External Memory Interface Handbook Volume 3

Section II. DDR3 SDRAM Controller with ALTMEMPHY IP User Guide

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

EMI_DDR3_UG-3.0

Document last updated for Altera Complete Design Suite version: 11.0
Document publication date: June 2011

Subscribe

Contents

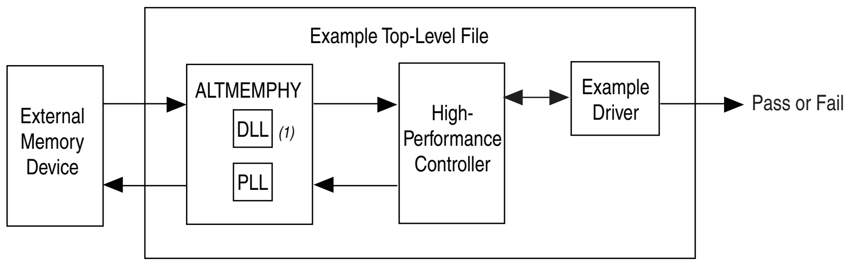
1. About This IP

The Altera® DDR3 SDRAM Controller with ALTMEMPHY IP provides simplified interfaces to industry-standard DDR3 SDRAM. The ALTMEMPHY megafunction is an interface between a memory controller and the memory devices, and performs read and write operations to the memory. The DDR3 SDRAM Controller with ALTMEMPHY IP works in conjunction with the Altera ALTMEMPHY megafunction.

The DDR3 SDRAM Controller with ALTMEMPHY IP and ALTMEMPHY megafunction support DDR3 SDRAM interfaces in half-rate mode. The DDR3 SDRAM Controller with ALTMEMPHY IP offers the high-performance controller II (HPC II), which provides high efficiency and advanced features.

Figure 1–1 on page 1–1 shows a system-level diagram including the example top-level file that the DDR3 SDRAM Controller with ALTMEMPHY IP creates for you.

Figure 1–1. System-Level Diagram



Note to Figure 1–1:

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

The MegaWizard™ Plug-In Manager generates an example top-level file, consisting of an example driver, and your DDR3 SDRAM high-performance controller custom variation. The controller instantiates an instance of the ALTMEMPHY megafunction which in turn instantiates a phase-locked loop (PLL) and DLL. You can also instantiate the DLL outside the ALTMEMPHY megafunction to share the DLL between multiple instances of the ALTMEMPHY megafunction. You cannot share a PLL between multiple instances of the ALTMEMPHY megafunction, but you may share some of the PLL clock outputs between these multiple instances.

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

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

Release Information

Table 1–1 provides information about this release of the DDR3 SDRAM Controller with ALTMEMPHY IP.

Table 1–1. Release Information

Item	Description
Version	11.0
Release Date	May 2011
Ordering Codes	IP-HPMCII (HPC II)
Product IDs	00C2 (DDR3 SDRAM) 00CO (ALTMEMPHY Megafunction)
Vendor ID	6AF7

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release. For information about issues on the DDR3 SDRAM high-performance controller and the ALTMEMPHY megafunction in a particular Quartus II version, refer to the *Quartus II Software Release Notes*.

Device Family Support

The MegaCore function provides either final or preliminary support for target Altera device families:

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

Table 1–2 shows the level of support offered by the DDR3 SDRAM Controller with ALTMEMPHY IP to each of the Altera device families.

Table 1–2. Device Family Support

Device Family	Support
Arria® II GX	Final
Other device families	No support

Features

The ALTMEMPHY megafunction offers the following features:

- Simple setup.
- Support for the Altera PHY Interface (AFI) for DDR3 SDRAM on all supported devices.
- Automated initial calibration eliminating complicated read data timing calculations.
- Voltage and temperature (VT) tracking that guarantees maximum stable performance for DDR3 SDRAM interface.
- Self-contained datapath that makes connection to an Altera controller or a third-party controller independent of the critical timing paths.
- Easy-to-use parameter editor.

The ALTMEMPHY megafunction supports DDR3 SDRAM components without leveling.

- ALTMEMPHY supports DDR3 SDRAM components without leveling for Arria II GX devices using T-topology for clock, address, and command bus:
  - Supports multiple chip selects.
- The DDR3 SDRAM PHY without leveling f_{MAX} is 400 MHz for single chip selects.
- No support for data-mask (DM) pins for ×4 DDR3 SDRAM DIMMs or components, so select **No** for **Drive DM pins from FPGA** when using ×4 devices.
- The ALTMEMPHY megafunction supports half-rate DDR3 SDRAM interfaces only.

In addition, Table 1–3 shows the features provided by the DDR3 SDRAM HPC and HPC II.

Table 1–3. DDR3 SDRAM HPC II Features (Part 1 of 2)

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

Table 1–3. DDR3 SDRAM HPC II Features (Part 2 of 2)

Features	HPC II
Additive latency	✓ (1)
Support for arbitrary Avalon burst length	✓
Built-in flexible memory burst adapter	✓
Configurable Local-to-Memory address mappings	✓
Optional run-time configuration of size and mode register settings, and memory timing	✓
Partial array self-refresh (PASR)	✓
Support for industry-standard DDR3 SDRAM devices	✓
Optional support for self-refresh command	✓
Optional support for user-controlled power-down command	—
Optional support for automatic power-down command with programmable time-out	✓
Optional support for auto-precharge read and auto-precharge write commands	✓
Optional support for user-controller refresh	✓
Optional multiple controller clock sharing in SOPC Builder Flow	✓
Integrated error correction coding (ECC) function 72-bit	✓
Integrated ECC function, 16, 24, and 40-bit	✓
Support for partial-word write with optional automatic error correction	✓
SOPC Builder ready	✓
Support for OpenCore Plus evaluation	—
IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulator	✓

Notes to Table 1–3:

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

(2) This feature is not supported with DDR3 SDRAM with leveling.

Unsupported Features

The DDR3 SDRAM Controller with ALTMEMPHY IP does not support the following features and devices:

- Timing simulation.
- Partial burst and unaligned burst in ECC and non-ECC mode when DM pins are disabled.
- Stratix III and Stratix IV
- DIMM support
- Full-rate interfaces

MegaCore Verification

Altera performs extensive random, directed tests with functional test coverage using industry-standard Denali models to ensure the functionality of the DDR3 SDRAM Controller with ALTMEMPHY IP.

Resource Utilization

Table 1–4 shows the resource utilization data for the ALTMEMPHY megafunction, and the DDR3 high-performance controller II.

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

Protocol	Memory Width (Bits)	Combinational ALUTS	Logic Registers	Mem ALUTs	M9K Blocks	M144K Blocks	Memory (Bits)
Controller							
DDR3 (Half rate)	8	1,883	1,505	10	2	0	4,352
	16	1,893	1,505	10	4	0	8,704
	64	1,946	1,521	18	15	0	34,560
	72	1,950	1,505	10	17	0	39,168
Controller+PHY							
DDR3 (Half rate)	8	3,389	2,760	12	4	0	4,672
	16	3,457	2,856	12	7	0	9,280
	64	3,793	3,696	20	24	0	36,672
	72	3,878	3,818	12	26	0	41,536

System Requirements

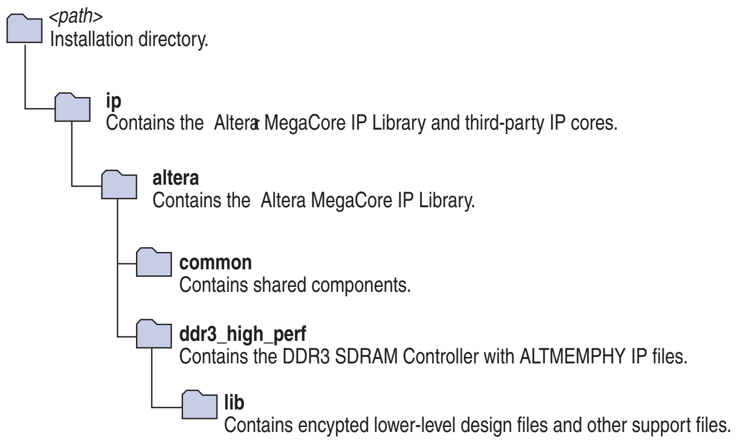
The DDR3 SDRAM Controller with ALTMEMPHY IP is a part of the MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

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 DDR3 SDRAM Controller with ALTMEMPHY IP, where <*path*> is the installation directory. The default installation directory on Windows is **c:\altera\**<*version*>; on Linux it is **/opt/altera**<*version*>.

Figure 1–2. Directory Structure


<path>
Installation directory.
ip
Contains the Altera MegaCore IP Library and third-party IP cores.
altera
Contains the Altera MegaCore IP Library.
common
Contains shared components.
ddr3_high_perf
Contains the DDR3 SDRAM Controller with ALTMEMPHY IP files.
lib
Contains encypted lower-level design files and other support files.

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

To use the DDR3 SDRAM HPC, you can request a license file from the Altera web site at www.altera.com/licensing and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local representative.

To use the DDR3 SDRAM HPC II, contact your local sales representative to order a license.

Free Evaluation

Altera's OpenCore Plus evaluation feature is only applicable to the DDR3 SDRAM HPC. With the OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP^{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.

2. Getting Started

Design Flow

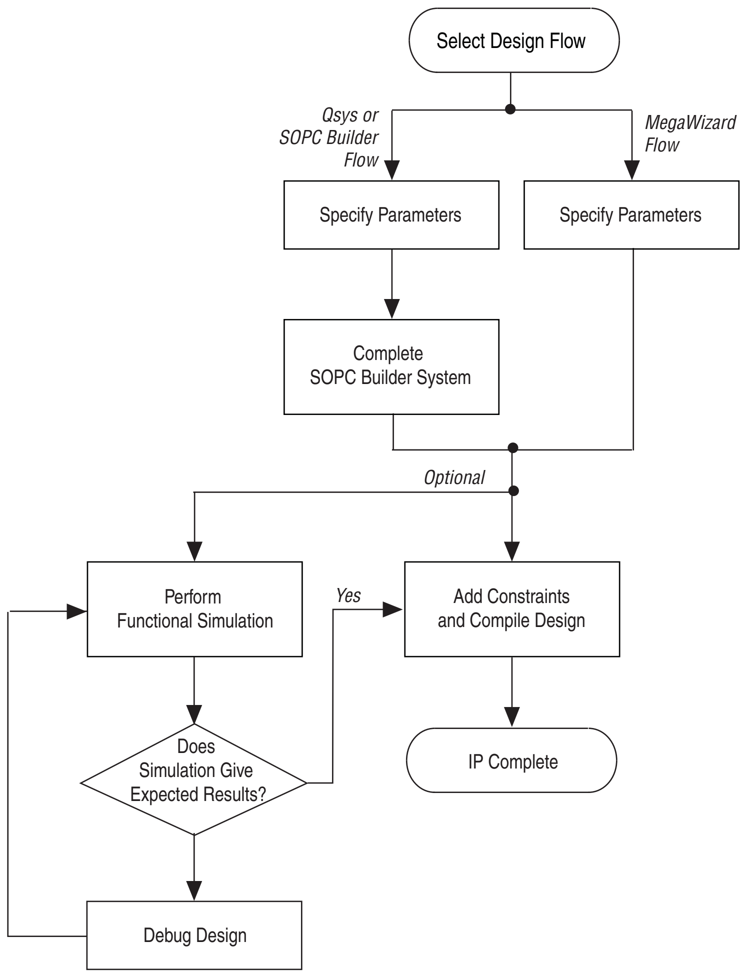
You can implement the DDR3 SDRAM Controller with ALTMEMPHY IP using any of the following flows:

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

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

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

Figure 2–1. Design Flow



Select Design Flow
Qsys or SOPC Builder Flow
MegaWizard Flow
Specify Parameters
Specify Parameters
Complete SOPC Builder System
Optional
Perform Functional Simulation
Yes
Add Constraints and Compile Design
Does Simulation Give Expected Results?
IP Complete
Debug Design

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 DDR3 SDRAM interface to peripheral device or devices
- Achieves higher-frequency operation

SOPC Builder Flow

The SOPC Builder flow allows you to add the DDR3 SDRAM Controller with ALTMEMPHY IP 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 DDR3 SDRAM controller, such as the Nios® II processor and scatter-gather direct memory access (SDMA) controllers. SOPC Builder automatically creates the system interconnect logic and system simulation environment.

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

Specifying Parameters

To specify the parameters for the DDR3 SDRAM Controller with ALTMEMPHY IP using the SOPC Builder flow, perform the following steps:

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. Add **DDR3 SDRAM Controller with ALTMEMPHY** to your system from the **System Contents** tab.

   The **DDR3 SDRAM Controller with ALTMEMPHY** is in the **SDRAM** folder under the **Memories and Memory Controllers** folder.

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

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

   For detailed explanation of the parameters, refer to the "Parameter Settings" on page 3–1.

6. Click **Finish** to complete parameterizing the DDR3 SDRAM Controller with ALTMEMPHY IP and add it to the system.

Completing the SOPC Builder System

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

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 `0×47`, 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	`0×60`	`0×80`
16	`0×A0`	`0×C0`
32	`0×120`	`0×140`
64	`0×240`	`0×260`

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 "Compiling and Simulating" on page 4–1.

Qsys Flow

The Qsys flow allows you to add the DDR3 SDRAM Controller with ALTMEMPHY directly to a new or existing Qsys system.

You can also easily add other available components to quickly create a Qsys system with a DDR3 SDRAM controller, such as the Nios II processor and scatter-gather direct memory access (SDMA) controllers. Qsys automatically creates the system interconnect logic and system simulation environment.

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

Specifying Parameters

To specify the parameters for the DDR3 SDRAM Controller with ALTMEMPHY IP using the Qsys flow, perform the following steps:

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click Qsys.
3. For a new system, specify the system name and language.
4. Add **DDR3 SDRAM Controller with ALTMEMPHY** to your system from the **System Contents** tab.

   The **DDR3 SDRAM Controller with ALTMEMPHY** is in the **SDRAM** folder under the **Memories and Memory Controllers** folder.

5. In the DDR3 SDRAM Controller with ALTMEMPHY parameter editor, specify the required parameters on all pages in the **Parameter Settings** tab.

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

For detailed explanation of the parameters, refer to the "Parameter Settings" on page 3–1.

6. Click **Finish** to complete parameterizing the DDR3 SDRAM Controller with ALTMEMPHY IP and add it to the system.

Completing the Qsys System

To complete the Qsys system, perform the following steps:

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

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 `0x0` to `0x47`, multiply the memory address by the width of the memory interface data bus in bytes. Refer to Table 2–2 for more Avalon-MM addresses.

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

External Memory Interface Width	Reset Vector Offset	Exception Vector Offset
8	`0x60`	`0x80`
16	`0xA0`	`0xC0`
32	`0x120`	`0x140`
64	`0x240`	`0x260`

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

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

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

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

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

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

10. Compile your design, refer to "Compiling and Simulating" on page 4–1.

MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize the DDR3 SDRAM Controller with ALTMEMPHY or the stand-alone PHY with the ALTMEMPHY megafunction, and manually integrate the function into your design.

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

Specifying Parameters

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

1. In the Quartus II software, create a new Quartus II project with the **New Project Wizard**.
2. On the Tools menu, click **MegaWizard Plug-In Manager** to start the MegaWizard Plug-In Manager.
   - The DDR3 SDRAM Controller with ALTMEMPHY is in the **Interfaces** folder under the **External Memory** folder.
   - The ALTMEMPHY megafunction is in the **I/O** folder.

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

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

   For detailed explanation of the parameters, refer to the "Parameter Settings" on page 3–1.

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

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

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

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

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

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

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

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

8. After you review the generation report, click **Exit** to close the MegaWizard Plug-In Manager.

9. For the high-performance controller, 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 2–3 shows the ALTMEMPHY generated files.

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

File Name	Description
alt_mem_phy_defines.v	Contains constants used in the interface. This file is always in Verilog HDL regardless of the language you chose in the MegaWizard Plug-In Manager.
<*variation_name*>**.html**	Lists the top-level files created and ports used in the megafunction.
<*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_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.
<*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.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_pll_bb.v/.cmp**	Black box file for the PLL used in your ALTMEMPHY variation. Typically unused.
<*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_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.

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

File Name	Description
<*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*>**_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*>**_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*>**_ddr_timing.sdc**	Contains timing constraints for your ALTMEMPHY variation.
<*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*>**_report_timing.tcl**	Script that reports timing for your ALTMEMPHY variation during compilation.

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

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

Module Name	Usage	Description
<*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`	DDR3 SDRAM ALTMEMPHY variation	Creates the VT tracking mechanism for DDR3 SDRAM PHYs.
<*variation_name*>`_alt_mem_phy_oct_delay`	DDR3 SDRAM ALTMEMPHY variation when dynamic OCT is enabled.	Generates the proper delay and duration for the OCT signals.
<*variation_name*>`_alt_mem_phy_postamble`	DDR3 SDRAM ALTMEMPHY variations	Generates the postamble enable and disable scheme for DDR3 PHYs.

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

Module Name	Usage	Description
<*variation_name*>`_alt_mem_phy_read_dp`	All ALTMEMPHY variations	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_rdata_valid`	DDR3 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 DDR3 SDRAM.
<*variation_name*>`_alt_mem_phy_write_dp`	All ALTMEMPHY variations	Generates the demultiplexing of data from half-rate to full-rate DDR data.

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

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

Filename	Description
alt_mem_ddrx_addr_cmd.v	Decodes internal protocol-related signals into memory address and command signals.
alt_mem_ddrx_addr_cmd_wrap.v	A wrapper that instantiates the **alt_mem_ddrx_addr_cmd.v** file.
alt_mem_ddrx_ddr2_odt_gen.v	Generates the on-die termination (ODT) control signal for DDR2 memory interfaces.
alt_mem_ddrx_ddr3_odt_gen.v	Generates the on-die termination (ODT) control signal for DDR3 memory interfaces.
alt_mem_ddrx_odt_gen.v	Wrapper that instantiates **alt_mem_ddrx_ddr2_odt_gen.v** and **alt_mem_ddrx_ddr3_odt_gen.v**. This file also controls the ODT addressing scheme.
alt_mem_ddrx_rdwr_data_tmg.v	Decodes internal data burst related signals to memory data signals.
alt_mem_ddrx_arbiter.v	Contains logic that determines which command to execute based on certain schemes.
alt_mem_ddrx_burst_gen.v	Converts internal DRAM-aware commands to AFI signals.
alt_mem_ddrx_cmd_gen.v	Converts user requests to DRAM-aware commands.
alt_mem_ddrx_csr.v	Contains configuration registers.
alt_mem_ddrx_buffer.v	Contains buffer for local data.
alt_mem_ddrx_buffer_manager.v	Manages the allocation of buffers.
alt_mem_ddrx_burst_tracking.v	Tracks data received per local burst command.
alt_mem_ddrx_dataid_manager.v	Manages the IDs associated with data stored in buffer.
alt_mem_ddrx_fifo.v	Contains the FIFO buffer to store local data to create a link; is also used in rdata_path to store the read address and error address.
alt_mem_ddrx_list.v	Tracks the DRAM commands associated with the data stored internally.
alt_mem_ddrx_rdata_path.v	Contains read data path logic.
alt_mem_ddrx_wdata_path.v	Contains write data path logic.

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

Filename	Description
alt_mem_ddrx_define.iv	Defines common parameters used in the RTL files.
alt_mem_ddrx_ecc_decoder.v	Instantiates appropriate width ECC decoder logic.
alt_mem_ddrx_ecc_decoder_32_syn.v	Contains synthesizable 32-bit version of ECC decoder.
alt_mem_ddrx_ecc_decoder_64_syn.v	Contains synthesizable 64-bit version of ECC decoder.
alt_mem_ddrx_ecc_encoder.v	Instantiates appropriate width ECC encoder logic.
alt_mem_ddrx_ecc_encoder_32_syn.v	Contains synthesizable 32-bit version of ECC decoder.
alt_mem_ddrx_ecc_encoder_64_syn.v	Contains synthesizable 64-bit version of ECC decoder.
alt_mem_ddrx_ecc_encoder_decoder_wrapper.v	Wrapper that instantiates all ECC logic.
alt_mem_ddrx_input_if.v	Contains local input interface logic.
alt_mem_ddrx_mm_st_converter.v	Contains supporting logic for Avalon-MM interface.
alt_mem_ddrx_rank_timer.v	Contains a timer associated with rank timing.
alt_mem_ddrx_sideband.v	Contains supporting logic for user-controlled refresh and precharge signals.
alt_mem_ddrx_tbp.v	Contains command queue and associated logic for reordering features.
alt_mem_ddrx_timing_param.v	Contains timer logic associated with nonrank timing.
alt_mem_ddrx_controller_st_top.v	Wrapper that instantiates all submodules amd configuration registers.
alt_mem_ddrx_controller_top.v	Wrapper that contains memory controller with Avalon-MM interface.
alt_mem_ddrx_controller.v	Wrapper that instantiates all submodules.

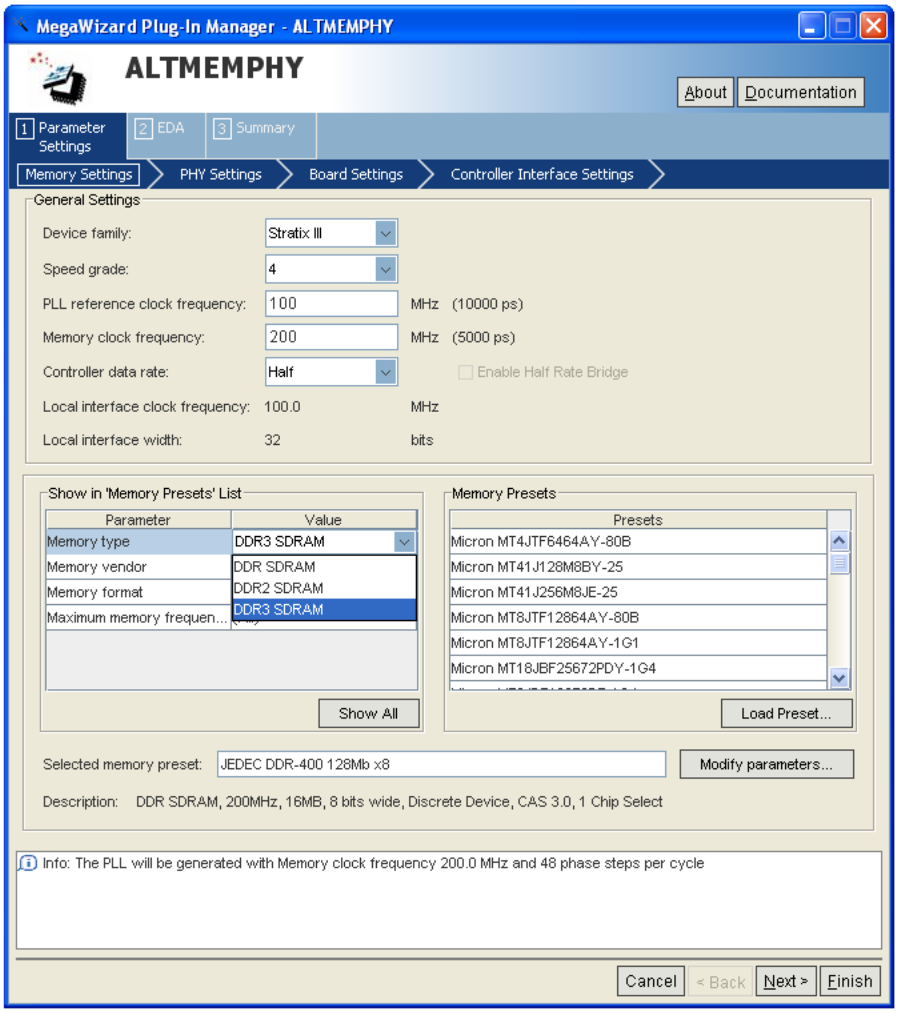
3. Parameter Settings

ALTMEMPHY Parameter Settings

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

- Memory Settings
- PHY Settings
- Board Settings

Figure 3–1. ALTMEMPHY Parameter Settings Page



General Settings

Device family:	Stratix III	
Speed grade:	4	
PLL reference clock frequency:	100	MHz (10000 ps)
Memory clock frequency:	200	MHz (5000 ps)
Controller data rate:	Half	☐ Enable Half Rate Bridge
Local interface clock frequency:	100.0	MHz
Local interface width:	32	bits

Show in 'Memory Presets' List

Parameter	Value
Memory type	DDR3 SDRAM
Memory vendor	DDR SDRAM
Memory format	DDR2 SDRAM
Maximum memory frequen...	DDR3 SDRAM

Memory Presets

Presets
Micron MT4JTF6464AY-80B
Micron MT41J128M8BY-25
Micron MT41J256M8JE-25
Micron MT8JTF12864AY-80B
Micron MT8JTF12864AY-1G1
Micron MT18JBF25672PDY-1G4

Selected memory preset: JEDEC DDR-400 128Mb x8

Description: DDR SDRAM, 200MHz, 16MB, 8 bits wide, Discrete Device, CAS 3.0, 1 Chip Select

Info: The PLL will be generated with Memory clock frequency 200.0 MHz and 48 phase steps per cycle

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

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

Memory Settings

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

Table 3–1 describes the **General Settings** available on the **Memory Settings** page of the ALTMEMPHY parameter editor.

Table 3–1. General Settings

Parameter Name	Description
Device family	Targets device family (for example, Arria II GX). Table 1–2 on page 1–3 shows supported device families. The device family selected here must match the device family selected on the MegaWizard page 2a.
Speed grade	Selects a particular speed grade of the device (for example, 2, 3, or 4 for the Arria II GX 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). The full-rate option is not available for DDR3 SDRAM devices.
Enable half rate bridge	This option is only available for HPC II full-rate controller. 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.
Local interface width	Value that depends on the memory clock frequency and controller data rate.

☞ When targeting a HardCopy device migration with performance improvement, the ALTMEMPHY IP should target the mid speed grade to ensure that the PLL and the PHY sequencer settings match. The compilation of the design can be executed in the faster speed grade.

Table 3–2 describes the options available to filter the **Memory Presets** that are displayed. This set of options is where you indicate whether you are creating a datapath for DDR3 SDRAM.

Table 3–2. Memory Presets List

Parameter Name	Description
Memory type	You can filter the type of memory to display, for example, DDR3 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 DDR3 SDRAM interfaces. Then, pick a device that has similar specifications to your chosen device and check the values of each parameter. Make sure you change the each parameter value to match your device specifications.
Memory format	You can filter the type of memory by format, for example, discrete devices or DIMM packages.
Maximum frequency	You can filter the type of memory by the maximum operating frequency.

Using the Preset Editor to Create a Custom Memory Preset

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

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

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

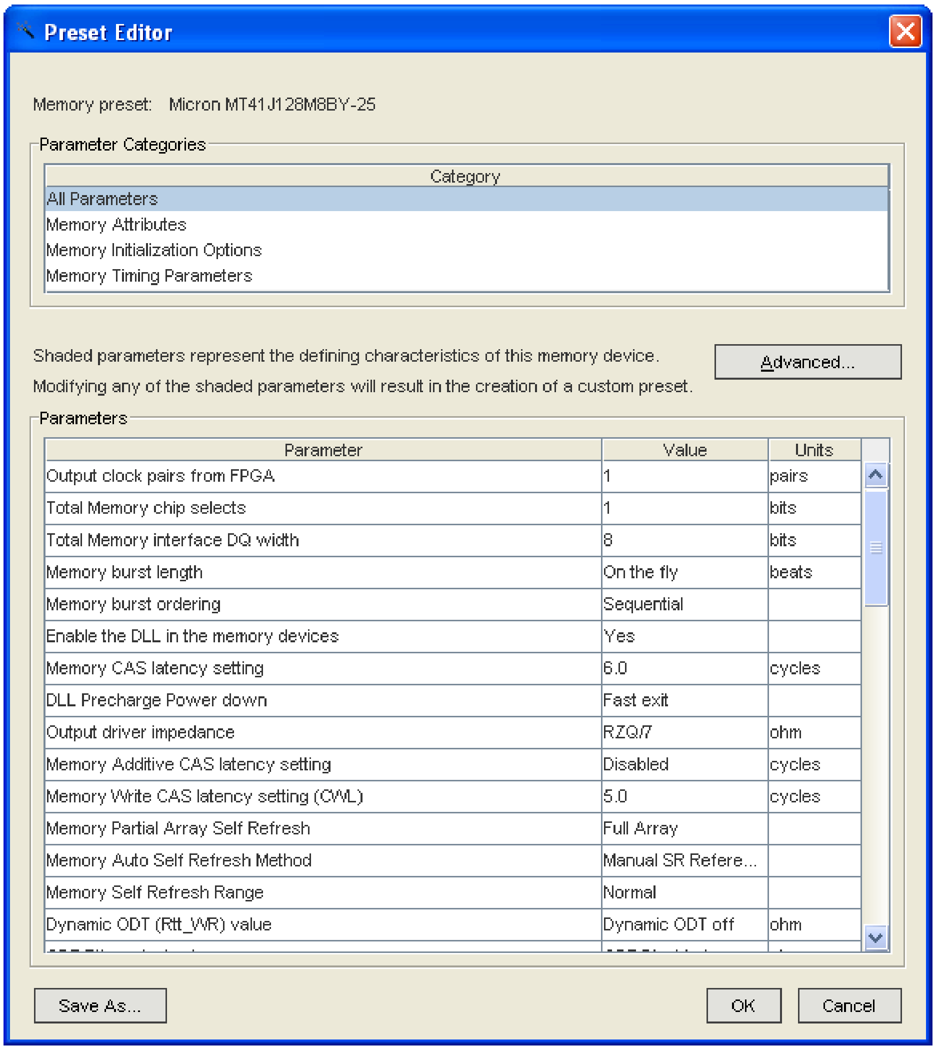
You can change the parameters with a white background to reflect your system. You can also change the parameters with a gray background so the device parameters match the device you are using. These parameters in gray background are characteristics of the chosen memory device and changing them creates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

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

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

Figure 3–2 shows the **Preset Editor** dialog box for a DDR3 SDRAM.

Figure 3–2. DDR3 SDRAM Preset Editor



Memory preset: Micron MT41J128M8BY-25

Parameter Categories

Category
All Parameters
Memory Attributes
Memory Initialization Options
Memory Timing Parameters

Shaded parameters represent the defining characteristics of this memory device.
Modifying any of the shaded parameters will result in the creation of a custom preset.

Parameters

Parameter	Value	Units
Output clock pairs from FPGA	1	pairs
Total Memory chip selects	1	bits
Total Memory interface DQ width	8	bits
Memory burst length	On the fly	beats
Memory burst ordering	Sequential	
Enable the DLL in the memory devices	Yes	
Memory CAS latency setting	6.0	cycles
DLL Precharge Power down	Fast exit	
Output driver impedance	RZQ/7	ohm
Memory Additive CAS latency setting	Disabled	cycles
Memory Write CAS latency setting (CWL)	5.0	cycles
Memory Partial Array Self Refresh	Full Array	
Memory Auto Self Refresh Method	Manual SR Refere...	
Memory Self Refresh Range	Normal	
Dynamic ODT (Rtt_WR) value	Dynamic ODT off	ohm

The **Advanced** 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 DDR3 SDRAM parameters available for memory attributes, initialization options, and timing parameters.

Table 3–3. DDR3 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. Memory clock pins use the signal splitter feature in Arria II GX devices for differential signaling. The ALTMEMPHY parameter editor displays an error on the bottom of the window if you choose more than one for DDR3 SDRAM interfaces.
Total Memory chip selects	1, 2, 4, or 8	bits	Sets the number of chip selects in your memory interface. The number of chip selects defines the depth of your memory. You are limited to the range shown as the local side binary encodes the chip select address.
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, DDR3 SDRAM variations are only supported up to 80-bit width due to restrictions in the board layout which affects timing at higher data width. Furthermore, 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.
Mirror addressing	—	—	On multiple rank DDR3 SDRAM DIMMs address signals are routed differently to each rank; referred to in the JEDEC specification as address mirroring. Enter ranks with mirrored addresses in this field. There is one bit per chip select. For example, for four chip selects, enter 1011 to mirror the address on chip select #3, #1, and #0.
Memory vendor	Elpida, JEDEC, Micron, Samsung, Hynix, Nanya, other	—	Lists the name of the memory vendor for all supported memory standards.
Memory format	Discrete Device	—	Arria II GX devices only support DDR3 SDRAM components without leveling, for example, **Discrete Device** memory format.
Maximum memory frequency	See the memory device datasheet	MHz	Sets the maximum frequency supported by the memory.
Column address width	10–12	bits	Defines the number of column address bits for your interface.
Row address width	12–16	bits	Defines the number of row address bits for your interface. If your DDR3 SDRAM device's row address bus is 12-bit wide, set the row address width to **13** and set the 13th bit to logic-level low (or leave the 13th bit unconnected to the memory device) in the top-level file.
Bank address width	3	bits	Defines the number of bank address bits for your interface.

Table 3–3. DDR3 SDRAM Attributes Settings (Part 2 of 2)

Parameter Name	Range (1)	Units	Description
Chip selects per device	1 or 2	bits	Defines the number of chip selects on each device in your interface. Currently, calibration is done with all ranks but you can only perform timing analysis with one.
DQ bits per DQS bit	4 or 8	bits	Defines the number of data (DQ) bits for each data strobe (DQS) pin.
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 with ×4 mode.
Maximum memory frequency for CAS latency 5.0	80–700	MHz	Specifies the frequency limits from the memory data sheet per given CAS latency. The ALTMEMPHY MegaWizard Plug-In Manager generates a warning if the operating frequency with your chosen CAS latency exceeds this number. The lowest frequency supported by DDR3 SDRAM devices is 300 MHz.
Maximum memory frequency for CAS latency 6.0			
Maximum memory frequency for CAS latency 7.0			
Maximum memory frequency for CAS latency 8.0			
Maximum memory frequency for CAS latency 9.0			
Maximum memory frequency for CAS latency 10.0			

Note to Table 3–3:

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

Table 3–4. DDR3 SDRAM Initialization Options (Part 1 of 2)

Parameter Name	Range	Units	Description
Memory burst length	4, 8, on-the-fly	beats	Sets the number of words read or written per transaction.
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.
DLL precharge power down	Fast exit or Slow exit	—	Sets the mode register setting to disable (**Slow exit**) or enable (**Fast exit**) the memory DLL when CKE is disabled.
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.
ODT `Rtt` nominal value	ODT disable, RZQ/4, RZQ/2, RZQ/6	W	RZQ in DDR3 SDRAM interfaces are set to 240 Ω. Sets the on-die termination (ODT) value to either 60 Ω (**RZQ/4**), 120 Ω (**RZQ/2**), or 40 Ω (**RZQ/6**). Set this to **ODT disable** if you are not planning to use ODT. For a single-ranked DIMM, set this to **RZQ/4**.

Table 3–4. DDR3 SDRAM Initialization Options (Part 2 of 2)

Parameter Name	Range	Units	Description
Dynamic ODT (`Rtt_WR`) value	Dynamic ODT off, RZQ/4, RZQ/2	W	RZQ in DDR3 SDRAM interfaces are set to 240 Ω. Sets the memory ODT value during write operations to 60 Ω (**RZQ/4**) or 120 Ω (**RZQ/2**). As ALTMEMPHY only supports single rank DIMMs, you do not need this option (set to **Dynamic ODT off**).
Output driver impedance	RZQ/6 (Reserved) or RZQ/7	W	RZQ in DDR3 SDRAM interfaces are set to 240 Ω. Sets the output driver impedance from the memory device. Some devices may not have **RZQ/6** available as an option. Be sure to check the memory device datasheet before choosing this option.
Memory CAS latency setting	5.0, 6.0, 7.0, 8.0, 9.0, 10.0	cycles	Sets the delay in clock cycles from the read command to the first output data from the memory.
Memory additive CAS latency setting	Disable, CL – 1, CL – 2	cycles	Allows you to add extra latency in addition to the CAS latency setting.
Memory write CAS latency setting (CWL)	5.0, 6.0, 7.0, 8.0	cycles	Sets the delay in clock cycles from the write command to the first expected data to the memory.
Memory partial array self refresh	Full array, Half array {BA[2:0]=000,001, 010,011}, Quarter array {BA[2:0]=000,001} , Eighth array {BA[2:0]=000}, Three Quarters array {BA[2:0]=010,011, 100,101,110,111}, Half array {BA[2:0]=100,101, 110,111}, Quarter array {BA[2:0]=110, 111}, Eighth array {BA[2:0]=111}	—	Determine whether you want to self-refresh only certain arrays instead of the full array. According to the DDR3 SDRAM specification, data located in the array beyond the specified address range are lost if **self refresh** is entered when you use this. This option is not supported by the DDR3 SDRAM Controller with ALTMEMPHY IP, so set to **Full Array** if you are using the Altera controller.
Memory auto self refresh method	Manual SR reference (SRT) or ASR enable (Optional)	—	Sets the auto self-refresh method for the memory device. The DDR3 SDRAM Controller with ALTMEMPHY IP currently does not support the ASR option that you need for extended temperature memory self-refresh.
Memory self refresh range	Normal or Extended	—	Determines the temperature range for self refresh. You need to also use the optional auto self refresh option when using this option. The Altera controller currently does not support the extended temperature self-refresh operation.

Table 3–5. DDR3 SDRAM Timing Parameter Settings (Part 1 of 2) ***(Note 1)***

Parameter Name	Range	Units	Description
Time to hold memory reset before beginning calibration	0–1000000	µs	Minimum time to hold the reset after a power cycle before issuing the MRS commands during the DDR3 SDRAM device initialization process.
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 DDR3 SDRAM high-performance controller and in terms of t_{CK} cycles in Micron's device datasheet. Convert t_{MRD} to ns by multiplying the number of cycles specified in the datasheet times t_{CK}, where t_{CK} is the memory operation frequency and not the memory device's t_{CK}.
t_{RAS}	8–200	ns	Minimum active to precharge time. The controller waits for this period of time after issuing an active command before issuing a precharge command to the same bank.
t_{RCD}	4–65	ns	Minimum active to read-write time. The controller does not issue read or write commands to a bank during this period of time after issuing an active command.
t_{RP}	4–65	ns	Minimum precharge command period. The controller does not access the bank for this period of time after issuing a precharge command.
t_{REFI}	1–65534	µs	Maximum interval between refresh commands. The controller performs regular refresh at this interval unless user-controlled refresh is turned on.
t_{RFC}	14–1651	ns	Minimum autorefresh command period. The length of time the controller waits before doing anything else after issuing an auto-refresh command.
t_{WR}	4–65	ns	Minimum write recovery time. The controller waits for this period of time after the end of a write transaction before issuing a precharge command.
t_{WTR}	1–6	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}	0–750	ps	DQ output access time.
t_{DQSCK}	50–750	ps	DQS output access time from CK/CK# signals.
t_{DQSQ}	50–500	ps	The maximum DQS to DQ skew; DQS to last DQ valid, per group, per access.
t_{DQSS}	0–0.3	t_{CK}	Positive DQS latching edge to associated clock edge.

Table 3–5. DDR3 SDRAM Timing Parameter Settings (Part 2 of 2) ***(Note 1)***

Parameter Name	Range	Units	Description
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 differential DQS and DQ/DM signals. Ensure that you are using the correct number and that the value entered is referenced to V_{REF}(dc), not V_{IH}(dc) min or V_{IL}(dc) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification.
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 differential DQS signals and DQ/DM signals. Ensure that you are using the correct number and that the value entered is referenced to V_{REF}(dc), not V_{IH}(ac) min or V_{IL}(ac) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification.
t_{DSH}	0.1–0.5	t_{CK}	DQS falling edge hold time from CK.
t_{DSS}	0.1–0.5	t_{CK}	DQS falling edge to CK setup.
t_{IH}	50–1000	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 $_{VREF}$(dc), not V_{IH}(dc) min or V_{IL}(dc) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification.
t_{IS}	65–1000	ps	Address and control input setup time, which has a derated value depending on the slew rate of the CK and CK# clocks and the address and command signals. Ensure that you are using the correct number and that the value entered is referenced to V_{REF}(dc), not V_{IH}(ac) min or V_{IL}(ac) max. Refer to "Derating Memory Setup and Hold Timing" on page 3–10 for more information about how to derate this specification.
t_{QHS}	0–700	ps	The maximum data hold skew factor.
t_{QH}	0.1–0.6	t_{CK}	DQ output hold time.
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).

Derating Memory Setup and Hold Timing

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

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

For Arria II GX 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}. 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.

The following memory device specifications and update the **Preset Editor** dialog box with the derated value:

For example, according to JEDEC, 533-MHz DDR3 SDRAM has the following specifications, assuming 1V/ns DQ slew rate rising signal and 2V/ns DQS-DQSn slew rate:

- Base $t_{DS} = 25$
- Base $t_{DH} = 100$
- $V_{IH}(ac) = V_{REF} + 0.175$ V
- $V_{IH}(dc) = V_{REF} + 0.100$ V
- $V_{IL}(ac) = V_{REF} – 0.175$ V
- $V_{IL}(dc) = V_{REF} – 0.100$ V

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 = 25 + 0 + 175 = 200 ps

t_{DH} (V_{REF}) = Base t_{DH} + delta t_{DH} + ($V_{IH}(dc) – V_{REF}$)/slew_rate = 100 + 0 + 100 = 200 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 + 88 + 87.5 = 200.5 ps

t_{DH} (V_{REF}) = Base t_{DH} + delta t_{DH} + ($V_{IH}(dc) – V_{REF}$)/slew_rate = 100 + 50 + 50 = 200 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 + 5 + 350 = 380 ps

t_{DH} (V_{REF}) = Base t_{DH} + delta t_{DH} + (V_{IH}(dc) – V_{REF})/slew_rate = 100 + 10 + 200 = 310 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

Parameter Name	Applicable Device Families	Description
Enable external access to reconfigure PLL prior to calibration	HardCopy II	When enabling this option for 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.	Use this option 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.
Clock phase	Arria 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.
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.

Table 3–7. ALTMEMPHY Board Settings (Part 1 of 2)

Parameter Name	Units	Description
Number of slots/discrete devices	—	Sets the single-rank or multi-rank configuration.
CK/CK# slew rate (differential)	V/ns	Sets the differential slew rate for the CK and CK# signals.
Addr/command slew rate	V/ns	Sets the slew rate for the address and command signals.
DQ/DQS# slew rate (differential)	V/ns	Sets the differential slew rate for the DQ and DQS# signals.

Table 3–7. ALTMEMPHY Board Settings (Part 2 of 2)

Parameter Name	Units	Description
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.
Max skew within DQS group	ns	Sets the largest skew between the DQ pins in a DQS group.
Max skew between DQS groups	ns	Sets the largest skew between DQS signals in different DQS groups.
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.

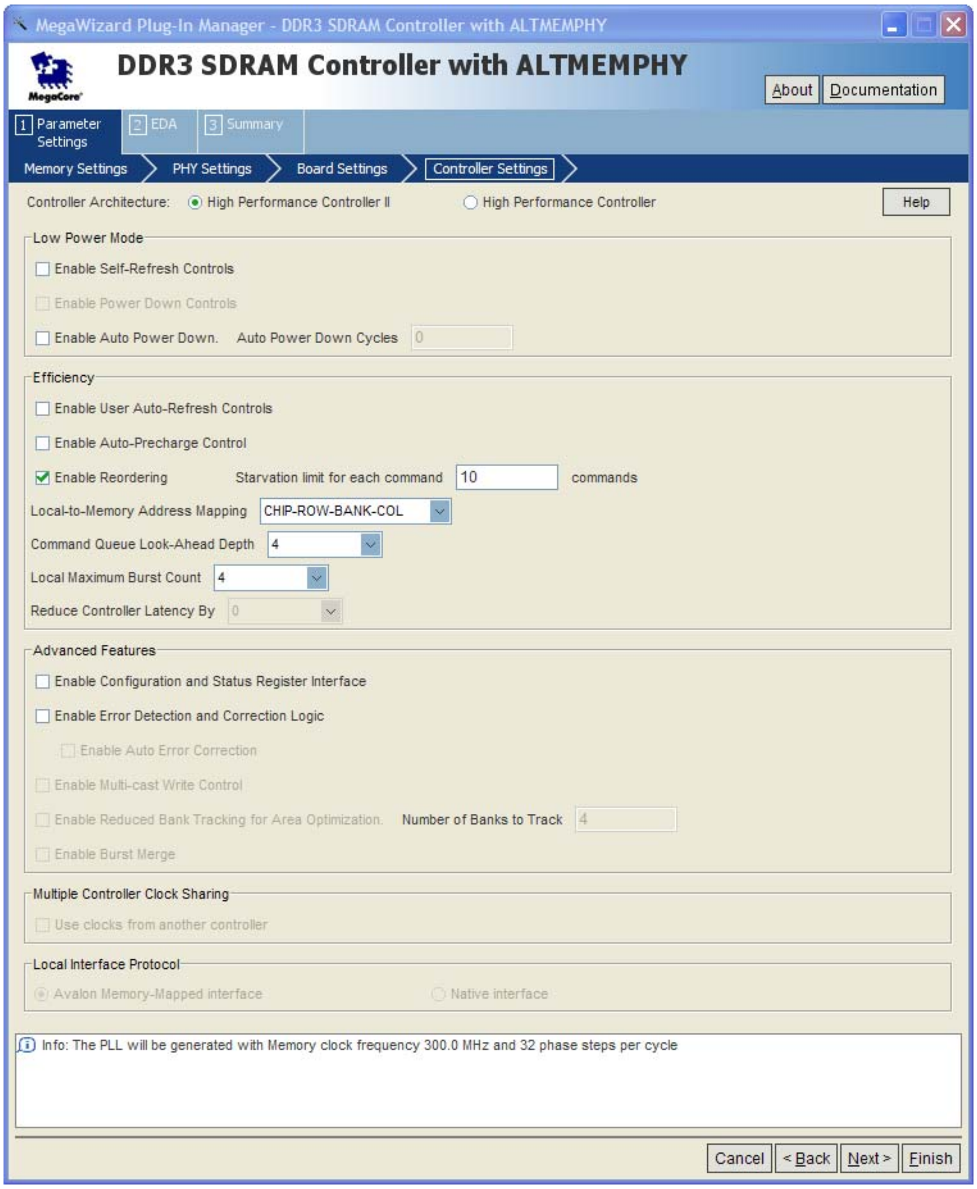
DDR3 SDRAM Controller with ALTMEMPHY Parameter Settings

The **Parameter Settings** page in the DDR3 SDRAM Controller with ALTMEMPHY parameter editor (Figure 3–3) allows you to parameterize the following settings:

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

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

Figure 3–3. DDR3 SDRAM Controller with ALTMEMPHY Settings


MegaWizard Plug-In Manager - DDR3 SDRAM Controller with ALTMEMPHY

DDR3 SDRAM Controller with ALTMEMPHY

About | Documentation

1 Parameter Settings | 2 EDA | 3 Summary

Memory Settings > PHY Settings > Board Settings > Controller Settings

Controller Architecture: (● High Performance Controller II) ○ High Performance Controller — Help

Low Power Mode
- ☐ Enable Self-Refresh Controls
- ☐ Enable Power Down Controls
- ☐ Enable Auto Power Down. Auto Power Down Cycles: 0

Efficiency
- ☐ Enable User Auto-Refresh Controls
- ☐ Enable Auto-Precharge Control
- ☑ Enable Reordering — Starvation limit for each command: 10 commands

Local-to-Memory Address Mapping	CHIP-ROW-BANK-COL
Command Queue Look-Ahead Depth	4
Local Maximum Burst Count	4
Reduce Controller Latency By	0

Advanced Features
- ☐ Enable Configuration and Status Register Interface
- ☐ Enable Error Detection and Correction Logic
  - ☐ Enable Auto Error Correction
- ☐ Enable Multi-cast Write Control
- ☐ Enable Reduced Bank Tracking for Area Optimization. Number of Banks to Track: 4
- ☐ Enable Burst Merge

Multiple Controller Clock Sharing
- ☐ Use clocks from another controller

Local Interface Protocol
- (● Avalon Memory-Mapped interface) ○ Native interface

Info: The PLL will be generated with Memory clock frequency 300.0 MHz and 32 phase steps per cycle

Cancel | < Back | Next > | Finish

Controller Settings

This section describes parameters for the High Performance Controller II (HPC II) with advanced features introduced in version 11.0 for designs generated in version 11.0. Designs created in earlier versions and regenerated in version 11.0 do not inherit the new advanced features; for information on parameters for HPC II without the version 11.0 advanced features, refer to the External Memory Interface Handbook for Quartus II version 10.1, available in the *External Memory Interfaces* section of the Altera Literature website.

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

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

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

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

Parameter	Description
Command queue look-ahead depth	Specifies a command queue look-ahead depth value to control the number of read or write requests the look-ahead bank management logic examines.
Local maximum burst count	Specifies a burst count to configure the maximum Avalon burst count that the controller slave port accepts.
Reduce controller latency by	Specifies, in controller clock cycles, a value by which to reduce the controller latency. The default value is **0** but you have the option to choose **1** to enhance the latency performance of your design at the expense of timing closure.
Enable configuration and status register interface	Turn on to enable run-time configuration and status retrieval of the memory controller. Enabling this option adds an additional Avalon-MM slave port to thememory 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 6–11.
Enable error detection and correction logic	Turn on to enable error correction coding (ECC) for single-bit error correction and double-bit error detection.
Enable auto error correction	Turn on to allow the controller to perform auto correction when the ECC logic detects a single-bit error. Alternatively, you can turn off this option and schedule the error correction at a desired time for better system efficiency.
Multiple controller clock sharing	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.
Local interface protocol	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.

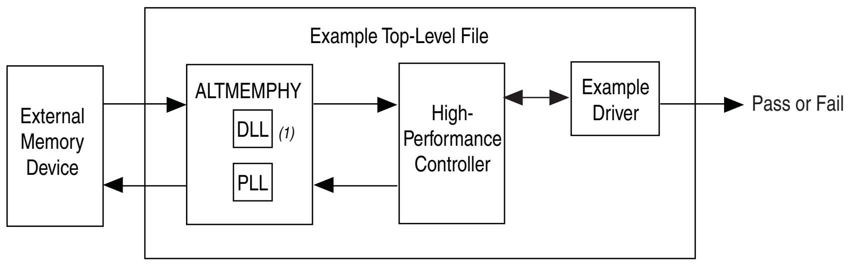
4. Compiling and Simulating

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

Compiling the Design

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

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



Note to Figure 4–1:

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

Before compiling a design with the ALTMEMPHY variation, you must edit some project settings, include the **.sdc** file, and make I/O assignments. I/O assignments include I/O standard, pin location, and other assignments, such as termination and drive strength settings. Some of these tasks are listed in the ALTMEMPHY **Generation** window.

To use the Quartus II software to compile the example top-level file in the Quartus II software and perform post-compilation timing analysis, perform the following 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. To edit the **.tcl** file, perform the following steps:

a. Open <*variation name*>**_pin_assignments.tcl** file.

b. Based on the flow you are using, set the `sopc_mode` value to Yes or No.

   ■ SOPC Builder System flow:

   ```
   if {![info exists 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 performing the following steps:

  a. On the Assignments menu, click **Pin Planner**.

  b. On the Edit menu, click **Create/Import Megafunction**.

  c. Select **Import an existing custom megafunction** and navigate to <*variation name*>**.ppf**.

  d. Type the prefix you want to use in **Instance name**. For example, change **mem_addr** to **core1_mem_addr**.

3. Set the top-level entity to the top-level design.

   a. On the File menu, click **Open**.

   b. Browse to your SOPC Builder system top-level design or <*variation name*>**_example_top** if you are using MegaWizard Plug-In Manager, and click **Open**.

   c. On the Project menu, click **Set as Top-Level Entity**.

4. Assign the DQ and DQS pin locations.

   a. You should assign pin locations to the pins in your design, so the Quartus II software can perform fitting and timing analysis correctly.

   b. Use either the Pin Planner or Assignment Editor to assign the clock source pin manually. Also choose which DQS pin groups should be used by assigning each DQS pin to the required pin. The Quartus II Fitter then automatically places the respective DQ signals onto suitable DQ pins within each group.

   ☞ To avoid no-fit errors when you compile your design, ensure that you place the `mem_clk` pins to the same edge as the `mem_dq` and `mem_dqs` pins, and set an appropriate I/O standard for the non-memory interfaces, such as the clock source and the reset inputs, when assigning pins in your design. For example, for DDR3 SDRAM select **1.5 V**. Also select in which bank or side of the device you want the Quartus II software to place them.

   The ×4 DIMM has the following mapping between DQS and DQ pins:

   - DQS[0] maps to DQ[3:0]
   - DQS[1] maps to DQ[7:4]
   - DQS[2] maps to DQ[11:8]
   - DQS[3] maps to DQ[15:12]

   The DQS pin index in other ×4 DIMM configurations typically increases sequentially with the DQ pin index (DQS[0]: DQ[3:0]; DQS[1]: DQ[7:4]; DQS[2]: DQ[11:8])

5. 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**.

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

Simulating the Design

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

For more information about simulating SOPC Builder systems, refer to volume 4 of the *Quartus II Handbook* and *AN 351: Simulating Nios II Embedded Processor Designs*. For more information about how to include your board simulation results in the Quartus II software and how to assign pins using pin planners, refer to *ALTMEMPHY Design Tutorials* section in volume 5 of the *External Memory Interface Handbook*.

You have the following three simulation options:

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

  Available for ×4 and ×8 DDR3 SDRAM. Skip calibration simulation is supported for 300 MHz through 400 MHz. There is no calibration in this simulation mode. As no phase calibration is performed, there must be no delays in the testbench.

  The ALTMEMPHY megafunction is statically configured to provide the correct write and read latencies. Skip calibration provides the fastest simulation time for DDR3 SDRAM interfaces. Use the generated or vendor DDR3 SDRAM simulation models for this simulation option.

  Skip calibration simulation between 300 MHz and 400 MHz supports CAS latency of 6 and a CAS write latency of 5.

  The additive latency must be disabled.

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

  Available for ×4 and ×8 DDR3 SDRAM. In quick calibration simulation mode, the sequencer only does clock cycle calibration. So there must be no delays (DDR3 DIMM modeling for example) in the testbench, because no phase calibration is performed. Quick calibration mode can be used between 300 MHz and 400 MHz. Both the generated or vendor DDR3 SDRAM simulation models support burst length on-the-fly changes during the calibration sequence.

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

  Available for ×4 and ×8 DDR3 SDRAM between 300 MHz and 400 MHz. You cannot use the wizard-generated memory model, if you select **Full Calibration**. You must use a memory-vendor provided memory model that supports write leveling calibration.

If you are simulating your ALTMEMPHY-based design with a Denali model, Altera recommends that you use full calibration mode.

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

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 the Altera high-performance controller. In addition, the ALTMEMPHY megafunction and the Altera high-performance controller are available for half-rate DDR3 SDRAM interfaces.

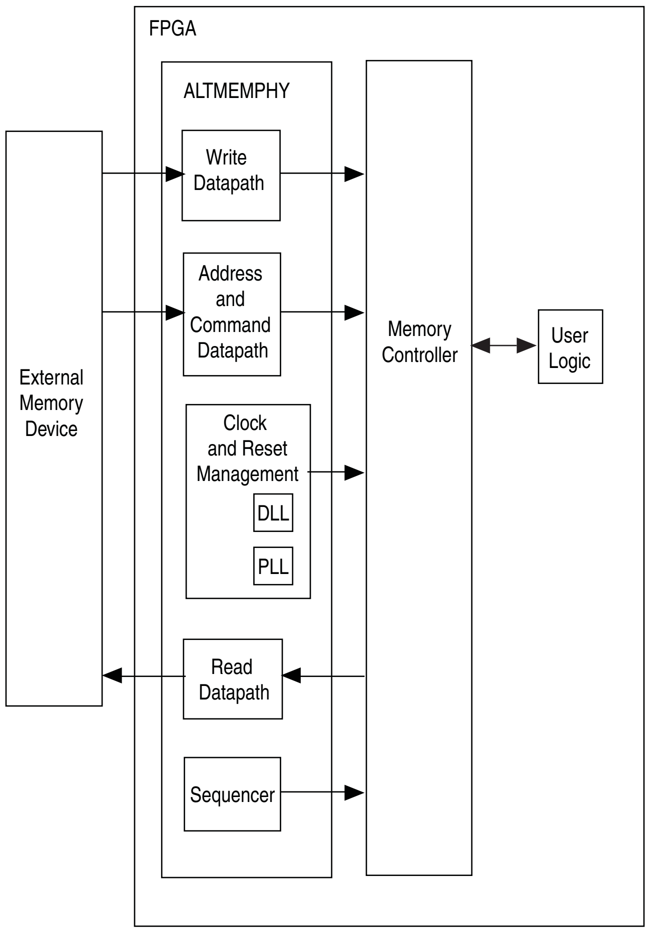
☞ If the ALTMEMPHY megafunction does not meet your requirements, you can also create your own memory interface datapath using the ALTDLL and ALTDQ_DQS megafunctions, available in the Quartus II software. However, you are then responsible for every aspect of the interface, including timing analysis and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

This chapter describes the DDR3 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

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

Calibration

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

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

Address and Command Datapath

This topic discusses the address and command datapath.

Arria II GX Devices

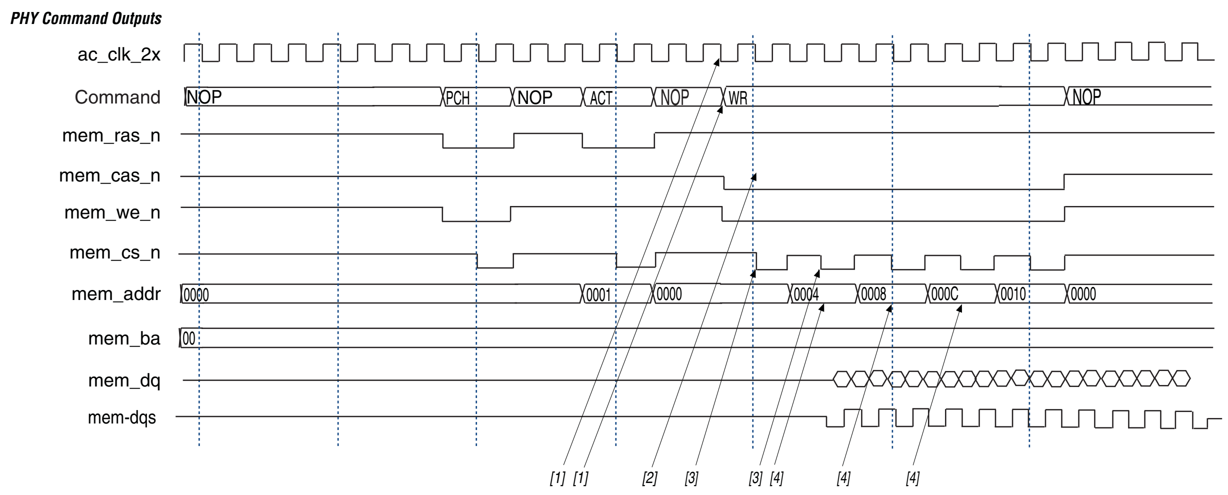
The address and command datapath is responsible for taking the address and command outputs from the controller and converting them from half-rate clock to full-rate clock. Two types of addressing are possible:

- 1T (full rate)—the duration of the address and command is a single memory clock cycle (`mem_clk_2x`, Figure 5–2). This applies to all address and command signals in full-rate designs or `mem_cs_n`, `mem_cke`, and `mem_odt` signals in half-rate designs.
- 2T (half rate)—the duration of the address and command is two memory clock cycles. For half-rate designs, the ALTMEMPHY megafunction supports only a burst size of four, which means the burst size on the local interface is always set to 1. The size of the data is 4n-bits wide on the local side and is *n*-bits wide on the memory side. To transfer all the 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 5–1 on page 5–6 to see the frequency relationship of `mem_clk_2x` with the rest of the clocks.

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

Figure 5–2. Arria II GX Address and Command Datapath



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

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

1. The commands are asserted either on the rising edge of `ac_clk_2x`. The `ac_clk_2x` is derived from either `mem_clk_2x` (0°), `write_clk_2x` (270°), or the inverted variations of those two clocks (for 180° and 90° phase shifts). This depends on the setting of the address and command clock in the ALTMEMPHY parameter editor. Refer to "Address and Command Datapath" on page 5–3 for illustrations of this clock in relation to the `mem_clk_2x` or `write_clk_2x` signals.
2. All address and command signals (except for `mem_cs_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.

Clock and Reset Management

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

Clock Management

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

Certain clocks require phase shifting during the ALTMEMPHY megafunction operation.

You can implement clock management circuitry using PLLs and DLLs.

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

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

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

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

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

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

Table 5–1 shows the clock outputs that Arria II GX devices use.

Table 5–1. DDR3 SDRAM Clocking in Arria II GX Devices (Part 1 of 2)

Clock Name (1)	Postscale Counter	Phase (Degrees)	Clock Rate	Clock Network Type		Notes
				All Quadrants	Any 3 Quadrants (2)	
`phy_clk_1x` and `aux_half_rate_clk`	C0	0°	Half-Rate	Global	Global	The only clocks parameterizable for the ALTMEMPHY megafunction. These clocks also feed into a divider circuit to provide the PLL `scan_clk` signal (for reconfiguration) that must be lower than 100 MHz.
`mem_clk_2x` and `aux_full_rate_clk`	C1	0°	Full-Rate	Global	Regional (3) Global (4)	This clock is for clocking DQS and as a reference clock for the memory devices.
`mem_clk_1x`	C2	0°	Half-Rate	Global	Regional	This clock is for clocking DQS and as a reference clock for the memory devices.
`write_clk_2x`	C3	–90°	Full-Rate	Global	Regional	This clock is for clocking the data out of the DDR I/O (DDIO) pins in advance of the DQS strobe (or equivalent). As a result, its phase leads that of the `mem_clk_2x` by 90°.
`ac_clk_2x`	C3	–90°	Full-Rate	Global	Regional	Address and command clock. The `ac_clk_2x` clock is derived from either `mem_clk_2x` (when you choose 0° or 180° phase shift) or `write_clk_2x` (when you choose 90° or 270° phase shift). Refer to "Address and Command Datapath" on page 5–3 for illustrations of the address and command clock relationship with the `mem_clk_2x` or `write_clk_2x` signals.
`cs_n_clk_2x`	C3	–90°	Full-Rate	Global	Global	Memory chip-select clock. The `cs_n_clk_2x` clock is derived from `ac_clk_2x`.
`resync_clk_2x`	C4	Calibrated	Full-Rate	Global	Regional	Clocks the resynchronization registers after the capture registers. Its phase is adjusted to the center of the data valid window across all the DQS-clocked DDIO groups.

Table 5–1. DDR3 SDRAM Clocking in Arria II GX Devices (Part 2 of 2)

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

Note to Table 5–1:

(1) The `_1x` clock represents a frequency that is half of the memory clock frequency; the `_2x` clock represents the memory clock frequency.

(2) The default clock network type is Global, however you can specify a regional clock network to improve clock jitter if your design uses any three quadrants.

(3) For mem_clk2x.

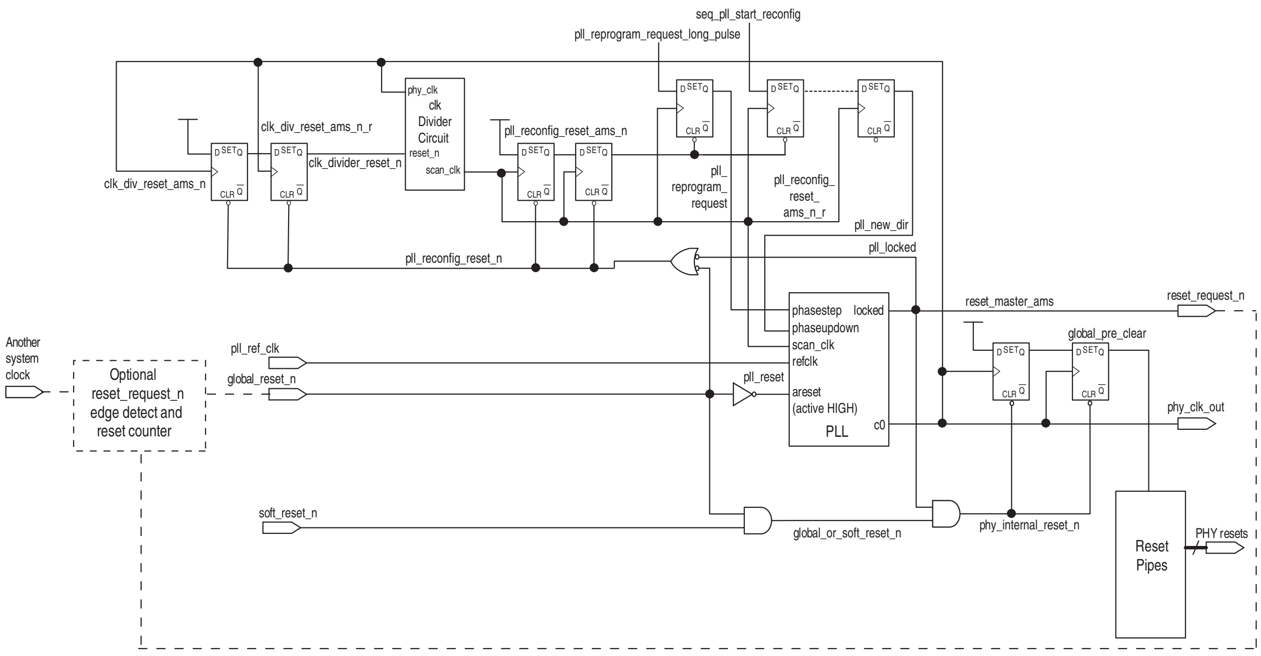
(4) For aux_full_rate_clk.

Reset Management

Figure 5–3 shows the main features of the reset management block for the DDR3 SDRAM PHY. You can use the `pll_ref_clk` input to feed the optional `reset_request_n` edge detect and reset counter module. However, this requires the `pll_ref_clk` signal to use a global clock network resource.

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

Figure 5–3. ALTMEMPHY Reset Management Block for Arria II GX Devices



Read Datapath

This topic discusses the read datapath.

Arria II GX Devices

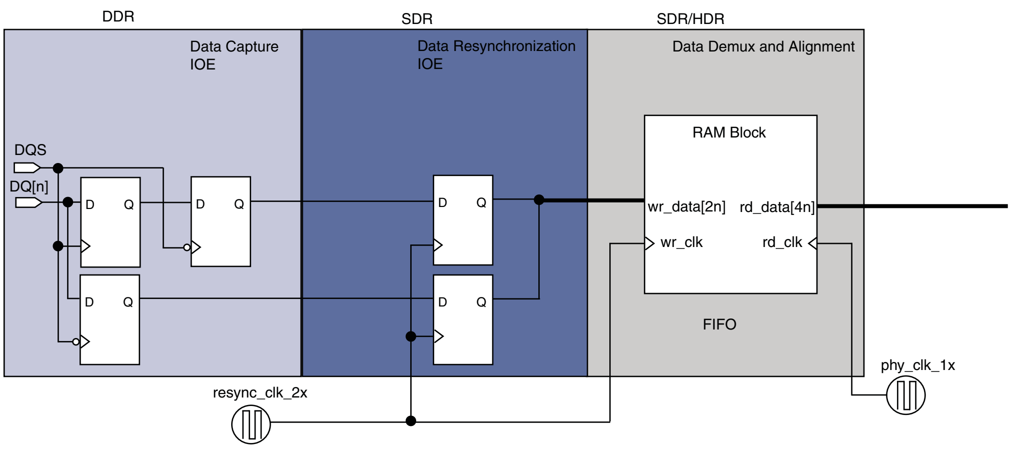
The read datapath logic captures data sent by the memory device and subsequently aligns the data back to the system clock domain. The read datapath for DDR3 SDRAM consists of the following three main blocks:

- Data capture
- Data resynchronization
- Data demultiplexing and alignment

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

Figure 5–4 shows the order of the functions performed by the read datapath and the frequency at which the read data is handled.

Figure 5–4. DDR3 SDRAM Read Datapath in Arria II GX Devices



Data Capture and Resynchronization

The data capture and resynchronization registers for Arria II GX devices are implemented in the I/O element (IOE) to achieve maximum performance. Data capture and resynchronization is the process of capturing the read data (DQ) with the DQS/DQSn strobes and resynchronizing the captured data to an internal free-running full-rate clock supplied by the enhanced PLL. The resynchronization clock is an intermediate clock whose phase shift is determined during the calibration stage. The captured data (`rdata_p_captured` and `rdata_n_captured`) is synchronized to the resynchronization clock (`resync_clk_2x`), refer to Figure 5–4. For Arria II GX devices, the ALTMEMPHY instances an ALTDQ_DQS megafunction that instantiates the required IOEs for all the DQ and DQS pins.

Data Demultiplexing

Data demultiplexing is the process of changing the SDR data into HDR data. Data demultiplexing is required to bring the frequency of the resynchronized data down to the frequency of the system clock, so that data from the external memory device can ultimately be brought into the FPGA controller clock domain. Before data capture, the data is DDR and *n*-bit wide. After data capture, the data is SDR and 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 4*n*-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 4*n*-bit wide read data follows the 2*n*-bit wide write data with a constant latency

Read Data Alignment

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

Postamble Protection

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

ALTMEMPHY Signals

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

Table 5–2 through Table 5–4 show the signals.

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

The signal lists include the following signal groups:

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

Table 5–2. Interface to the DDR3 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.

Table 5–2. Interface to the DDR3 SDRAM Devices *(Note 1)*

Signal Name	Type	Width *(2)*	Description
`mem_dq`	Bidirectional	`MEM_IF_DWIDTH`	The memory bidirectional data bus.
`mem_dqs`	Bidirectional	`MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS`	The memory bidirectional data strobe bus.
`mem_dqs_n`	Bidirectional	`MEM_IF_DWIDTH/ MEM_IF_DQ_PER_DQS`	The memory bidirectional data strobe bus.
`mem_odt`	Output	`MEM_IF_CS_WIDTH`	The memory on-die termination control signal.
`mem_ras_n`	Output	1	The memory row address strobe.
`mem_reset_n`	Output	1	The memory reset signal. This signal is derived from the PHY's internal reset signal, which is generated by gating the global reset, soft reset, and the PLL locked signal.
`mem_we_n`	Output	1	The memory write enable signal.
`mem_ac_parity` *(4)*	Output	1	The address or command parity signal generated by the PHY and sent to the DIMM.
parity_error_n *(4)*	Output	1	The active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset.
mem_err_out_n *(4)*	Input	1	The signal sent from the DIMM to the PHY to indicate that a parity error has occured for a particular cycle.

Notes to Table 5–2:

(1) Connected to I/O pads.

(2) Refer to Table 5–5 for parameter description.

(3) Output is for memory device, and input path is fed back to ALTMEMPHY megafunction for VT tracking.

(4) This signal is for Registered DIMMs only.

Table 5–3. AFI Signals (Part 1 of 4)

Signal Name	Type	Width *(1)*	Description
Clocks and Resets			
`pll_ref_clk`	Input	1	The reference clock input to the PHY PLL.
`global_reset_n`	Input	1	Active-low global reset for PLL and all logic in the PHY. A level set reset signal, which causes a complete reset of the whole system. The PLL may maintain some state information.
`soft_reset_n`	Input	1	Edge detect reset input intended for SOPC Builder use or to be controlled by other system reset logic. Causes a complete reset of PHY, but not the PLL used in the PHY.

Table 5–3. AFI Signals (Part 2 of 4)

Signal Name	Type	Width (1)	Description
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.
aux_full_rate_clk	Output	1	In full-rate designs, a copy of the mem_clk_2x signal that you can use in other parts of your design.
aux_scan_clk	Output	1	Low frequency scan clock supplied primarily to clock any user logic that interfaces to the PLL and DLL reconfiguration interfaces.
aux_scan_clk_reset_n	Output	1	This reset output asynchronously asserts (drives low) when global_reset_n is asserted and de-assert (drives high) synchronous to aux_scan_clk when global_reset_n is de-asserted. It allows you to reset any external circuitry clocked by aux_scan_clk.
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 the ctl_wdata_valid signal and must be driven for the correct duration to generate a correctly timed mem_dqs signal.
ctl_wdata_valid	Input	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.

Table 5–3. AFI Signals (Part 3 of 4)

Signal Name	Type	Width (1)	Description
`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 DDR3 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.
`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 signal is unused by the Altera high-performance controller.
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.

Table 5–3. AFI Signals (Part 4 of 4)

Signal Name	Type	Width (1)	Description
`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.
`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 5–2:

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

Table 5–4. Other Interface Signals (Part 1 of 2)

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.

Table 5–4. Other Interface Signals (Part 2 of 2)

Signal Name	Type	Width	Description
`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.
Calibration Interface Signals—without leveling only			
`rsu_codvw_phase`	Output	—	The sequencer sweeps the phase of a resynchronization clock across 360° or 720° of a memory clock cycle. Data reads from the DIMM are performed for each phase position, and a data valid window is located, which is the set of resynchronization clock phase positions where data is successfully read. The final resynchronization clock phase is set at the center of this range: the center of the data valid window or CODVW. This output is set to the current calculated value for the CODVW, and represents how many phase steps were performed by the PLL to offset the resynchronization clock from the memory clock.
`rsu_codvw_size`	Output	—	The final centre of data valid window size (`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 5–4:

(1) The debug interface uses the simple Avalon-MM interface protocol.

(2) These ports exist in the Quartus II software, even though the debug interface is for Altera's use only.

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

Table 5–5. 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.

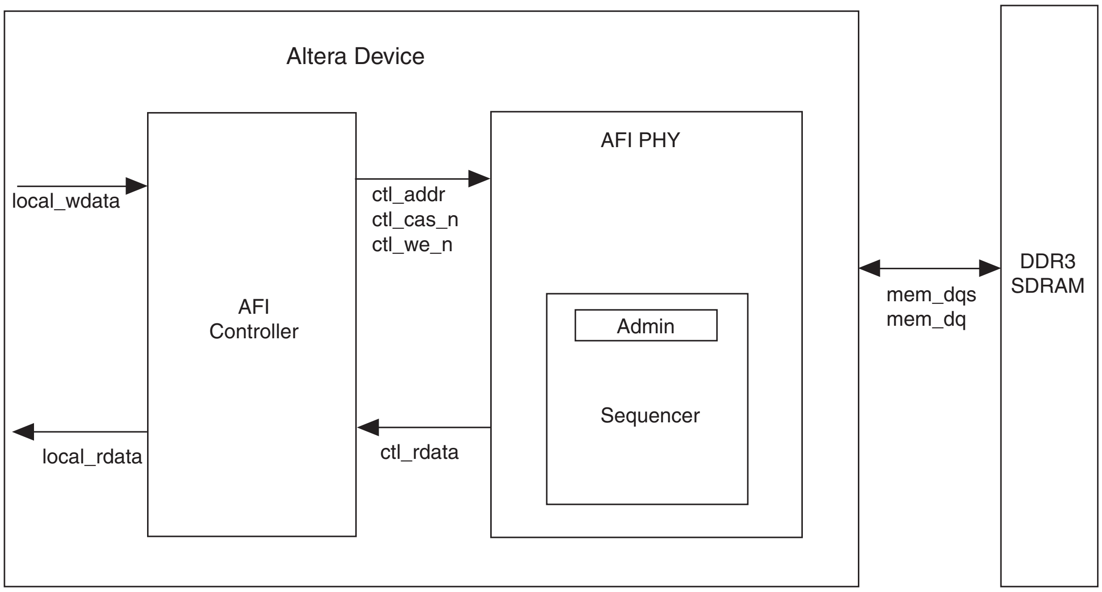
PHY-to-Controller Interfaces

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 controller. The AFI PHY includes an administration block that configures the memory for calibration and performs necessary mode registers accesses to configure the memory as required (these calibration processes are different). Figure 5–5 shows an overview of the connections between the PHY, the controller, and the memory device.

Altera recommends that you use the AFI for new designs.

Figure 5–5. AFI PHY Connections



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

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

For half-rate designs, the AFI allows the controller to issue reads and writes that are aligned to either half-cycle of the half-rate `phy_clk`, which means that the datapaths can support multiple data alignments—word-unaligned and word-aligned writes and reads. Figure 5–6 and Figure 5–7 display the half-rate write operation.

Figure 5–6. Half-Rate Write with Word-Unaligned Data

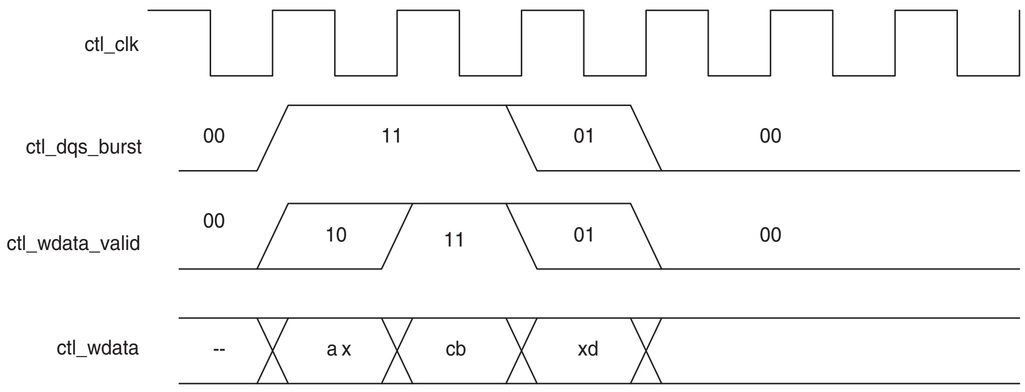
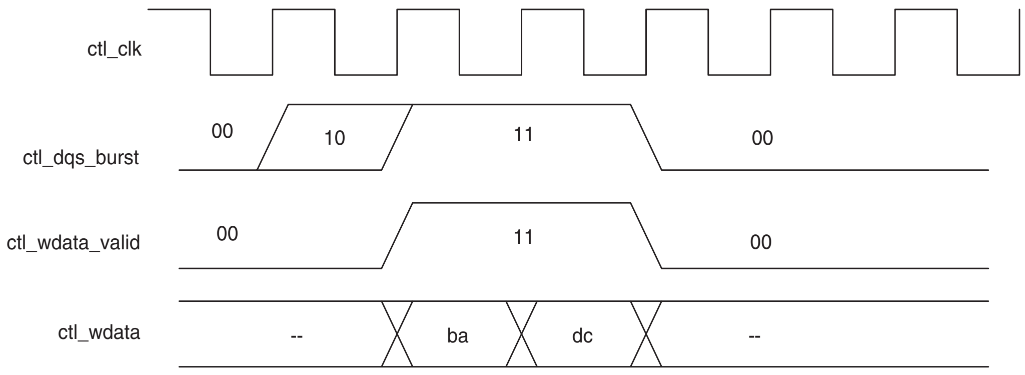


Figure 5–7. Half-Rate Write with Word-Aligned Data



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

Figure 5–8 and Figure 5–9 show word-aligned writes and reads. In the following read and write examples the data is written to and read from the same address. In each example, `ctl_rdata` and `ctl_wdata` are aligned with controller clock (`ctl_clk`) cycles. All the data in the bit vector is valid at once. For comparison, refer Figure 5–10 and Figure 5–11 that show the word-unaligned writes and reads.

☞ The `ctl_doing_rd` is represented as a half-rate signal when passed into the PHY. Therefore, the lower half of this bit vector represents one memory clock cycle and the upper half the next memory clock cycle. Figure 5–11 on page 5–23 shows separated word-unaligned reads as an example of two `ctl_doing_rd` bits are different. Therefore, for each x16 device, at least two `ctl_doing_rd` bits need to be driven, and two `ctl_rdata_valid` bits need to be interpreted.

The AFI has the following conventions:

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

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

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

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

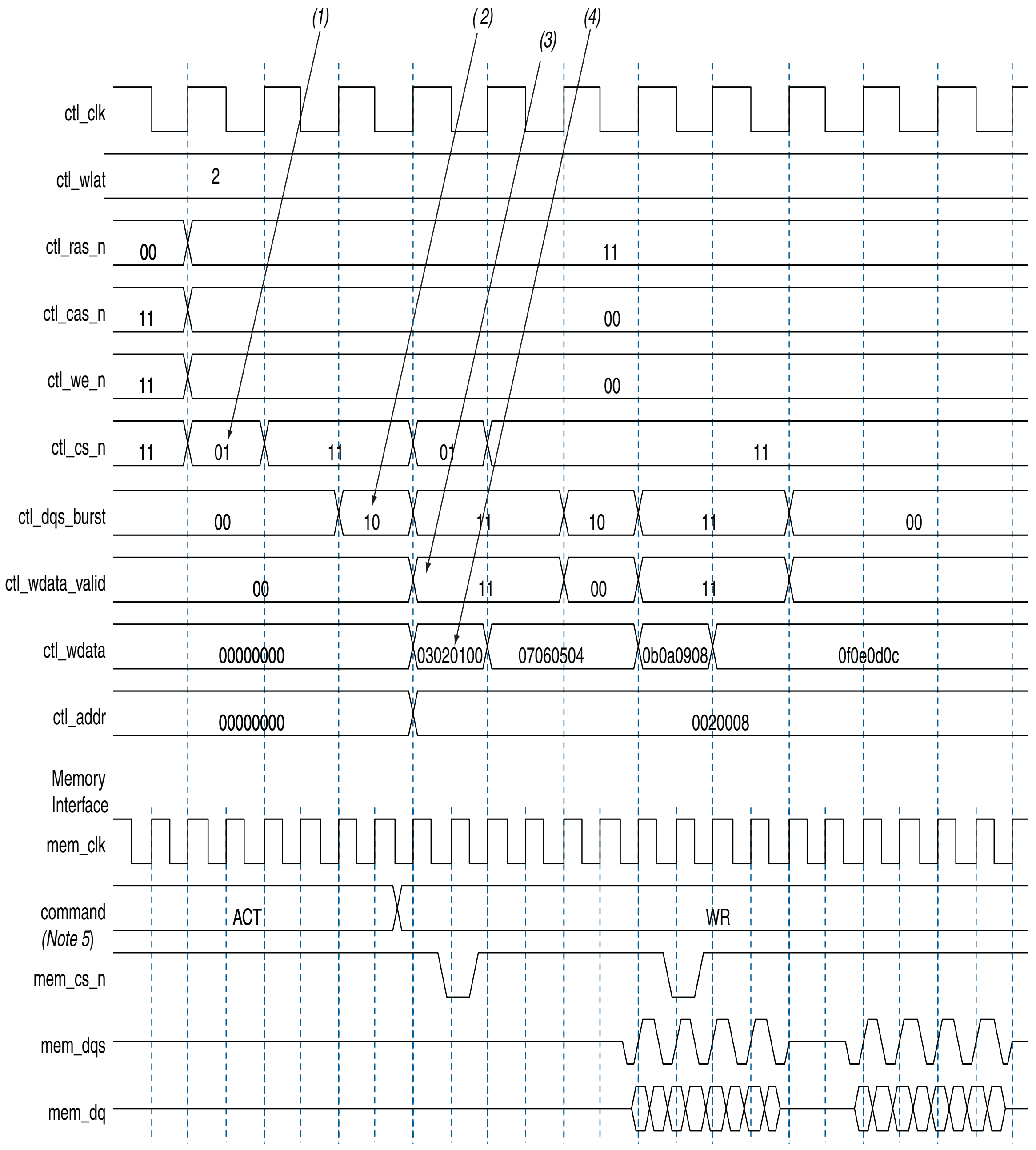
  ☞ The timing analysis script does not support word-unaligned reads and writes.

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

Figure 5–8 through Figure 5–11 assume the following general points:

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

Figure 5–8. Word-Aligned Writes



Notes to Figure 5–8:

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

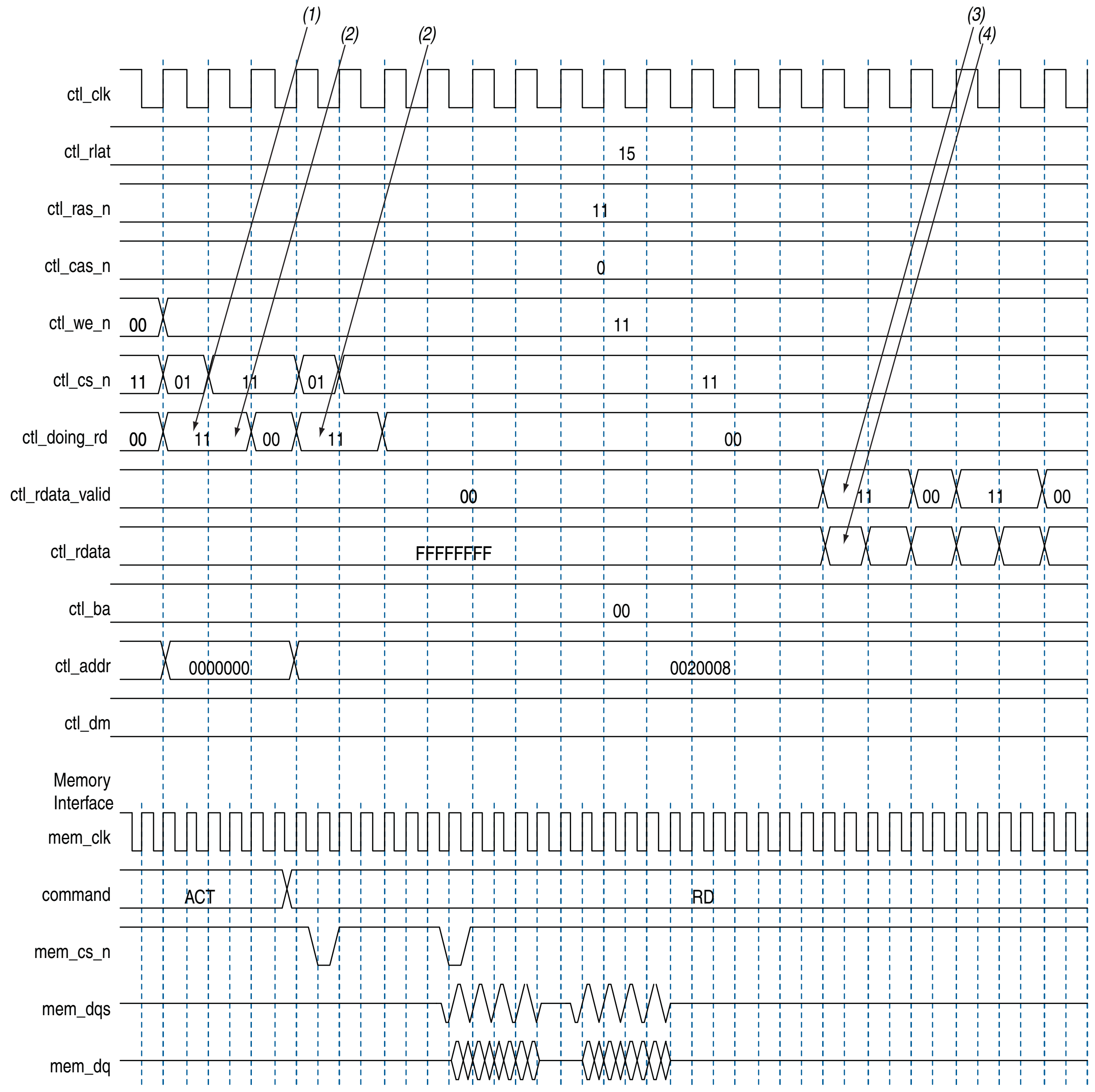
(2) The `ctl_dqs_burst` must go high one memory clock cycle before `ctl_wdata_valid`. Compare with the word-unaligned case.

(3) The `ctl_wdata_valid` is asserted two `ctl_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 5–9. Word-Aligned Reads



Notes to Figure 5–9:

(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`.

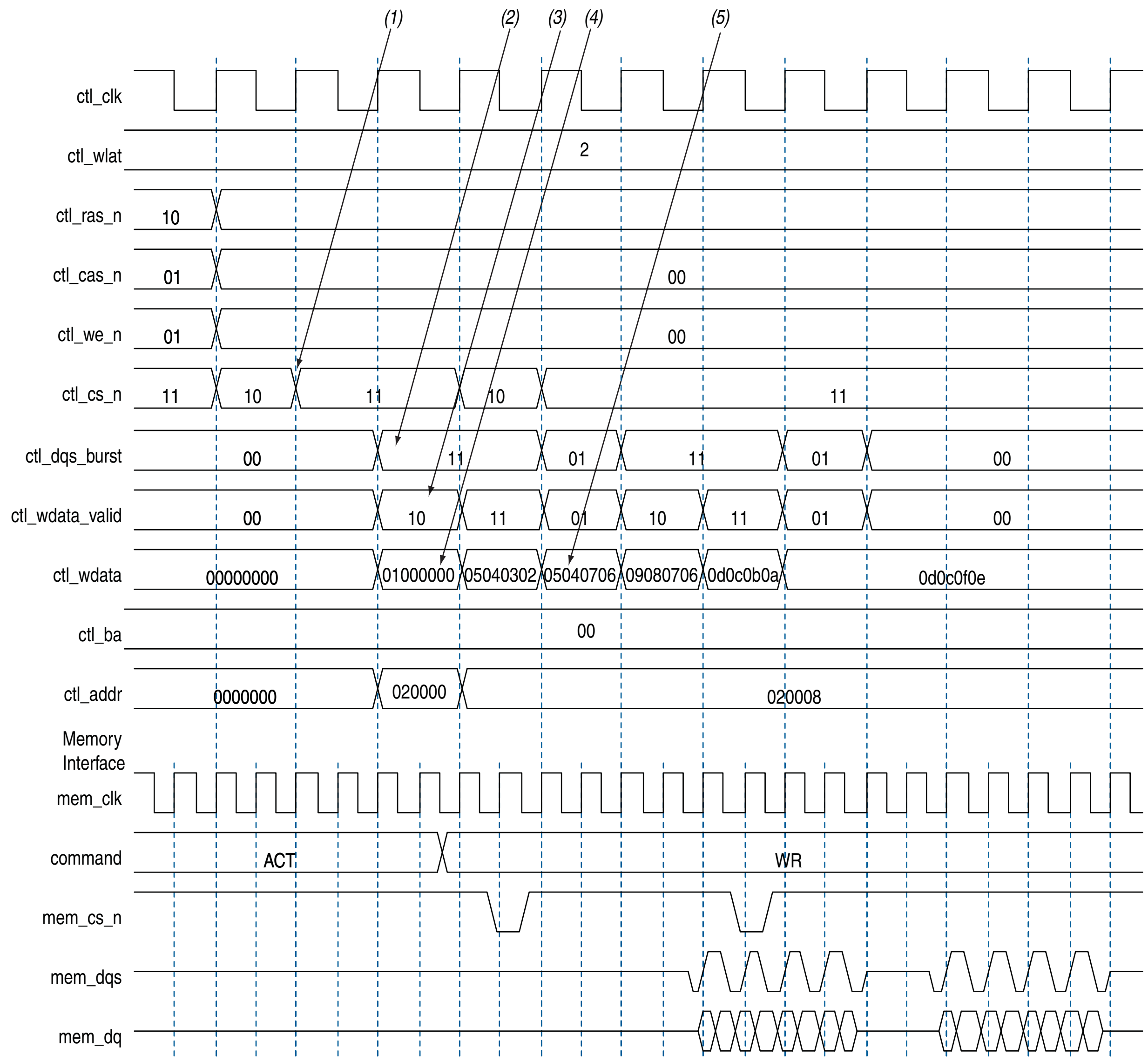
(2) AFI requires that `ctl_doing_rd` is driven for the duration of the read. In this example, it is driven to `11` for two half-rate `ctl_clks`, which equates to driving to `1`, for the four memory clock cycles of this four-beat burst.

(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 5–10 and Figure 5–11 show spaced word-unaligned writes and reads.

Figure 5–10. Word-Unaligned Writes



Notes to Figure 5–10:

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

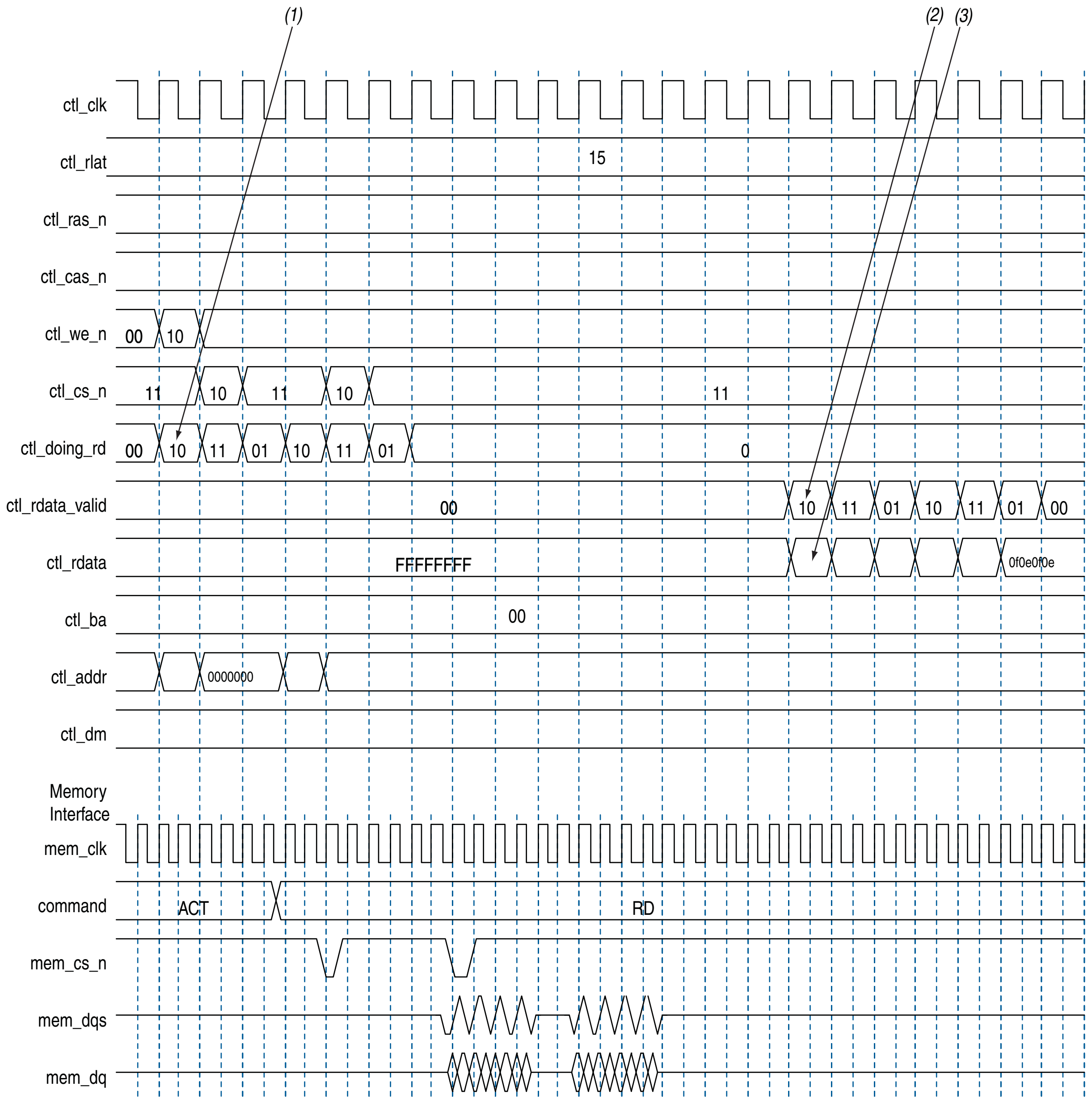
(2) As with word- aligned writes, `ctl_dqs_burst` is asserted one memory clock cycle before `ctl_wdata_valid`. You can see `ctl_dqs_burst` is 11 in the same cycle where `ctl_wdata_valid` is 10. The LSB of these two becomes the first value the signal takes in the `mem_clk` domain. You can see that `ctl_dqs_burst`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

(3) The latency between ctl_cs_n being asserted and `ctl_wdata_valid` going high is effectively `ctl_wlat` (in this example, two) controller clock (ctl_clk) cycles. This can be thought of in terms of relative memory clock (`mem_clk`) cycles, in which case the latency is four `mem_clk` cycles.

(4) Only the upper half is valid (as the `ctl_wdata_valid` signal demonstrates, there is one `ctl_wdata_valid` bit to two 8-bit words). The write data bits go out on the bus in order, least significant byte first. So for a continuous burst of write data on the DQ pins, the most significant half of write data is used, which goes out on the bus last and is therefore contiguous with the following data. The converse is true for the end of the burst. Write data is spread across three controller clock (`ctl_clk`) cycles, but still only four memory clock (`mem_clk`) cycles. However, in relative memory clock cycles the latency is equivalent in the word-aligned and word-unaligned cases.

(5) The `0504` here is residual from the previous clock cycle. In the same way that only the upper half of the write data is used for the first beat of the write, only the lower half of the write data is used in the last beat of the write. These upper bits can be driven to any value in this alignment.

Figure 5–11. Word-Unaligned Reads



Notes to Figure 5–11:

(1) Similar to word-aligned reads, `ctl_doing_rd` is asserted one memory clock cycle before chip select (`ctl_cs_n`) is asserted, which for a word-unaligned read is in the previous controller clock (`ctl_clk`) cycle. In this example the `ctl_doing_rd` signal is now spread over three controller clock (`ctl_clk`) cycles, the high bits in the sequence '10','11','01','10','11','01' providing the required four memory clock cycles of assertion for `ctl_doing_rd` for the two 4-beat reads in the full-rate memory clock domain, '011110','011110'.

(2) The return pattern of `ctl_rdata_valid` is a delayed version of `ctl_doing_rd`. Advertised read latency (`ctl_rlat`) is the number of controller clock (`ctl_clk`) cycles delay inserted between `ctl_doing_rd` and `ctl_rdata_valid`.

(3) The read data (`ctl_rdata`) is spread over three controller clock cycles and in the pointed to vector only the upper half of the `ctl_rdata` bit vector is valid (denoted by `ctl_rdata_valid`).

Using a Custom Controller

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

Preliminary Steps

Perform the following steps to generate the ALTMEMPHY megafunction:

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

Design Considerations

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

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

Clocks and Resets

The ALTMEMPHY megafunction automatically generates a PLL instance, but you must still provide the reference clock input (`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, is provided and all inputs and outputs of the ALTMEMPHY megafunction are synchronous to this clock. For AFIs, this signal is called `ctl_clk`.

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

Calibration Process Requirements

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

Other Local Interface Requirements

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

Address and Command Interfacing

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

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

Handshake Mechanism Between Read Commands and Read Data

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

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

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

The ALTMEMPHY provides a signal, `ctl_rdata_valid`, to indicate that the data on read data bus is valid. The width of this signal varies between half-rate and full-rate designs to support the option to indicate that the read data is not word aligned. Figure 5–12 and Figure 5–13 show these relationships.

Figure 5–12. Address and Command and Read-Path Timing—Full-Rate Design

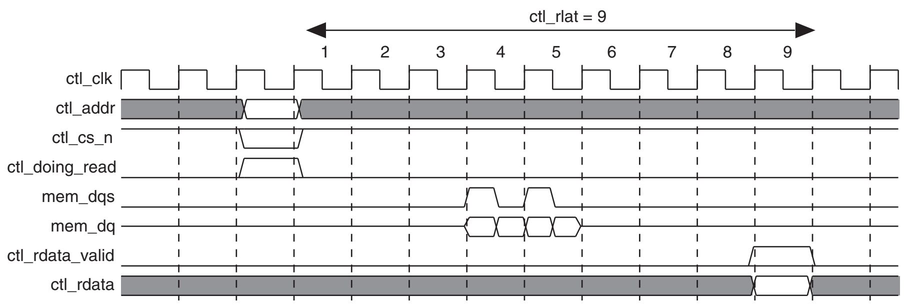
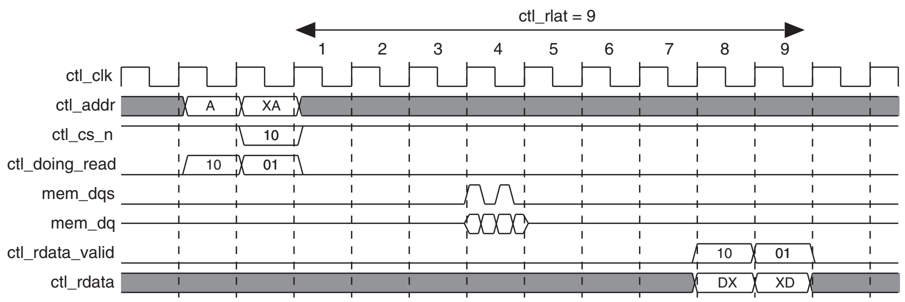


Figure 5–13. Second Read Alignment—Half-Rate Design



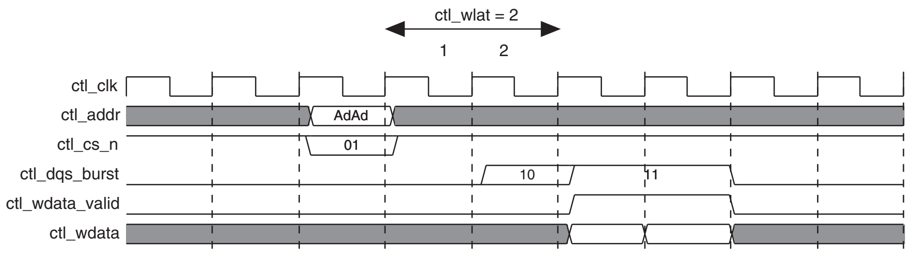
Handshake Mechanism Between Write Commands and Write Data

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

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

The ctl_wlat signal is only valid when the calibration has been successfully completed by the ALTMEMPHY sequencer and does not change at any point during normal user mode operation. Figure 5–14 shows the operation of ctl_wlat port.

Figure 5–14. Timing for ctl_dqs_burst, ctl_wdata_valid, Address, and Command—Half-Rate Design



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

Partial Writes

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

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

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

6. Functional Description— High-Performance Controller II

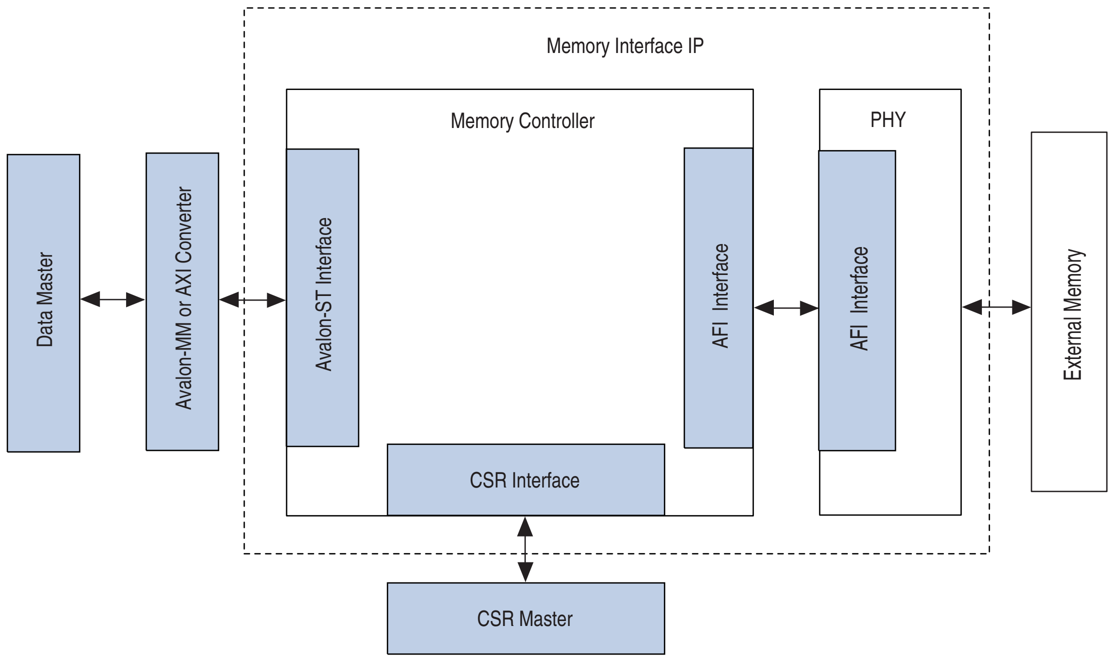
This chapter describes the High Performance Controller II (HPC II) with advanced features introduced in version 11.0 for designs generated in the Quatus II software version 11.0. Designs created in earlier versions and regenerated in version 11.0 do not inherit the new advanced features; for information on HPC II without the version 11.0 advanced features, refer to the External Memory Interface Handbook for Quartus II version 10.1, available in the *External Memory Interfaces* section of the Altera Literature website.

The memory controller provides high memory bandwidth, high clock rate performance, and run-time programmability. The controller can reorder data to reduce row conflicts and bus turn-around time by grouping reads and writes together, allowing for efficient traffic patterns and reduced latency.

Memory Controller Architecture

Figure 6–1 shows a high-level block diagram of the overall memory interface architecture.

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

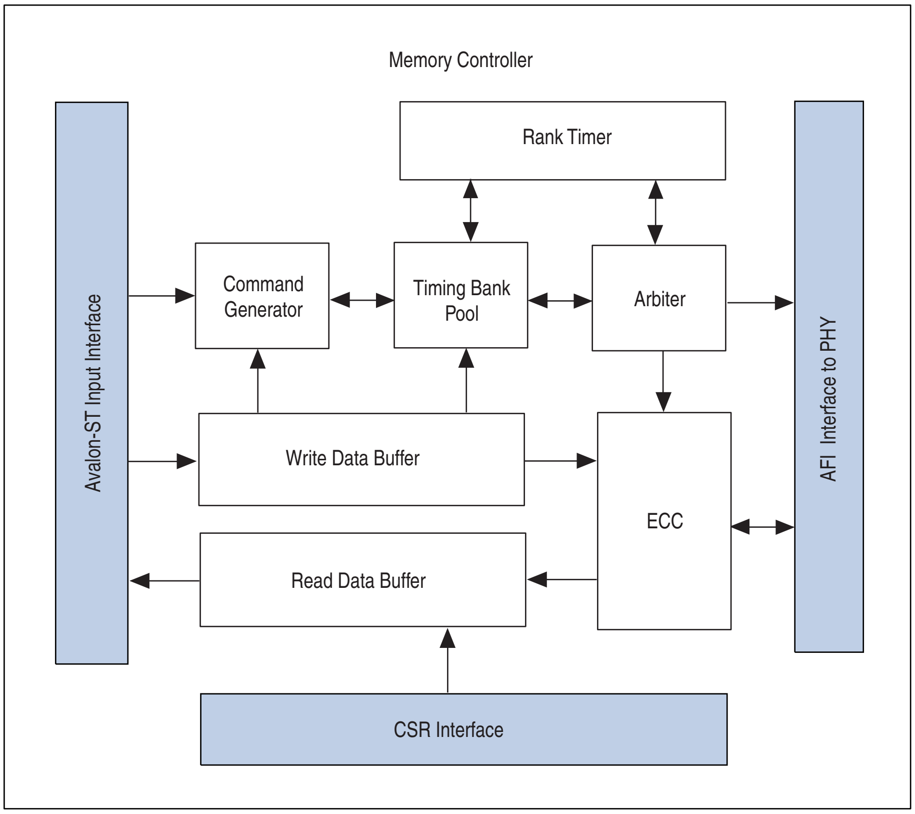


The memory interface consists of the memory controller logic block, the physical (PHY) logic layer, and their associated interfaces.

The memory controller logic block uses an Avalon Streaming (Avalon-ST) interface as its native interface, and communicates with the PHY layer by the Altera PHY Interface (AFI). The controller supports an Avalon Memory Mapped (Avalon-MM) bus protocol.

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

Figure 6–2. Memory Controller Architecture Block Diagram



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

Avalon-ST Input Interface

The Avalon-ST interface serves as the entry point to the memory controller, and provides communication with the requesting data masters.

For information about the Avalon interface, refer to *Avalon Interface Specifications*.

Command Generator

The command generator accepts commands from the front-end Avalon-ST interface and from local ECC internal logic, and provides those commands to the timing bank pool.

Timing Bank Pool

The timing bank pool is a parallel queue that works with the arbiter to enable data reordering. The timing bank pool tracks incoming requests, ensures that all timing requirements are met and, upon receiving write-data-ready notification from the write data buffer, passes the requests to the arbiter in an ordered and efficient manner.

Arbiter

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

Arbitration Rules

The arbiter follows the following arbitration rules:

- If only one master is issuing a request, grant that request immediately.
- If there are outstanding requests from two or more masters, the arbiter applies the following tests, in order:
    a. Is there a read request? If so, the arbiter grants the read request ahead of any write requests.
    b. If neither of the above conditions apply, the arbiter grants the oldest request first.

Rank Timer

The rank timer maintains rank-specific timing information, and performs the following functions:

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

Read Data Buffer

The read data buffer receives data from the PHY and passes that data through the input interface to the master.

Write Data Buffer

The write data buffer receives write data from the input interface and passes that data to the PHY, upon approval of the write request.

ECC Block

The error-correcting code (ECC) block comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC block can remedy errors resulting from noise or other impairments during data transmission.

AFI Interface

The AFI interface provides communication with the memory device, through the physical layer logic (PHY).

For more information about AFI signals, refer to "AFI Signals" on page 5–11.

CSR Interface

The CSR interface provides communication with your system's internal control status registers.

Controller Features Descriptions

The following sections describe main features of the memory controller.

Data Reordering

The controller implements data reordering to maximize efficiency for read and write commands. The controller can reorder read and write commands as necessary to mitigate bus turn-around time and reduce conflict between rows.

Inter-bank data reordering reorders commands going to different bank addresses. Commands going to the same bank address are not reordered. This reordering method implements simple hazard detection on the bank address level.

The controller implements logic to limit the length of time that a command can go unserved. This logic is known as starvation control. In starvation control, a counter is incremented for every command served. You can set a starvation limit, to ensure that a waiting command is served immediately, when the starvation counter reaches the specified limit.

Pre-emptive Bank Management

Data reordering allows the controller to issue bank-management commands pre-emptively, based on the patterns of incoming commands; consequently, the desired page in memory can be already open when a command reaches the AFI interface.

Quasi-1T and Quasi-2T

One controller clock cycle equals two memory clock cycles in a half-rate interface, and to four memory clock cycles in a quarter-rate interface. To fully utilize the command bandwidth, the controlller can operate in Quasi-1T half-rate and Quasi-2T quarter-rate modes.

In Quasi-1T and Quasi-2T modes, the controller issues two commands on every controller clock cycle. The controller is constrained to issue a row command on the first clock phase and a column command on the second clock phase, or vice versa. Row commands include activate and precharge commands; column commands include read and write commands.

User Autoprecharge Commands

The autoprecharge read and autoprecharge write commands allow you to indicate to the memory device that this read or write command is the last access to the currently open row. The memory device automatically closes or autoprecharges the page it is currently accessing so that the next access to the same bank is quicker.

This command is useful for applications that require fast random accesses.

Since the controller can reorder transactions for best efficiency, when you assert the `local_autopch_req` signal, the controller evaluates the current command and buffered commands to determine the best autoprecharge operation.

Address and Command Decoding Logic

When the main state machine issues a command to the memory, it asserts a set of internal signals. The address and command decoding logic turns these signals into AFI-specific commands and address. This block generates the following signals:

- Clock enable and reset signals: `afi_cke`, `afi_rst_n`
- Command and address signals: `afi_cs_n`, `afi_ba`, `afi_addr`, `afi_ras_n`, `afi_cas_n`, `afi_we_n`

Low-Power Logic

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

User-Controlled Self-Refresh

When you assert the `local_self_rfsh_req signal`, the controller completes any currently executing reads and writes, and then interrupts the command queue and immediately places the memory into self-refresh mode. When the controller places the memory into self-refresh mode, it responds by asserting an acknowledge signal, `local_self_rfsh_ack`. You can leave the memory in self-refresh mode for as long as you choose.

To bring the memory out of self-refresh mode, you must deassert the request signal, and the controller responds by deasserting the acknowledge signal when the memory is no longer in self-refresh mode.

☞ If a user-controlled refresh request and a system-generated refresh request occur at the same time, the user-controlled refresh takes priority; the system-generated refresh is processed only after the user-controlled refresh request is completed.

Automatic Power-Down with Programmable Time-Out

The controller automatically places the memory in power-down mode to save power if the requested number of idle controller clock cycles is observed in the controller. The **Auto Power Down Cycles** parameter on the **Controller Settings** tab allows you to specify a range between 1 to 65,535 idle controller clock cycles. The counter for the programmable time-out starts when there are no user read or write requests in the command queue. Once the controller places the memory in power-down mode, it responds by asserting the acknowledge signal, `local_power_down_ack`.

ODT Generation Logic

The on-die termination (ODT) generation logic generates the necessary ODT signals for the controller, based on the scheme that Altera recmmends.

DDR3 SDRAM

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

☞ There is no ODT for reads.

Table 6–1. ODT—DDR3 SDRAM Single Slot Single Chip-select Per DIMM (Write)

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

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

☞ There is no ODT for reads.

Table 6–2. ODT—DDR3 SDRAM Single Slot Dual Chip-select Per DIMM (Write)

Write On	ODT Enabled
`mem_cs[0]`	`mem_odt[0]`
`mem_cs[1]`	`mem_odt[1]`

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

Table 6–3. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Write)

Write On	ODT Enabled
`mem_cs[0]`	`mem_odt[0]` and `mem_odt[1]`
`mem_cs[1]`	`mem_odt[0]` and `mem_odt[1]`

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

Table 6–4. ODT—DDR3 SDRAM Dual Slot Single Chip-select Per DIMM (Read)

Read On	ODT Enabled
`mem_cs[0]`	`mem_odt[1]`
`mem_cs[1]`	`mem_odt[0]`

Table 6–5 shows which ODT signal is enabled for dual-slot dual chip-select per DIMM.

Table 6–5. ODT—DDR3 SDRAM Dual Slot Dual Chip-select Per DIMM (Write)

Write On	ODT Enabled
`mem_cs[0]`	`mem_odt[0]` and `mem_odt[2]`
`mem_cs[1]`	`mem_odt[1]` and `mem_odt[3]`
`mem_cs[2]`	`mem_odt[0]` and `mem_odt[2]`
`mem_cs[3]`	`mem_odt[1]` and `mem_odt[3]`

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

Table 6–6. ODT—DDR3 SDRAM Dual Slot Dual Rank Per DIMM (Read)

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

ECC

The ECC logic comprises an encoder and a decoder-corrector, which can detect and correct single-bit errors, and detect double-bit errors. The ECC logic is available in two widths: 64/72 bit and 32/40 bit. The ECC logic has the following features:

- Has Hamming code ECC logic that encodes every 64, 32, 16, or 8 bits of data into 72, 40, 24, or 16 bits of codeword.
- Has a latency increase of one clock for both writes and reads.
- For a 128-bit interface, ECC is generated as one 64-bit data path with 8-bits of ECC path, plus a second 64-bit data path with 8-bits of ECC path.
- Detects and corrects all single-bit errors.
- Detects all double-bit errors.
- Counts the number of single-bit and double-bit errors.
- Accepts partial writes, which trigger a read-modify-write cycle, for memory devices with DM pins.
- Can inject single-bit and double-bit errors to trigger ECC correction for testing and debugging purposes.
- Generates an interrupt signal when an error occurs.

☞ When using ECC, you must initialize memory before writing to it.

When a single-bit or double-bit error occurs, the ECC logic triggers the `ecc_interrupt` signal to inform you that an ECC error has occurred. When a single-bit error occurs, the ECC logic reads the error address, and writes back the corrected data. When a double-bit error occurs, the ECC logic does not do any error correction but it asserts the `avl_rdata_error` signal to indicate that the data is incorrect. The `avl_rdata_error` signal follows the same timing as the `avl_rdata_valid` signal.

Enabling autocorrection allows the ECC logic to delay all controller pending activities until the correction completes. You can disable autocorrection and schedule the correction manually when the controller is idle to ensure better system efficiency. To manually correct ECC errors, follow these steps:

1. When an interrupt occurs, read out the `SBE_ERROR` register. When a single-bit error occurs, the `SBE_ERROR` register is equal to one.
2. Read out the `ERR_ADDR` register.
3. Correct the single-bit error by issuing a dummy write to the memory address stored in the `ERR_ADDR` register. A dummy write is a write request with the `local_be` signal zero, that triggers a partial write which is effectively a read-modify-write event. The partial write corrects the data at that address and writes it back.

Partial Writes

The ECC logic supports partial writes. Along with the address, data, and burst signals, the Avalon-MM interface also supports a signal vector, `local_be`, that is responsible for byte-enable. Every bit of this signal vector represents a byte on the data-bus. Thus, a logic low on any of these bits instructs the controller not to write to that particular byte, resulting in a partial write. The ECC code is calculated on all bytes of the data-bus. If any bytes are changed, the IP core must recalculate the ECC code and write the new code back to the memory.

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

1. The ECC logic sends a read command to the partial write address.
2. Upon receiving a return data from the memory for the particular address, the ECC logic decodes the data, checks for errors, and then merges the corrected or correct dataword with the incoming information.
3. The ECC logic issues a write to write back the updated data and the new ECC code.

The following corner cases can occur:

- A single-bit error during the read phase of the read-modify-write process. In this case, the IP core corrects the single-bit error first, increments the single-bit error counter and then performs a partial write to this corrected decoded data word.
- A double-bit error during the read phase of the read-modify-write process. In this case, the IP core increments the double-bit error counter and issues an interrupt. The IP core writes a new write word to the location of the error. The ECC status register keeps track of the error information.

Figure 6–3 and Figure 6–4 show partial write operations for the controller, for full and half rate configurations, respectively.

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

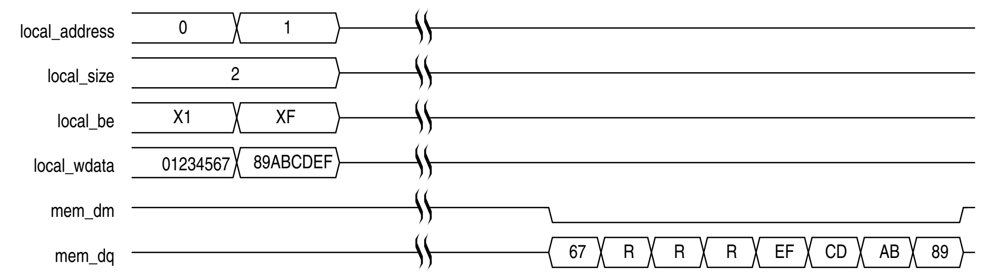
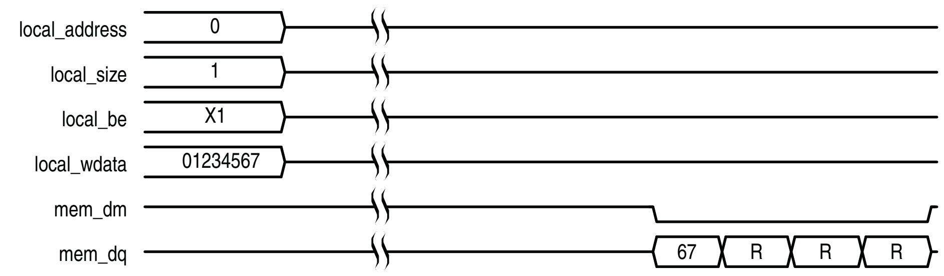


Figure 6–4. Partial Write for the Controller—Half Rate

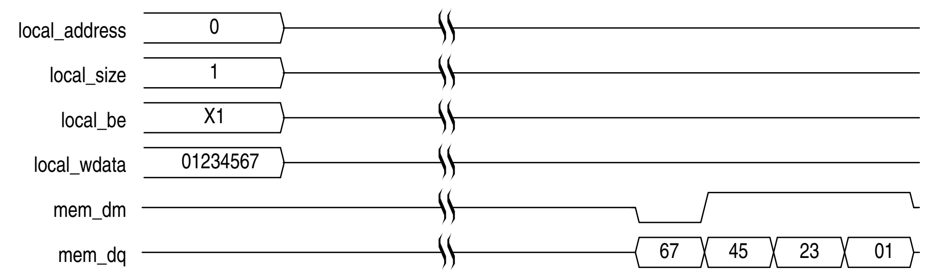


Partial Bursts

Devices that do not have the DM pins do not support partial bursts. You must write a minimum (or multiples) of memory-burst-length-equivalent words to the memory at the same time..

Figure 6–5 shows a partial burst operation for the controller.

Figure 6–5. Partial Burst for Controller



External Interfaces

This section discusses the interfaces between the controller and other external memory interface components.

Clock and Reset Interface

The clock and reset interface is part of the AFI interface.

The controller can have up to two clock domains, which are synchronous to each other. The controller operates with a single clock domain when there is no integrated half-rate bridge, and with two-clock domains when there is an integrated half-rate bridge. The clocks are provided by ALTMEMPHY.

The main controller clock is `afi_clk`, and the optional half-rate controller clock is `afi_half_clk`. The main and half-rate clocks must be synchronous and have a 2:1 frequency ratio. The optional quarter-rate controller clock is `afi_quarter_clk`, which must also be synchronous and have a 4:1 frequency ratio.

Avalon-ST Data Slave Interface

The Avalon-ST data slave interface consists of the following Avalon-ST channels, which together form a single data slave:

- The command channel, which serves as command and address for both read and write operations.
- The write data channel, which carries write data.
- The read data channel, which carries read data.
- The write response channel, which is an AXI protocol.

For information about the Avalon interface, refer to *Avalon Interface Specifications*.

Controller-PHY Interface

The interface between the controller and the PHY is part of the AFI interface.

The controller assumes that the PHY performs all necessary calibration processes without any interaction with the controller.

For more information about AFI signals, refer to "AFI Signals" on page 5–11.

Memory Side-Band Signals

This section describes supported side-band signals.

Self-Refresh (Low Power) Interface

The optional low power self-refresh interface consists of a request signal and an acknowledgement signal, which you can use to instruct the controller to place the memory device into self-refresh mode. This interface is clocked by `afi_clk`.

When you assert the request signal, the controller places the memory device into self-refresh mode and asserts the acknowledge signal. To bring the memory device out of self-refresh mode, you deassert the request signal; the controller then deasserts the acknowledge signal when the memory device is no longer in self-refresh mode.

User-Controller Refresh Interface

The optional user-controlled refresh interface consists of a request signal and an acknowledgement signal, which allow you to determine when the controller issues refresh signals to the memory device. This interface gives you increased control over worst-case read latency, and enables you to issue refresh bursts during idle periods. This interface is clocked by `afi_clk`.

When you assert a refresh request signal to instruct the controller to perform a refresh operation, that request takes priority over any outstanding read or write requests that might be in the command queue. Once the controller successfully issues a refresh command to the memory device, the controller asserts the refresh acknowledge signal for one clock cycle.

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

Configuration and Status Register (CSR) Interface

The controller has a configuration and status register (CSR) interface that allows you to configure timing parameters, address widths, and the behavior of the controller. The CSR interface is a 32-bit Avalon-MM slave of fixed address width; if you do not need this feature, you can disable it to save area.

This interface is clocked by `csr_clk`, which is the same as `afi_clk`, and is always synchronous relative to the main data slave interface.

Table 6–7 summarizes the controller's external interfaces.

Table 6–7. Summary of Controller External Interfaces (Part 1 of 2)

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

Table 6–7. Summary of Controller External Interfaces (Part 2 of 2)

Interface Name	Display Name	Type	Description
Configuration and Status Register (CSR) Interface			
CSR	Configuration and Status Register Interface	Avalon-MM *(2)*	Enables on-the-fly configuration of memory timing parameters, address widths, and controller behaviour.

Notes:

(1) For information about AFI signals, refer to AFI Signals.

(2) For information about Avalon signals, refer to *Avalon Interface Specifications*.

Top-Level Signals Description

Table 6–8 shows the clock and reset signals.

☞ The suffix _n denotes active low signals.

Table 6–8. Clock and Reset Signals (Part 1 of 2)

Name	Direction	Description
`global_reset_n`	Input	The asynchronous reset input to the controller. The IP core derives all other reset signals from resynchronized versions of this signal. This signal holds the PHY, including the PLL, in reset while low.
`pll_ref_clk`	Input	The reference clock input to PLL.
`phy_clk`	Output	The system clock that the PHY provides to the user. All user inputs to and outputs from the controller must be synchronous to this clock.
`reset_phy_clk_n`	Output	The reset signal that the PHY provides to the user. The IP core asserts `reset_phy_clk_n` asynchronously and deasserts synchronously to `phy_clk` clock domain.
`aux_full_rate_clk`	Output	An alternative clock that the PHY provides to the user. This clock always runs at the same frequency as the external memory interface. In half-rate designs, this clock is twice the frequency of the `phy_clk` and you can use it whenever you require a 2x clock. In full-rate designs, the same PLL output as the `phy_clk` signal drives this clock.
`aux_half_rate_clk`	Output	An alternative clock that the PHY provides to the user. This clock always runs at half the frequency as the external memory interface. In full-rate designs, this clock is half the frequency of the `phy_clk` and you can use it, for example to clock the user side of a half-rate bridge. In half-rate designs, or if the **Enable Half Rate Bridge** option is turned on. The same PLL output that drives the `phy_clk` signal drives this clock.
`dll_reference_clk`	Output	Reference clock to feed to an externally instantiated DLL.
`reset_request_n`	Output	Reset request output that indicates when the PLL outputs are not locked. Use this signal as a reset request input to any system-level reset controller you may have. This signal is always low when the PLL is trying to lock, and so any reset logic using Altera advises you detect a reset request on a falling edge rather than by level detection.
`soft_reset_n`	Input	Edge detect reset input for SOPC Builder or for control by other system reset logic. Assert to cause a complete reset to the PHY, but not to the PLL that the PHY uses.

Table 6–8. Clock and Reset Signals (Part 2 of 2)

Name	Direction	Description
`seriesterminationcontrol`	Input (for OCT slave)	Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode.
	Output(for OCT master)	Unconnected PHY signal, available for sharing with another PHY.
`parallelterminationcontrol`	Input (for OCT slave)	Required signal for PHY to provide series termination calibration value. Must be connected to a user-instantiated OCT control block (alt_oct) or another UniPHY instance that is set to OCT master mode.
	Output (for OCT master)	Unconnected PHY signal, available for sharing with another PHY.
`oct_rdn`	Input (for OCT master)	Must connect to calibration resistor tied to GND on the appropriate RDN pin on the device. (Refer to appropriate device handbook.)
`oct_rup`	Input (for OCT master)	Must connect to calibration resistor tied to V_{ccio} on the appropriate RUP pin on the device. (See appropriate device handbook.)
`dqs_delay_ctrl_import`	Input	Allows the use of DLL in another PHY instance in this PHY instance. Connect the `export` port on the PHY instance with a DLL to the `import` port on the other PHY instance.

Table 6–9 on page 6–14 shows the controller local interface signals.

Table 6–9. 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 IP core maps local address to the bank interleaving scheme. You can change the ordering via the **Local-to-Memory Address Mapping** option in the **Controller Settings** page.
		The IP core sizes the width of this bus according to the following equations:
		■ Full rate controllers
		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, 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 IP core ignores the least significant bit (LSB) of the column address (multiples of four) on the memory side, because the local data width is twice that of the memory data bus width.
		■ Half rate controllers
		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, 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]`
		The IP core ignores two LSBs of the column address on the memory side, because the local data width is four times that of the memory data bus width.

`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 >`

Table 6–9. Local Interface Signals (Part 2 of 4)

Signal Name	Direction	Description
`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 deasserts `local_ready`. The IP core samples this signal at the rising edge of `phy_clk` when `local_write_req` is asserted. After the slave deasserts the `local_ready` signal, the master keeps all the write request signals asserted until `local_ready` signal becomes high again. For read transactions, assert this signal for one clock cycle when read request is asserted and `local_address` from which the data should be read is given to the memory. After the slave deasserts `local_ready` (`waitrequest_n` in Avalon interface), the master keeps all the read request signals asserted until `local_ready` becomes high again.
`local_read_req`	Input	Read request signal. You cannot assert read request and write request signals at the same time. The controller must deassert `reset_phy_clk_n` before you can assert `local_autopch_req`.
`local_refresh_req`	Input	User-controlled refresh request. If **Enable User Auto-Refresh Controls** option is turned on, `local_refresh_req` becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including grouping together multiple refresh commands. Refresh requests take priority over read and write requests, unless the IP core is already processing the requests.
`local_refresh_chip`	Input	Controls which chip to issue the user refresh to. The IP core uses this active high signal with `local_refresh_req`. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip. For example: If `local_refresh_chip` signal is assigned with a value of `4’b0101`, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.
`local_size[]`	Input	Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The IP core supports Avalon burst lengths from 1 to 64. The IP core derives the width of this signal based on the burst count that you specify in the **Local Maximum Burst Count** option. With the derived width, you specify a value ranging from 1 to the local maximum burst count specified.
`local_wdata[]`	Input	Write data bus. The width of `local_wdata` is twice that of the memory data bus for a full-rate controller; four times the memory data bus for a half-rate controller.
`local_write_req`	Input	Write request signal. You cannot assert read request and write request signal at the same time. The controller must deassert `reset_phy_clk_n` before you can assert `local_write_req`.

Table 6–9. Local Interface Signals (Part 3 of 4)

Signal Name	Direction	Description
`local_autopch_req`	Input	User control of autoprecharge. If you turn on **Enable Auto-Precharge Control**, the `local_autopch_req` signal becomes available and you can request the controller to issue an autoprecharge write or autoprecharge read command. These commands cause the memory to issue a precharge command to the current bank at the appropriate time without an explicit precharge command from the controller. This feature is particularly useful if you know the current read or write is the last one you intend to issue to the currently open row. The next time you need to use that bank, the access could be quicker as the controller does not need to precharge the bank before activating the row you wish to access. Upon receipt of the `local_autopch_req` signal, the controller evaluates the pending commands in the command buffer and determines the most efficient autoprecharge operation to perform, reordering commands if necessary.
`local_self_rfsh_chip`	Input	Controls which chip to issue the user refresh to. The IP core uses this active high signal with `local_self_rfsh_req`. This signal is as wide as the memory chip select. This signal asserts a high value to each bit that represents the refresh for the corresponding memory chip. For example: If `local_self_rfsh_chip` signal is assigned with a value of `4’b0101`, the controller refreshes the memory chips 0 and 2, and memory chips 1 and 3 are not refreshed.
`local_self_rfsh_req`	Input	User control of the self-refresh feature. If you turn on **Enable Self-Refresh Controls**, you can request that the controller place the memory devices into a self-refresh state by asserting this signal. The controller places the memory in the self-refresh state as soon as it can without violating the relevant timing parameters and responds by asserting `local_self_rfsh_ack`. You can hold the memory in the self-refresh state by keeping this signal asserted. You can release the memory from the self-refresh state at any time by deasserting `local_self_rfsh_req` and the controller responds by deasserting `local__self_rfsh_ack` when it has successfully brought the memory out of the self-refresh state.
`local_init_done`	Output	When the memory initialization, training, and calibration are complete, the PHY sequencer asserts `ctrl_usr_mode_rdy` to the memory controller, which then asserts this signal to indicate that the memory interface is ready for use. The controller still accepts read and write requests before `local_init_done` is asserted, however it does not issue them to the memory until it is safe to do so. 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 you turn on **Enable Error Detection and Correction Logic**. The controller asserts this signal 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 6–9. Local Interface Signals (Part 4 of 4)

Signal Name	Direction	Description
`local_ready`	Output	The `local_ready` signal indicates that the controller is ready to accept request signals. If controller asserts the `local_ready` signal in the clock cycle that it asserts a read or write request, the controller accepts that request. The controller deasserts the `local_ready` signal to indicate that it cannot accept any more requests. The controller can buffer eight read or write requests, after which the `local_ready` signal goes low.
`local_refresh_ack`	Output	Refresh request acknowledge, which the controller asserts for one clock cycle every time it issues a refresh. Even if you do not turn on **Enable User Auto-Refresh Controls**, `local_refresh_ack` still indicates to the local interface that the controller has just issued a refresh command.
`local_self_rfsh_ack`	Output	Self refresh request acknowledge signal. The controller asserts and deasserts this signal in response to the `local_self_rfsh_req` signal.
`local_power_down_ack`	Output	Auto power-down acknowledge signal. The controller asserts this signal for one clock cycle every time auto power-down is issued.
`ecc_interrupt`	Output	Interrupt signal from the ECC logic. The controller asserts this signal when the ECC feature is turned on, and the controller detects an error.

Table 6–10 shows the controller interface signals.

Table 6–10. Interface Signals (Part 1 of 2)

Signal Name	Direction	Description
`mem_dq[]`	Bidirectional	Memory data bus. This bus is half the width of the local read and write data busses.
`mem_dqs[]`	Bidirectional	Memory data strobe signal, which writes data into the DDR3 SDRAM and captures read data into the Altera device.
`mem_dqs_n[]`	Bidirectional	Inverted memory data strobe signal, which with the `mem_dqs` signal improves signal integrity.
`mem_clk`	Bidirectional	Clock for the memory device.
`mem_clk_n`	Bidirectional	Inverted clock for the memory device.
`mem_addr[]`	Output	Memory address bus.
`mem_ac_parity` *(1)*	Output	Address or command parity signal generated by the PHY and sent to the DIMM. DDR3 SDRAM only.
`mem_ba[]`	Output	Memory bank address bus.
`mem_cas_n`	Output	Memory column address strobe signal.
`mem_cke[]`	Output	Memory clock enable signals.
`mem_cs_n[]`	Output	Memory chip select signals.
`mem_dm[]`	Output	Memory data mask signal, which masks individual bytes during writes.
`mem_odt`	Output	Memory on-die termination control signal.
`mem_ras_n`	Output	Memory row address strobe signal.
`mem_we_n`	Output	Memory write enable signal.
`parity_error_n` *(1)*	Output	Active-low signal that is asserted when a parity error occurs and stays asserted until the PHY is reset. DDR3 SDRAM only

Table 6–10. Interface Signals (Part 2 of 2)

Signal Name	Direction	Description
`mem_err_out_n` *(1)*	Input	Signal sent from the DIMM to the PHY to indicate that a parity error has occured for a particular cycle. DDR3 SDRAM only.

Notes to Table 6–10:

(1) This signal is for registered DIMMs only.

Table 6–11 shows the CSR interface signals.

Table 6–11. 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.

Sequence of Operations

This section explains how the various blocks pass information in common situations.

Write Command

When a requesting master issues a write command together with write data, the following events occur:

- The input interface accepts the write command and the write data.
- The input interface passes the write command to the command generator and the write data to the write data buffer.
- The command generator processes the command and sends it to the timing bank pool.
- Once all timing requirements are met and a write-data-ready notification has been received from the write data buffer, the timing bank pool sends the command to the arbiter.
- When rank timing requirements are met, the arbiter grants the command request from the timing bank pool and passes the write command to the AFI interface.

- The AFI interface receives the write command from the arbiter and requests the corresponding write data from the write data buffer.
- The PHY receives the write command and the write data, through the AFI interface.

Read Command

When a requesting master issues a read command, the following events occur:

- The input interface accepts the read command.
- The input interface passes the read command to the command generator.
- The command generator processes the command and sends it to the timing bank pool.
- Once all timing requirements are met, the timing bank pool sends the command to the arbiter.
- When rank timing requirements are met, the arbiter grants the command request from the timing bank pool and passes the read command to the AFI interface.
- The AFI interface receives the read command from the arbiter and passes the command to the PHY.
- The PHY receives the read command through the AFI interface, and returns read data through the AFI interface.
- The AFI interface passes the read data from the PHY to the read data buffer.
- The read data buffer sends the read data to the master through the input interface.

Read-Modify-Write Command

A read-modify-write command can occur when enabling ECC for partial write, and for ECC correction commands. When a read-modify-write command is issued, the following events occur:

- The command generator issues a read command to the timing bank pool.
- The timing bank pool and arbiter passes the read command to the PHY through the AFI interface.
- The PHY receives the read command, reads data from the memory device, and returns the read data through the AFI interface.
- The read data received from the PHY passes to the ECC block.
- The read data is processed by the write data buffer.
- When the write data buffer issues a read-modify-write data ready notification to the command generator, the command generator issues a write command to the timing bank pool; the arbiter can then issue the write request to the PHY through the AFI interface.
- When the PHY receives the write request, it passes the data to the memory device.

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 DDR3 SDRAM HPC II. The example top-level file consists of the DDR3 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. The example top-level file is a working system that you can compile and use for both static timing checks and board tests.

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

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

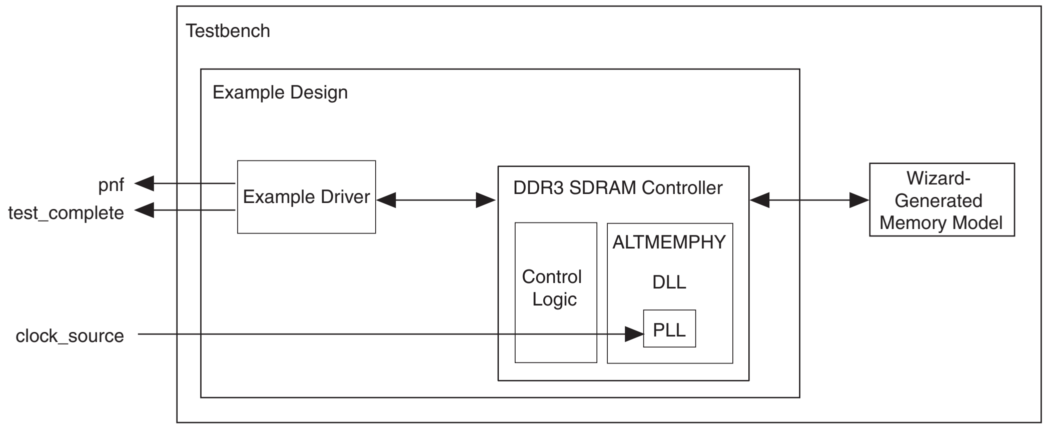


Table 6–12 describes the files that are associated with the example top-level file and the testbench.

Table 6–12. Example Top-Level File and Testbench Files

Filename	Description
*<variation name>*_**example_top_tb.v** or **.vhd**	Testbench for the example top-level file.
*<variation name>*_**example_top.v** or **.vhd**	Example top-level file.
*<variation name>*_**mem_model.v** or **.vhd**	Associative-array memory model.
*<variation name>*_**full_mem_model.v** or **.vhd**	Full-array memory model.
*<variation name>*_**example_driver.v** or **.vhd**	Example driver.
<variation name> **.v** or **.vhd**	Top-level description of the custom MegaCore function.
<variation name>.**qip**	Contains Quartus II project information for your MegaCore function variations.

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

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

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

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

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

Example Driver

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

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

- Sequential addressing writes and reads

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

- Incomplete write operation

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

- Byte enable/data mask pin operation

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

- Address pin operation

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

- Low-power mode operation

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

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

Table 6–13 shows the bit mapping for each test status.

Table 6–13. Test Status[] Bit Mapping

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

Table 6–14 shows the ALTMEMPHY Debug interface signals, which are located in <*variation_name*>**_phy.v/vhd** file.

Table 6–14. ALTMEMPHY Debug Interface Signals

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

Register Maps

Table 6–15 shows the overall register mapping for the DDR3 SDRAM Controller with ALTMEMPHY.

Table 6–15. Register Map

Address	Description
ALTMEMPHY Register Map	
0x005	Mode register 0-1.
0x006	Mode register 2-3.
Controller Register Map	
0x100	ALTMEMPHY status and control register.
0x110	Controller status and configuration register.
0x120	Memory address size register 0.
0x121	Memory address size register 1.
0x122	Memory address size register 2.
0x123	Memory timing parameters register 0.
0x124	Memory timing parameters register 1.
0x125	Memory timing parameters register 2.
0x126	Memory timing parameters register 3.
0x130	ECC control register.
0x131	ECC status register.
0x132	ECC error address register.

ALTMEMPHY Register Map

The ALTMEMPHY register map allows you to control the memory components' mode register settings. Table 6–16 shows the register map for ALTMEMPHY.

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

Table 6–16. ALTMEMPHY Register Map (Part 1 of 2)

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

Table 6–16. ALTMEMPHY Register Map (Part 2 of 2)

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

Controller Register Map

The controller register map allows you to control the memory controller settings. To access the controller register map, connect the CSR interface signals using the Avalon-MM protocol. Table 6–17 shows the register map for the controller.

Table 6–17. Controller Register Map (Part 1 of 5)

Address	Bit	Name	Default	Access	Description
0x100	0	CAL_SUCCESS	—	Read only	This bit reports the value of the ALTMEMPHY `ctl_cal_success` output. Writing to this bit has no effect.
	1	CAL_FAIL	—	Read only	This bit reports the value of the ALTMEMPHY `ctl_cal_fail` output. Writing to this bit has no effect.
	2	CAL_REQ	0	Read write	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. You must not use this register during the ALTMEMPHY megafunction calibration. You must wait until the CAL_SUCCESS or CAL_FAIL register shows a value of 1.
	7:3	Reserved.	0	—	Reserved for future use.
	13:8	Reserved.	0	—	Reserved for future use.
	30:14	Reserved.	0	—	Reserved for future use.

Table 6–17. Controller Register Map (Part 2 of 5)

Address	Bit	Name	Default	Access	Description
0x110	15:0	AUTO_PD_CYCLES	0x0	Read write	The number of idle clock cycles after which the controller should place the memory into power-down mode. The controller is considered to be idle if there are no commands in the command queue. Setting this register to 0 disables the auto power-down mode. The default value of this register depends on the values set during the generation of the design.
	16	Reserved.	0	—	Reserved for future use.
	17	Reserved.	0	—	Reserved for future use.
	18	Reserved.	0	—	Reserved for future use.
	19	Reserved.	0	—	Reserved for future use.
	21:20	ADDR_ORDER	00	Read write	00 - Chip, row, bank, column. 01 - Chip, bank, row, column. 10 - reserved for future use. 11 - Reserved for future use.
	22	REGDIMM	0	Read write	Setting this bit to 1 enables REGDIMM support in the controller.
	24:23	Reserved.	0	—	Reserved for future use.
	30:24	Reserved	0	—	Reserved for future use.
0x120	7:0	Column address width	—	Read write	The number of column address bits for the memory devices in your memory interface. The range of legal values is 7-12.
	15:8	Row address width	—	Read write	The number of row address bits for the memory devices in your memory interface. The range of legal values is 12-16.
	19:16	Bank address width	—	Read write	The number of bank address bits for the memory devices in your memory interface. The range of legal values is 2-3.
	23:20	Chip select address width	—	Read write	The number of chip select address bits for the memory devices in your memory interface. The range of legal values is 0-2. If there is only one single chip select in the memory interface, set this bit to 0.
	31:24	Reserved.	0	—	Reserved for future use.
0x121	31:0	Data width representation (word)	—	Read only	The number of DQS bits in the memory interface. This bit can be used to derive the width of the memory interface by multiplying this value by the number of DQ pins per DQS pin (typically 8).
0x122	7:0	Chip select representation	—	Read only	The number of chip select in binary representation. For example, a design with 2 chip selects has the value of 00000011.
	31:8	Reserved.	0	—	Reserved for future use.

Table 6–17. Controller Register Map (Part 3 of 5)

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

Table 6–17. Controller Register Map (Part 4 of 5)

Address	Bit	Name	Default	Access	Description
0x130	0	ENABLE_ECC	1	Read write	When this bit equals 1, it enables the generation and checking of ECC. This bit is only active if ECC was enabled during IP parameterization.
	1	ENABLE_AUTO_CORR	—	Read write	When this bit equals 1, it enables auto-correction when a single-bit error is detected.
	2	GEN_SBE	0	Read write	When this bit equals 1, it enables the deliberate insertion of single-bit errors, bit 0, in the data written to memory. This bit is used only for testing purposes.
	3	GEN_DBE	0	Read write	When this bit equals 1, it enables the deliberate insertion of double-bit errors, bits 0 and 1, in the data written to memory. This bit is used only for testing purposes.
	4	ENABLE_INTR	1	Read write	When this bit equals 1, it enables the interrupt output.
	5	MASK_SBE_INTR	0	Read write	When this bit equals 1, it masks the single-bit error interrupt.
	6	MASK_DBE_INTR	0	Read write	When this bit equals 1, it masks the double-bit error interrupt
	7	CLEAR	0	Read write	When this bit equals 1, writing to this self-clearing bit clears the interrupt signal, and the error status and error address registers.
	8	MASK_CORDROP_INTR	0	Read write	When this bit equals 1, the dropped autocorrection error interrupt is dropped.
	9	Reserved.	0	—	Reserved for future use.
0x131	0	SBE_ERROR	0	Read only	Set to 1 when any single-bit errors occur.
	1	DBE_ERROR	0	Read only	Set to 1 when any double-bit errors occur.
	2	CORDROP_ERROR	0	Read only	Value is set to 1 when any controller-scheduled autoconnections are dropped.
	7:3	Reserved.	0	—	Reserved for future use.
	15:8	SBE_COUNT	0	Read only	Reports the number of single-bit errors that have occurred since the status register counters were last cleared.
	23:16	DBE_COUNT	0	Read only	Reports the number of double-bit errors that have occurred since the status register counters were last cleared.
	31:24	CORDROP_COUNT.	0	Read only	Reports the number of controller-scheduled autocorrections dropped since the status register counters were last cleared.
0x132	31:0	ERR_ADDR	0	Read only	The address of the most recent ECC error. This address is a memory burst-aligned local address.

Table 6–17. Controller Register Map (Part 5 of 5)

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

7. 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.

Altera defines read and write latencies in terms of the local interface clock frequency and by the absolute time for the memory controllers. These latencies apply to supported device families with the half-rate DDR3 high-performance controller.

The latency defined in this section uses the following assumptions:

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

The latency for the high-performance controller comprises many different stages of the memory interface.

Figure 7–1 shows a typical memory interface read latency path showing read latency from the time a `local_read_req` assertion is detected by the controller up to data available to be read from the dual-port RAM (DPRAM) module.

Figure 7–1. Typical Latency Path

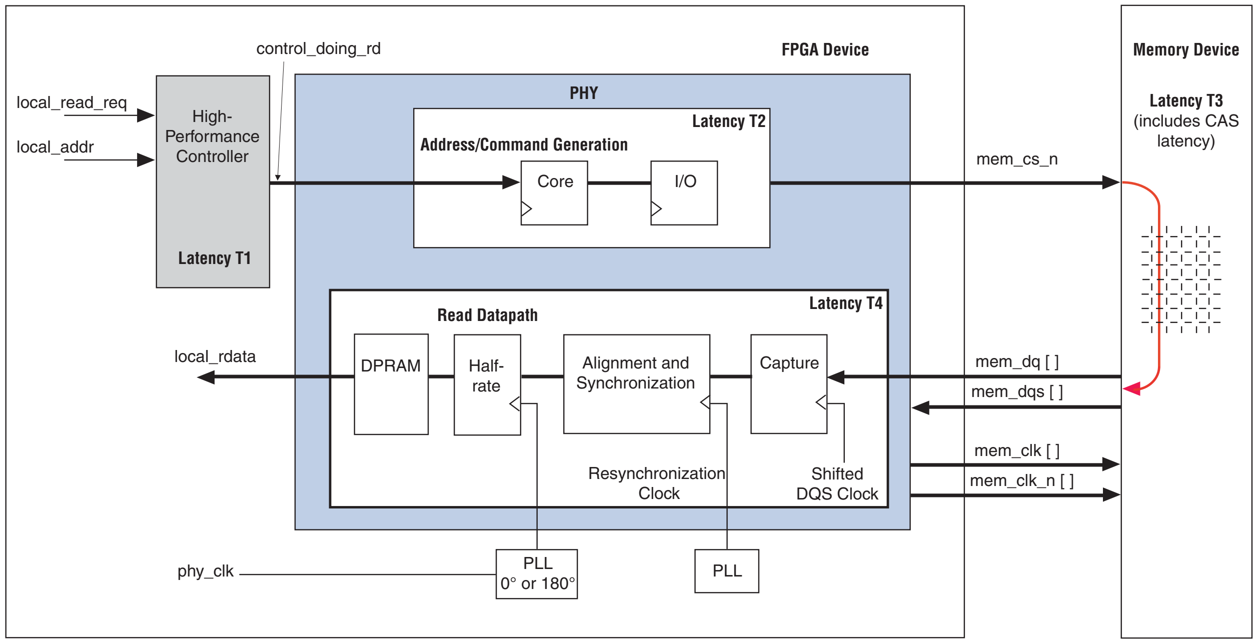


Table 7–1 shows the different stages that make up the whole read and write latency that Figure 7–1 shows.

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

Latency Number	Latency Stage	Description
T1	Controller	`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 7–1, the read latency in the high-performance controller is made up of four components:

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

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

write latency = controller latency (T1) + write data latency (T2+T3)

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

☞ The exact latency depends on your precise configuration. You should obtain precise latency from simulation, but this figure may vary in hardware because of the automatic calibration process.

8. Timing Diagrams

This chapter details the timing diagrams for the DDR3 SDRAM high-performance controller.

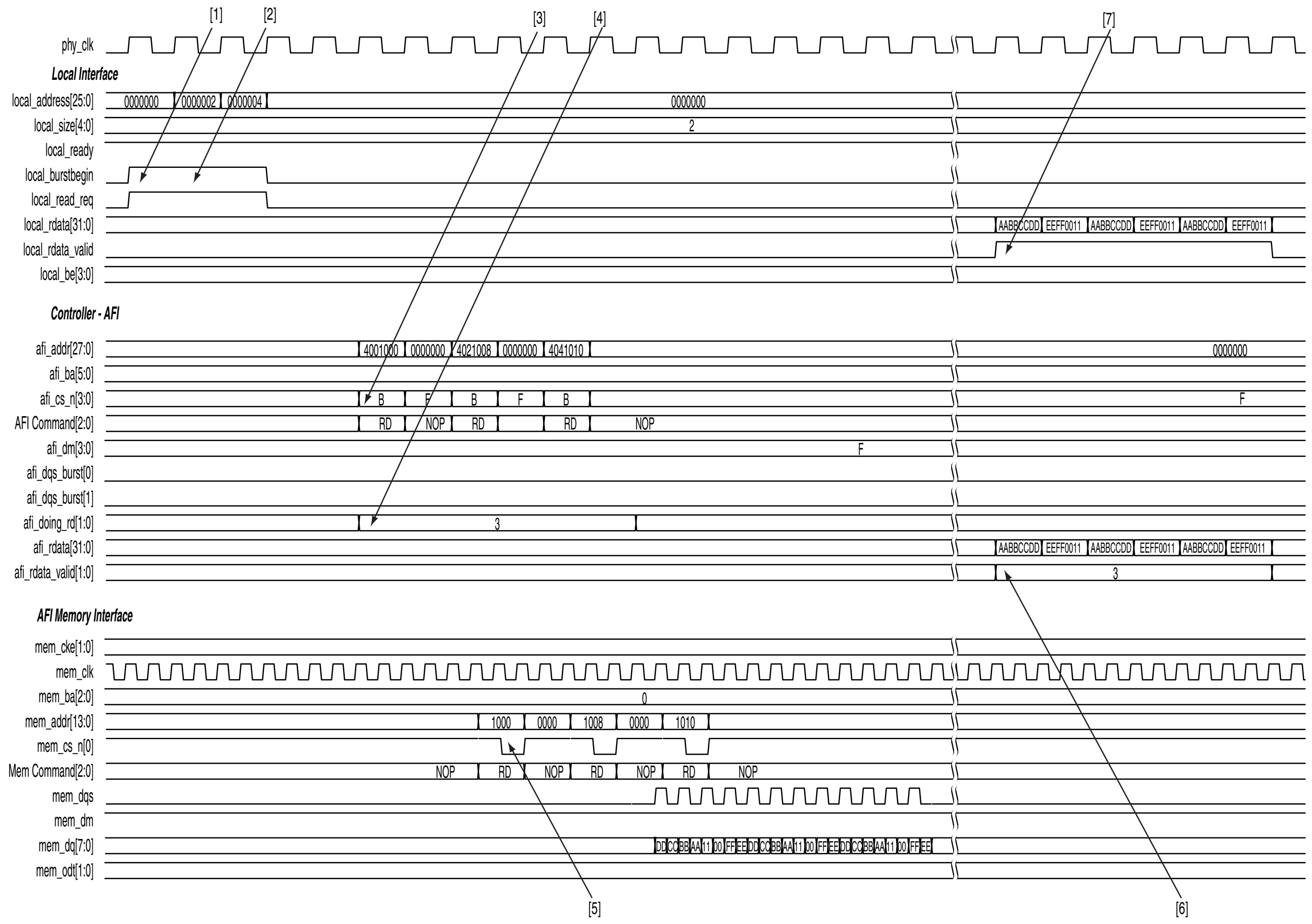
DDR3 High-Performance Controller II

This section discusses the following timing diagrams for HPC II:

- "Half-Rate Read (Burst-Aligned Address)"
- "Half-Rate Write (Burst-Aligned Address)"
- "Half-Rate Read (Non Burst-Aligned Address)"
- "Half-Rate Write (Non Burst-Aligned Address)"
- "Half-Rate Read With Gaps"
- "Half-Rate Write With Gaps"
- "Half-Rate Write Operation (Merging Writes)"
- "Write-Read-Write-Read Operation"

Half-Rate Read (Burst-Aligned Address)

Figure 8–1. Half-Rate Read Operation for HPC II—Burst-Aligned Address



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

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

   ```
   mem_row_address = 0x000000

   mem_col_address = 0x0000

   mem_bank_address = 0x00
   ```

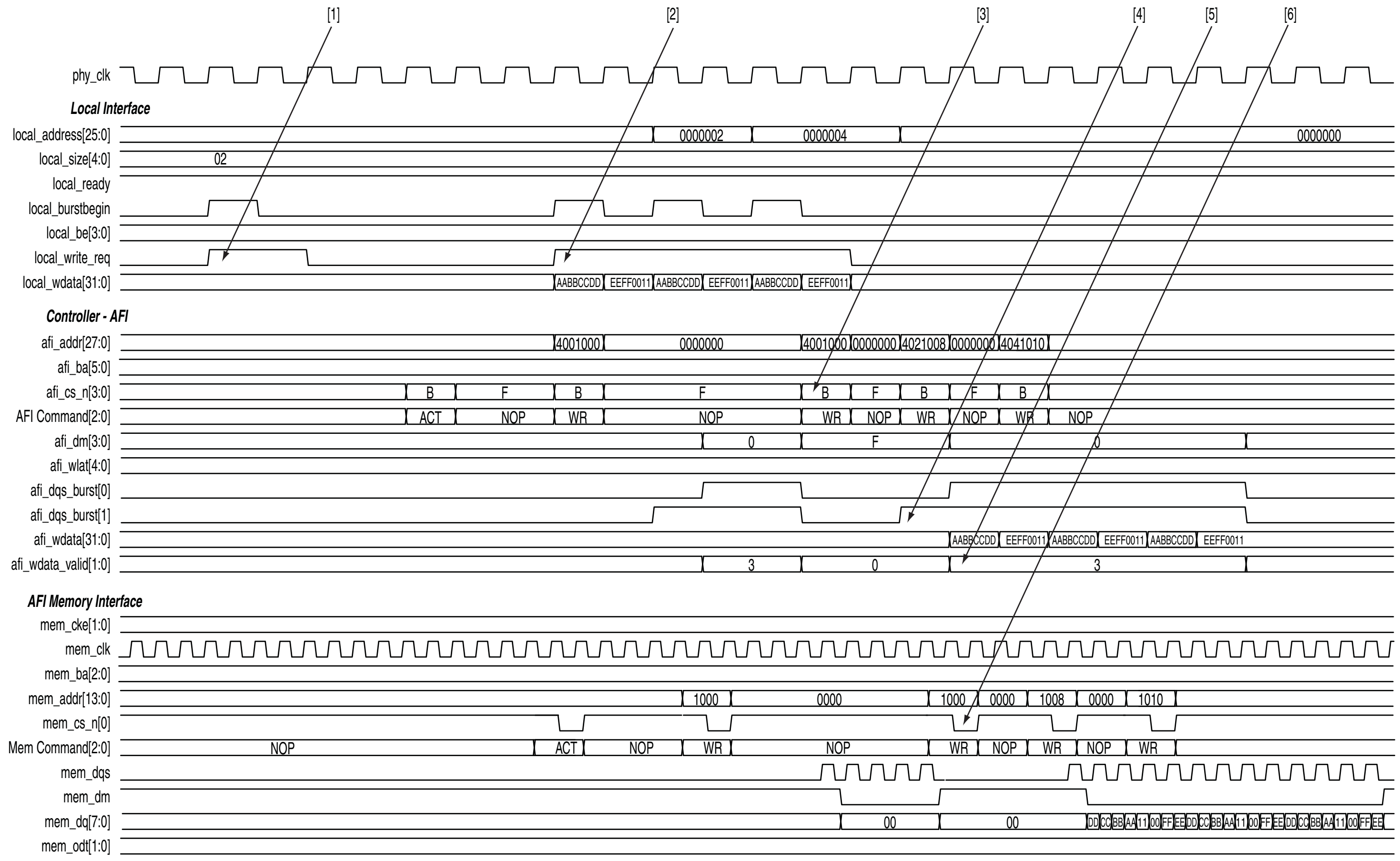
2. The user logic initiates a second read to a different memory column within the same row. The request for the second read is a burst length of 2. In this example, the user logic continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the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

   ```
   mem_row_address = 0x0000

   mem_col_address = 0x0002<<2 = 0x0008

   mem_bank_address = 0x00
   ```

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

Half-Rate Write (Burst-Aligned Address)

Figure 8–2. Half-Rate Write Operation for HPC II—Burst-Aligned Address

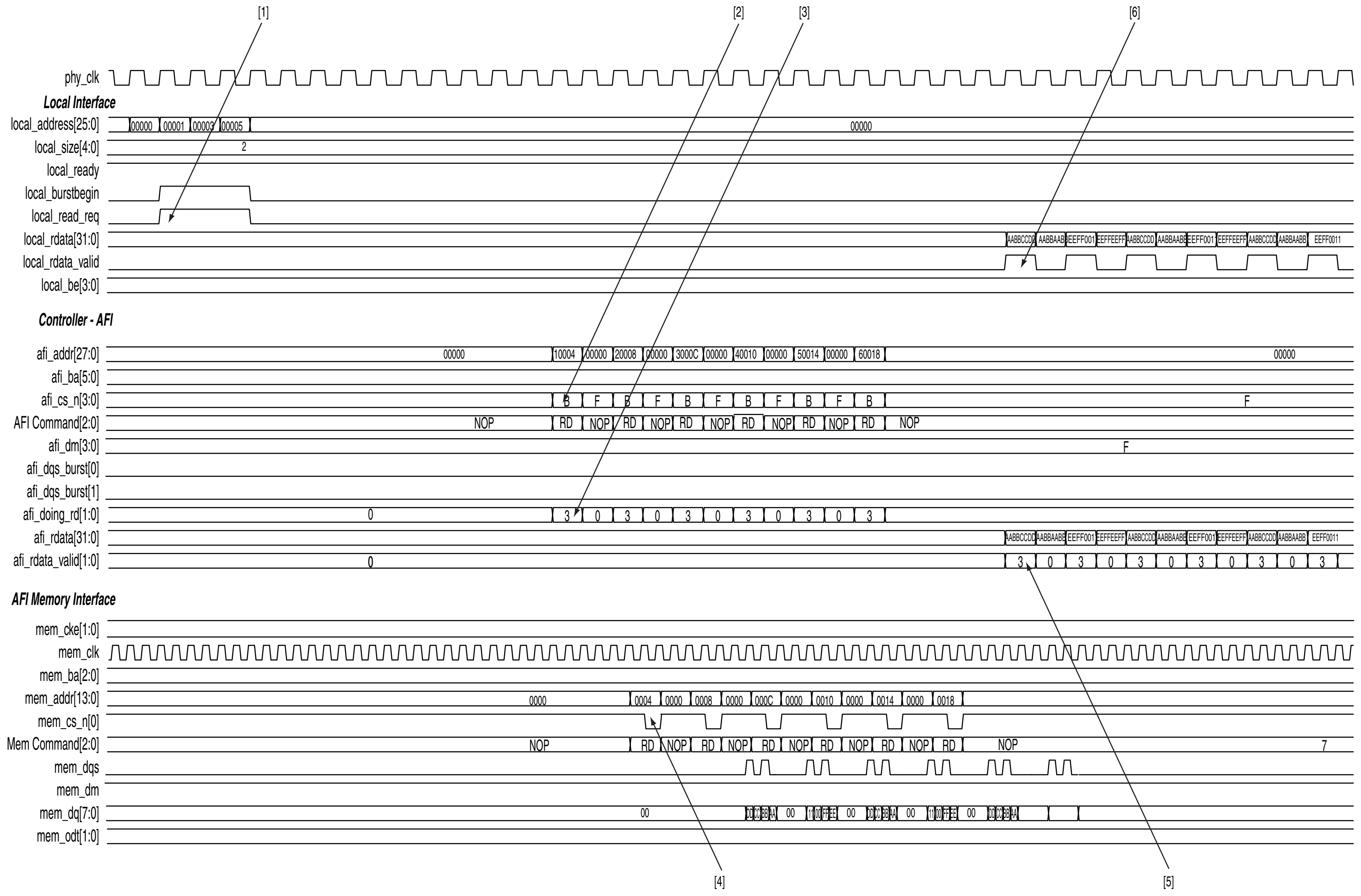


The following sequence corresponds with the numbered items in Figure 8–2:

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

Half-Rate Read (Non Burst-Aligned Address)

Figure 8–3. Half-Rate Read Operation for HPC II—Non Burst-Aligned Address



The following sequence corresponds with the numbered items in Figure 8–3:

1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0x000001`. This local address is mapped to the following memory address in half-rate mode:

   `mem_row_address = 0x0000`

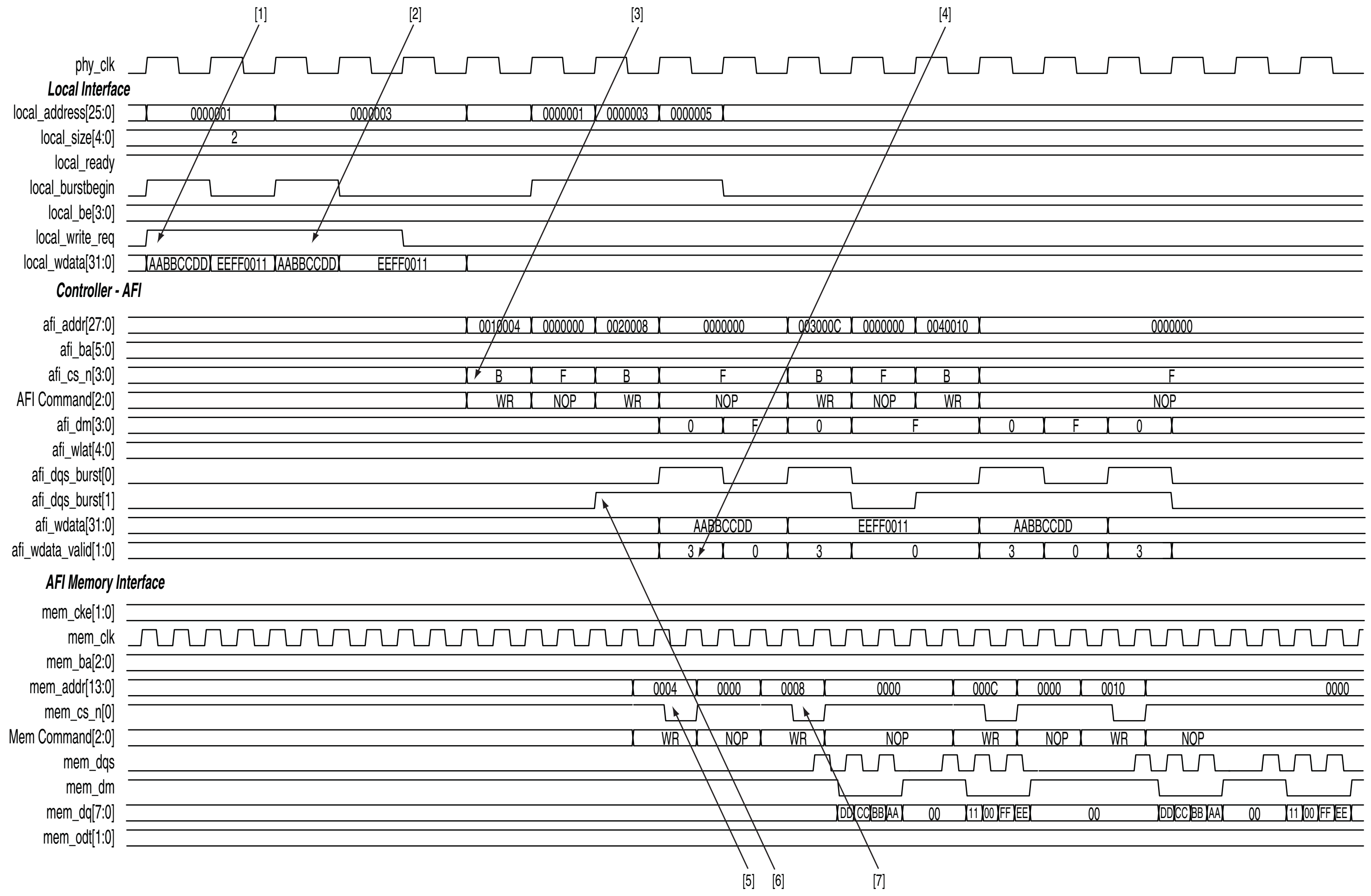
   `mem_col_address = 0x0001<<2 = 0x0004`

   `mem_bank_address = 0x00`

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

Half-Rate Write (Non Burst-Aligned Address)

Figure 8–4. Half-Rate Write Operation for HPC II—Non Burst-Aligned Address



The following sequence corresponds with the numbered items in Figure 8–4:

1. The user logic asserts the first `local_write_req` signal with a size of 2 and an address of `0×000001`. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high. The local address 0x000001 is mapped to the following memory address in half-rate mode:

   `mem_row_address = 0×0000`

   `mem_col_address = 0×000001<<2 = 0×000004`

   `mem_bank_address = 0×00`

2. The user logic asserts the second `local_write_req` signal with a size of 2 and an address of `0×000003`. The local address `0×000003` is mapped to the following memory address in half-rate mode:

   `mem_row_address = 0×0000`

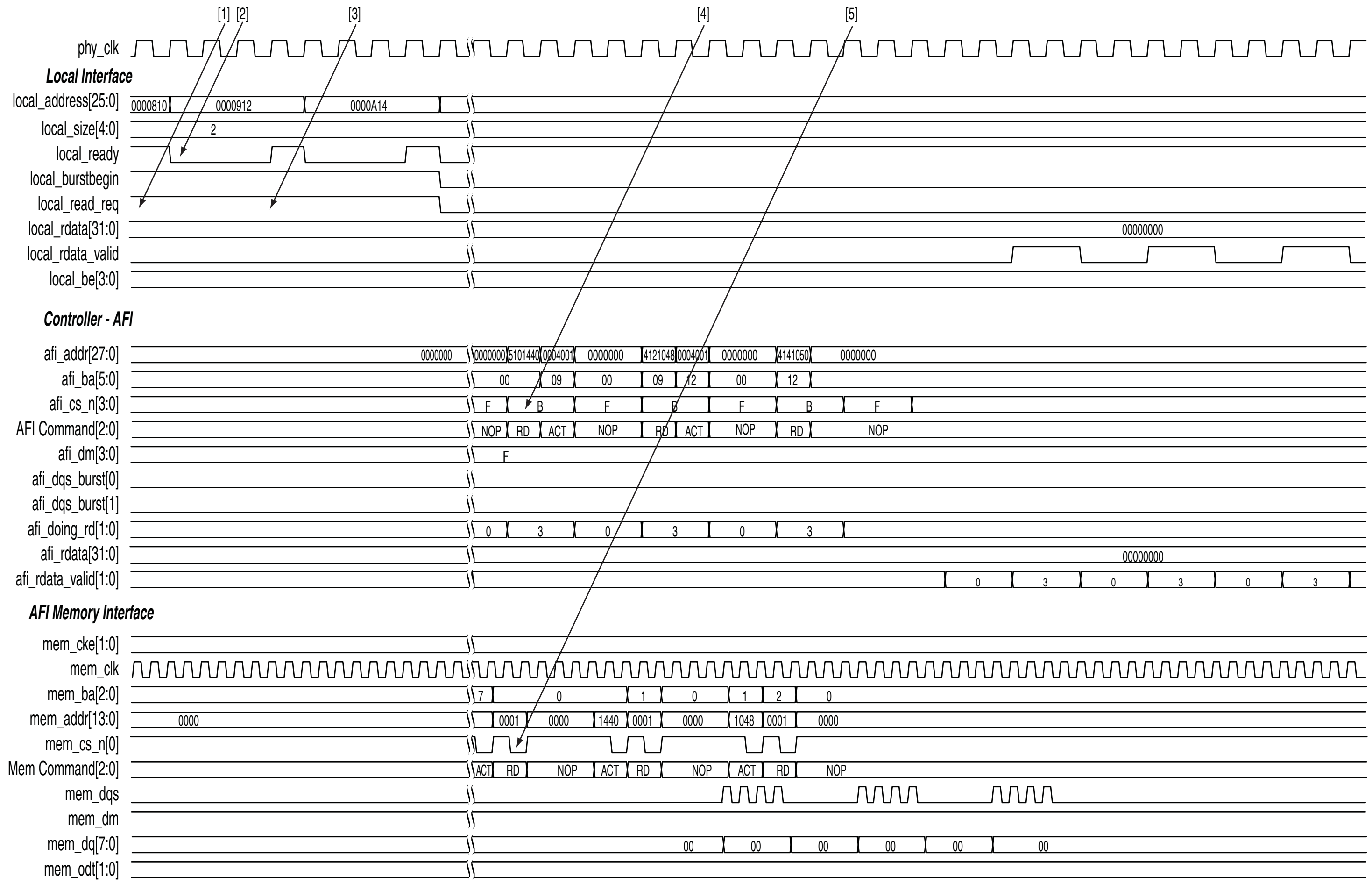
   `mem_col_address = 0×000003<<2 = 0×00000C`

   `mem_bank_address = 0×00`

3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
7. The controller generates another write because the first write is to a non-aligned memory address, `0×0004`. The controller performs the second write burst at the memory address of `0×0008`.

Half-Rate Read With Gaps

Figure 8–5. Half-Rate Read Operation for HPC II—With Gaps



The following sequence corresponds with the numbered items in Figure 8–5:

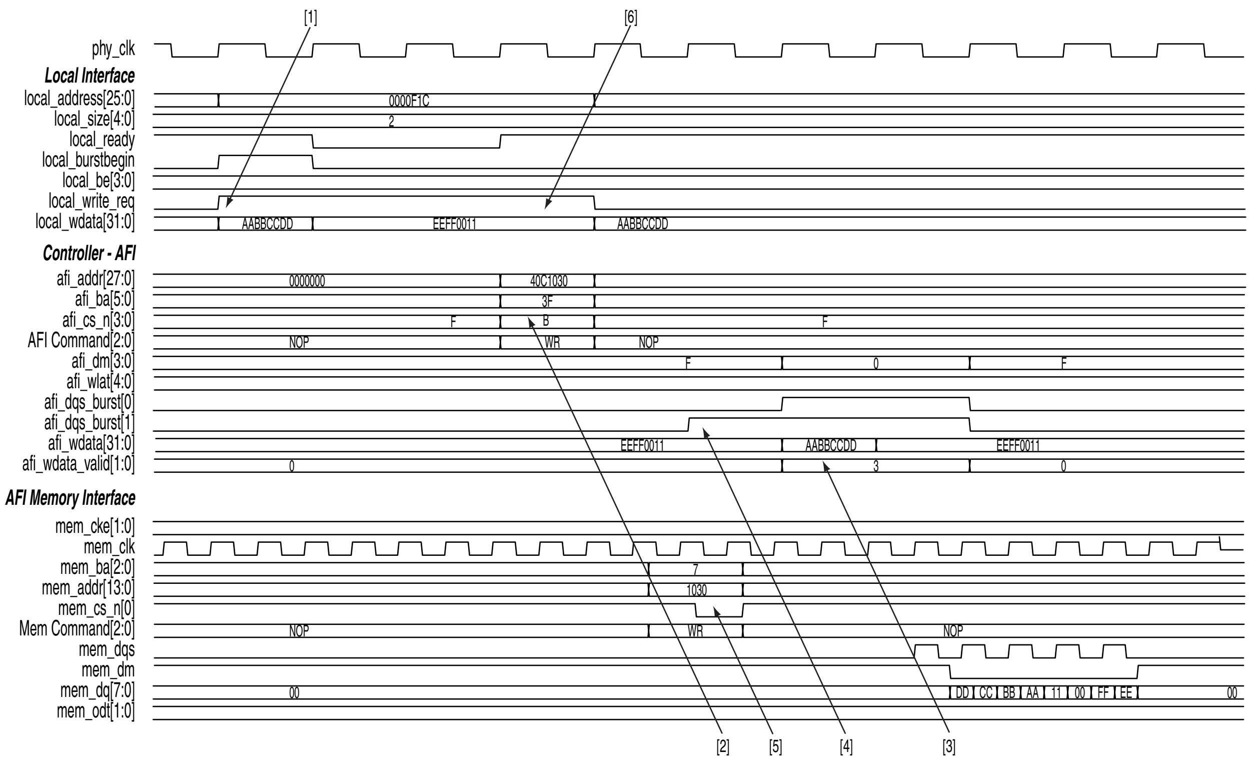
1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length of 2 to the local address `0×0000810`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0×0001

   mem_col_address = 0×0010<<2 = 0×0040

   mem_bank_address = 0×00
   ```

2. When the command queue is full, the controller deasserts the `local_ready` signal to indicate that the controller has not accepted the command. The user logic must keep the read request, size, and address signal until the `local_ready` signal is asserted again.
3. The user logic asserts a second `local_read_req` signal with a size of 2 and address of `0×0000912`.
4. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
5. The ALTMEMPHY megafunction issues the read command to the memory and captures the read data from the memory.

Half-Rate Write With Gaps

Figure 8–6. Half-Rate Write Operation for HPC II—With Gaps

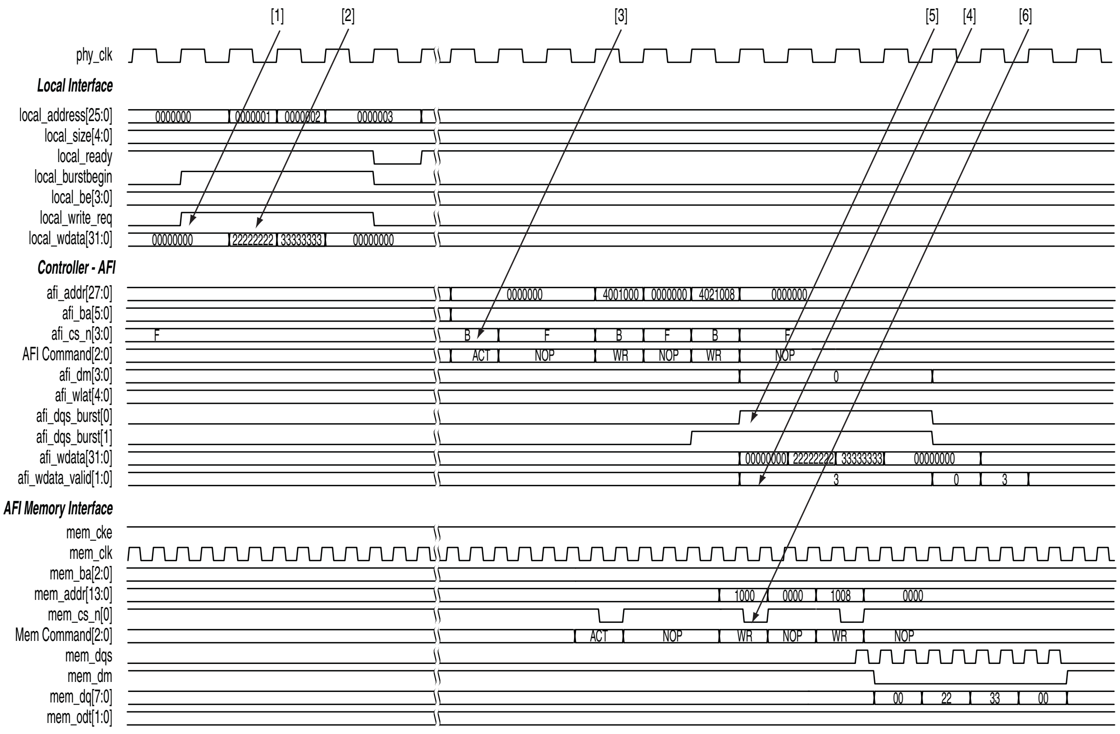


The following sequence corresponds with the numbered items in Figure 8–6:

1. The user logic asserts a `local_write_req` signal with a size of 2 and an address of `0×0000F1C`.
2. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
3. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
4. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
5. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
6. For transactions with a local size of two, the `local_write_req` and `local_ready` signals must be high for two clock cycles so that all the write data can be transferred to the controller.

Half-Rate Write Operation (Merging Writes)

Figure 8–7. Write Operation for HPC II—Merging Writes



The following sequence corresponds with the numbered items in Figure 8–7:

1. The user logic asserts the first `local_write_req` signal with a size of 1 and an address of `0x000000`. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request, and the user logic can request another read or write in the following clock cycle. If the `local_ready` signal was not asserted, the user logic must keep the write request, size, and address signals asserted until the `local_ready` signal is registered high. The local address `0x000000` is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000
   ```

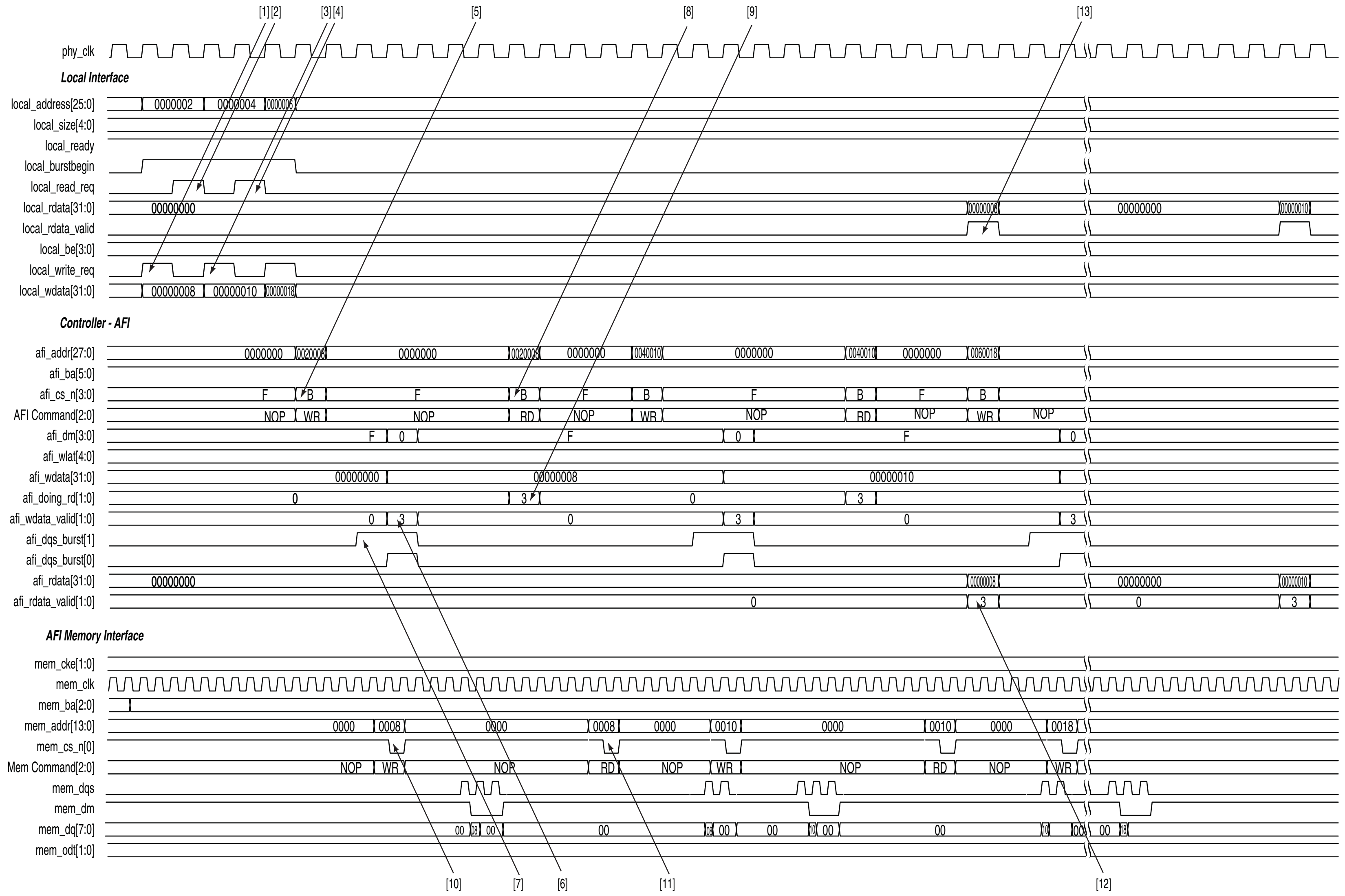
   ```
   mem_col_address = 0x0000<<2 = 0x0000
   ```

   ```
   mem_bank_address = 0x00
   ```

2. The user logic asserts a second `local_write_req` signal with a size of 1 and address of 1. The `local_ready` signal is asserted along with the `local_write_req` signal, which indicates that the controller has accepted this request. Since the second write request is to a sequential address (same row, same bank, and column increment by 1), this write and the first write can be merged at the memory transaction.
3. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
4. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
5. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signal that the ALTMEMPHY megafunction issues to the memory.
6. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.

Write-Read-Write-Read Operation

Figure 8–8. Write-Read Sequential Operation for HPC II



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

1. The user logic requests the first write by asserting the `local_write_req` signal, and the size and address for this write. In this example, the request is a burst length of 1 to a local address `0x000002`. This local address is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000
   mem_col_address = 0x0002<<2 = 0x0008
   mem_bank_address = 0x00
   ```

2. The user logic initiates the first read to the same address as the first write. The request for the read is a burst length of 1. The controller continues to accept commands until the command queue is full. When the command queue is full, the controller deasserts the `local_ready` signal. The starting local address 0x000002 is mapped to the following memory address in half-rate mode:

   ```
   mem_row_address = 0x0000
   mem_col_address = 0x0002<<2 = 0x0008
   mem_bank_address = 0x00
   ```

3. The user logic asserts a second `local_write_req` signal with a size of 1 and address of `0x000004`.
4. The user logic asserts a second `local_read_req` signal with a size of 1 and address of `0x000004`.
5. The controller issues the necessary memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
6. The controller asserts the `afi_wdata_valid` signal to indicate to the ALTMEMPHY megafunction that valid write data and write data masks are present on the inputs to the ALTMEMPHY megafunction.
7. The controller asserts the `afi_dqs_burst` signals to control the timing of the DQS signals that the ALTMEMPHY megafunction issues to the memory.
8. The controller issues the first read memory command and address signals to the ALTMEMPHY megafunction for it to send to the memory device.
9. The controller asserts the `afi_doing_rd` signal to indicate to the ALTMEMPHY megafunction the number of clock cycles of read data it must expect for the first read. The ALTMEMPHY megafunction uses the `afi_doing_rd` signal to enable its capture registers for the expected duration of memory burst.
10. The ALTMEMPHY megafunction issues the write command, and sends the write data and write DQS to the memory.
11. The ALTMEMPHY megafunction issues the first read command to the memory and captures the read data from the memory.
12. The ALTMEMPHY megafunction returns the first data read to the controller after resynchronizing the data to the `phy_clk` domain, by asserting the `afi_rdata_valid` signal when there is valid read data on the `afi_rdata` bus.

13. The controller returns the first read data to the user by asserting the `local_rdata_valid` signal when there is valid read data on the `local_rdata` bus. If the ECC logic is disabled, there is no delay between the `afi_rdata` and the `local_rdata` buses. If there is ECC logic in the controller, there is one or three clock cycles of delay between the `afi_rdata` and `local_rdata` buses.

Additional Information

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
June 2011	3.0	■ Removed High-Performance Controller information. ■ Updated High-Performance Controller II information. ■ Removed HardCopy III, HardCopy IV E, HardCopy IV GX, Stratix III, and Stratix IV support.
December 2010	2.1	Updated for 10.1.
July 2010	2.0	■ Added information for new GUI parameters: **Controller latency**, **Enable reduced bank tracking for area optimization**, and **Number of banks to track**. ■ Removed information about IP Advisor. This feature is removed from the DDR3 SDRAM IP support for version 10.0.
February 2010	1.3	Corrected typos.
February 2010	1.2	■ Full support for Stratix IV devices. ■ Added information for **Register Control Word** parameters. ■ Added descriptions for `mem_ac_parity`, `mem_err_out_n`, and `parity_error_n` signals. ■ Added timing diagrams for initialization and calibration stages for HPC.
November 2009	1.1	Minor corrections.
November 2009	1.0	First published.

How to Contact Altera

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

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

Note to Table:

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

Typographic Conventions

The following table shows the typographic conventions this document uses.

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