



DDR and DDR2 SDRAM

High-Performance Controller User Guide



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com

MegaCore Version:	7.1
Document Version:	7.1
Document Date:	May 2007

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



UG-01010-3.0



About This User Guide

Revision History	v
How to Contact Altera	v
Typographic Conventions	v

Chapter 1. About These MegaCore Functions

Release Information	1-1
Device Family Support	1-1
Features	1-2
General Description	1-2
OpenCore Plus Evaluation	1-3
Performance	1-4

Chapter 2. Getting Started

Design Flow	2-1
DDR & DDR2 SDRAM High-Performance Controller Walkthrough	2-2
Create a New Quartus II Project	2-3
Launch the MegaWizard Plug-In Manager	2-4
Launch the MegaWizard Plug-In from SOPC Builder	2-7
Parameterize	2-11
Set Up Simulation	2-14
Generate Files	2-16
Simulate the Example Design	2-18
Simulate with IP Functional Simulation Models	2-18
Simulating in Third-Party Simulation Tools Using NativeLink	2-18
Compile the Example Design	2-19
Program a Device	2-22
Implement Your Design	2-22
Set Up Licensing	2-22

Chapter 3. Specifications

Functional Description	3-1
Control Logic	3-1
Latency	3-3
ECC	3-4
OpenCore Plus Time-Out Behavior	3-8
Example Design	3-9
Interfaces & Signals	3-10
Interface Description	3-10
Signals	3-22

Contents

Parameters	3–25
MegaCore Verification	3–25
Appendix A. ECC Register Description	A–1



About This User Guide

Revision History

The table below displays the revision history for the chapters in this user guide.

Date	Version	Changes Made
May 2007	7.1	<ul style="list-style-type: none">• Updated device support• Added description of ECC• Added Appendix A
December 2006	7.0	Added Cyclone® III support.
December 2006	6.1	First release.

How to Contact Altera

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

Information Type	Contact <i>Note (1)</i>
Technical support	www.altera.com/mysupport/
Technical training	www.altera.com/training/
Technical training services	custrain@altera.com
Product literature	www.altera.com/literature
Product literature services	literature@altera.com
FTP site	ftp.altera.com





Note to table:

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

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: Save As dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX} , lqdesigns directory, d: drive, chiptrip.gdf file.

Visual Cue	Meaning
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75: High-Speed Board Design</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA} , $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <file name>, <project name>.pdf file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix n, e.g., resetn. Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword SUBDESIGN), as well as logic function names (e.g., TRI) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● ●	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
	A warning calls attention to a condition or possible situation that can cause injury to the user.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.



1. About These MegaCore Functions

Release Information

Table 1–1 provides information about this release of the DDR and DDR2 SDRAM High-Performance Controller MegaCore® functions.

Table 1–1. DDR & DDR2 SDRAM High-Performance Controller Release Information

Item	Description
Version	7.1
Release Date	May 2007
Ordering Codes	IP-SDRAM/HPDDR (DDR SDRAM) IP-SDRAM/HPDDR2 (DDR2 SDRAM)
Product IDs	00BE (DDR SDRAM) 00BF (DDR2 SDRAM) 00CO (ALTMEMPHY Megafunction)
Vendor ID	6AF7

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera® device families, as described below:

- *Full support* means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs
- *Preliminary support* means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution

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

Table 1–2. Device Family Support		
Device Family	Support	
	DDR SDRAM	DDR2 SDRAM
Arria™ GX	Preliminary	Preliminary
Cyclone® III	Preliminary	Preliminary
HardCopy® II	Preliminary	Preliminary
Stratix® II	Full	Full
Stratix II GX	Full	Full
Stratix III	Preliminary	Preliminary
Other device families	No support	No support

Features

- Integrated error correction coding (ECC) function
- Power-up on-chip termination (OCT) support for Cyclone III and Stratix III devices
- Full-rate support for Arria GX, Cyclone III, HardCopy II, Stratix II, and Stratix II GX devices
- SOPC Builder ready
- Support for ALTMEMPHY megafunction
- Support for industry-standard DDR and DDR2 SDRAM devices and modules
- Optional user-controller refresh
- Optional Avalon® Memory-Mapped (Avalon-MM) local interface
- Easy-to-use MegaWizard® interface
- Support for OpenCore Plus evaluation
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

General Description

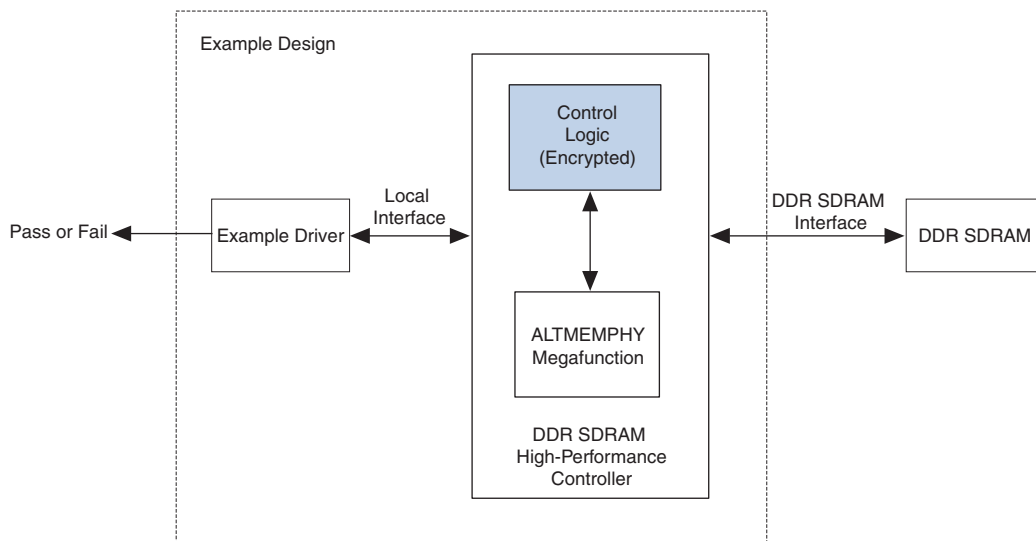
The Altera DDR and DDR2 SDRAM High-Performance Controller MegaCore functions provide simplified interfaces to industry-standard DDR SDRAM and DDR2 SDRAM. The MegaCore functions work in conjunction with the Altera ALTMEMPHY megafunction.



For more information on the ALTMEMPHY megafunction, refer to the *ALTMEMPHY Megafunction User Guide*.

Figure 1–1 shows a system-level diagram including the example design that the DDR or DDR2 SDRAM high-performance controller MegaCore functions create for you.

Figure 1–1. System-Level Diagram



The MegaWizard Plug-In Manager generates an example design, instantiates a phase-locked loop (PLL), an example driver, your DDR or DDR2 SDRAM high-performance controller custom variation, and an optional DLL (for Stratix series only). The example design is a fully-functional design that can be simulated, synthesized, and used in hardware. The example driver is a self-test module that issues read and write commands to the controller and checks the read data to produce the pass/fail and test complete signals.

OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

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

- Program a device and verify your design in hardware

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



For more information on OpenCore Plus hardware evaluation using the DDR and DDR2 SDRAM high-performance controller, see “[OpenCore Plus Time-Out Behavior](#)” on page 3–8 and *Application Note 320: OpenCore Plus Evaluation of Megafunctions*.

Performance

[Table 1–3](#) shows typical performance results for the DDR SDRAM high-performance controller using the Quartus® II software version 7.1.

Table 1–3. Typical Performance		
Device	System f_{MAX} (MHz)	
	DDR SDRAM	DDR2 SDRAM
Cyclone III	167 (1)	200 (1)
Stratix II	200	333
Stratix III	200 (1)	400 (1)

Note to [Table 1–3](#):

(1) Pending device characterization.



For more information on device performance, see the relevant device handbook.

[Table 1–4](#) shows typical sizes for the DDR SDRAM high-performance controller on Cyclone III devices.

Table 1–4. Typical Size—Cyclone III Devices			
Local Data Width (Bits)	Memory Width (Bits)	LEs	Memory Blocks (M9K)
32	8	1,929	3
64	16	2,083	6
256	64	2,983	21
288	72	3,167	22

Table 1–5 shows typical sizes for the DDR SDRAM high-performance controller on Stratix II devices.

Table 1–5. Typical Size—Stratix II Devices				
Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Memory Blocks (M4K)
32	8	1,420	1,480	3
64	16	1,530	1693	6
256	64	2,136	2,914	20
288	72	2,291	3,119	22

Table 1–6 shows typical sizes for the DDR SDRAM high-performance controller on Stratix III devices (with no calibration logic).

Table 1–6. Typical Size—Stratix III Devices				
Local Data Width (Bits)	Memory Width (Bits)	Combinational ALUTs	Logic Registers	Memory Blocks (MLAB)
32	8	796	1,163	5
64	16	937	1,440	10
256	64	1,773	3,110	39
288	72	1,907	3,383	44

Design Flow

To evaluate the DDR and DDR2 SDRAM High-Performance Controller MegaCore® functions using the OpenCore Plus feature, include these steps in your design flow:

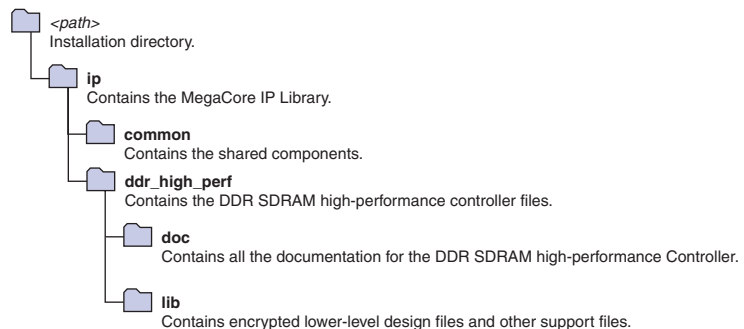
1. Obtain and install the DDR and DDR2 SDRAM High-Performance Controller MegaCore.

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

For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows* or *Quartus II Installation & Licensing for UNIX & Linux* on the Altera website at www.altera.com/literature/lit-qts.jsp.

Figure 2–1 shows the directory structure after you install the DDR and DDR2 SDRAM High-Performance Controller MegaCore functions, where *<path>* is the installation directory. The default installation directory on Windows is *c:\altera\71*; on UNIX and Solaris it is */opt/altera/71*.

Figure 2–1. Directory Structure



2. Create a custom variation of the DDR or DDR2 SDRAM High-Performance Controller MegaCore function.

3. Use the IP functional simulation model to verify the operation of the example design and the example driver.



For more information on IP functional simulation models, see the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

4. Use the Quartus II software to add constraints to the example design, compile the example design, and perform post-compilation timing analysis.
5. Perform gate-level timing simulation, or if you have a suitable development board, you can generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of the example design in hardware.
6. Purchase a license for the DDR or DDR2 SDRAM High-Performance Controller MegaCore function.

After you have purchased a license for the DDR and DDR2 SDRAM High-Performance Controller, follow these additional steps:

1. Set up licensing.
2. Generate a programming file for the Altera® device(s) on your board.
3. Program the Altera device(s) with the completed design.

DDR & DDR2 SDRAM High-Performance Controller Walkthrough

This walkthrough explains how to create a custom variation of the DDR or DDR2 SDRAM high-performance controller MegaCore function using the MegaWizard® Plug-In Manager and the Quartus II software. As you go through the wizard, each step is described in detail.

This walkthrough requires the following steps:

- “Create a New Quartus II Project” on page 2–3
- “Launch the MegaWizard Plug-In Manager” on page 2–4
- “Parameterize” on page 2–11
- “Set Up Simulation” on page 2–14
- “Generate Files” on page 2–16

Create a New Quartus II Project

You need to create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity. To create a new project follow these steps:

1. Choose **Programs > Altera > Quartus II <version>** (Windows Start menu) to run the Quartus II software. Alternatively, you can use the Quartus II Web Edition software.
2. Choose **New Project Wizard** (File menu).
3. Click **Next** in the **New Project Wizard Introduction** page (the introduction page does not display if you turned it off previously).
4. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:
 - a. Specify the working directory for your project. For example, this walkthrough uses the **c:\altera\projects\ddr_project** directory.
 - b. Specify the name of the project. This walkthrough uses **project** for the project name.



The Quartus II software automatically specifies a top-level design entity that has the same name as the project. Do not change it.

5. Click **Next** to close this page and display the **New Project Wizard: Add Files** page.



When you specify a directory that does not already exist, a message asks if the specified directory should be created. Click **Yes** to create the directory.

6. If you installed the MegaCore IP Library in a different directory from where you installed the Quartus II software, you must add the user libraries:
 - a. Click **User Libraries**.
 - b. Type `<path>\ip` into the **Library name** box, where `<path>` is the directory in which you installed the DDR and DDR2 SDRAM High-Performance Controller.
 - c. Click **Add to** add the path to the Quartus II project.
 - d. Click **OK** to save the library path in the project.
7. Click **Next** to close this page and display the **New Project Wizard: Family & Device Settings** page.
8. On the **New Project Wizard: Family & Device Settings** page, choose the target device family in the **Family** list.
9. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

You have finished creating your new Quartus II project. You can now launch the MegaWizard Plug-In in either the MegaWizard Plug-In Manager or from SOPC Builder.

Launch the MegaWizard Plug-In Manager

To launch the MegaWizard Plug-In Manager in the Quartus II software, follow these steps:

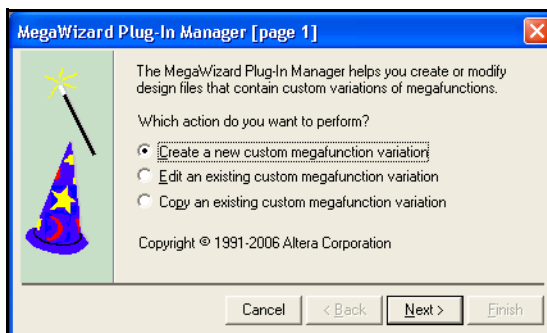


To launch the MegaWizard Plug-In from SOPC Builder, see [“Launch the MegaWizard Plug-In from SOPC Builder” on page 2–7](#).

1. Start the MegaWizard Plug-In Manager by choosing the **MegaWizard Plug-In Manager** command (Tools menu). The **MegaWizard Plug-In Manager** dialog box displays (see [Figure 2–2](#)).



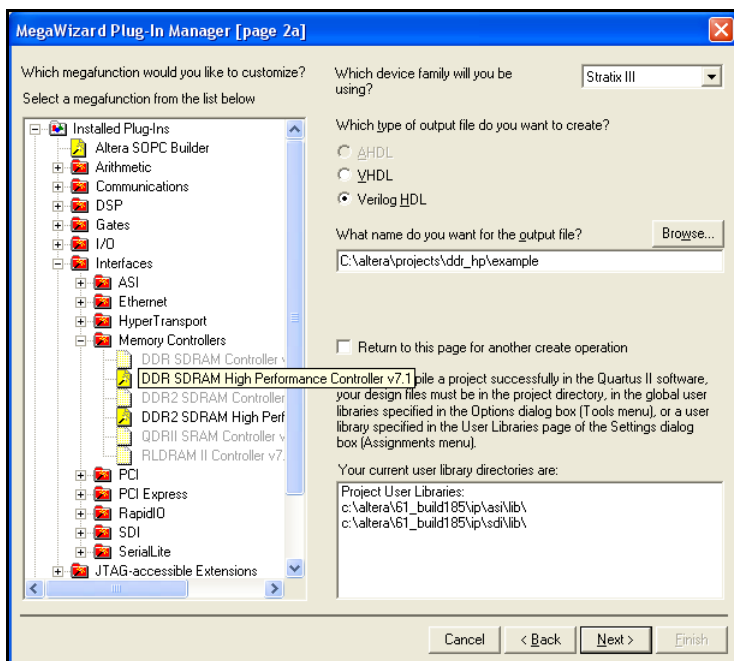
Refer to Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

Figure 2–2. MegaWizard Plug-In Manager

2. Specify that you want to create a new custom megafunction variation and click **Next**.
3. Expand the **Interfaces > Memory Controllers** directory then click either **DDR SDRAM High-Performance Controller v7.1** or **DDR2 SDRAM High-Performance Controller v7.1**.
4. Select the output file type for your design; the wizard supports VHDL and Verilog HDL.
5. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files `<project path>\<variation name>`. [Figure 2–3 on page 2–6](#) shows the wizard after you have made these settings.



The `<variation name>` should be a different name from the project name and the top-level design entity name. The wizard generates an example top-level design, `<variation name>_example_top.v(hd)`. You should use this example as a starting point for your own design. See [“Compile the Example Design” on page 2–19](#).

Figure 2–3. Select the MegaCore Function

6. Click **Next** to display the **Parameter Settings** page for the DDR and DDR2 SDRAM High-Performance Controller MegaCore functions (see [Figure 2–4](#)).



You can change the page that the MegaWizard Plug-In Manager displays by clicking **Next** or **Back** at the bottom of the dialog box. You can move directly to a named page by clicking the **Parameter Settings**, **Simulation Model**, or **Summary** tab.

Also, you can directly display individual parameter settings by clicking on the **Memory Settings**, **PHY Settings**, or **Controller Settings** tabs.

Figure 2–4. Parameters

MegaWizard Plug-In Manager - DDR SDRAM High Performance Controller
Version 7.1

Parameter Settings | Simulation Model | Summary

Memory Settings | PHY Settings | Controller Settings

General Settings

Device family: Stratix III

Speed grade: 4

PLL reference clock frequency: 100 MHz (10000 ps)

Memory clock frequency: 200 MHz (5000 ps)

Local interface clock frequency: Half (100.0 MHz)

Local interface width: 32 bits

Show in 'Memory Presets' List

Parameter	Value
Memory vendor	(All)
Memory format	(All)
Maximum memory frequency	(All)

Show All

Memory Presets

Presets

- JEDEC DDR-400 128Mb x8
- JEDEC DDR-400 128Mb x4
- JEDEC DDR-400 256Mb x8
- JEDEC DDR-400 256Mb x4
- JEDEC DDR-400 512Mb x8
- JEDEC DDR-400 512Mb x4

Load Preset...

Selected memory preset: JEDEC DDR-400 128Mb x8

Modify parameters...

Description: DDR-SDRAM, 200MHz, 16MB, 8 bits wide, Discrete Device, CAS 3.0, x 1 CS

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

Cancel < Back Next > Finish

- Go to **“Parameterize”** on page 2–11.

Launch the MegaWizard Plug-In from SOPC Builder

To create a new SOPC Builder system and launch the MegaWizard Plug-In from SOPC Builder, follow these steps:

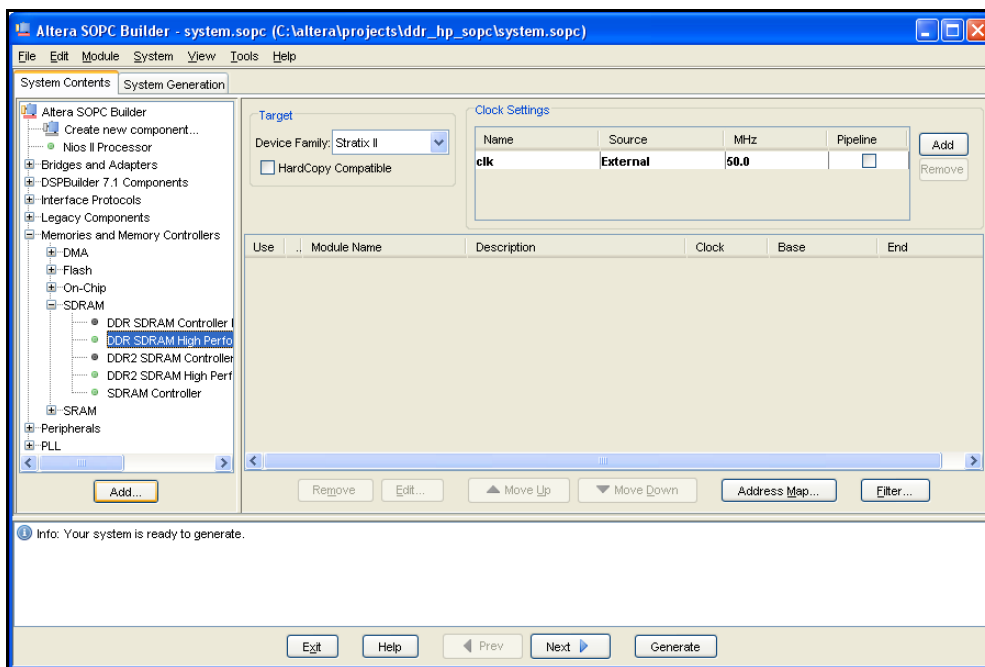
- On the tools menu click **SOPC Builder**.
- Enter a **System Name**.



The system name must not be the same as the Quartus II project name (and therefore the top-level design entity name).

3. Expand **Memories and Memory Controllers** and expand **SDRAM**.
4. Click **DDR SDRAM High Performance Controller** and click **Add** (see [Figure 2-5 on page 2-8](#)).

Figure 2-5. Add a DDR SDRAM High-Performance Controller in SOPC Builder



5. Enter the PLL reference clock frequency.



For all Nios II development boards, enter 50 MHz for the PLL reference clock frequency (see [Figure 2-6 on page 2-9](#)).

Figure 2–6. Parameterize

DDR SDRAM High Performance Controller - altmemddr

DDR SDRAM High Performance Controller
Version 7.1

About Documentation

Parameter Settings

Memory Settings > PHY Settings > Controller Settings >

General Settings

Device family: Stratix II

Speed grade: 4

PLL reference clock frequency: 50 MHz (20000 ps)

Memory clock frequency: 200 MHz (5000 ps)

Local interface clock frequency: Half (100.0 MHz)

Local interface width: 32 bits

Show in "Memory Presets" List

Parameter	Value
Memory vendor	(All)
Memory format	(All)
Maximum memory frequency	(All)

Show All

Memory Presets

Presets

- JEDEC DDR-400 128Mb x8
- JEDEC DDR-400 128Mb x4
- JEDEC DDR-400 256Mb x8
- JEDEC DDR-400 256Mb x4
- JEDEC DDR-400 512Mb x8
- JEDEC DDR-400 512Mb x4

Load Preset...

Selected memory preset: JEDEC DDR-400 128Mb x8

Modify parameters...

Description: DDR-SDRAM, 200MHz, 16MB, 8 bits wide, Discrete Device, CAS 3.0, x 1 CS

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

Cancel < Back Next > Finish

6. Select your memory preset (for example **MT46V16M16** for the Nios II Development Board, Stratix II RoHS Edition).
7. Select either **Half** of **Full** for the local clock rate frequency. This parameter affects the width and frequency of the local Avalon-MM interface. If you select half rate, ensure you reduce the local data width to 32 by clicking **Modify Parameters** and selecting 8 bits for the **Memory DQ width**.



For more information on the parameters, refer to the *ALTMEMPHY Megafunction User Guide*.

8. Select the other parameters for the MegaCore function (see [“Parameterize” on page 2–11](#)).
9. Click **Finish**.
10. In SOPC Builder, select **Nios II processor** and click **Add**.
11. For the **Reset Vector** and the **Exception Vector** select **altmemddr**.
12. Change the **Reset Vector Offset** to **0x20** and the **Exception Vector Offset** to **0x40**.



The ALTMEMPHY megafunction performs memory interface calibration every time it is reset and in doing so writes to addresses `0x0` to `0x1f`. If you want your memory contents to remain intact through a system reset, you should avoid using the memory addresses below `0x20`. This step is not necessary, if you reload your SDRAM memory contents from flash every time you reset.

13. Click **Finish**.
14. In SOPC Builder expand **Interface Protocols** and expand **Serial**.
15. Select **JTAG UART** and click **Add**.
16. 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**.

17. For this example system, ensure all the other modules are clocked on the `altmemddr_sysclk`, to avoid any unnecessary clock-domain crossing logic.
18. Click **Generate**.
19. You must now edit the `altmemddr_example_top.v` file to replace the example driver and the DDR SDRAM High-Performance controller and instantiate your SOPC Builder-generated system.
20. Now go to [“Compile the Example Design” on page 2–19](#).

Parameterize

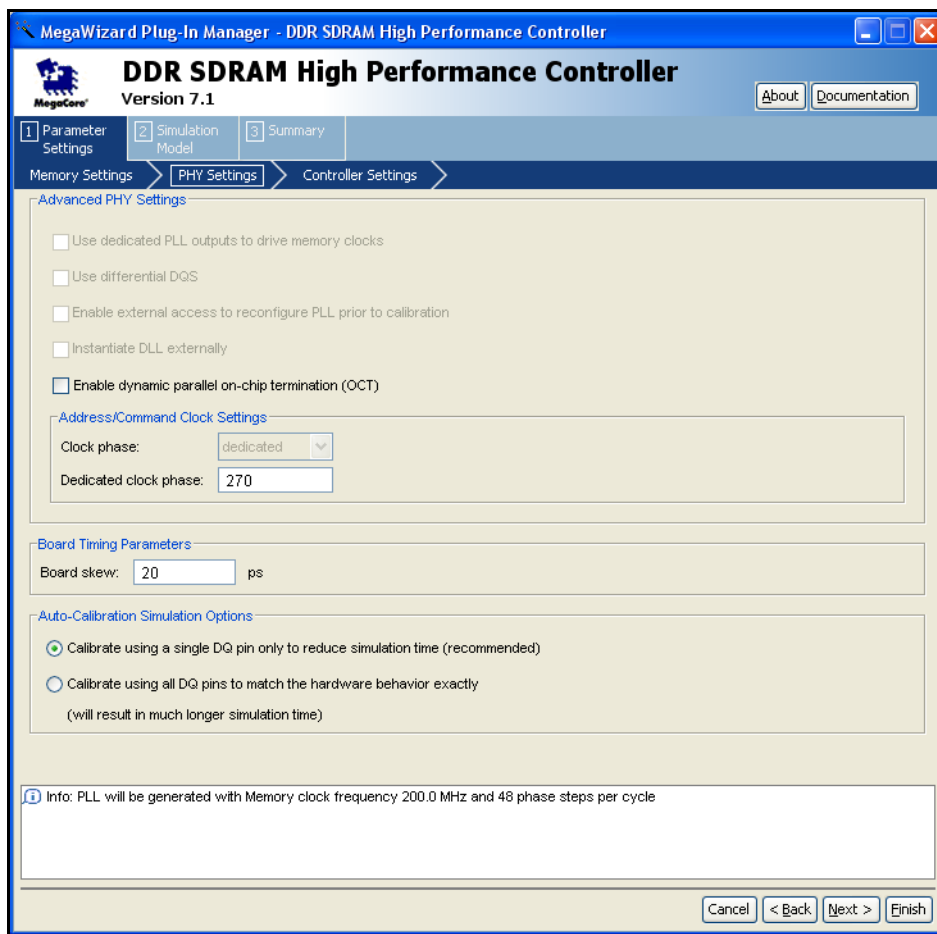
To parameterize your MegaCore function, follow these steps:



For more information on the ALTMEMPHY megafunction, refer to the *ALTMEMPHY Megafunction User Guide*.

1. Select your memory settings.
2. Click the **PHY Settings** tab (see [Figure 2–7 on page 2–11](#)).

Figure 2–7. PHY Settings



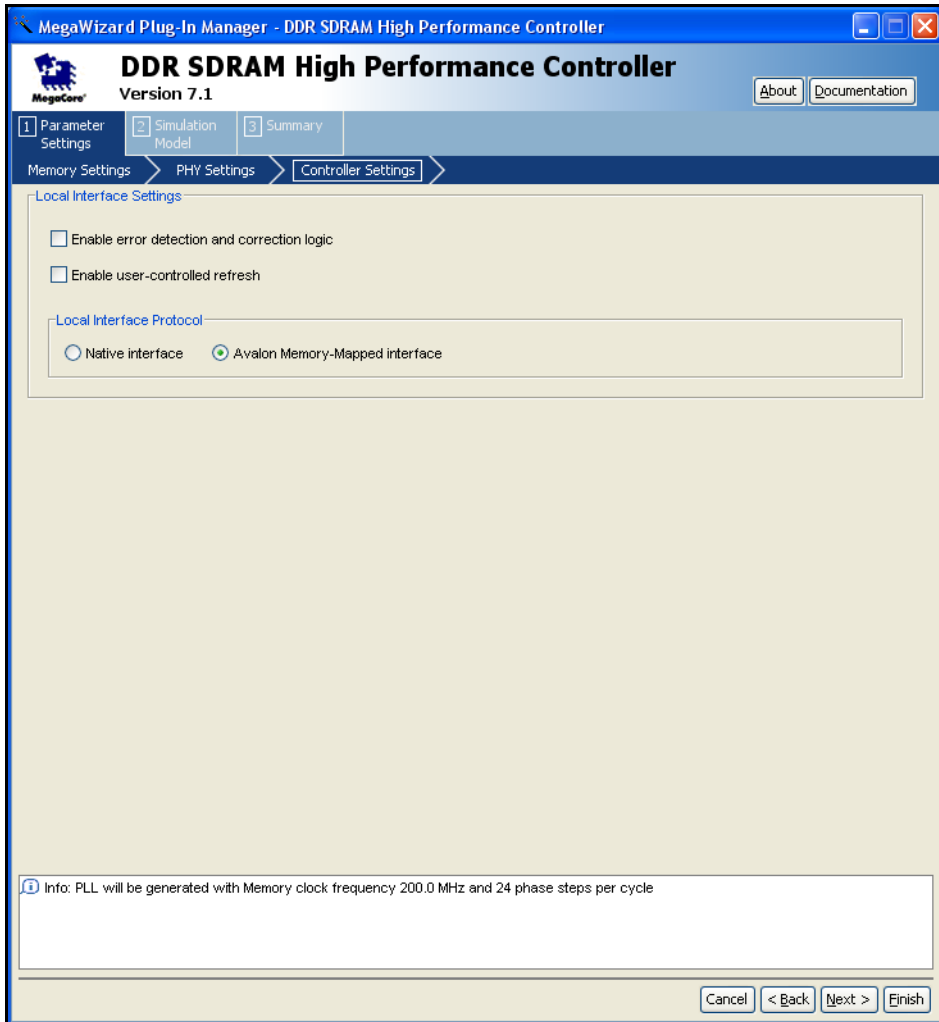
3. Select your PHY settings.

- Click the **Controller Settings** tab (see [Figure 2-8](#)).



For more information on controller settings, see [“Controller Settings” on page 3-25](#).

Figure 2-8. Controller Parameters



- Choose your local interface settings.



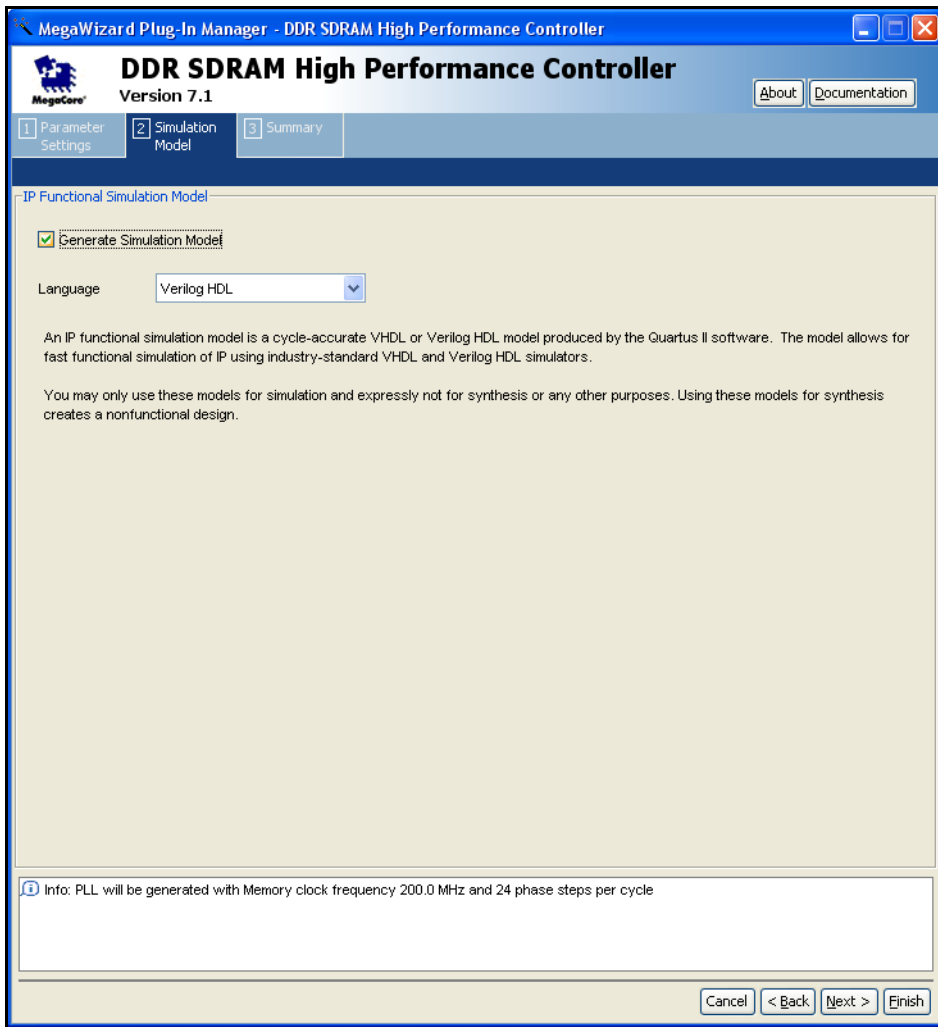
For more information on the ECC, see [“ECC” on page 3-4](#).

6. Select **Native** or **Avalon Memory-Mapped** interface. The Avalon® Memory-Mapped (MM) interface allows you to easily connect to other Avalon-MM peripherals.



For more information on the Avalon-MM interface, see the *Avalon Memory-Mapped Interface Specification*.

7. Click **Next** (or the **Simulation Model** tab) to display the simulation setup page (see [Figure 2-9](#)).

Figure 2–9. Simulation Model

Set Up Simulation

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software. The model allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

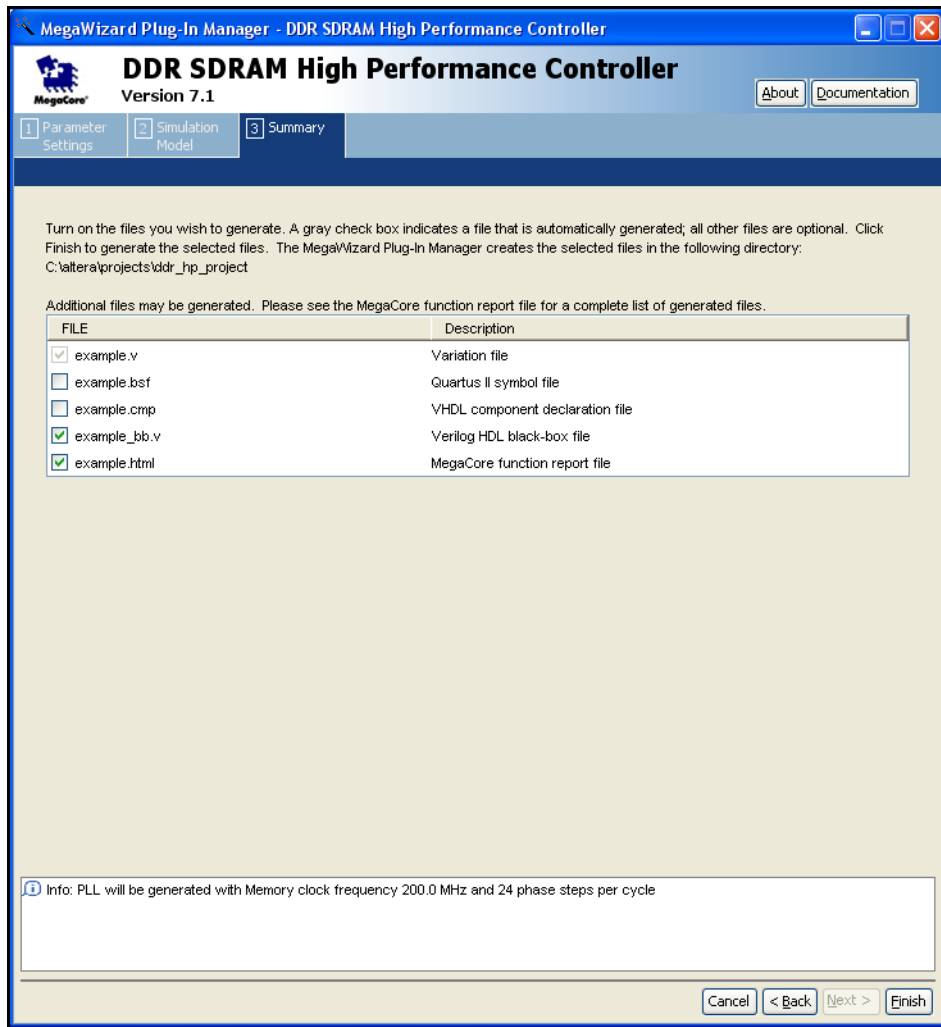


You may only use these models for simulation and expressly not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Turn on **Generate Simulation Model**.
2. Choose the language from the **Language** list.
3. Click **Next** (or the **Summary** tab) to display the summary page (see [Figure 2-10](#)).

Figure 2–10. Summary



Generate Files

You can use the check boxes on the **Summary** page to enable or disable the generation of specified files. A gray checkmark indicates a file that is automatically generated; a red checkmark indicates an optional file.

You can click **Back** to display the previous page or click **Parameters Setting**, **Simulation Library** or **Summary**, if you want to change any of the MegaWizard options.

To generate the files, follow these steps:

1. Turn on the files you wish to generate.



At this stage you can still click **Back** or the pages to display any of the other pages in the MegaWizard Plug-In Manager, if you want to change any of the parameters.

2. To generate the specified files and close the MegaWizard Plug-In Manager, click **Finish**.



The generation phase may take several minutes to complete.

Table 2–1 describes the generated files and other files that may be in your project directory. The names and types of files specified in the MegaWizard Plug-In Manager report vary based on whether you created your design with VHDL or Verilog HDL.

Table 2–1. Generated Files <i>Note (1)</i>	
Filename	Description
<code><variation name>.bsf</code>	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<code><variation name>.html</code>	MegaCore function report file.
<code><variation name>.vo</code> or <code>.vho</code>	VHDL or Verilog HDL IP functional simulation model.
<code><variation name>.vhd</code> , or <code>.v</code>	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
<code><variation name>_bb.v</code>	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<code><variation name>_example_driver.vhd</code> , or <code>.v</code>	Example driver.
<code><variation name>_example_top.vhd</code> , or <code>.v</code>	Example design.

Notes to Table 2–1:

(1) `<variation name>` is the variation name.

3. After you review the generation report, click **Exit** to close the MegaWizard Plug-In Manager.
4. Set the `<variation name>_example_top.v` or `.vhd` file to be the project top-level design file.

You have finished the walkthrough. Now, simulate the example design (see “[Simulate the Example Design](#)” on page 2–18), edit the PLL(s), and compile (see “[Compile the Example Design](#)” on page 2–19).

Simulate the Example Design

This section describes the following simulation techniques:

- [Simulate with IP Functional Simulation Models](#)
- [Simulating in Third-Party Simulation Tools Using NativeLink](#)

Simulate with IP Functional Simulation Models

You can simulate the example design with the MegaWizard Plug-In Manager-generated IP functional simulation models. The MegaWizard Plug-In Manager generates a VHDL or Verilog HDL testbench for your example design, which is in the **testbench** directory in your project directory.



For more information on the testbench, see “[Example Design](#)” on page 3–9.

You can use the IP functional simulation model with any Altera-supported VHDL or Verilog HDL simulator.

Simulating in Third-Party Simulation Tools Using NativeLink

You can perform a simulation in a third-party simulation tool from within the Quartus II software, using NativeLink.



For more information on NativeLink, refer to the *Simulating Altera IP Using NativeLink* chapter in volume 3 of the *Quartus II Handbook*.

To set up simulation in the Quartus II software using NativeLink, follow these steps:

1. Create a custom variation with an IP functional simulation model.
2. Set the generated top-level file `<variation name>_example_top` to be the project top-level file.

3. Obtain and copy a memory model to a suitable location, for example, the testbench directory.



Before running the simulation you may also need to edit the testbench to match the chosen memory model.

4. Check that the absolute path to your third-party simulator executable is set. On the Tools menu click **Options** and select **EDA Tools Options**.
5. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
6. On the Assignments menu click **Settings**, expand **EDA Tool Settings** and select **Simulation**. Select a simulator under **Tool Name** and in **NativeLink Settings**, select **Compile Test Bench** and click **Test Benches**.
7. Click **New**.
8. Enter a name for the **Test bench name**.
9. Enter the name of the automatically generated testbench, *<variation name>_example_top_tb*, in **Test bench entity**.
10. Enter the name of the top-level instance in **Instance**.
11. Change **Run for** to 500 μ s.
12. Add the testbench files. In the **File name** field browse to the location of the memory model and the testbench, click **OK** and click **Add**.
13. Click **OK**.
14. Click **OK**.
15. On the Tools menu point to **EDA Simulation Tool** and click **Run EDA RTL Simulation**.

Compile the Example Design

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

1. Enable TimeQuest.

- a. On the Assignments menu click **Settings**, expand **Timing Analysis Settings**, and select **Use TimeQuest Timing Analyzer during compilation** and click **OK**.
 - b. Add the Synopsys design constraints file, *<variation name>_phy_dds_timing.sdc*, to your project. On the Project menu click **Add/Remove Files in Project** and browse to the file.
2. Add pin I/O standard assignments:



The I/O standard pin assignment script is not run automatically. As a result, bidirectional pins have the wrong I/O standard assigned. Ultimately, the Quartus II fitter fails. Therefore, you must run the I/O standard assignment script manually before running the Quartus II fitter.

- a. For Stratix III devices you must run *<variation name>_pin_assignments.tcl*.
 - b. For all other devices, run *<variation name>_pin_assignments.tcl*, or follow these steps:
 - On the Assignments menu, click **Pin Planner**. In the Quartus II Pin Planner, edit your top-level design to add a prefix to all DDR or DDR2 signal names. For example, change **mem_addr** to **core1_mem_addr**.
 - On the Assignments menu click **Pins**. Right-click in the window and click **Create/Import Megafunction**. Select **Import an existing custom megafunction** and navigate to *<variation name>.ppf*.
 - Type the prefix that you added to your top-level DDR or DDR2 signal names into the **Instance name** box and click **OK**.
3. Set the top-level entity to the example project.
 - a. On the File menu click **Open**.
 - b. Browse to *<variation name>_example_top* and click **Open**.
 - c. On the Project menu click **Set as top-level entity**.
 4. On the Processing menu, point to **Start** and click **Start Analysis and Synthesis**.
 5. Assign the DQ and DQS pin groups.

- a. For Stratix III devices only, add the DQ group assignments, to relate the DQ and DQS pin groups together for the Quartus II fitter to place them correctly, by running `<variation name>_assign_dq_groups.tcl`.
- b. For all other device families, 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.

or

- c. Manually specify all DQ and DQS pins to align your project with your PCB requirements.

or

- d. Manually specify all project pin locations to align your project with your PCB requirements.



When you are assigning pins, ensure the IO standard is set to SSTL18-II and not LVTTTL, for example for the clock source, the reset, and the address and command signals. Also select which bank or side of the device you want the Quartus II software to place them in.

6. Set the output pin loading for all memory interface pins.
7. Select your required IO driver strength (derived from simulation) to ensure that you correctly drive each signal and do not suffer from overshoot or undershoot.
8. On the Processing menu click **Start Compilation**, to compile the design.
9. *Optional.* Run the report timing to get detailed a DDR SDRAM interface timing report. Run `<variation name>_report_timing.tcl`:
 - a. On the Tools menu click **Tcl scripts**.

or

 - b. On the Tools menu click **TimeQuest Timing Analyzer**. On the Script menu click **Run Tcl Script**.



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

Program a Device

After you have compiled the example design, you can perform gate-level simulation (see [“Simulate the Example Design” on page 2–18](#)) or program your targeted Altera device to verify the example design in hardware.

With Altera's free OpenCore Plus evaluation feature, you can evaluate the DDR or DDR2 SDRAM high-performance controller MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file.



For more information on OpenCore Plus hardware evaluation using the DDR or DDR2 SDRAM high-performance controller MegaCore function, see [“OpenCore Plus Evaluation” on page 1–3](#), [“OpenCore Plus Time-Out Behavior” on page 3–8](#), and *Application Note 320: OpenCore Plus Evaluation of Megafunctions*.

Implement Your Design

To implement your design based on the example design, replace the example driver in the example design with your own logic.

Set Up Licensing

You need to purchase 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.

After you purchase a license for DDR or DDR2 SDRAM high-performance controller MegaCore function, 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 Altera representative.

Functional Description

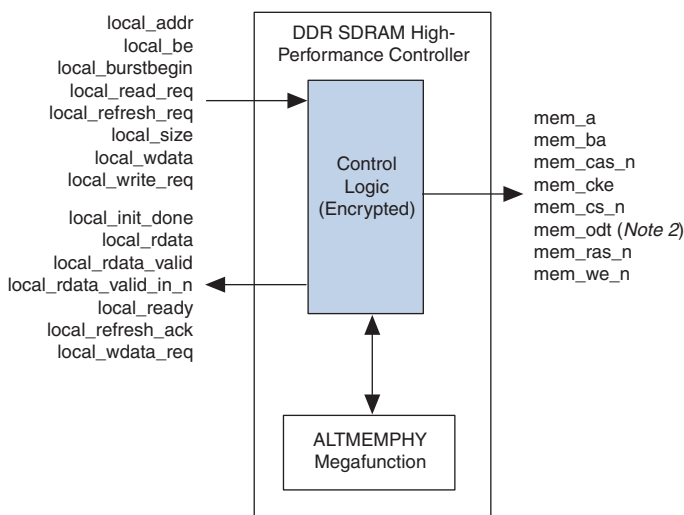
The DDR and DDR2 SDRAM high-performance controllers instantiate encrypted control logic and the ALTMEMPHY megafunction.



For more information on the ALTMEMPHY megafunction, refer to the *ALTMEMPHY Megafunction User Guide*.

Figure 3–1 shows a block diagram of the DDR & DDR2 SDRAM high-performance controller.

Figure 3–1. DDR & DDR2 SDRAM High-Performance Controller Block Diagram



Notes to Figure 3–1:

- (1) DDR2 SDRAM high-performance controller only.

Control Logic

Bus commands control SDRAM devices using combinations of the `mem_ras_n`, `mem_cas_n`, and `mem_we_n` signals. For example, on a clock cycle where all three signals are high, the associated command is a

no operation (NOP). A NOP command is also indicated when the chip select signal is not asserted. Table 3–1 shows the standard SDRAM bus commands.

Table 3–1. Bus Commands				
Command	Acronym	ras_n	cas_n	we_n
No operation	NOP	High	High	High
Active	ACT	Low	High	High
Read	RD	High	Low	High
Write	WR	High	Low	Low
Burst terminate	BT	High	High	Low
Precharge	PCH	Low	High	Low
Auto refresh	ARF	Low	Low	High
Load mode register	LMR	Low	Low	Low

The DDR and DDR2 SDRAM high-performance controllers must open SDRAM banks before they access addresses in that bank. The row and bank to be opened are registered at the same time as the active (ACT) command. The DDR and DDR2 SDRAM high-performance controllers close the bank and open it again if they need to access a different row. The precharge (PCH) command closes a bank.

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

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



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

Latency

Table 3–2 shows some approximate values for latency, all measured in the local interface clock domain.

Table 3–2. Latency		
Rate	Operation	Local Latency (Clock Cycles)
Half	Read	17
	Write	11
Full	Read	19
	Write	10

The read latency measures the time from a read request on the local interface until the read data is presented at the local interface. The values assume a CAS latency of 3 and an open bank. The write latency is measured from a write request on the local interface until the data is written to the memory.



The read latency for Cyclone III devices is likely to be less than for Stratix II and Stratix III devices, because there is no resynchronization. The 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.

The following equations give approximate calculations for latency:

$$\text{Half rate latency} = [\text{cycles in PHY} + (\text{CAS latency}/2) + \text{controller}] \times 1/\text{user side frequency}$$

$$\text{Full rate latency} = [\text{cycles in PHY} + \text{CAS latency} + \text{controller}] \times 1/\text{user side frequency}$$

For example, for a half-rate controller running at 333-MHz memory clock frequency and 167-MHz local local clock frequency:

$$\text{Latency} = 17 \times 1/167 \text{ MHz} = 102 \text{ ns}$$

For a full rate controller running at 333-MHz memory clock frequency and 333-MHz local local clock frequency:

$$\text{Latency} = 19 \times 1/333 \text{ MHz} = 57 \text{ ns}$$

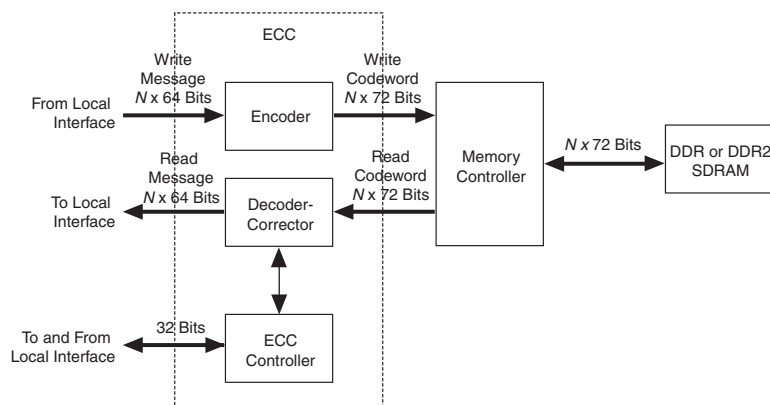
ECC

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

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

Figure 4 shows the ECC block diagram.

Figure 4. ECC Block Diagram



The ECC comprises the following blocks:

- The encoder—encodes the 64-bit message to a 72-bit codeword
- The decoder-corrector—decodes and corrects the 72-bit codeword if possible

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



For more information on the ECC registers, see [Appendix A, ECC Register Description](#).

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

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

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

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

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

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

Interrupts

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

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

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

Partial Writes

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

- Stalls further read or write commands from the Avalon-MM interface when it receives a partial write condition.
- Simultaneously sends a self-generated read command, for the partial write address, to the memory controller.
- Upon receiving a return data from the memory controller for the particular address, the ECC decodes the data, checks for errors, and then sends it to the ECC controller.
- The ECC controller merges the corrected or correct dataword with the incoming information.
- Sends the updated dataword to the encoder for encoding and then sends to the memory controller with a write command.
- Releases the stall of commands from the Avalon-MM interface, which allows it to receive new commands.

The following corner cases can occur:

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

Partial Bursts

Some DIMMs do not have the DM pins and so do not support partial bursts. A minimum of four words have to be written to the memory at the same time. In cases of partial burst write, the ECC offers a mechanism similar to the partial write.

In cases of partial bursts, the write data from the native interface is stored in a 64-bit wide FIFO buffer of maximum burst size depth, while in parallel a read command of the corresponding addresses is sent to the DIMM. Further commands from native interface are stalled until the current burst is read, modified, and written back to the memory controller.

ECC Latency

Using the ECC results in the following latency changes.

Burst Length 1

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

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

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

Table 3–3. Burst Lengths & Rates		
Local Burst Length	Rate	Memory Burst Length
1	Half	4
2	Full	4

Burst Length 2

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

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

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

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 timeout is 1 hour; the tethered timeout value is indefinite.

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



For more information on OpenCore Plus hardware evaluation, see [“OpenCore Plus Evaluation” on page 1–3](#) and *Application Note 320: OpenCore Plus Evaluation of Megafunctions*.

Example Design

The MegaWizard® Plug-In Manager creates an example design that shows you how to instantiate and connect up the DDR or DDR2 SDRAM high-performance controller. The example design consists of the DDR or DDR2 SDRAM high-performance controller, some driver logic to issue read and write requests to the controller, a PLL to create the necessary clocks and a DLL (Stratix series only). The example design is a working system that can be compiled and used for both static timing checks and board tests.

Figure 3–1 shows the testbench and the example design.

Figure 3–1. Testbench & Example Design

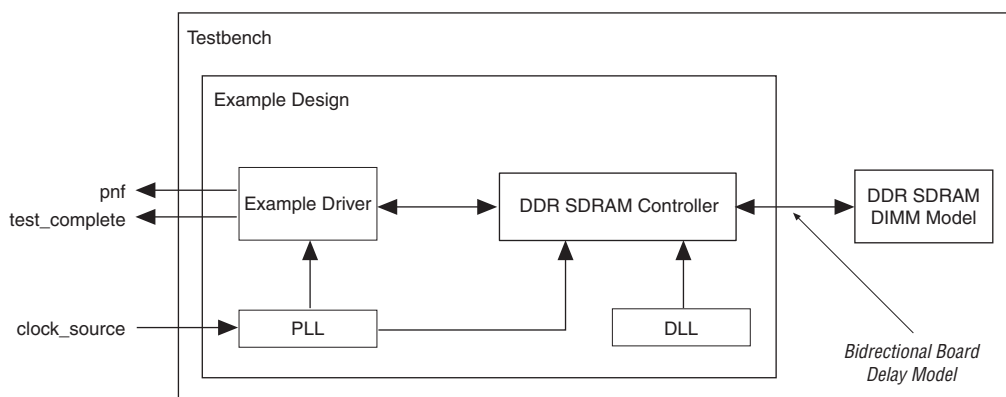


Table 3–4 describes the files that are associated with the example design and the testbench.

Table 3–4. Example Design & Testbench Files

Filename	Description
<variation name>_example_top_tb.v or .vhd	Testbench for the example design.
<variation name>_example_top.v or .vhd	Example design.
<variation name>_phy_alt_mem_phy_pll_<device>.v or .vhd (1)	Example PLL.
<variation name>_example_driver.v or .vhd	Example driver.

Table 3–4. Example Design & Testbench Files

Filename	Description
<variation name> .v or .vhd	Top-level description of the custom MegaCore function.

Notes to Table 3–4:

(1) <device> is the short-hand name of device, for example `sii` for Stratix II devices.

The example driver is a self-checking test generator for the DDR or DDR2 SDRAM high-performance controller. It uses a state machine to write data patterns to a range of column addresses, within a range of row addresses in all memory banks. It then reads back the data from the same locations, and checks that the data matches. The pass not fail (`pnf`) output transitions low if any read data fails the comparison. There is also a `pnf_per_byte` output, which shows the comparison on a per byte basis. The `test_complete` output transitions high for a clock cycle at the end of the write or read test sequence. After this transition the test restarts from the beginning.

The data patterns used are generated using an 8-bit LFSR per byte, with each LFSR having a different initialization seed.

The testbench instantiates a placeholder for the DDR or DDR2 SDRAM model, a reference clock for the PLL. When `test_complete` is detected high, a test finished message is printed out, which shows whether the test has passed.



Altera does not provide a memory simulation model. You must obtain one from your memory vendor.



For more details on how to run the simulation script, see “[Simulate the Example Design](#)” on page 2–18.

Interfaces & Signals

This section describes the following topics:

- “Interface Description” on page 3–10
- “Signals” on page 3–22

Interface Description

This section describes the following local-side interface requests:

- “Writes” on page 3–11
- “Reads” on page 3–13
- “Read-Write-Read-Write” on page 3–15
- “Read-Write-Read-Write” on page 3–15

- “DDR SDRAM Initialization Timing” on page 3–18
- “DDR2 SDRAM Initialization Timing” on page 3–20



These interface requests are for the native interface. For the Avalon™ Memory-Mapped(Avalon-MM) interface see the *Avalon Memory-Mapped Interface Specification*.



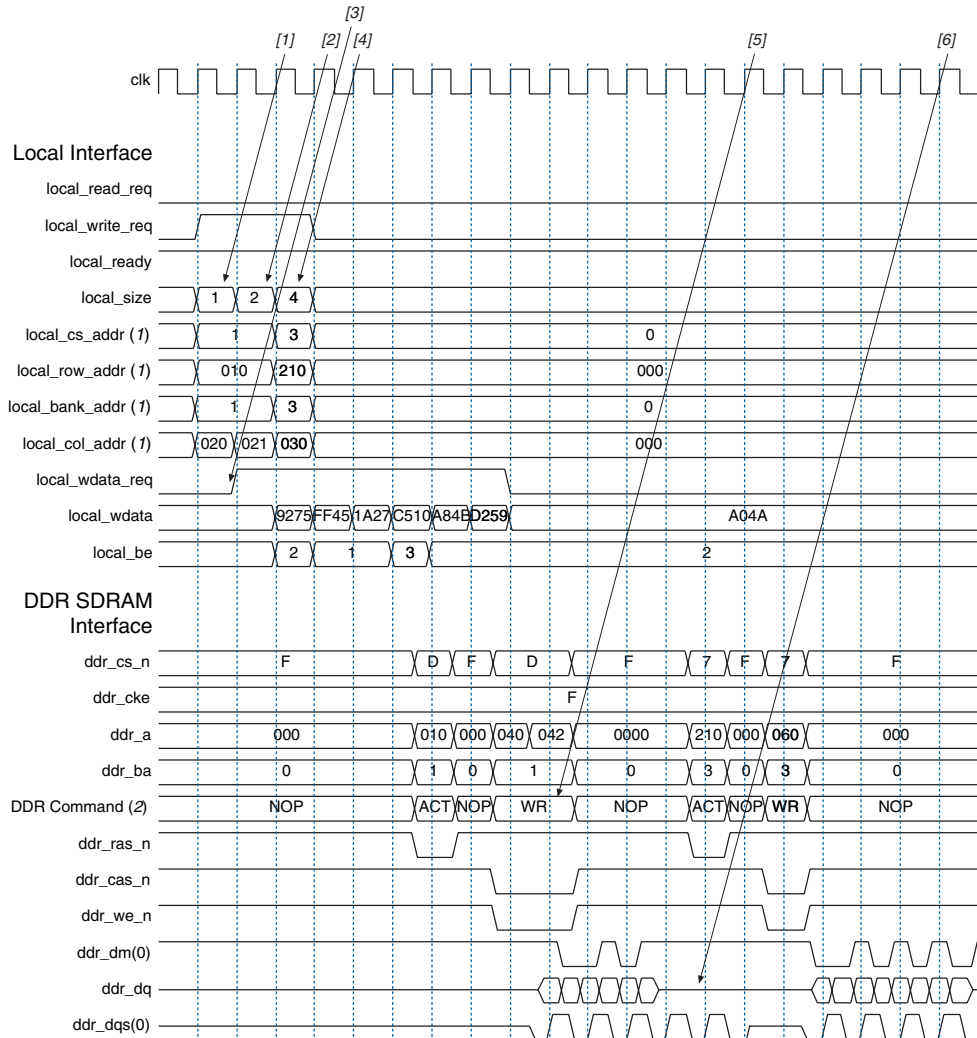
All figures show full rate controllers.

Writes

Figure 3–2 on page 3–12 shows three write requests of different sizes, the first two to sequential addresses and the third to a new row and bank. The controller allows you to use any burst length up to the maximum burst length set on the memory device. For example, if you select burst length of 8 for your DDR SDRAM memory, the controller allows bursts of length 1, 2, 3, and 4 (2, 4, 6, and 8 on the DDR SDRAM side).



The concept is similar for DDR2 SDRAM although only burst lengths 1 and 2 (2 and 4 on the DDR2 SDRAM side) are available.

Figure 3–2. Writes

Notes to Figure 3–2:

- (1) The `local_cs_addr`, `local_row_addr`, `local_bank_addr`, and `local_col_addr` signals are a representation of the `local_addr` signal.
- (2) DDR Command shows the command that the command signals (`mem_ras_n`, `mem_cas_n` and `mem_we_n`) are issuing.

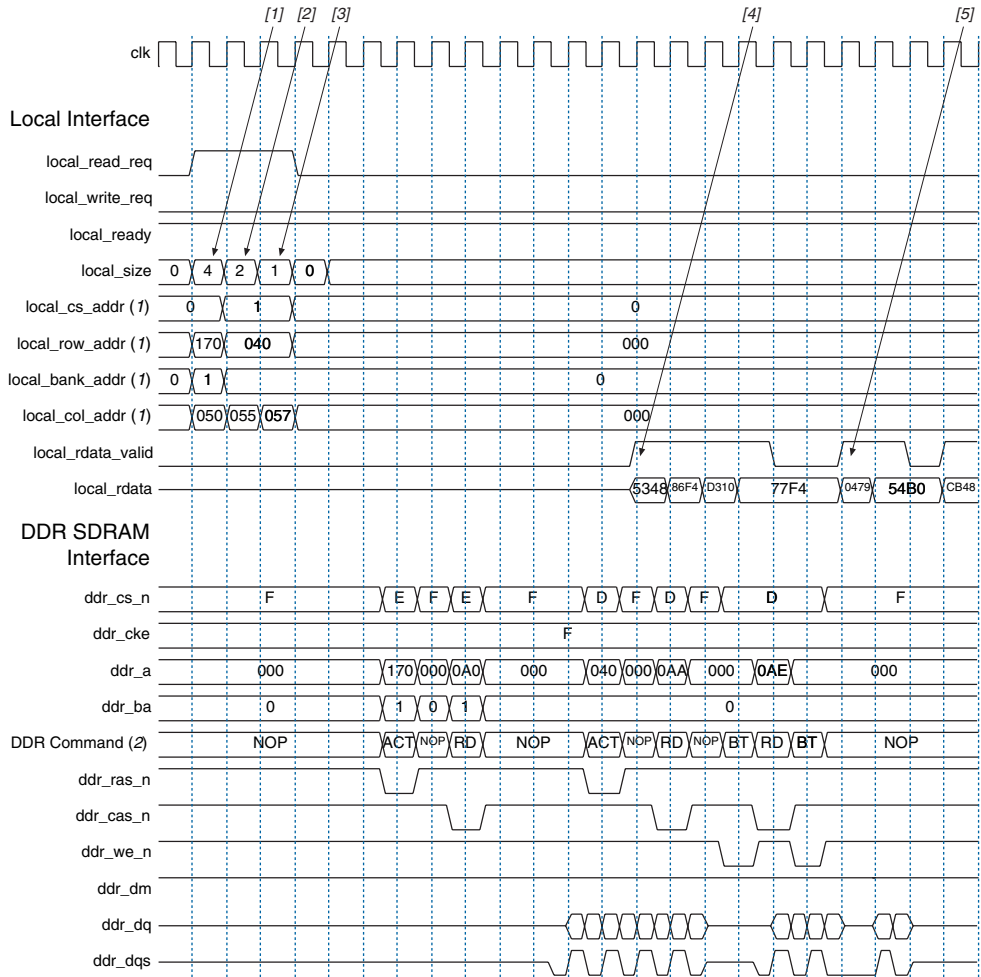
1. The user logic requests the first write, by asserting the `local_write_req` signal, and the size and address for this write. In this example, the request is a burst of length 1 (2 on the DDR SDRAM side) to chip select 1. The `local_ready` signal is asserted, which indicates that the controller has 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.
2. The user logic requests a second write to a sequential address, this time of size 2 (4 on the DDR SDRAM side). The `local_ready` signal remains asserted, which indicates that the controller has accepted the request.
3. The controller requests the write data and byte enables for the first write from the user logic. The write data and byte enables must be presented in the clock cycle after the request. In this example, the controller also continues to request write data for the subsequent writes. The user logic must be able to supply the write data for the entire burst when it requests a write.
4. The user logic requests the third write to a different chip select. The controller is able to buffer up to four requests so the `local_ready` signal stays high and the request is accepted.
5. When it has issued the necessary bank activation command, the controller issues the first two write requests sequentially to the memory device.
6. Even though no data is being written to memory, the `mem_dqs` signal must continue toggling for the entire length of the memory device's burst length (8 in this example).

Reads

Figure 3–3 on page 3–14 shows three read requests of different sizes. The controller allows you to use any burst length up to the maximum burst length set on the memory device. For example, if you select burst length of 8 for your DDR SDRAM memory, the controller allows bursts of length 1, 2, 3, and 4 (2, 4, 6, and 8 on the DDR SDRAM side).



The concept is similar for DDR2 SDRAM although only burst lengths 1 and 2 (2 and 4 on the DDR2 SDRAM side) are available.

Figure 3–3. Reads**Notes to Figure 3–3:**

- (1) The `local_cs_addr`, `local_row_addr`, `local_bank_addr`, and `local_col_addr` signals are a representation of the `local_addr` signal.
- (2) DDR Command shows the command that the command signals are issuing.

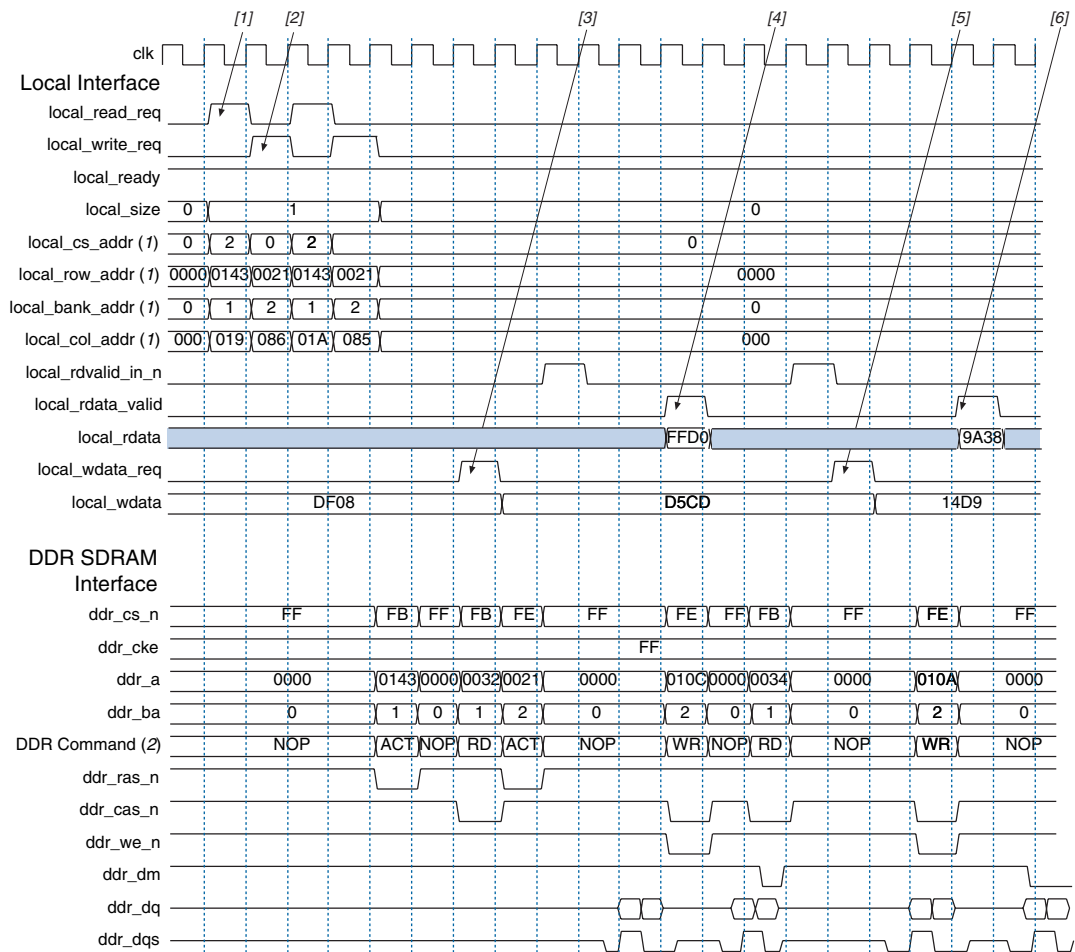
1. The user logic requests the first read by asserting the `local_read_req` signal, and the size and address for this read. In this example, the request is a burst of length 4 (8 on the DDR SDRAM side). The `local_ready` signal is asserted, which

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 read request, size, and address signals asserted.

2. The user logic requests a second read to a different address, this time of size 2 (4 on the DDR SDRAM side). The `local_ready` signal remains asserted, which indicates that the controller has accepted the request.
3. The user logic requests a third read to a different address, this time of size 1 (2 on the DDR SDRAM side). The `local_ready` signal remains asserted, which indicates that the controller has accepted the request.
4. The controller returns the read data for the first request by asserting the `local_rdata_valid` signal. The exact number of clock cycles between the controller accepting the request and returning the data depends on the number of other requests pending in the controller, the state the memory is in, and the timing requirements of the memory (e.g., the CAS latency).
5. The controller returns the read data for the subsequent read requests.

Read-Write-Read-Write

Figure 3–4 on page 3–16 shows a sequence of interleaved reads and writes.

Figure 3–4. Read-Write-Read-Write

Notes to Figure 3–4:

- (1) The local_cs_addr, local_row_addr, local_bank_addr, and local_col_addr signals are a representation of the local_addr signal.
- (2) DDR Command shows the command that the command signals are issuing.

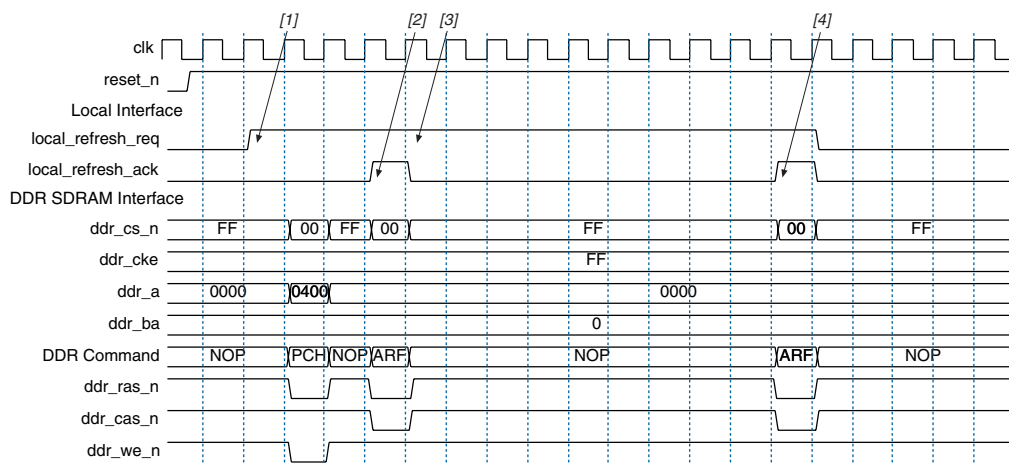
1. The user logic requests a read request by asserting the local_read_req signal along with the size and address for that read. Because the local_ready signal is high, that request can be considered accepted.
2. The user logic requests a write, a read, and another write request, which are accepted.

3. The controller asserts the write data request signal to ask the user logic to present valid write data and byte enables on the next clock edge.
4. The read data from the first read request is returned and marked as valid by the read data valid signal.
5. The controller again asserts the write data request for the second write request.
6. The read data from the second read request is returned.

User Refresh Control

Figure 3–5 shows the user refresh control interface. This feature allows you to control when the controller issues refreshes to the memory. This feature allows better control of worst case latency and allows refreshes to be issued in bursts to take advantage of idle periods.

Figure 3–5. User Refresh Control



Note to Figure 3–5:

- (1) DDR Command shows the command that the command signals are issuing.

1. The user logic asserts the refresh request signal to indicate to the controller that it should perform a refresh. The state of the read and write requests signal does not matter as the controller gives priority to the refresh request (although it completes any currently active reads or writes).

2. The controller asserts the refresh acknowledge signal to indicate that it has issued a refresh. This signal is still available even if the user refresh control option is not switched on, allowing the user logic to keep track of when the controller is issuing refreshes.
3. The user logic keeps the refresh request signal asserted to indicate that it wishes to perform another refresh request.

The controller again asserts the refresh acknowledge signal to indicate that it has issued a refresh. At this point the user logic deasserts the refresh request signal and the controller continues with the reads and writes in its buffers.

DDR SDRAM Initialization Timing



DDR SDRAM and DDR2 SDRAM initialization timing is different. For DDR2 SDRAM initialization timing, see [“DDR2 SDRAM Initialization Timing” on page 3–20](#).

The DDR SDRAM high-performance controller initializes the SDRAM devices by issuing the following memory command sequence:

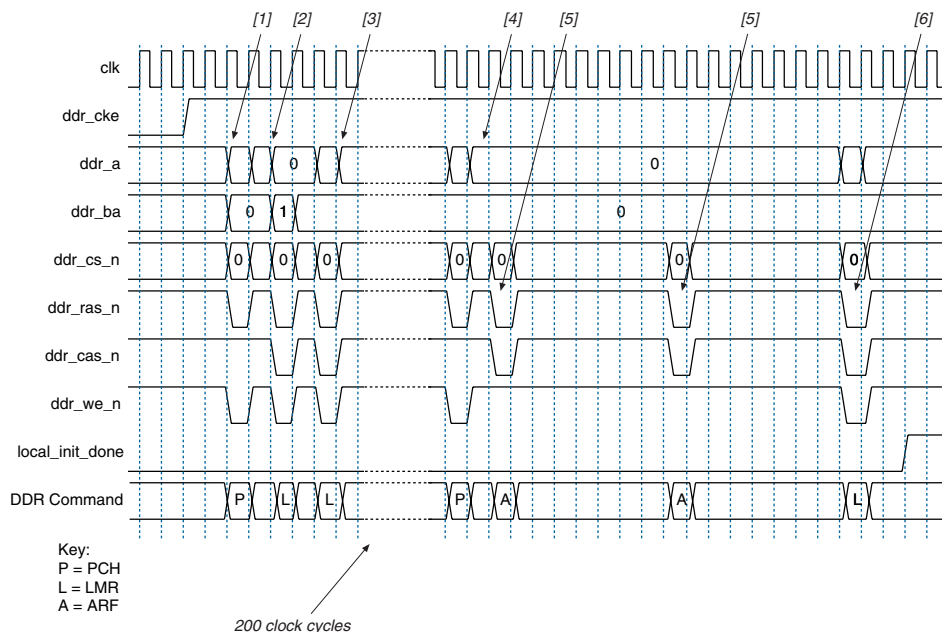
- NOP (for 200 μ s, programmable)
- PCH
- Extended LMR (ELMR)
- LMR
- NOP (for 200 clock cycles, fixed)
- PCH
- ARF
- ARF
- LMR

After issuing the final LMR command, the memory controller hands over control of the memory to the ALTMEMPHY megafunction to allow it to carry out its calibration process.



For more information, refer to the *ALTMEMPHY Megafunction User Guide*.

[Figure 3–6 on page 3–19](#) shows a typical initialization timing sequence, which is described below. The length of time between the reset and the first PCH command should be 200 μ s. This time can be reduced for simulation testing by setting the start-up timer parameter in the MegaWizard Plug-In Manager.

Figure 3–6. DDR SDRAM Device Initialization Timing

1. A PCH command is sent to all banks by setting the precharge pin, the address bit a [10], or a [8] high.
2. An ELMR command is issued to enable the internal delay-locked loop (DLL) in the memory devices. An ELMR command is an LMR command with the bank address bits set to address the extended mode register.
3. An LMR command sets the operating parameters of the memory such as CAS latency and burst length. This LMR command is also used to reset the internal memory device DLL. The DDR SDRAM high-performance controller allows 200 clock cycles to elapse after a DLL reset and before it issues the next command to the memory.
4. A further PCH command places all the banks in their idle state.
5. Two ARF commands must follow the PCH command.
6. The final LMR command programs the operating parameters without resetting the DLL.

7. After issuing the final LMR command, the memory controller hands over control of the memory to the ALTMEMPHY megafunction to allow it to carry out its calibration process.



For more information, refer to the *ALTMEMPHY Megafunction User Guide*.

When ALTMEMPHY has finished calibrating, the memory controller asserts the `local_init_done` signal, which shows that it has initialized the memory devices.

DDR2 SDRAM Initialization Timing

The DDR2 SDRAM high-performance controller initializes the memory devices by issuing the following command sequence:

- NOP (for 200 μ s, programmable)
- PCH
- ELMR, register 2
- ELMR, register 3
- ELMR, register 1
- LMR
- PCH
- ARF
- ARF
- LMR
- ELMR, register 1
- ELMR, register 1

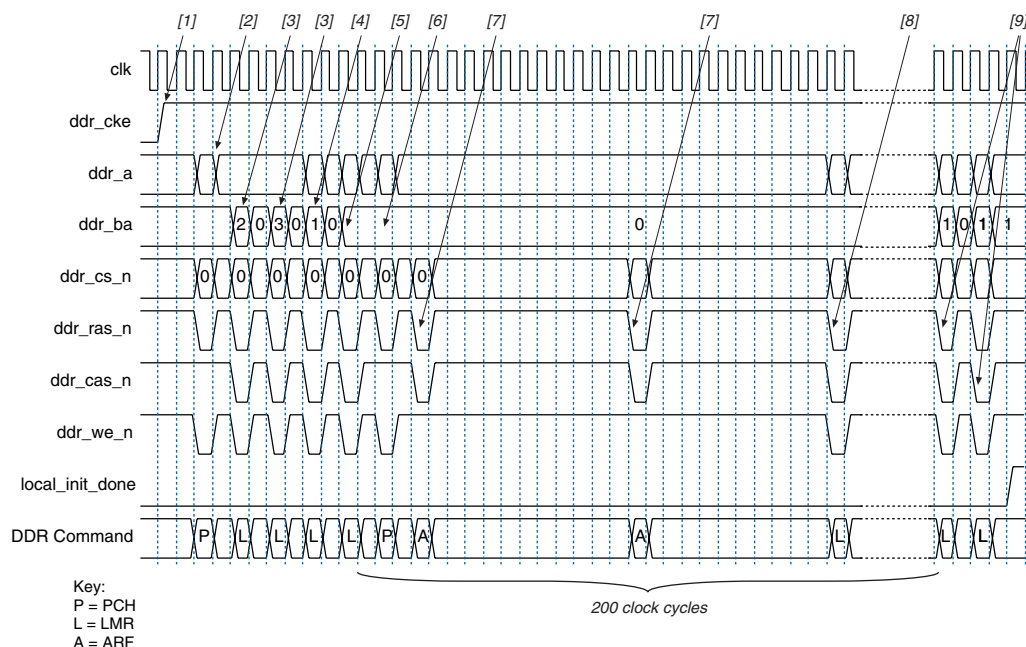
After issuing the final ELMR command, the memory controller hands over control of the memory to the ALTMEMPHY megafunction to allow it to carry out its calibration process.



For more information, refer to the *ALTMEMPHY Megafunction User Guide*.

Figure 3–7 on page 3–21 shows a typical DDR2 SDRAM initialization timing sequence, which is described below. The length of time between the reset and the clock enable signal going high should be 200 μ s. This time can be reduced for simulation testing by setting the start-up timer parameter in the MegaWizard Plug-In Manager.

Figure 3–7. DDR2 SDRAM Device Initialization Timing



1. The clock enable signal (CKE) is asserted 200 μ s after coming out of reset.
2. The controller then waits 400 ns and then issues the first PCH command by setting the precharge pin, the address bit a[10] or a[8] high. The 400 ns is calculated by taking the number of clock cycles calculated by the wizard for the 200 μ s delay and dividing this by 500. If a small initialization time is selected for simulation purposes, this delay is always at least 1 clock cycle.
3. Two ELMR commands are issued to load extend mode registers 2 and 3 with zeros.
4. An ELMR command is issued to extend mode register 1 to enable the internal DLL in the memory devices.
5. An LMR command is issued to set the operating parameters of the memory such as CAS latency and burst length. This LMR command is also used to reset the internal memory device DLL.
6. A further PCH command places all the banks in their idle state.

7. Two ARF commands must follow the PCH command.
8. A final LMR command is issued to program the operating parameters without resetting the DLL.
9. 200 clock cycles after step 5, two ELMR commands are issued to set the memory device off-chip driver (OCD) impedance to the default setting.
10. After issuing the final ELMR command, the memory controller hands over control of the memory to the ALTMEMPHY megafunction to allow it to carry out its calibration process.



For more information, refer to the *ALTMEMPHY Megafunction User Guide*.

When ALTMEMPHY has finished calibrating, the memory controller asserts the `local_init_done` signal, which shows that it has initialized the memory devices.

Signals

Table 3–5 shows the DDR and DDR2 SDRAM high-performance controller local interface signals.

Table 3–5. Local Interface Signals (Part 1 of 3)		
Signal Name	Direction	Description
<code>local_addr[]</code>	Input	Memory address at which the burst should start. The width of this bus is sized using the following equation: For one chip select: $\text{width} = \text{bank bits} + \text{row bits} + \text{column bits} - 1$ For multiple chip selects: $\text{width} = \text{chip bits} + \text{bank bits} + \text{row bits} + \text{column bits} - 1$ The least significant bit (LSB) of the column address on the memory side is ignored, because the local data width is twice that of the memory data bus width.
<code>local_be[]</code>	Input	Byte enable signal, which you use to mask off individual bytes during writes.
<code>local_burstbegin</code>	Input	Avalon burst begin strobe, which indicates the beginning of an Avalon burst. This signal is only available when the local interface is an Avalon-MM interface and the memory burst length is greater than 2. Unlike all other Avalon-MM signals, the burst begin signal does not stay asserted if <code>local_ready</code> is deasserted.

Table 3–5. Local Interface Signals (Part 2 of 3)

Signal Name	Direction	Description
<code>local_read_req</code>	Input	Read request signal.
<code>local_refresh_req</code>	Input	User controlled refresh request. If User Controlled Refresh is turned on, <code>local_refresh_req</code> becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including ganging together multiple refresh commands. Refresh requests take priority over read and write requests unless they are already being processed.
<code>local_size[]</code>	Input	Controls the number of beats in the requested read or write access to memory, encoded as a binary number. The range of values depend on the memory burst length and whether you select full or half rate in the wizard. If you select a memory burst length 4 and half rate, the local burst length is 1 and so <code>local_size</code> should always be driven with 1. If you select a memory burst length 4 and full rate, the local burst length is 2 and you should set the <code>local_size</code> to either 1 or 2 for each read or write request.
<code>local_wdata[]</code>	Input	Write data bus. The width of <code>local_wdata</code> is twice that of the memory data bus.
<code>local_write_req</code>	Input	Write request signal.
<code>local_init_done</code>	Output	Memory initialization complete signal, which is asserted once the controller has completed its initialization of the memory. Read and write requests are still accepted before <code>local_init_done</code> is asserted, however they are not issued to the memory until it is safe to do so.
<code>local_rdata[]</code>	Output	Read data bus. The width of <code>local_rdata</code> is twice that of the memory data bus.
<code>local_rdata_error</code>	Output	Asserted if the current read data has an error.
<code>local_rdata_valid</code>	Output	Read data valid signal. The <code>local_rdata_valid</code> signal indicates that valid data is present on the read data bus. The timing of <code>local_rdata_valid</code> is automatically adjusted to cope with your choice of resynchronization and pipelining options.
<code>local_rdvalid_in_n</code>	Output	An early version of the read data valid signal which appears three cycles before it. Only present with native interfaces.
<code>local_ready</code>	Output	The <code>local_ready</code> signal indicates that the DDR or DDR2 SDRAM high-performance controller is ready to accept request signals. If <code>local_ready</code> is asserted in the clock cycle that a read or write request is asserted, that request has been accepted. The <code>local_ready</code> signal is deasserted to indicate that the DDR or DDR2 SDRAM high-performance controller cannot accept any more requests.

Table 3–5. Local Interface Signals (Part 3 of 3)

Signal Name	Direction	Description
local_refresh_ack	Output	Refresh request acknowledge, which is asserted for one clock cycle every time a refresh is issued. Even if the User Controlled Refresh option is not selected, local_refresh_ack still indicates to the local interface that the controller has just issued a refresh command.
local_wdata_req	Output	Write data request signal, which indicates to the local interface that it should present valid write data on the next clock edge. Not present in Avalon mode.

Table 3–6 shows the DDR and DDR2 SDRAM interface signals.

Table 3–6. DDR & DDR2 SDRAM Interface Signals *Note (1)*

Signal Name	Direction	Description
mem_dq[]	Bidirectional	Memory data bus. This bus is half the width of the local read and write data busses.
mem_dqs[]	Bidirectional	Memory data strobe signal, which writes data into the DDR or DDR2 SDRAM and captures read data into the Altera device.
mem_clk	Output	Clock for the memory device.
mem_clk_n	Output	Inverted clock for the memory device.
mem_a[]	Output	Memory address bus.
mem_ba[]	Output	Memory bank address bus.
mem_cas_n	Output	Memory column address strobe signal.
mem_cke[]	Output	Memory clock enable signals.
mem_cs_n[]	Output	Memory chip select signals.
mem_dm[]	Output	Memory data mask signal, which masks individual bytes during writes.
mem_odt[]	Output	Memory on-die termination control signal (DDR2 SDRAM only).
mem_ras_n	Output	Memory row address strobe signal.
mem_we_n	Output	Memory write enable signal.

Note to Table 3–6:

- (1) You can change the mem_ signal name prefix in the MegaWizard Plug-In.

Table 3–7 shows the ECC controller signals.

Table 3–7. ECC Controller Signals		
Signal Name	Direction	Description
<code>ecc_addr[]</code>	Input	Address for ECC controller.
<code>ecc_be[]</code>	Input	ECC controller byte enable.
<code>ecc_interrupt</code>	Output	Interrupt from ECC controller.
<code>ecc_rdata[]</code>	Output	Return data from ECC controller.
<code>ecc_read_req</code>	Input	Read request for ECC controller.
<code>ecc_wdata[]</code>	Input	ECC controller write data.
<code>ecc_write_req</code>	Input	Write request for ECC controller.

Parameters

The parameters can be set only in the MegaWizard Plug-In Manager (see “[DDR & DDR2 SDRAM High-Performance Controller Walkthrough](#)” on page 2–2).

Table 3–8 shows the controller settings.

Table 3–8. Controller Settings		
Parameter	Range	Description
Enable error correction and detection logic	On or off	Turn on to add an ECC to the design, see “ ECC ” on page 3–4.
Enable user controlled refresh	On or off	Turn on for user control of the refreshes, see “ User Refresh Control ” on page 3–17.
Local Interface Protocol	Native or Avalon Memory-Mapped	Specifies the local side interface between the user logic and the memory controller.

MegaCore Verification

MegaCore verification involves simulation testing. Altera has carried out extensive random, directed tests with functional test coverage using industry-standard Denali models to ensure the functionality of the DDR and DDR2 SDRAM high-performance controller. In addition, Altera has carried out a wide variety of gate-level tests of the DDR and DDR2 SDRAM high-performance controllers to verify the post-compilation functionality of the controllers.



Appendix A. ECC Register Description

This appendix describes the ECC registers and then describes the register bits.

Table A–1 shows the ECC registers.

Table A–1. ECC Registers (Part 1 of 3)					
Name	Address	Size (Bits)	Attribute	Default	Description
Control word specifications	00	32	R/W	0000000F	This register contains all commands for the ECC functioning.
Maximum single-bit error counter threshold	01	32	R/W	00000001	The single-bit error counter increments (when a single-bit error occurs) until the maximum threshold, as defined by this register. When this threshold is crossed, the ECC generates an interrupt.
Maximum double-bit error counter threshold	02	32	R/W	00000001	The double-bit error counter increments (when a double-bit error occurs) until the maximum threshold, as defined by this register. When this threshold is crossed, the ECC generates an interrupt.
Current single-bit error count	03	32	RO	00000000	The single-bit error counter increments (when a single-bit error occurs) until the maximum threshold. You can find the value of the count by reading this status register.
Current double-bit error count	04	32	RO	00000000	The double-bit error counter increments (when a double-bit error occurs) until the maximum threshold. You can find the value of the count by reading this status register.
Last or first single-bit error error address	05	32	RO	00000000	This status register stores the last single-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored.

Table A–1. ECC Registers (Part 2 of 3)

Name	Address	Size (Bits)	Attribute	Default	Description
Last or first double-bit error error address	06	32	RO	00000000	This status register stores the last double-bit error error address. It can be cleared using the control word clear. If bit 10 of the control word is set high, the first occurred address is stored.
Last single-bit error error data	07	32	RO	00000000	This status register stores the last single-bit error error data word. As the data word is an M th multiple of 64, the data word is stored in a $2N$ -deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear.
Last single-bit error syndrome	08	32	RO	00000000	This status register stores the last single-bit error syndrome, which specifies the location of the error bit on a 64-bit data word. As the data word is an M th multiple of 64, the syndrome is stored in a N deep, 8-bit wide FIFO buffer where each syndrome represents errors in every 64-bit part of the data word. The register gets updated with the correct syndrome depending on which part of the data word is shown on the last single-bit error error data register. It can be cleared individually by using the control word clear.
Last double-bit error error data	09	32	RO	00000000	This status register stores the last double-bit error error data word. As the data word is an M th multiple of 64, the data word is stored in a $2N$ deep, 32-bit wide FIFO buffer with the least significant 32-bit sub word stored first. It can be cleared individually by using the control word clear.
Interrupt status register	0A	5	RO	00000000	This status register stores the interrupt status in four fields (see Table A–3). These status bits can be cleared by writing a 1 in the respective locations.
Interrupt mask register	0B	5	WO	00000001	This register stores the interrupt mask in four fields (see Table A–4).

Table A–1. ECC Registers (Part 3 of 3)

Name	Address	Size (Bits)	Attribute	Default	Description
Single-bit error location status register	0C	32	R/W	00000000	This status register stores the occurrence of single-bit error for each 64-bit part of the data word in every bit (see Table A–5). These status bits can be cleared by writing a 1 in the respective locations.
Double-bit error location status register	0D	32	R/W	00000000	This status register stores the occurrence of double-bit error for each 64-bit part of the data word in every bit (see Table A–6). These status bits can be cleared by writing a 1 in the respective locations.

[Table A–2](#) shows the control word specification register.

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

Bit	Name	Direction	Description
0	Count single-bit error	Decoder-corrector	When 1, count single-bit errors.
1	Correct single-bit error	Decoder-corrector	When 1, correct single-bit errors.
2	Double-bit error enable	Decoder-corrector	When 1, detect all double-bit errors and increment double-bit error counter.
3	Hamming code enable	Encoder	When 1, encode incoming data.
4	Clear all status registers	Controller	When 1, clear counters single-bit error and double-bit error status registers for first and last error address.
5	Single-bit error interrupt mask	Controller	When 1, masks the single-bit error interrupt.
6	Double-bit error interrupt mask	Controller	When 1, masks the double-bit error Interrupt. The single-bit error and double-bit error interrupts are ORed and sent as output to the Avalon-MM interface.
7	Counter clear on read	Controller	When 1, enables counters to clear on read feature.
8	Corrupt ECC enable	Controller	When 1, enables deliberate ECC corruption during encoding, to test the ECC.
9	ECC corruption type	Controller	When 0, creates single-bit errors in all ECC codewords; when 1, creates double-bit errors in all ECC codewords.

Table A–2. Control Word Specification Register (Part 2 of 2)			
Bit	Name	Direction	Description
10	First or last error	Controller	When 1, stores the first error address rather than the last error address of single-bit error or double-bit error.
11	Clear interrupt	Controller	When 1, clears the interrupt.

Table A–3 shows the interrupt status register.

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

Table A–4 shows the interrupt mask register.

Table A–4. Interrupt Mask Register		
Bit	Name	Description
0	Single-bit error	When 1, masks single-bit error.
1	Double-bit error	When 1, masks double-bit error.
2	Maximum single-bit error	When 1, masks single-bit error maximum threshold exceeding condition.
3	Maximum double-bit error	When 1, masks double-bit error maximum threshold exceeding condition.
4	Double-bit error during read-modify-write	When 1, masks interrupt when double-bit error occurs during a read-modify-write condition. (partial write).
Others	Reserved	Reserved.

Table A–5 shows the single-bit error location status register.

Table A–5. Single-Bit Error Location Status Register		
Bit	Name	Description
Bits $N - 1$ down to 0	Interrupt	When 0, no single-bit error; when 1, single-bit error occurred in this 64-bit part.
Others	Reserved	Reserved.

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

Table A–6. Double-Bit Error Location Status Register		
Bit	Name	Description
Bits $N-1$ down to 0	Cause of Interrupt	When 0, no double-bit error; when 1, double-bit error occurred in this 64-bit part.
Others	Reserved	Reserved.

