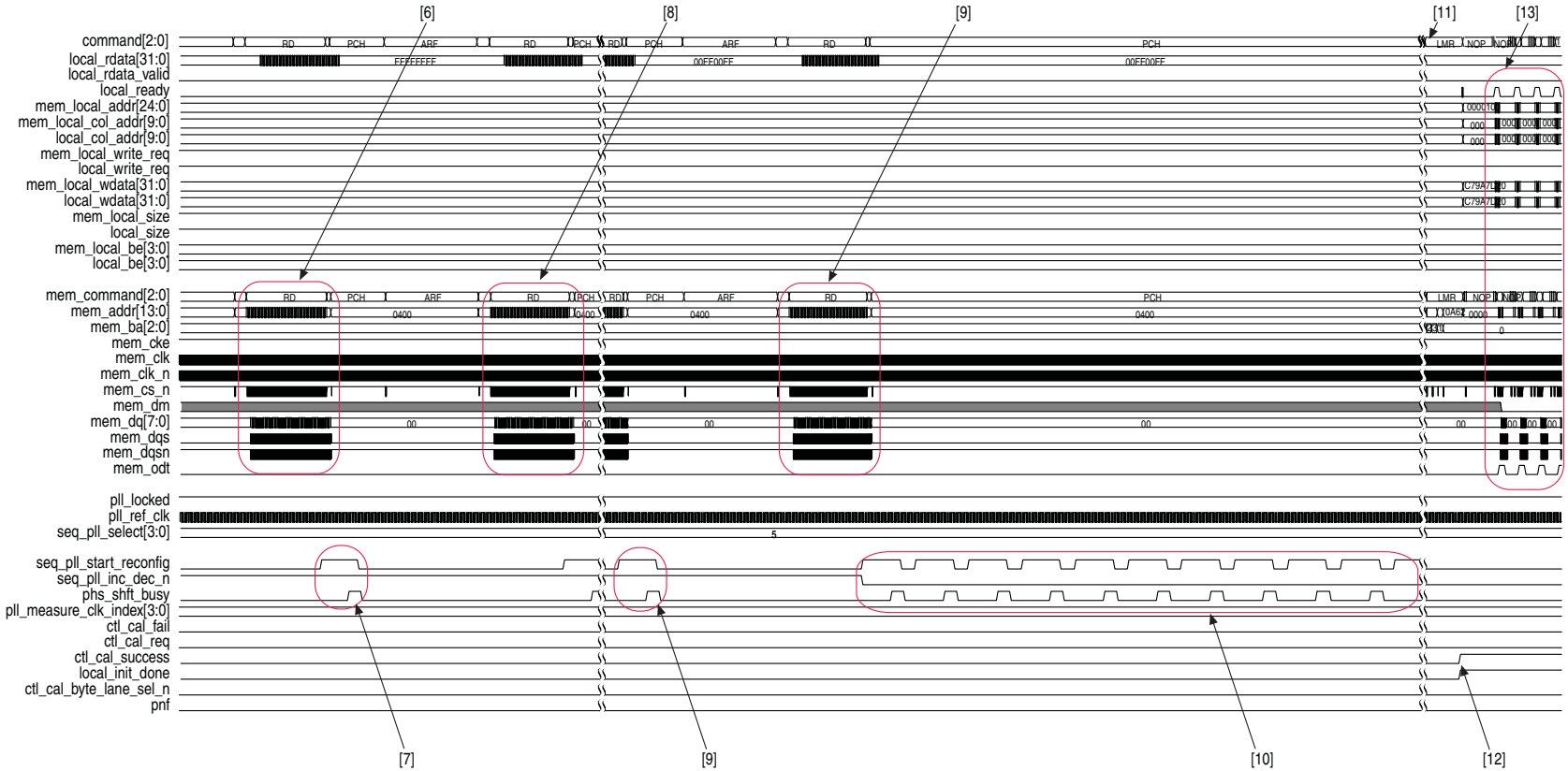


The following sequence corresponds with the numbered items in Figure 9–7:

1. The `pll_locked` signal goes high.
2. The `mem_cke` signal is asserted.
3. The first sequence of the initialization commands: PCH, LMR, PCH, ARF, LMR.
4. Write training data.
5. The first read command to read back training pattern.

Calibration Timing

Figure 9–8. Calibration Timing for HPC



The following sequence corresponds with the numbered items in Figure 9–8:

1. The first read calibration at zero degrees.
2. The PPL phase.
3. The second read calibration after the PLL phase.
4. The final read calibration and final PLL phase.
5. The burst of the PLL phase to center the clock.
6. The second initialization sequence (LMR) to load the settings.
7. The `ctl_cal_success` signal goes high.
8. The functional memory stage.

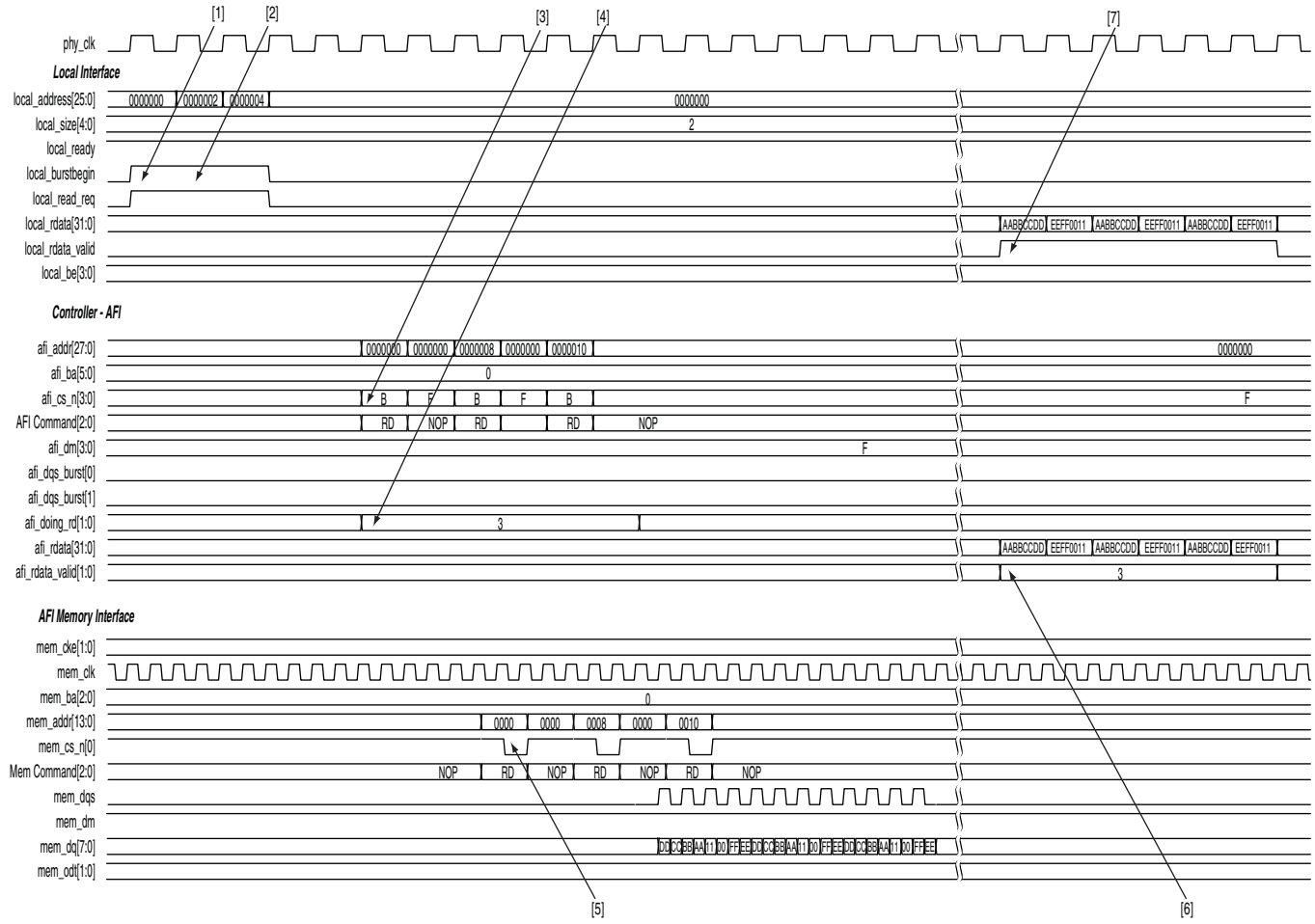
DDR and DDR2 High-Performance Controllers II

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 9–9. Half-Rate Read Operation for HPC II



The following sequence corresponds with the numbered items in Figure 9–9:

1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0×000000`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0×000000
   mem_col_address = 0×0000
   mem_bank_address = 0×00
   ```

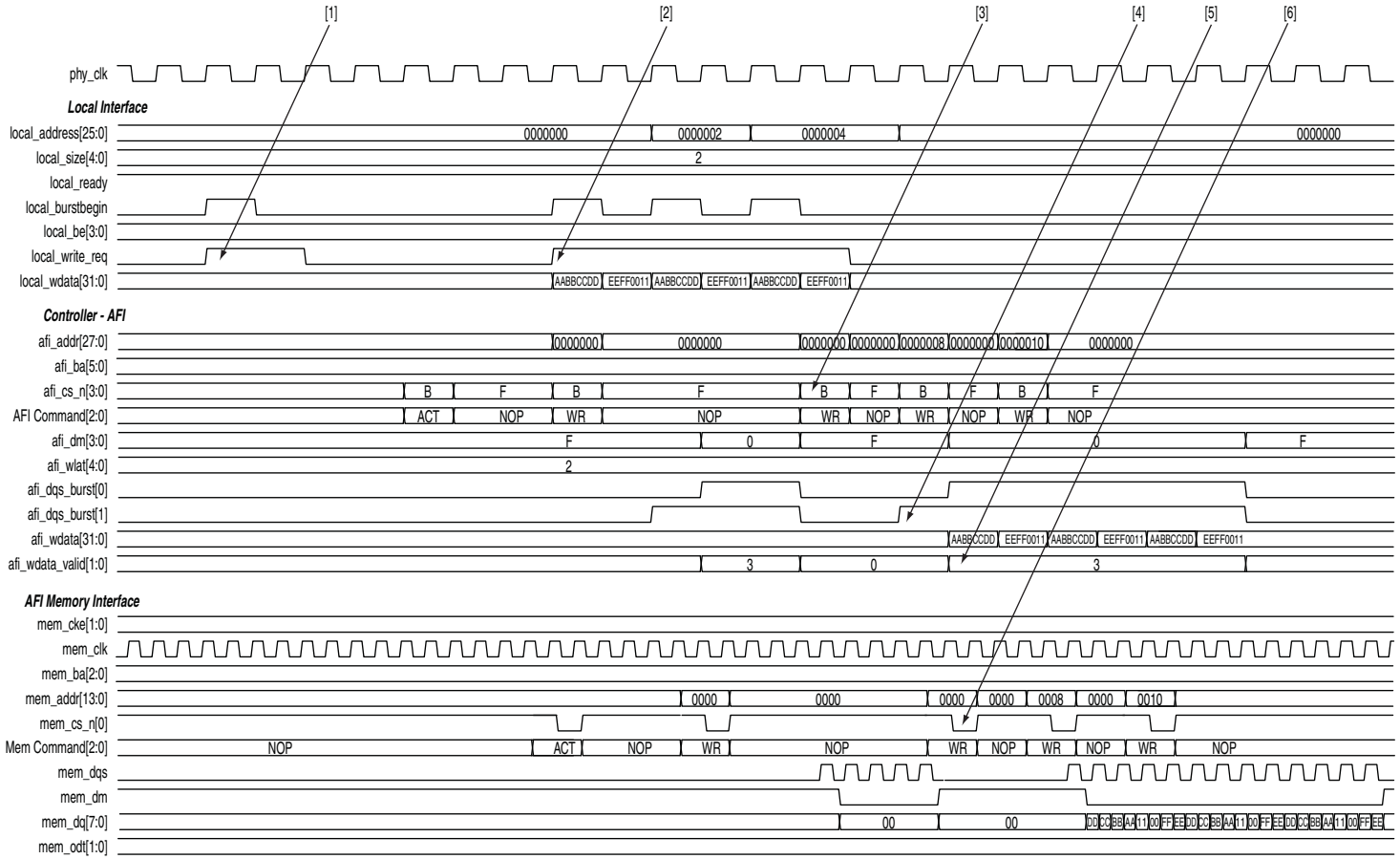
2. The user logic initiates a second read to a different memory column within the same row. The request for the second write is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the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

   ```
   mem_row_address = 0×0000
   mem_col_address = 0×0002<<2 = 0×0008
   mem_bank_address = 0×00
   ```

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 9–10. Half-Rate Write Operation for HPC II

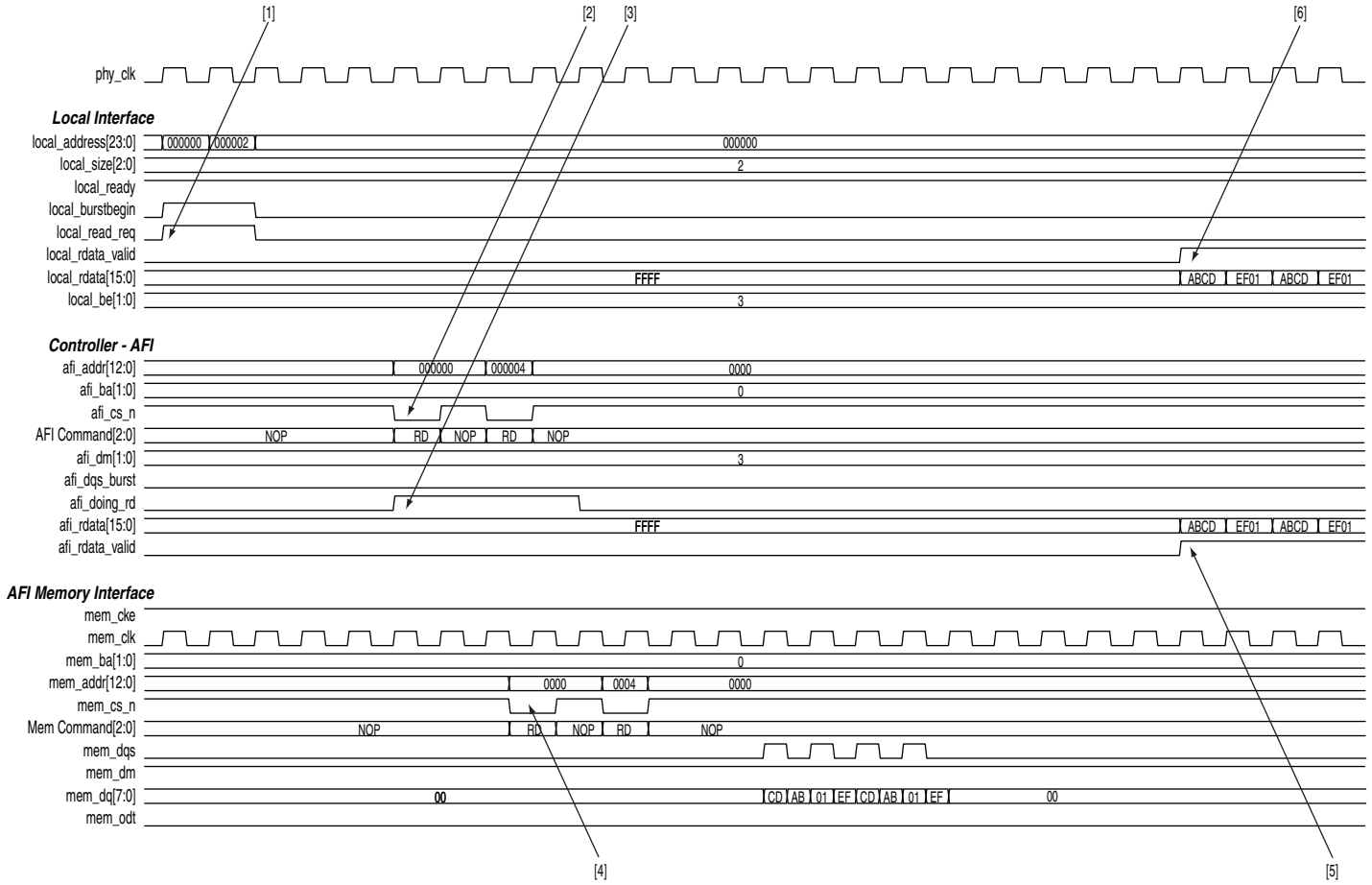


The following sequence corresponds with the numbered items in Figure 9–10:

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 9–11. Full-Rate Read Operation for HPC II



The following sequence corresponds with the numbered items in Figure 9–11:

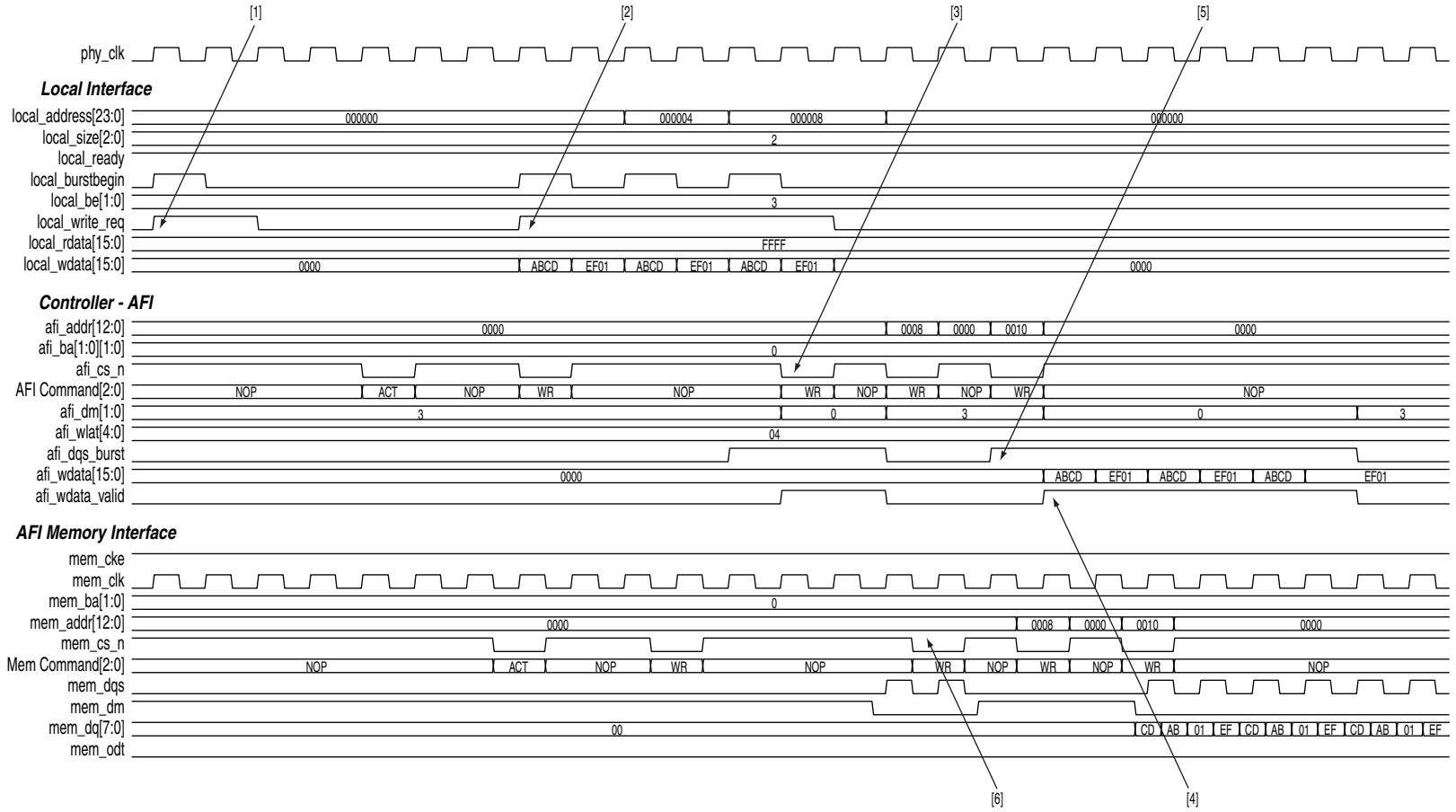
1. The user logic requests the first read by asserting `local_read_req` signal, and the size and address for this read. In this example, the request is a burst length of 2 to a local address `0x000000`. This local address is mapped to the following memory address in full-rate mode:

   ```
   mem_row_address = 0×0000
   mem_col_address = 0×0000<<2 = 0×0000
   mem_bank_address = 0×00
   ```

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 9–12. Full-Rate Write Operation for HPC II



The following sequence corresponds with the numbered items in Figure 9–12:

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.

How to Contact Altera

For the most up-to-date information about Altera® products, see the following table.

Contact *(Note 1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Altera literature services	Email	literature@altera.com
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note:

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

Typographic Conventions

The following table shows the typographic conventions that this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicates 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, dialog box options, software utility names, and other GUI labels. For example, **\qdesigns** directory, **d:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicates document titles. For example, *AN 519: Stratix IV Design Guidelines.*
Italic type	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	Indicates keyboard keys and menu names. For example, 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`. Active-low signals are denoted by suffix `n`. 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`).

Visual Cue	Meaning
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 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 angled arrow instructs you to press Enter.
	The feet direct you to more information about a particular topic.