MegaCore®

DDR3 SDRAM Controller User Guide

ALTERA®

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

MegaCore Version: 7.2
Document Date: October 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.

I.S. EN ISO 9001

UG-DDR3-1.0

Contents

1. About This MegaCore Function~~s~~

Release Information

Table 1–1 provides information about this release of the DDR3 SDRAM Controller MegaCore® functions.

Table 1–1. DDR3 SDRAM Controller Release Information

Item	Description
Version	7.2
Release Date	October 2007
Ordering Codes	IP-SDRAM/DDR3
Product IDs	00C2 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 DDR3 SDRAM controller to each of the Altera device families.

Table 1–2. Device Family Support

Device Family	Support
Stratix® III	Preliminary
Other device families	No support

Features

- Power-up calibrated on-chip termination (OCT) support
- Half-rate support for Stratix III devices
- SOPC Builder ready

- Support for altmemphy megafunction
- Support for industry-standard DDR3 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

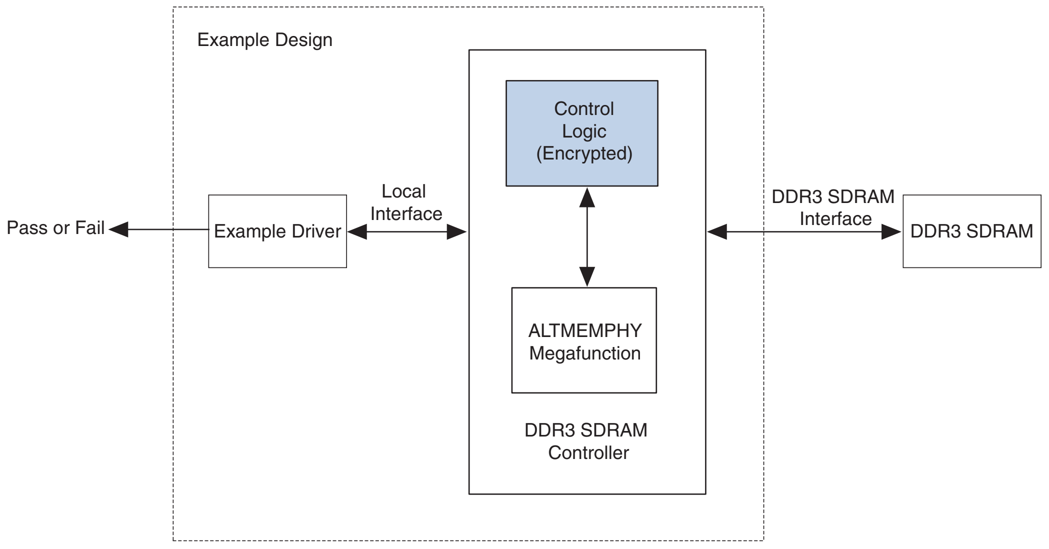
General Description

The Altera DDR3 SDRAM Controller MegaCore functions provide simplified interfaces to industry-standard DDR3 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 DDR3 SDRAM controller MegaCore functions create for you.

Figure 1–1. System-Level Diagram



The MegaWizard Plug-In Manager generates an example design that instantiates an example driver and your DDR3 SDRAM high-performance controller custom variation. 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.

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 DDR3 SDRAM controller. In addition, Altera has carried out a wide variety of gate-level tests of the DDR3 SDRAM controller to verify the post-compilation functionality of the controller.

Performance and Resource Utilization

Table 1–3 shows typical performance results for the DDR3 SDRAM controller using the Quartus® II software version MegaCore Version 7.2.

Table 1–3. Typical Performance

Device	System f_{MAX} (MHz)
Stratix III	400 *(1)*

Note to Table 1–3:
(1) Pending device characterization.

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

Table 1–6 shows typical sizes for the DDR3 SDRAM controller on Stratix III devices ~~(with no calibration logic)~~.

Table 1–4. 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

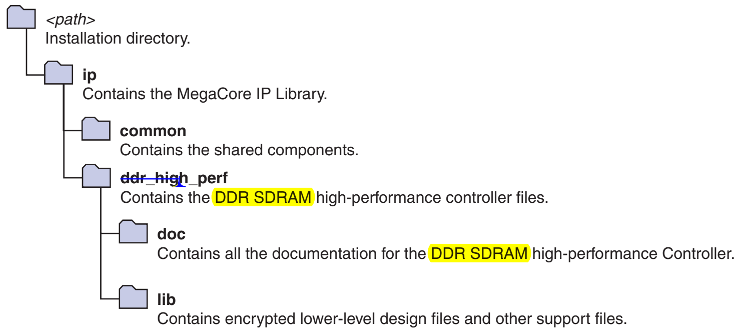
Installation and Licensing

The DDR3 SDRAM 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 Workstations*.

Figure 1–2 shows the directory structure after you install the DDR3 SDRAM Controller MegaCore functions, where <*path*> is the installation directory. The default installation directory on Windows is **c:\altera\72**; on UNIX and Solaris it is **/opt/altera/72**.

Figure 1–2. Directory Structure


<path>
Installation directory.
ip
Contains the MegaCore IP Library.
common
Contains the shared components.
~~ddr_high~~_perf
Contains the DDR SDRAM high-performance controller files.
doc
Contains all the documentation for the DDR SDRAM high-performance Controller.
lib
Contains encrypted lower-level design files and other support files.

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 DDR3 SDRAM 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.

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 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 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 DDR3 SDRAM controller, refer to *Application Note 320: OpenCore Plus Evaluation of Megafunctions*.

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.

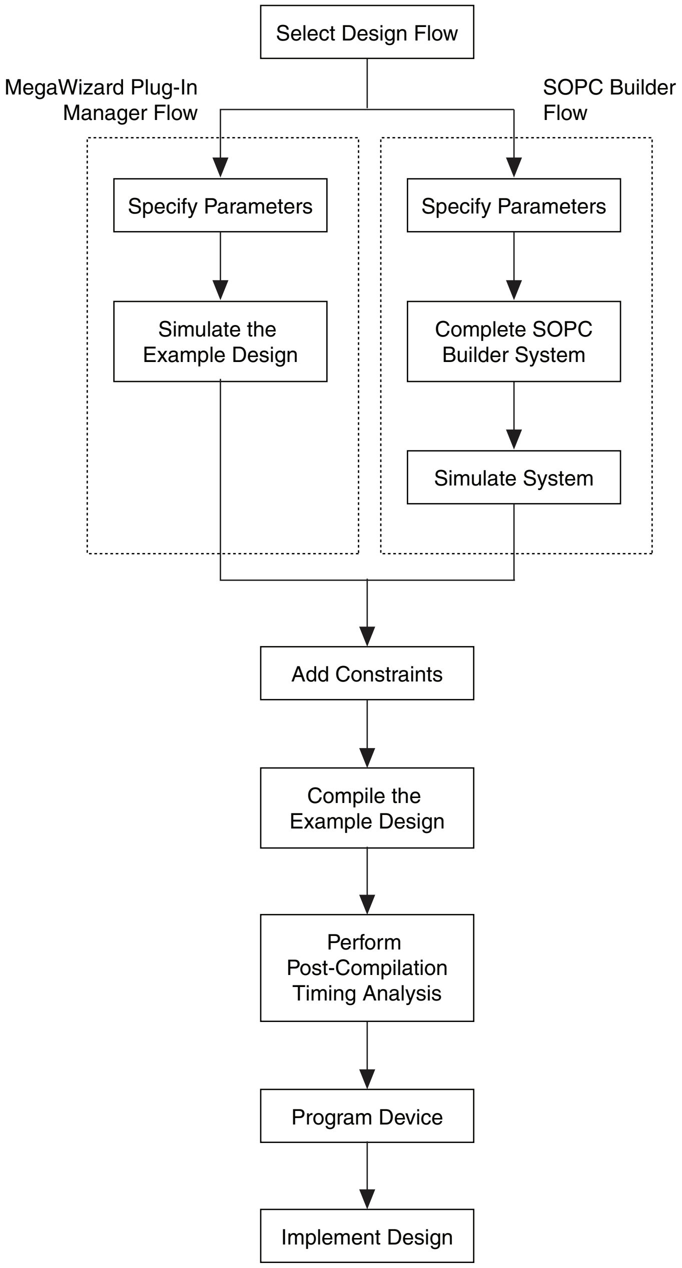
Your design stops working after the hardware evaluation time expires and the `local_ready` output goes low.

2. Getting Started

Design Flow

Figure 2–1 shows the stages for creating a system with the DDR3 SDRAM Controller MegaCore® function and the Quartus® II software. The sections in this chapter describe each stage.

Figure 2–1. Design Flow


Select Design Flow
MegaWizard Plug-In Manager Flow
SOPC Builder Flow
Specify Parameters
Simulate the Example Design
Specify Parameters
Complete SOPC Builder System
Simulate System
Add Constraints
Compile the Example Design
Perform Post-Compilation Timing Analysis
Program Device
Implement Design

Select Flow

You can parameterize the DDR3 SDRAM Controller MegaCore function using either one of the following flows:

- SOPC Builder flow
- MegaWizard Plug-in Manager flow

The SOPC Builder flow offers the following advantages:

- Automatically-generated simulation environment
- Create custom components and integrate them via the component wizard
- All components are automatically interconnected with the Avalon-MM interface

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

- Design directly from the DDR3 SDRAM interface to peripheral device(s)
- Achieves higher-frequency operation

SOPC Builder Flow

The SOPC Builder flow allows you to add the DDR3 SDRAM Controller MegaCore function 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 an DDR3 SDRAM Controller, such as the Nios II processor, external memory controllers and scatter/gather DMA controllers. SOPC Builder automatically creates the system interconnect logic and system simulation environment.

For Information About	Refer To
SOPC Builder	Volume 4 of the *Quartus II Handbook*
Quartus II software	Quartus II Help

Specify Parameters

To specify DDR3 SDRAM Controller parameters using the SOPC Builder flow, follow these 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** to your system from the **System Contents** tab.

   The **DDR3 SDRAM Controller** is in the **Memories and Memory Controllers > SDRAM** directory.

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

   For detailed explanation of the parameters, refer to the "Parameter Settings" on pag[illegible]1.

6. Click **Finish** to complete the DDR3 SDRAM Controller MegaCore function and add it to the system.

Complete the SOPC Builder System

To complete the SOPC Builder system, follow these steps:

7. In SOPC Builder, select **Nios II processor** and click **Add**.

8. For the **Reset Vector** and the **Exception Vector** select **altmemddr**.

9. Click **Finish**.

10. In SOPC Builder expand **Interface Protocols** and expand **Serial**.

11. Select **JTAG UART** and click **Add**.

12. Click **Finish**.

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

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

14. Click **Generate**.

15. To ensure the automatically-generated constraints work you must ensure the pin names and pin group assignments match, otherwise ~~will~~ get a no fit when you compile your design. ~~Also, you must now edit the **altmemddr_example_top.v**~~ file to replace the example driver and the DDR3 SDRAM controller and instantiate your SOPC Builder-generated system.

Simulate the System

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. SOPC Builder 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.

For Information About	Refer To
Simulating SOPC Builder systems	Volume 4 of the *Quartus II Handbook*
	AN 351 :Simulating Nios II Systems

MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize the DDR3 SDRAM Controller MegaCore function, and manually integrate the function into your design.

For Information About	Refer To
MegaWizard Plug-In Manager	Quartus II Help
Quartus II software	

Specify Parameters

To specify DDR3 SDRAM Controller parameters using the MegaWizard Plug-in Manager flow, follow these 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** and follow the steps to start the MegaWizard Plug-In Manager.

   The DDR3 SDRAM Controller MegaCore function is in the **Interfaces** > **Memory Controllers** directory.

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, select **Generate Simulation Model** and **Language** 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.

5. On the **Summary** tab, select the files you want to generate. A grey checkmark indicates a file that is automatically generated. All other files are optional.

   For more information about the files generated in your project directory, see Table 2–1.

6. Click **Finish** to generate the MegaCore function and supporting files.

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

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 *Note (1)*

Filename	Description
<variation name>.**bsf**	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<variation name>.**html**	MegaCore function report file.
<variation name>.**vo or .vho**	VHDL or Verilog HDL gate-level simulation model.
<variation name>.**vhd**, or **.v**	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
*<variation name>*_**auk_ddr_hp_controller_wrapper.vo or .vho**	VHDL or Verilog HDL IP functional simulation model.
*<variation name>*_**bb.v**	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
*<variation name>*_**example_driver.vhd**, or **.v**	Example driver.
*<variation name>*_**example_top.vhd**, or **.v**	Example design.

Notes to Table 2–1:
(1) *<variation name>* is the variation name.

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

9. Set the *<variation name>*_**example_top.v** or **.vhd** file to be the project top-level design file.

Now, simulate the example design (see "Simulate the Example Design" on page 2–8) and compile (see "Compile the Example Design" on page 2–9).

Simulate the Example Design

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 4–8.

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

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. Check that the absolute path to your third-party simulator executable is set. On the Tools menu click **Options** and select **EDA Tools Options**.
4. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
5. 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**.
6. Click **New**.
7. Enter a name for the **Test bench name**.
8. Enter the name of the automatically generated testbench, <*variation name*>`_example_top_tb`, in **Test bench entity**.

9. Enter the name of the top-level instance in **Instance**.

10. Change **Run for** to **500** µ**s**.

11. 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**.

    NativeLink creates this list for you.

12. Click **OK**.

13. Click **OK**.

14. 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_ddr_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 DDR3 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 DDR3 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 LVTTL, 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 or ODT setting 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 DDR3 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 2–8) or program your targeted Altera device to verify the example design in hardware.

Implement Your Design

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

3. Parameter Settings

Memory Settings

The memory settings page is the same as the altmemphy megafunction memory settings page.

For more information on the memory settings, refer to the *altmemphy Megafunction User Guide*.

PHY Settings

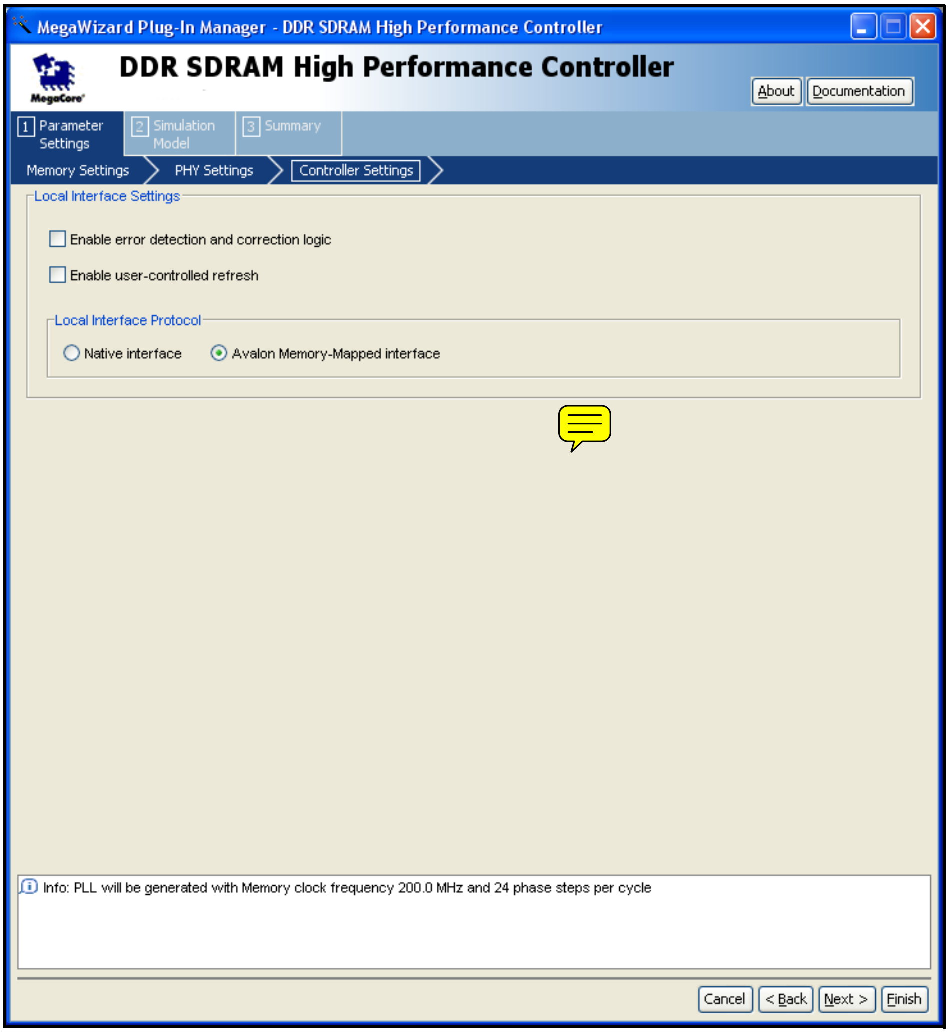
Board skew is the skew across all the memory interface signals, which includes clock, address, command, data, mask, and strobe signals.

For more information on the PHY settings, refer to the *altmemphy Megafunction User Guide*.

Controller Settings

Figure 3–1 shows the controller settings.

Figure 3–1. Controller Settings



DDR SDRAM High Performance Controller

Local Interface Settings

- [ ] Enable error detection and correction logic
- [ ] Enable user-controlled refresh

Local Interface Protocol

Native interface [✓ Avalon Memory-Mapped interface]

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

Table 3–1 shows the controller settings.

Table 3–1. Controller Settings

Parameter	Range	Description
Enable user controlled refresh	On or off	Turn on for user control of the refreshes, see "User Refresh Control" on page 4–17.
Local Interface Protocol	Native or Avalon Memory-Mapped	Specifies the local side interface between the user logic and the memory controller. The Avalon® Memory-Mapped (MM) interface allows you to easily connect to other Avalon-MM peripherals.

4. Functional Description

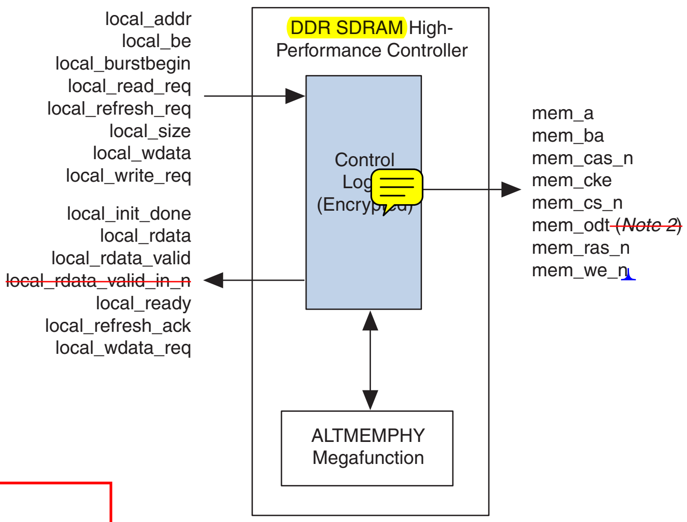
The DDR3 SDRAM controller instantiates encrypted control logic and the altmemphy megafunction.

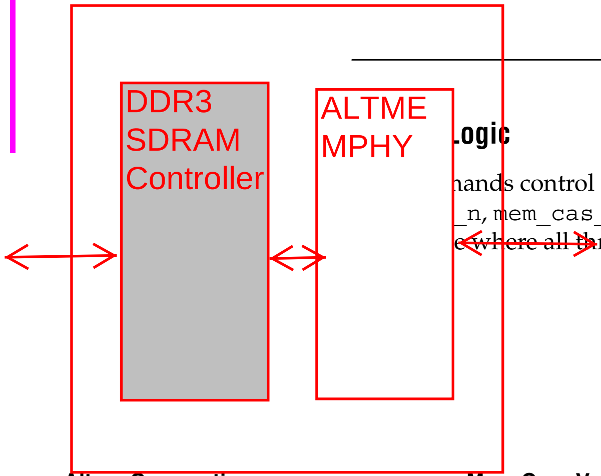
For more information on the altmemphy megafunction, refer to the *altmemphy Megafunction User Guide*.

Block Description

Figure 4–1 shows a block diagram of the DDR3 SDRAM controller.

Figure 4–1. DDR3 SDRAM Controller Block Diagram





Logic

…ands control SDRAM devices using combinations of the …_n, mem_cas_n, and mem_we_n signals. For example, on a …e 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 4–1 shows the standard SDRAM bus commands.

Table 4–1. 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 DDR3 SDRAM controller 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 DDR3 SDRAM controller closes the bank and open it again if they need to access a different row. The precharge (PCH) command closes only a bank.

The primary commands used to access SDRAM are read (RD) and write (WR). When the WR command is issued, the initial column address and data word is registered. When a RD command is issued, the initial address is registered. The initial data appears on the data bus ~~2 to 3~~ clock cycles later ~~(3 to 5 for DDR2 SDRAM)~~. This delay is the column address strobe (CAS) latency and is due to the time required to read the internal DRAM core and register the data on the bus. The CAS latency depends on the speed of the SDRAM and the frequency of the memory clock. In general, the faster the clock, the more cycles of CAS latency are required. After the initial RD or WR command, sequential reads and writes continue until the burst length is reached ~~or a burst terminate (BT) command is issued~~. DDR3 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 DDR3 SDRAM 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

There are two types of latency that you must consider for memory controller designs—read and write latencies. We define the read and write latencies as follows:

- Read latency is the time it takes for the read data to appear at the local interface after you initiate the read request
- Write latency is the time it takes for the write data to appear at the memory interface after you initiate the write request

Latency calculations have the following assumptions:

- Reading and writing to the rows that are already open
- The `local_ready` signal is asserted high (no wait states)
- The latency is defined using the local side frequency and absolute time (ns)

☞ For the half-rate controller the local side frequency is half the memory interface frequency.

Altera defines the read and write latencies in terms of the local interface clock frequency and by the absolute time for the memory controllers.

Read latency has the following definition.

Read latency = controller latency + command output latency + CAS latency + PHY read data input latency

Write latency has the following definition:

Write latency = controller latency + command output latency + write data latency

Table 4–2 shows the read and write latency component definitions.

Table 4–2. Latency Definitions

Term	Description
Controller latency	`local_read_req` to `control_doing_rd`.
Command output latency	`control_doing_rd` to `mem_cs_n`.
CAS latency	Read command to DQ data appearing on the bus.
PHY read data input latency	Read data that appears on the local interface.
Write data latency	Write data that appears on the memory inteface.

Table 4–3 shows read and write latency derived from the write and read latency definitions for half-rate controllers and for Stratix III devices.

Table 4–3. Typical Latency

Controller Rate	Frequency (MHz)	Latency Type	Latency (Cycles)	Latency (ns)
Half	~~333~~	Read	6 + 4 + 1.5 + 8.5 = 20	~~60~~
		Write	8 + 0 + 2 = 10	~~30~~

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.

Example Design

The MegaWizard® Plug-In Manager creates an example design that shows you how to instantiate and connect up the DDR3 SDRAM controller. The example design consists of the DDR3 SDRAM controller and some driver logic to issue read and write requests to the controller. The example design is a working system that can be compiled and used for both static timing checks and board tests.

Figure 4–1 shows the testbench and the example design.

Figure 4–2. Testbench & Example Design

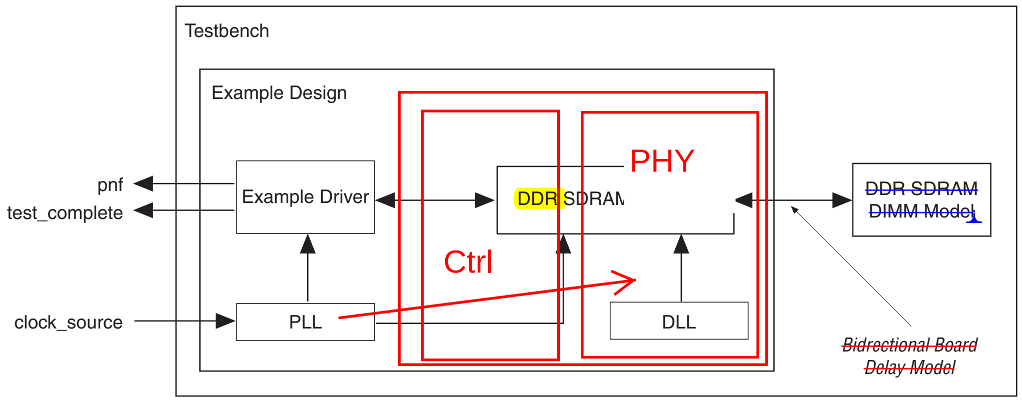


Table 4–5 describes the files that are associated with the example design and the testbench.

Table 4–4. 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*>**_example_driver.v** or **.vhd**	Example driver.
<*variation name*> **.v** or **.vhd**	Top-level description of the custom MegaCore function.

The example driver is a self-checking test generator for the DDR3 SDRAM 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.

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 2–8.

Interfaces & Signals

This section describes the following topics:

- "Interface Description" on page 4–10
- "Signals" on page 4–22

Interface Description

This section describes the following local-side interface requests:

- "Writes" on page 4–11

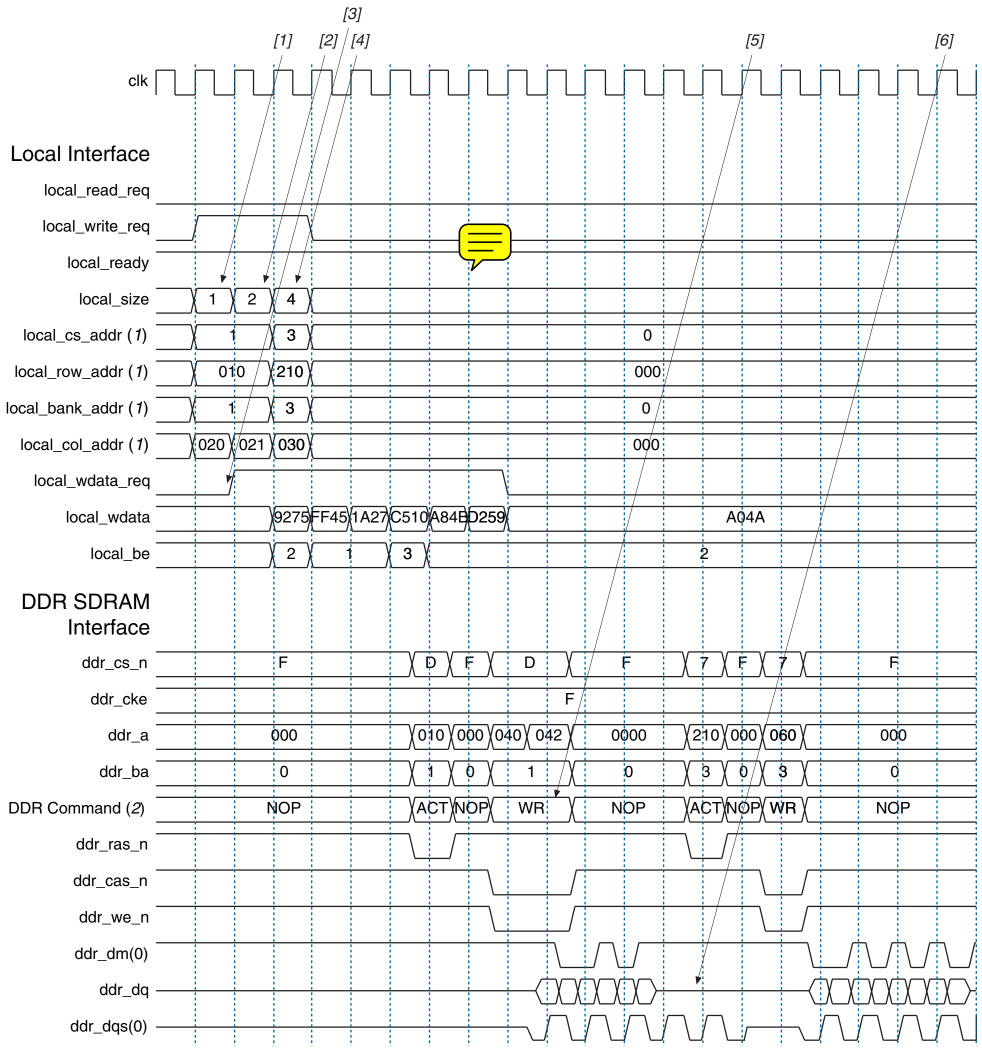
- "Reads" on page 4–13
- "Read-Write-Read-Write" on page 4–15
- "Read-Write-Read-Write" on page 4–15
- "DDR SDRAM Initialization Timing" on page 4–18
- "DDR2 SDRAM Initialization Timing" on page 4–20

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

Writes

Figure 4–2 on page 4–12 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 DDR3 SDRAM, the controller allows bursts of length 1, 2, 3, and 4 (2, 4, 6, and 8 on the DDR3 SDRAM side).~~

Figure 4–3. Writes



Notes to Figure 4–2:

(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 DDR3 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 DDR3 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.

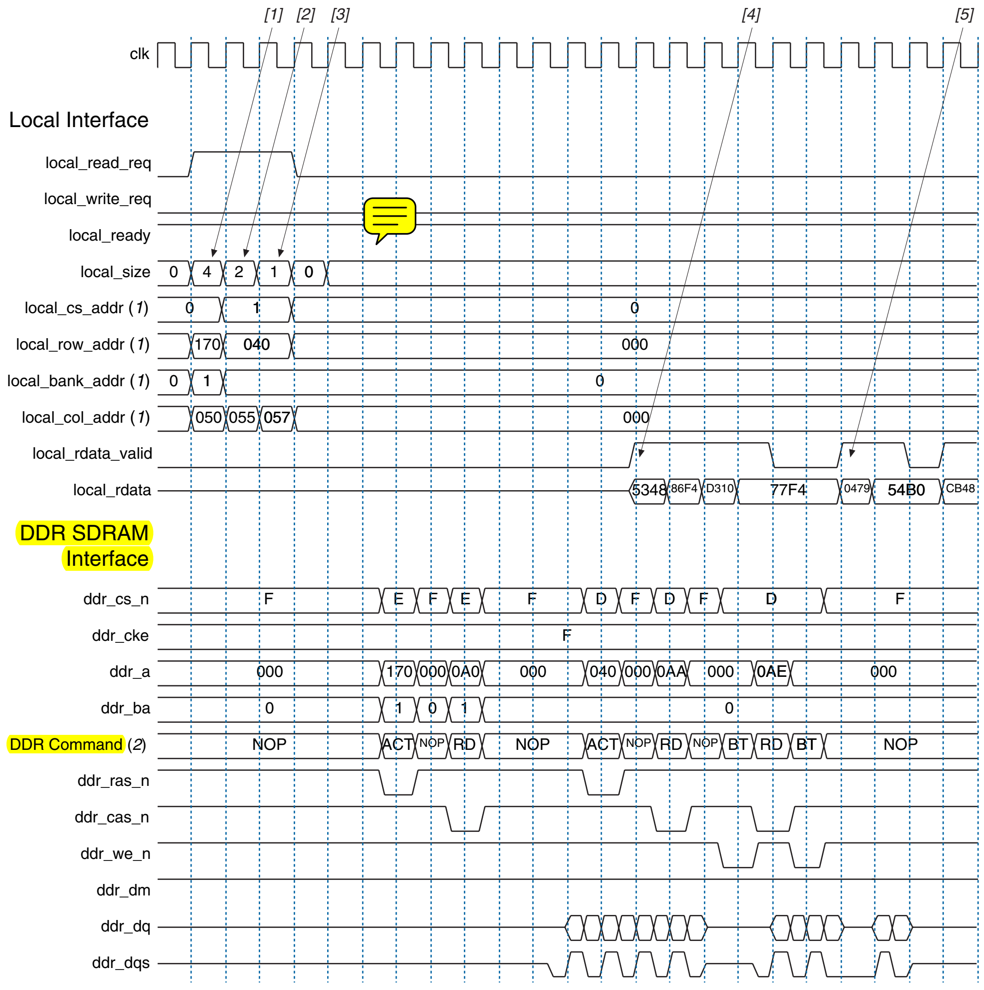
5. When it has issued the necessary bank activation command, the controller issues the first two write requests sequentially to the memory device.

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 4–3 on page 4–14 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 DDR3 SDRAM memory, the controller allows bursts of length 1, 2, 3, and 4 (2, 4, 6, and 8 on the DDR3 SDRAM side).~~

Figure 4–4. Reads



Notes to Figure 4–3:

(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 DDR3 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 DDR3 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 DDR3 SDRAM side). The `local_ready` signal remains asserted, which indicates that the controller has accepted the request.

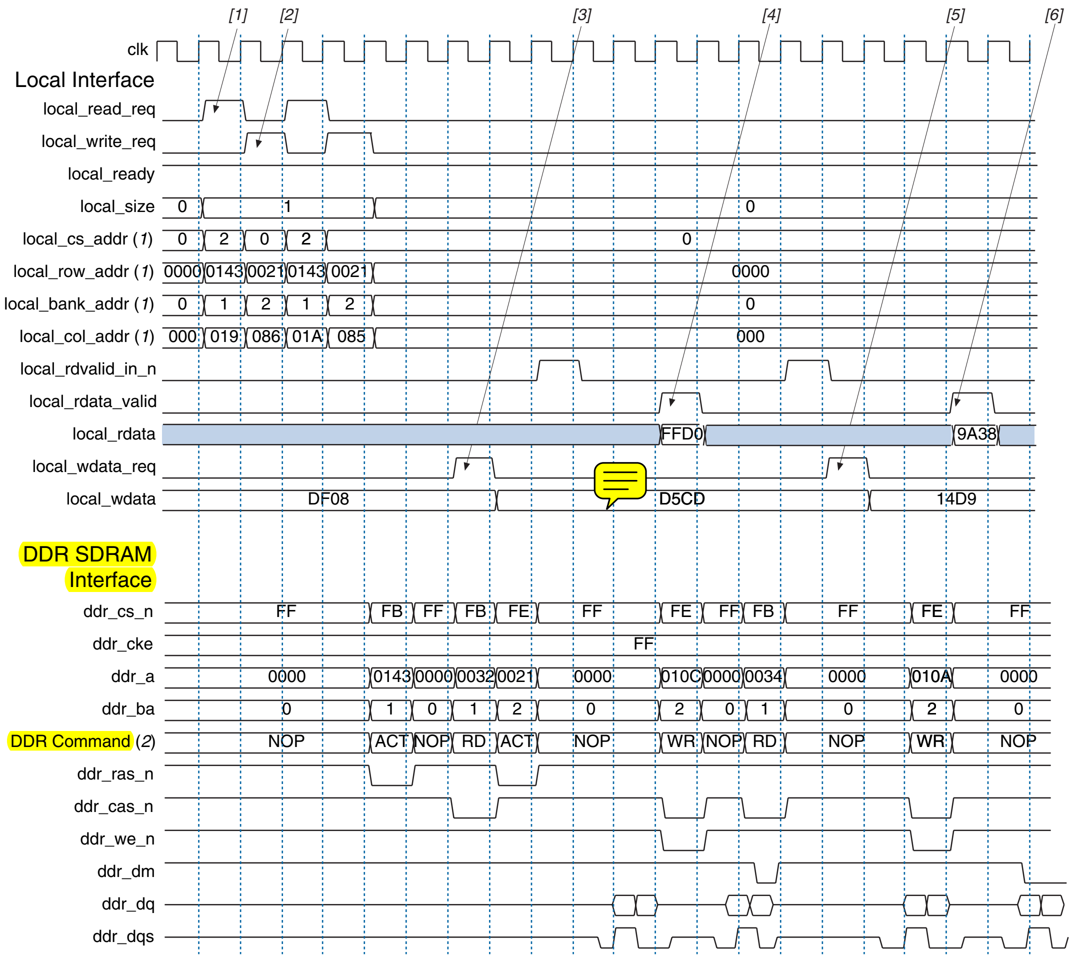
4. The controller returns the read data for the first request by asserting the `local_rdata_valid` signal. The exact number of clock cycles between the controller accepting the request and returning the data depends on the number of other requests pending in the controller, the state the memory is in, 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 4–4 on page 4–16 shows a sequence of interleaved reads and writes.

Figure 4–5. Read-Write-Read-Write



Notes to *Figure 4–4*:

(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.

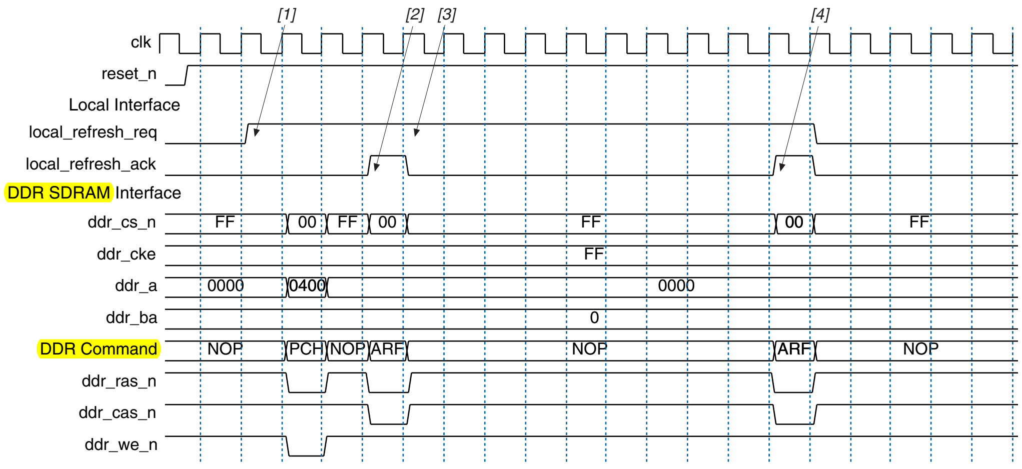
5. The controller again asserts the write data request for the second write request.

6. The read data from the second read request is returned.

User Refresh Control

Figure 4–5 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 4–6. User Refresh Control



Note to Figure 4–5:
(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

The DDR3 SDRAM ~~controller~~ ~~initializes the SDRAM devices by issuing the following memory command sequence:~~

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

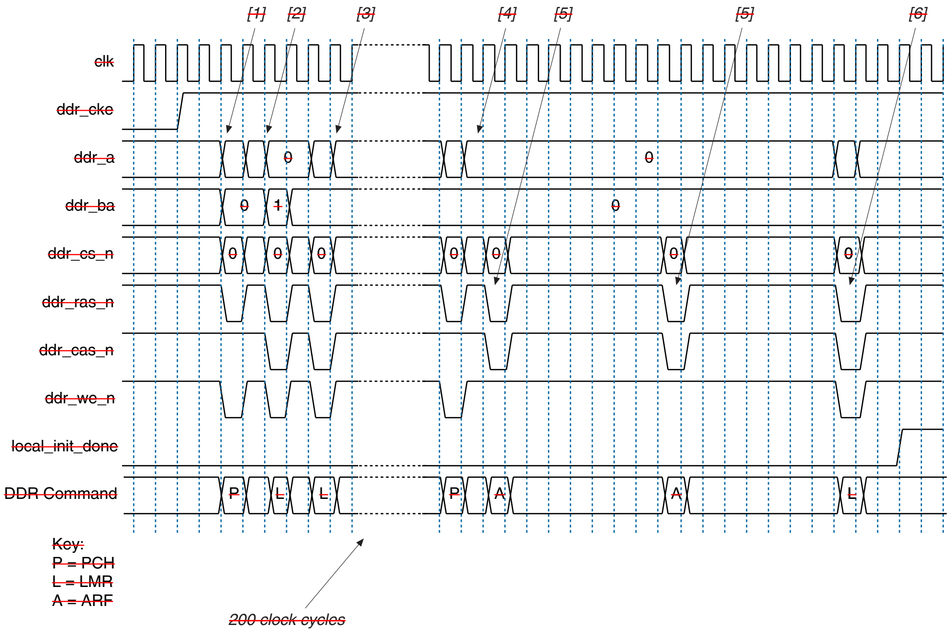
~~After issuing the final LMR command, the memory controller hands over control of the memory to the altmemphy megafunction to allow it to carry out its calibration process.~~

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

~~Figure 4–6 on page 4–19 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 µs. This time can be reduced for simulation testing by setting the **memory initialization time at power up t(INIt)** parameter in the MegaWizard Plug-In Manager.~~

~~Do not set **tINIT** to zero.~~

~~*Figure 4–7. 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:~~

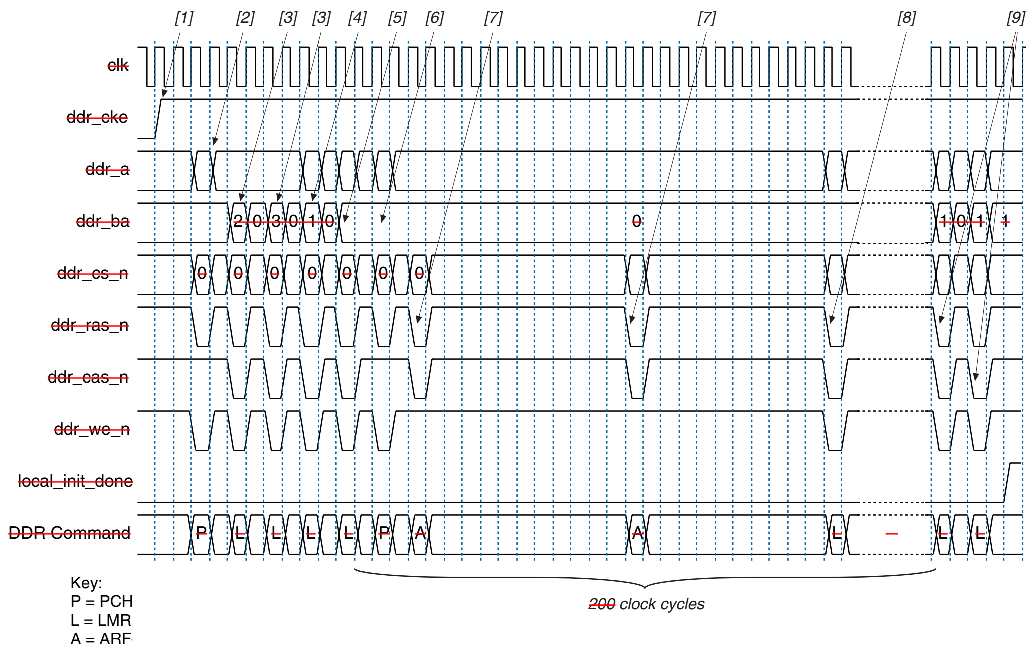
- ~~NOP (for 200 µs, programmable)~~
- ~~PCH~~
- ~~ELMR, register 2~~
- ~~ELMR, register 3~~
- ~~ELMR, register 1~~
- ~~LMR~~
- ~~PCH~~
- ~~ARF~~
- ~~ARF~~
- ~~LMR~~
- ~~ELMR, register 1~~
- ~~ELMR, register 1~~

~~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 4–7 on page 4–21 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 µs. This time can be reduced for simulation testing by setting the **memory initialization time at power up t(INIt)** parameter in the MegaWizard Plug-In Manager.~~

~~*Figure 4–8. DDR2 SDRAM Device Initialization Timing*~~



~~*Note to Figure 4–7:*~~
~~(1) local_init_done only goes high when calibration has completed.~~

1. ~~The clock enable signal (`CKE`) is asserted 200 µs 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 µs 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 4–6 shows the DDR3 SDRAM controller local interface signals.

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

Signal Name	Direction	Description
`local_addr[]`	Input	Memory address at which the burst should start. The width of this bus is sized using the following equation: For one chip select: width = bank bits + row bits + column bits – 1 For multiple chip selects: width = chip bits + bank bits + row bits + column bits – 1 For half-rate controllers, two least significant bits (LSB) of the column address on the memory side are ignored, because the local data width is four times that of the memory data bus width
`local_be[]`	Input	Byte enable signal, which you use to mask off individual bytes during writes.
`local_burstbegin`	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 `local_ready` is deasserted.

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

Signal Name	Direction	Description
`local_read_req`	Input	Read request signal.
`local_refresh_req`	Input	User controlled refresh request. If User Controlled Refresh is turned on, `local_refresh_req` becomes available and you are responsible for issuing sufficient refresh requests to meet the memory requirements. This option allows complete control over when refreshes are issued to the memory including ganging together multiple refresh commands. Refresh requests take priority over read and write requests unless they are already being processed.
`local_size[]`	Input	Controls the number of beats in the requested read or write access to memory, encoded as a binary number. ~~The range of values depend on the memory burst length.~~ ~~If you select a memory burst length 4 and half rate, the local burst length is 1 and so `local_size` should always be driven with 1.~~
`local_wdata[]`	Input	Write data bus. The width of `local_wdata` is four times the memory data bus for half rate controller.
`local_write_req`	Input	Write request signal.
`local_init_done`	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 `local_init_done` is asserted, however they are not issued to the memory until it is safe to do so.
`local_rdata[]`	Output	Read data bus. The width of `local_rdata` is ~~twice~~ that of the memory data bus.
~~`local_rdata_error`~~	~~Output~~	~~Asserted if the current read data has an error.~~
`local_rdata_valid`	Output	Read data valid signal. The `local_rdata_valid` signal indicates that valid data is present on the read data bus. The timing of `local_rdata_valid` is automatically adjusted to cope with your choice of resynchronization and pipelining options.
`local_rdvalid_in_n`	Output	An early version of the read data valid signal which appears three cycles before it. Only present with native interfaces.
`local_ready`	Output	The `local_ready` signal indicates that the DDR3 SDRAM controller is ready to accept request signals. If `local_ready` is asserted in the clock cycle that a read or write request is asserted, that request has been accepted. The `local_ready` signal is deasserted to indicate that the DDR3 SDRAM controller cannot accept any more requests.
`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. .

Table 4–7 shows the DDR and DDR2 SDRAM interface signals.

Table 4–6. DDR & DDR2 SDRAM Interface Signals

Signal Name	Direction	Description
`mem_dq[]`	Bidirectional	Memory data bus. This bus is half the width of the local read and write data busses.
`mem_dqs[]`	Bidirectional	Memory data strobe signal, which writes data into the DDR or DDR2 SDRAM and captures read data into the Altera device.
`mem_clk` *(1)*	Bidirectional	Clock for the memory device.
`mem_clk_n` *(1)*	Bidirectional	Inverted clock for the memory device.
`mem_a[]`	Output	Memory address bus.
`mem_ba[]`	Output	Memory bank address bus.
`mem_cas_n`	Output	Memory column address strobe signal.
`mem_cke[]`	Output	Memory clock enable signals.
`mem_cs_n[]`	Output	Memory chip select signals.
`mem_dm[]`	Output	Memory data mask signal, which masks individual bytes during writes.
`mem_odt[]`	Output	Memory on-die termination control signal (DDR2 SDRAM only).
`mem_ras_n`	Output	Memory row address strobe signal.
`mem_we_n`	Output	Memory write enable signal.

Note to Table 4–7:

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

Additional Information

Revision History

The following table shows the revision history for this user guide.

Date	Version	Changes Made
October 2007	7.2	First release.

How to Contact Altera

For the most up-to-date information about Altera® products, see 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
Altera literature services	Email	literature@altera.com
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

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

Typographic Conventions

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

Table Info–1. Typographic Conventions (Part 1 of 2)

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, file names, file name extensions, and software utility names are shown in bold type. Examples: **f_{MAX}**, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.

Table Info–1. Typographic Conventions (Part 2 of 2)

Visual Cue	Meaning
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	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>*.**pof** 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.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user’s work.
⚠ WARNING	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.