### Contents

**About This User Guide**
- Revision History ................................................................. v
- How to Contact Altera .......................................................... v
- Typographic Conventions ....................................................... vi

**Chapter 1. About This MegaCore Function**
- Release Information ........................................................... 1–1
- Device Family Support .......................................................... 1–1
- Introduction .......................................................................... 1–2
- Features ............................................................................... 1–2
- General Description .............................................................. 1–2
  - OpenCore Plus Evaluation ................................................... 1–4
- Performance .......................................................................... 1–4

**Chapter 2. Getting Started**
- System Requirements .......................................................... 2–1
- Design Flow .......................................................................... 2–1
- Obtain & Install the RLDRAM II Controller ......................... 2–2
  - Download the RLDRAM II Controller MegaCore Function .... 2–2
  - Install the RLDRAM II Controller MegaCore Function Files ..... 2–3
- Directory Structure ............................................................... 2–4
- RLDRAM II Controller Walkthrough ..................................... 2–4
  - Create a New Quartus II Project ........................................... 2–5
  - Launch IP Toolbench .......................................................... 2–6
  - Step 1: Parameterize .......................................................... 2–7
  - Step 2: Constraints ............................................................ 2–12
  - Step 3: Set Up Simulation .................................................... 2–14
  - Step 4: Generate ............................................................... 2–16
- Simulate the Example Design .................................................. 2–19
- Edit the PLL .......................................................................... 2–19
- Compile the Example Design .................................................. 2–20
- Program a Device ................................................................. 2–21
- Implement Your Design .......................................................... 2–22
- Set Up Licensing ................................................................. 2–22
  - Append the License to Your license.dat File ......................... 2–22
  - Specify the License File in the Quartus II Software ................. 2–23

**Chapter 3. Specifications**
- Functional Description .......................................................... 3–1
- Control Logic ......................................................................... 3–2
## Contents

Datapath .................................................................................................................................................. 3–3
OpenCore Plus Time-Out Behavior ................................................................................................... 3–12
Device-Level Configuration .................................................................................................................. 3–12
   PLL Configuration .......................................................................................................................... 3–12
   Example Design ............................................................................................................................. 3–14
Constraints .......................................................................................................................................... 3–16
Interfaces .......................................................................................................................................... 3–16
   Initialization ................................................................................................................................... 3–16
   Writes ........................................................................................................................................... 3–17
   Reads ............................................................................................................................................... 3–19
   Refreshes ......................................................................................................................................... 3–21
Signals ............................................................................................................................................... 3–22
Parameters .......................................................................................................................................... 3–22
   Memory ......................................................................................................................................... 3–27
   Timing ........................................................................................................................................... 3–29
   Project Settings .......................................................................................................................... 3–30
MegaCore Verification ....................................................................................................................... 3–31
About This User Guide

Revision History

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

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Date</th>
<th>Version</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 3</td>
<td>October 2005</td>
<td>1.0.0</td>
<td>First published.</td>
</tr>
</tbody>
</table>

How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>USA &amp; Canada</th>
<th>All Other Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(800) 800-EPLD (3753)</td>
<td>+1 408-544-8767</td>
</tr>
<tr>
<td></td>
<td>(7:00 a.m. to 5:00 p.m. Pacific Time)</td>
<td>7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time</td>
</tr>
<tr>
<td>Product literature</td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
</tr>
<tr>
<td>Altera literature services</td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
</tr>
<tr>
<td>Non-technical customer service</td>
<td>(800) 767-3753</td>
<td>+ 1 408-544-7000</td>
</tr>
<tr>
<td></td>
<td>(7:00 a.m. to 5:00 p.m. Pacific Time)</td>
<td>(GMT -8:00) Pacific Time</td>
</tr>
<tr>
<td>FTP site</td>
<td>ftp.altera.com</td>
<td>ftp.altera.com</td>
</tr>
</tbody>
</table>
This document uses the typographic conventions shown below.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <strong>Save As</strong> dialog box.</td>
</tr>
<tr>
<td><em>bold type</em></td>
<td>External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: ( f_{\text{MAX}} ), `qdesigns` directory, \textit{d:} drive, \textit{chiptrip.gdf} file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Document titles are shown in italic type with initial capital letters. Example: <em>AN 75: High-Speed Board Design.</em></td>
</tr>
<tr>
<td><em>Italic type</em></td>
<td>Internal timing parameters and variables are shown in italic type. Examples: ( t_{\text{PIA}}, n + 1 ).</td>
</tr>
<tr>
<td></td>
<td>Variable names are enclosed in angle brackets (&lt; &gt;) and shown in italic type. Example: &lt;\textit{file name}&gt;, &lt;\textit{project name}&gt;.pot file.</td>
</tr>
<tr>
<td><em>Initial Capital Letters</em></td>
<td>Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.</td>
</tr>
<tr>
<td>“Subheading Title”</td>
<td>References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: ”Typographic Conventions.”</td>
</tr>
<tr>
<td><em>Courier type</em></td>
<td>Signal and port names are shown in lowercase Courier type. Examples: \textit{data1}, \textit{tdi}, \textit{input}. Active-low signals are denoted by suffix \textit{n}, e.g., \textit{resetn}.</td>
</tr>
<tr>
<td></td>
<td>Anything that must be typed exactly as it appears is shown in Courier type. For example: \textit{c:}`\textbackslash qdesigns\textbackslash tutorial\textbackslash chiptrip.gdf}. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword \textit{SUBDESIGN}), as well as logic function names (e.g., \textit{TRI}) are shown in Courier.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., etc.</td>
<td>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.</td>
</tr>
<tr>
<td>■ •</td>
<td>Bullets are used in a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td>✓</td>
<td>The checkmark indicates a procedure that consists of one step only.</td>
</tr>
<tr>
<td>🔄</td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td>⚠</td>
<td>The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</td>
</tr>
<tr>
<td>⚠</td>
<td>The warning indicates information that should be read prior to starting or continuing the procedure or processes.</td>
</tr>
<tr>
<td>↔</td>
<td>The angled arrow indicates you should press the Enter key.</td>
</tr>
<tr>
<td>≈</td>
<td>The feet direct you to more information on a particular topic.</td>
</tr>
</tbody>
</table>
1. About This MegaCore Function

Release Information

Table 1–1 provides information about this release of the Altera® RLDAM II Controller MegaCore® function.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Release Date</td>
<td>October 2005</td>
</tr>
<tr>
<td>Ordering Code</td>
<td>IP-RLDRAMII</td>
</tr>
<tr>
<td>Product ID</td>
<td>00AC</td>
</tr>
<tr>
<td>Vendor ID</td>
<td>6AF7</td>
</tr>
</tbody>
</table>

Device Family Support

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

- 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 RLDAM II Controller MegaCore function to each Altera device family.

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix® II</td>
<td>Full</td>
</tr>
<tr>
<td>Stratix II GX</td>
<td>Preliminary</td>
</tr>
<tr>
<td>HardCopy® II</td>
<td>Preliminary</td>
</tr>
<tr>
<td>Other device families</td>
<td>No support</td>
</tr>
</tbody>
</table>
Introduction

The RLDRAM II controller MegaCore function handles the complex aspects of using RLDRAM II—initializing the memory devices and translating read and write requests from the local interface into all the necessary RLDRAM II command signals.

Features

- Common I/O (CIO) and separate I/O (SIO) device support
- Memory burst length 2, 4, and 8-beat support
- Nonmultiplexed addressing
- Datapath generation
- Data strobe signal (DQS) and non-DQS capture modes
- Automatic constraint generation
- Easy-to-use IP Toolbench interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore® Plus evaluation

General Description

The RLDRAM II controller is optimized for Altera Stratix II devices and has preliminary support for Stratix II GX and HardCopy II devices. The advanced features available in these devices allow you to interface directly to RLDRAM II devices.

Figure 1–1 shows a system-level diagram including the example design that the RLDRAM II Controller MegaCore function creates for you.
**Note to Figure 1–1:**

(1) Non-DQS mode only.

IP Toolbench generates the following items:

- A testbench, which instantiates the example design
- A synthesizable example design which instantiates the following modules:
  - RLDRAM II controller:
    - Encrypted control logic, which takes transaction requests from the local interface and issues writes, reads, and refreshes to the memory interface
    - A clear-text datapath
  - Example driver—generates write, read and refresh requests and outputs a *pass_fail* signal to indicate that the tests are passing or failing
  - System phase-locked loop (PLL)—generates the RLDRAM II controller clocks
  - Delay locked loop (DLL)—instantiated in DQS mode and generates the DQS delay control signal for the dedicated DQS delay circuitry
Performance

- Optional feedback clock PLL—instantiated in non-DQS mode and generates a capture clock for the datapath read capture and logic path

OpenCore Plus Evaluation

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

- Simulate the behavior of a MegaCore function within your system
- Verify the functionality of your design, as well as quickly and easily evaluate its size and speed
- 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 obtain a license for the MegaCore function 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 RLDRAM II controller, see “OpenCore Plus Time-Out Behavior” on page 3–12 and AN 320: OpenCore Plus Evaluation of Megafunctions.

Performance

Table 1–3 shows typical expected performance for the RLDRAM II Controller MegaCore function, with the Quartus II software version 5.1.

<table>
<thead>
<tr>
<th>Device</th>
<th>Capture Mode</th>
<th>fMAX (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix II (EP2S60F1020C3)</td>
<td>Non-DQS</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>DQS</td>
<td>300</td>
</tr>
</tbody>
</table>
2. Getting Started

System Requirements

The instructions in this section require the following hardware and software:

- A computer running any of the following operating systems:
  - Windows 2000/XP
  - Red Hat Linux 8.0
  - Red Hat Enterprise Linux 3 WS (with support for 32-bit, AMD64, or Intel EM64T workstations)
  - Solaris 8 or 9 (32-bit or 64-bit)
- Quartus® II software version 5.1 or higher

Design Flow

To evaluate the RLDRAM II Controller MegaCore function using the OpenCore® Plus feature, the design flow involves the following steps:

1. Obtain and install the RLDRAM II Controller MegaCore function.

2. Create a custom variation of the RLDRAM II Controller MegaCore function using IP Toolbench.

   IP Toolbench is a toolbar from which you can quickly and easily view documentation, specify parameters, and generate all of the files necessary for integrating the parameterized MegaCore function into your design. You can launch IP Toolbench from within the Quartus II software.

3. Implement the rest of your design using the design entry method of your choice.

4. Use the IP Toolbench-generated IP functional simulation model to verify the operation of your design.

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

5. Edit the PLL.

6. Use the Quartus II software to add constraints to the example design and compile the example design.
Obtain & Install the RLDRAM II Controller

7. Perform gate-level timing simulation, or if you have a suitable development board, you may generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of the example design in hardware.

8. Either obtain a license for the RLDRAM II controller MegaCore function or replace the encrypted RLDRAM II controller control logic with your own logic and use the clear-text datapath.

   If you obtain a license for the RLDRAM II controller, you must set up licensing.

9. Generate a programming file for the Altera® device(s) on your board.

10. Program the Altera device(s) with the completed design.

11. Perform design verification.

Obtain & Install the RLDRAM II Controller

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your computer. Altera MegaCore functions can be installed from the MegaCore IP Library CD-ROM either during or after Quartus II installation, or downloaded individually from the Altera web site and installed separately.

   The following instructions describe the process of downloading and installing the RLDRAM II controller. If you have already installed the RLDRAM II controller from the MegaCore IP Library CD-ROM, skip to “Directory Structure” on page 2–4.

Download the RLDRAM II Controller MegaCore Function

If you have Internet access, you can download MegaCore functions from Altera’s web site at www.altera.com. Follow the instructions below to obtain the RLDRAM II controller via the Internet. If you do not have Internet access, contact your local Altera representative to obtain the MegaCore IP Library CD-ROM.

1. Point your web browser to www.altera.com/ipmegastore.

2. Type RLDRAM in the IP MegaSearch box.

3. Click Go.

4. Choose RLDRAM II Controller from the search results page. The product description web page displays.
5. Click **Download Free Evaluation** on the top right of the product description web page.

6. Fill out the registration form and click **Submit Request**.

7. Read the Altera MegaCore license agreement, turn on the **I have read the license agreement** check box, and click **Proceed to Download Page**.

8. Follow the instructions on the RLDRAM II controller download and installation page to download the MegaCore function and save it to your hard disk.

   There is a specific MegaCore function download file for each supported operating system.

### Install the RLDRAM II Controller MegaCore Function Files

The following instructions describe how to install the RLDRAM II controller on computers running the Windows, Linux, or Solaris operating systems.

**Windows**

Follow these steps to install the RLDRAM II controller on a PC running a supported version of the Windows operating system:

1. Choose **Run** (Windows Start menu).

2. Type `<path name>`\rldram_ii_controller-v1.0.0.exe, where `<path name>` is the location of the downloaded MegaCore function.

3. Click **OK**. The **RLDRAM II Controller Installation** dialog box appears. Follow the on-screen instructions to finish installation.

**Solaris & Linux**

Follow these steps to install the RLDRAM II controller on a computer running supported versions of the Solaris and Linux operating systems:

1. Move the compressed files to the desired installation directory and make that directory your current directory.

2. Decompress the package by typing the following command:

   ```
   gzip -d rldram_ii_controller-v1.0.0_linux.tar.gz
   ```
or

```
gzip -d rldram_ii_controller-v1.0.0_solaris.tar.gz
```

3. Extract the package by typing the following command:

```
tar xvf rldram_ii_controller-v1.0.0_linux.tar
```
or

```
tar xvf rldram_ii_controller-v1.0.0_solaris.tar
```

Directory Structure

Figure 2–1 shows the directory structure for the RLDRAM II controller.

---

**Figure 2–1. RLDRAM II Controller Directory Structure**

- `<path>`
- **common**
  - Contains the common MegaCore function files.
- **ip-toolbench**
  - Contains the common IP Toolbench files.
- **rldram_ii_controller-v1.0.0**
  - Contains the RLDRAM II Controller files and documentation.
- **constraints**
  - Contains scripts that generate an instance-specific Tcl script for each instance of the RLDRAM II Controller in various Altera devices.
- **dat**
  - Contains a data file for each Altera device combination that is used by the Tcl script to generate the instance-specific Tcl script.
- **doc**
  - Contains the documentation for the MegaCore functions.
- **lib**
  - Contains encrypted lower-level design files and some clear-text example files that are used in the design flow.

---

**RLDRAM II Controller Walkthrough**

This walkthrough explains how to create a RLDRAM II controller using the Altera RLDRAM II controller IP Toolbench and the Quartus II software on a PC. When you are finished generating a RLDRAM II controller, you can incorporate it into your overall project.

This walkthrough involves the following steps:

- “Create a New Quartus II Project” on page 2–5
Create a New Quartus II Project

Before you begin, you must create a new Quartus II project. With the New Project wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You will also specify the RLDRAM II controller user library. To create a new project, follow these steps:

1. Choose Programs > Altera > Quartus II <version> (Windows Start menu) to run the Quartus II software. You can also use the Quartus II Web Edition software.

2. Choose New Project Wizard (File menu).

3. Click Next in the introduction (the introduction will not display if you turned it off previously).

4. Specify the working directory for your project. This walkthrough uses the directory d:\temp.

5. Specify the name of the project. This walkthrough uses project.

6. Click Next.

7. For Linux and Solaris operating systems, add the user libraries:
   a. Click User Library Pathnames.
   b. Type <path>\rlDRAM_ii_controller-v1.0.0\lib\ into the Library name box, where <path> is the directory in which you installed the RLDRAM II controller. The default installation directory is c:\altera\MegaCore.
   c. Click Add.
   d. Click OK.

8. Click Next.

9. Choose the target device family in the Family list.
10. Click Finish.

You have finished creating your new Quartus II project.

**Launch IP Toolbench**

To launch IP Toolbench in the Quartus II software, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing MegaWizard Plug-In Manager (Tools menu). The MegaWizard Plug-In Manager dialog box is displayed.

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

2. Specify that you want to create a new custom megafunction variation and click Next.

3. Select RLDRAM II Controller v1.0.0 in the Interfaces > Memory Controllers directory.

4. Select the output file type for your design; the wizard supports VHDL and Verilog HDL.

5. Specify a name for MegaCore function files, `<directory name>\<variation name>`. Figure 2–2 shows the wizard after you have made these settings.

   The `<variation name>` must be a different name from the project name and the top-level design entity name.
6. Click **Next** to launch IP Toolbench.

**Step 1: Parameterize**

To parameterize your MegaCore function, follow these steps:

For more information on parameters, see “Parameters” on page 3–27.

1. Click **Step 1: Parameterize** in IP Toolbench (see Figure 2–3).
2. Choose the memory type (see Figure 2–4 on page 2–9):

   a. Choose the memory device.

   ![Caveat]

   You can add your own memory devices to this list by editing the `memory_types.dat` file in the `\constraints` directory.

   b. Enter the clock speed.

   c. Choose the interface voltage.

   d. Choose the data bus width.

   e. Choose the **DQ per DQS** (CIO devices), or the **Q per DQS** (SIO devices).
3. Choose the memory initialization options.

4. Choose your memory interface parameters.

5. Click the Timing tab (see Figure 2–5 on page 2–10).

For more information on timing parameters, see “Timing” on page 3–29.
6. Enter the datapath pipeline options.

7. Choose the clocking modes.

8. Turn on the appropriate capture mode—DQS or non-DQS capture mode. If you turn off Enable DQS mode (non-DQS capture mode), you can turn on Use migratable bytegroups.

9. Click the Project Settings tab (see Figure 2–6 on page 2–11).

For more information on project settings, see “Project Settings” on page 3–30.
10. Altera recommends that you turn on **Automatically apply RLDRAm II controller-specific constraints to the Quartus II project** so that the Quartus II software automatically applies the constraints script when you compile the example design.

11. Ensure **Update the example design file that instantiates the RLDRAm II controller variation** is turned on, for IP Toolbench to automatically update the example design file.

12. Turn off **Update example design system PLL**, if you have edited the PLL and you do not want the wizard to regenerate the PLL when you regenerate the variation.
13. The constraints script automatically detects the hierarchy of your design. The constraints script analyzes and elaborates your design to automatically extract the hierarchy to your variation. To prevent the constraints script analyzing and elaborating your design, turn on **Enable Hierarchy Control**, and enter the correct hierarchy path to your datapath. Figure 2–7 shows the following example hierarchy:

```
my_system:my_system_inst|my_sub_system:my_sub_system_inst|
my_rldramii:my_rldramii_inst|datapath:datapath_inst|
```

**Figure 2–7. System Naming**  
*Note (1)*

*Note to Figure 2–7:*
(1) The names match the names in Figure 2–6.

14. IP Toolbench uses a prefix (for example, `rldramii_`) for the names of all memory interface pins. Enter a prefix for all memory interface pins associated with this custom variation.

15. Enter the pin loading for the FPGA pins.  

   ![Diagram](image)

   You must enter suitable values for the pin loading, because the values affect timing.

16. Click **Finish**.

**Step 2: Constraints**

To choose the constraints for your device, follow these steps:
1. Click **Step 2: Constraints** in IP Toolbench (see Figure 2–8).

2. Choose the positions on the device for each of the RLDRAM II byte groups (see Figure 2–9 on page 2–14). To place a byte group, select the byte group in the drop-down box at your chosen position.

   The floorplan matches the orientation of the Quartus II floorplanner. The layout represents the die as viewed from above. A byte group consists of data (DQ) pins for CIO devices; or data (Q) pins for SIO devices, and a data strobe signal (DQS) pin. The number of data pins per byte group matches your choice of **DQ** (or **Q**) per **DQS**.
Step 3: Set Up Simulation

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

You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

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

1. Click Step 3: Set Up Simulation in IP Toolbench (see Figure 2–10).
2. Turn on **Generate Simulation Model** (see Figure 2–11).

3. Choose the language in the **Language** list.
4. Click OK.

**Step 4: Generate**

To generate your MegaCore function, follow these steps:

1. Click **Step 4: Generate** in IP Toolbench (see Figure 2–12).

![Figure 2–12. IP Toolbench—Generate](image)

2. The generation report lists the design files that IP Toolbench creates (see Figure 2–13). Click **Exit**.
Table 2–1 describes the IP Toolbench-generated files.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;variation name&gt;.vhd, or .v</td>
<td>A MegaCore function variation file, which defines a VHDL or Verilog HDL 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.</td>
</tr>
<tr>
<td>&lt;variation name&gt;.cmp</td>
<td>A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.</td>
</tr>
<tr>
<td>&lt;variation name&gt;.bsf</td>
<td>Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.</td>
</tr>
</tbody>
</table>
Table 2–1. IP Toolbench-Generated Files (Part 2 of 2)  Notes (1), 2, & 3

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altera_vhdl_support.vhd</td>
<td>A VHDL package that contains functions for the generated entities. This file can be shared between MegaCore functions.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_example_driver.vhd or .v</td>
<td>Example driver.</td>
</tr>
<tr>
<td>&lt;top-level name&gt;.vhd or .v</td>
<td>Example design file.</td>
</tr>
<tr>
<td>add_constraints_for_&lt;variation name&gt;.tcl</td>
<td>Add constraints script.</td>
</tr>
<tr>
<td>rldramii_pll_&lt;device name&gt;.vhd or .v</td>
<td>System PLL.</td>
</tr>
<tr>
<td>rldramii_fbpll_&lt;device name&gt;.vhd or .v</td>
<td>Feedback PLL.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_addr_cmd_reg.vhd or .v</td>
<td>Address and command output registers.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_clk_gen.vhd or .v</td>
<td>Memory clock generator.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_controller_ipfs_wrapper.vhd or .v</td>
<td>A file that instantiates the controller.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_controller_ipfs_wrapper.vhd or .vo</td>
<td>VHDL or Verilog HDL IP functional simulation model.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_datapath.vhd or .v</td>
<td>Datapath.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_dm_group.vhd or .v</td>
<td>Data mask (DM) group.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_dqs_group.vhd or .v</td>
<td>DQS group.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_pipeline_addr_cmd.vhd or .v</td>
<td>Address and command pipeline registers.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_pipeline_qvld.vhd or .v</td>
<td>Valid data flag (QVLD) pipeline registers.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_pipeline_rdata.vhd or .v</td>
<td>Read data pipeline registers.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_pipeline_wdata.vhd or .v</td>
<td>Write data pipeline registers.</td>
</tr>
<tr>
<td>&lt;variation name&gt;_auk_rldramii_qvld_group.vhd or .v</td>
<td>QVLD group.</td>
</tr>
<tr>
<td>&lt;variation name&gt;.html</td>
<td>MegaCore function report file.</td>
</tr>
</tbody>
</table>

Notes to Table 2–1:
(1) <top-level name> is the name of the Quartus® II project top-level entity.
(2) <variation name> is the variation name.
(3) <device name> is the device family name.
Now, simulate the example design (see “Simulate the Example Design” on page 2–19), edit the PLL(s), and compile (see “Compile the Example Design” on page 2–20).

Simulate the Example Design

You can simulate the example design using the IP Toolbench-generated IP functional simulation models. IP Toolbench 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 3–14.

You can use the IP functional simulation model with any Altera-supported VHDL or Verilog HDL simulator. To simulate the example design with the ModelSim® simulator, follow these steps:

1. Obtain a memory model that matches your chosen parameters and save it to the `<directory name>`\testbench directory. For example, you can download a RLDRAM II model from the Micron web site at www.micron.com.

2. Start the ModelSim-Altera simulator.

3. Change your working directory to your IP Toolbench-generated file directory `<directory name>`\testbench\modelsim.

4. To simulate with an IP functional simulation model simulation, type the following command:

```
source <variation name>_vsim.tcl
```

5. For a gate-level timing simulation (VHDL or Verilog HDL ModelSim output from the Quartus II software), type the following commands:

```
set use_gate_model 1
source <variation name>_vsim.tcl
```

Edit the PLL

The IP Toolbench-generated example design includes a PLL, which has an input to output clock ratio of 1:1 and a clock frequency that you entered in IP Toolbench. In addition, IP Toolbench correctly sets all the phase offsets of all the relevant clock outputs for your design. You can edit the PLL input clock to make it conform to your system requirements. If you re-run IP Toolbench and wish to save your PLL edits, turn off Update example design system PLL.
If you turn off Enable DQS mode, IP Toolbench generates a second PLL—the feedback PLL. You need not edit the feedback PLL.

For more information on the PLL, see “PLL Configuration” on page 3–12.

To edit the example PLL, follow these steps:

1. Choose **MegaWizard Plug-In Manager** (Tools menu).

2. Select **Edit an existing custom megafuntion variation** and click Next.

3. In your Quartus II project directory, for VHDL choose `rldramii_pll_<device name>.vhd`; for Verilog HDL choose `rldramii_pll_<device name>.v`.

4. Click Next.

5. Edit the PLL parameters in the **altpll** MegaWizard Plug-In.

For more information on the **altpll** megafuntion, refer to the Quartus II Help or click **Documentation** in the **altpll** MegaWizard Plug-In.

**Compile the Example Design**

Before the Quartus II software compiles the example design it runs the IP Toolbench-generated Tcl constraints script, **auto_add_rldramii_constraints.tcl**, which calls the **add_constraints_for_<variation name>.tcl** script for each variation in your design. The **add_constraints_for_<variation name>.tcl** script checks for any previously added constraints, removes them, and then adds constraints for that variation.

The constraints script analyzes and elaborates your design, to automatically extract the hierarchy to your variation. To prevent the constraints script analyzing and elaborating your design, turn on **Enable Hierarchy Control** in the wizard, and enter the correct hierarchy path to your datapath (see step 13 on page 2–12).

When the constraints script runs, it creates another script, **remove_constraints_for_<variation name>.tcl**, which you can use to remove the constraints from your design.

To compile the example instance, follow these steps:
Getting Started

1. Choose **Start Compilation** (Processing menu), which runs the add constraints scripts, compiles the example design, and performs timing analysis.

2. View the Timing Analyzer to verify your design meets timing.

If the compilation does not reach the frequency requirements, follow these steps:

1. Choose **Settings** (Assignments menu).

2. Choose **Analysis and Synthesis Settings** in the category list.

3. Select **Speed** in Optimization Technique.

4. Click **OK**.

5. Re-compile the example design by choosing **Start Compilation** (Processing menu).

   To achieve a higher frequency, increase the number of address and command and write data pipeline registers, or increase the number read data pipeline registers, see step 6 on page 2–10.

To view the constraints in the Quartus II Assignment Editor, choose **Assignment Editor** (Assignments menu).

   If you have “?” characters in the Quartus II Assignment Editor, the Quartus II software cannot find the entity to which it is applying the constraints, probably because of a hierarchy mismatch. Either edit the constraints script, or enter the correct hierarchy path in the Project Settings tab (see step 13 on page 2–12).

For more information on constraints, see “Constraints” on page 3–16.

Program a Device

After you have compiled the example design, you can perform gate-level simulation (see “Simulate the Example Design” on page 2–19) 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 RLDRAM II Controller MegaCore function before you obtain a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file.
Implement Your Design

For more information on OpenCore Plus hardware evaluation using the RLDRAM II Controller MegaCore function, see “OpenCore Plus Evaluation” on page 1–4, “OpenCore Plus Time-Out Behavior” on page 3–12, and Application Note 320: OpenCore Plus Evaluation of Megafunctions.

Implement Your Design

In the MegaWizard flow, 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 obtain 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 obtain a license for RLDRAM II controller, 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 e-mails you a license.dat file. If you do not have Internet access, contact your local Altera representative.

To install your license, you can either append the license to your license.dat file or you can specify the MegaCore function’s license.dat file in the Quartus II software.

Before you set up licensing for the RLDRAM II controller, you must already have the Quartus II software installed on your computer with licensing set up.

Append the License to Your license.dat File

To append the license, follow these steps:

1. Close the following software if it is running on your PC:
   - Quartus II software
   - MAX+PLUS® II software
   - LeonardoSpectrum™ synthesis tool
   - Synplify software
   - ModelSim® simulator

2. Open the RLDRAM II controller license file in a text editor. The file should contain one FEATURE line, spanning 2 lines.

3. Open your Quartus II license.dat file in a text editor.
4. Copy the FEATURE line from the RLDRAM II controller license file and paste it into the Quartus II license file.

[fläche] Do not delete any FEATURE lines from the Quartus II license file.

5. Save the Quartus II license file.

[fläche] When using editors such as Microsoft Word or Notepad, ensure that the file does not have extra extensions appended to it after you save (e.g., license.dat.txt or license.dat.doc). Verify the filename in a DOS box or at a command prompt.

**Specify the License File in the Quartus II Software**

To specify the MegaCore function’s license file, follow these steps:

[fläche] Altera recommends that you give the file a unique name, e.g., <MegaCore name>_license.dat.

1. Run the Quartus II software.

2. Choose License Setup (Tools menu). The Options dialog box opens to the License Setup page.

3. In the License file box, add a semicolon to the end of the existing license path and filename.

4. Type the path and filename of the MegaCore function license file after the semicolon.

[fläche] Do not include any spaces either around the semicolon or in the path/filename.

5. Click OK to save your changes.
3. Specifications

Functional Description

Figure 3–1 shows the RLDRAM II Controller MegaCore® function block diagram.

**Figure 3–1. RLDRAM II Controller Block Diagram**

**Notes (1) & (2)**

**Notes to Figure 3–1:**
1. You can edit the `rldramii` prefix in IP Toolbench.
2. The default signal is `<signal>_0`. When you specify additional address and command busses, both `<signal>_0` and `<signal>_1` are present.
3. Non-DQS mode only.
4. DQS mode only.
The RLDRAM II controller comprises the following two blocks:

- Control logic (encrypted)
- Datapath (clear text)

The control logic performs the following actions:

- Generates initialization sequence using the RLDRAM II initialization values set in IP Toolbench
- Generates write, read, or refresh accesses when requested at the local interface
- Generates datapath control signals that ensure that the write data is output on the memory `rldramii_dq[]` (CIO devices) or `rldramii_d[]` (SIO devices) bus during the correct clock cycles

The datapath performs the following actions:

- Interfaces to common I/O (CIO) or separate I/O (SIO) RLDRAM II devices
- Generates RLDRAM II clocks
- Places RLDRAM II commands onto the memory command bus using one of the following system PLL clocks on either the rising or falling edge:
  - System clock
  - Write clock
  - Dedicated clock
- Places write data onto the `rldramii_dq[]` or `rldramii_d[]` bus during the correct clock cycles
- Captures the read data using dedicated data strobe signal (DQS) delay circuitry during DQS mode or an external capture clock in non-DQS mode

**Control Logic**

The control logic is responsible for controlling transactions at the memory interface. The control logic accepts read, write, and refresh requests and executes them immediately as RLDRAM II transactions.

In addition to reads, writes, and refreshes the control logic is also responsible for controlling initialization of the RLDRAM II devices.

For more information on reads, writes, refreshes, and initialization, see “Interfaces” on page 3–16.
Table 3–1 shows the RLDRAM II control signals generated by the control logic for each operation.

### Table 3–1. Control Signals

<table>
<thead>
<tr>
<th>Operation</th>
<th>Acronym</th>
<th>rldramii_cs_n 0</th>
<th>rldramii_we_n 0</th>
<th>rldramii_ref_n 0</th>
<th>rldramii_a_0[20:0]</th>
<th>rldramii_ba_0[2:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>NOP</td>
<td>High</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
<td>Don’t care</td>
</tr>
<tr>
<td>Mode Register Set</td>
<td>MRS</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>See your RLDRAM data sheet</td>
<td>Don’t care</td>
</tr>
<tr>
<td>Read</td>
<td>READ</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Address</td>
<td>Bank address</td>
</tr>
<tr>
<td>Write</td>
<td>WRITE</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Address</td>
<td>Bank address</td>
</tr>
<tr>
<td>Auto Refresh</td>
<td>AREF</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>Don’t care</td>
<td>Bank address</td>
</tr>
</tbody>
</table>

**Datapath**

Figure 3–2 shows the datapath block diagram.
The datapath performs the following functions:

- Interfaces to CIO or SIO RLDRAM II devices
- Outputs write data to the RLDRAM II interface
- Captures RLDRAM II read data and data valid (QVLD) signals with:
  - In DQS mode, a delayed `rldramii_qk[]` generated by the dedicated DQS delay circuitry
  - In non-DQS mode, an external capture clock

Note to Figure 3–2:
(1) The default signal is `<signal>_0`. When you specify additional address and command busses, both `<signal>_0` and `<signal>_1` are present.
Specifications

- Generates the RLDRAM II clocks
- Generates addresses and commands on:
  - System, dedicated, or write clock
  - Rising or falling edge
- Inserts pipeline registers in address and command and write data path
- Inserts pipeline registers in read data and QVLD path

The datapath provides the interface between the read and write data busses of the datapath interface and the double-clocked, bidirectional data bus of the memory interface. The datapath data busses are twice the width of the memory data bus, because the memory interface transfers data on both the rising and falling edges of the clock.

IP Toolbench generates a clear-text VHDL or Verilog HDL datapath, which matches your custom variation. If you are designing your own controller, Altera recommends that you use this module as your datapath.

Figure 3–3 shows the write control signals timing relationship when writing to the datapath.

![Figure 3–3. Datapath Write Control Signal Timing](image)

Memory Clock Generator

The memory clock generator generates memory clocks. There can be up to four memory clocks and they are generated with an altddio_out megafunction.

Address & Command Output Registers

The address and command output registers can have the following options:

- System, write, or dedicated clock clocking for the output registers.
- Rising or falling edge clocking
**Pipeline Registers**

IP Toolbench can insert pipeline registers into the datapath to help meet timing at higher frequencies. IP Toolbench offers the following pipeline options:

- Insert address and command and write data pipeline registers. The pipeline depth is the same for the write-data path and the address and command path. The write data and address and command pipeline registers are clocked off the system clock.
- Insert read data and QVLD pipeline registers. The pipeline depth is the same for the read-data path and the QVLD path. The read data and QVLD pipeline registers are clocked off the clock that captures the read data—the delayed `rldramii_qk[]` signal in DQS mode; the external capture clock in non-DQS mode.

**DQS Group**

The datapath instantiates one or more DQS groups, which generates write data, `rldramii_dq[]` (CIO devices), or `rldramii_d[]` (SIO devices) and captures read data `rldramii_dq[]` (CIO devices), or `rldramii_q[]` (SIO devices). The IP Toolbench `DQ per DQS` (CIO devices) or `Q per DQS` (SIO devices) parameter determines the DQS group width. For example, if `DQ per DQS` is 9 bits, the `control_wdata[]` and `control_rdata[]` signals are 18-bits wide. To build larger widths, the datapath instantiates multiple DQS group modules to increase the data-bus width in increments of `DQ per DQS` (or `Q per DQS`) bits.

- The datapath generates the DM output, `rldramii_dm[]`, in the DM group module. It generates the DM output in the same way as the write data.
- The datapath captures the QVLD input, `rldramii_qvld[]`, in the QVLD group module. The `rldramii_qvld[]` signal is captured in the same way that the DQS group module captures the read data. In DQS mode, the delayed `rldramii_qk[]` captures `rldramii_qvld[]`; in non-DQS mode, the external clock captures `rldramii_qvld[]`.

Figure 3–4 on page 3–7 shows the Stratix® II series and HardCopy® II devices DQS group block diagram (DQS mode, CIO devices).
Notes to Figure 3–4:
(1) This figure shows the logic for one DQ output only.
(2) All clocks are clk, unless marked otherwise.
(3) Bus width W is dependent on the DQ per DQS parameter.
(4) Invert combout of the I/O element (IOE) for the dq pin before feeding in to inclock of the IOE for the DQ pin.
This inversion is automatic if you use an altdq megafunction for the DQ pins.

Figure 3–5 on page 3–8 shows the Stratix II series and HardCopy II devices DQS group block diagram (DQS mode, SIO devices).
Figure 3-5. DQS Group Block Diagram—DQS Mode, SIO Devices  

Notes (1), (2), & (3)

Notes to Figure 3-4:
1. This figure shows the logic for one Q output and one D input only.
2. All clocks are clk, unless marked otherwise.
3. Bus width W is dependent on the Q per DQS parameter.
4. Invert combout of the I/O element (IOE) for the dqpin before feeding into inclock of the IOE for the Q pin. This inversion is automatic if you use an altdq megafunction for the Q pins.

Datapath Example

Figure 3-6 shows an example datapath. The example RLDRAM II controller and memory configuration has the following parameters:

- DQS mode
- Two 18-bit CIO RLDRAM II devices. Each RLDRAM II device has two rldramii_qk[] data strobes, each associated with 9-bits of data
- 36-bit RLDRAM II interface, which requires a 72-bit datapath interface
Figure 3–6. Example Datapath

Figure 3–6 shows the following points, which are applicable for all interface configurations:

- Each DQS rldramii_dq[] byte group is captured by the delayed version of its associated rldramii_qk[] data strobe:

```plaintext
control_rdata[35:18] control_rdata[71:54]
capture_clk[1]
capture_clk[0]
control_qvld[1] control_qvld[0]
control_rdata[17:0] control_rdata[53:36]
```
Functional Description

- `rldramii_dq[8:0]` is captured by the delayed `rldramii_qk[0]`
- `rldramii_dq[17:9]` is captured by the delayed `rldramii_qk[1]`
- `rldramii_dq[26:18]` is captured by the delayed `rldramii_qk[2]`
- `rldramii_dq[35:27]` is captured by the delayed `rldramii_qk[3]`

- **QVLD** is always captured by the delayed version of `rldramii_qk[0]` for the associated RLDRAM II device. In Figure 3–6 there are four `rldramii_qk[]` signals. Only `rldramii_qk[0]` per RLDRAM II device captures the associated QVLD signal:
  - `rldramii_qvld[0]` is captured by the delayed `rldramii_qk[0]`
  - `rldramii_qvld[1]` is captured by the delayed `rldramii_qk[2]`

- After the capture registers all captured read data is clocked off the delayed `rldramii_qk[]` signal that captures the QVLD signal for a particular RLDRAM II device:
  - All RLDRAM II 0 captured data is clocked off the delayed `rldramii_qk[0]`
  - All RLDRAM II 1 captured data is clocked off the delayed `rldramii_qk[2]`

- Only one `capture_clk[]` per attached RLDRAM II device is output from the datapath:
  - RLDRAM II 0 capture data is associated with `capture_clk[0]`, which is the delayed `rldramii_qk[0]`
  - RLDRAM II 1 capture data is associated with `capture_clk[1]`, which is the delayed `rldramii_qk[2]`

Read Data Capture Clock Association

Figure 3–7 shows the read data and data strobes at the memory interface for the example datapath in Figure 3–6. Figure 3–8 shows how the `capture_clk[]` associates with the captured read data, `control_rdata[]` at the datapath interface.
Figure 3–7. Memory Interface

Figure 3–8. Datapath Interface

Figure 3–8 shows that any read data captured on the rising edge of the delayed rldramii_qk[] signal is located in the lower half-bit locations of control_rdata[]. Any read data captured on the falling edge of the delayed rldramii_qk[] signal is located in the upper half-bit locations...
Device-Level Configuration

of control_rdata[], which means different bit ranges of the control_rdata[] are associated with different capture_clk[] signals.

Figure 3–8 is a specific example but the mapping and clock association applies to any RLDRAM II controller interface and memory configuration.

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 controller issues no read commands at the memory interface.

For more information on OpenCore Plus hardware evaluation, see “OpenCore Plus Evaluation” on page 1–4 and AN 320: OpenCore Plus Evaluation of Megafunctions.

PLL Configuration

This section describes the following topics:

- “PLL Configuration” on page 3–12
- “Example Design” on page 3–14
- “Constraints” on page 3–16

IP Toolbench creates up to two example PLLs in your project directory, which you can parameterize to meet your exact requirements. IP Toolbench generates the example PLLs with an input to output clock ratio of 1:1 and a clock frequency you entered in IP Toolbench. In addition IP Toolbench sets the correct phase outputs on the PLLs’ clocks. You can edit
the PLLs to meet your requirements with the `altpll` MegaWizard® Plug-In. IP Toolbench overwrites your PLLs in your project directory unless you turn off `Update example design system PLL`.

The external clocks are generated using standard I/O pins in double data rate I/O (DDIO) mode (using the `altddio_out` megafunction). This generation matches the way in which the write data is generated and allows better control of the skew between the clock and the data to meet the timing requirements of the RLDRAM II device.

The PLL has the following outputs:

- Output `c0` drives the system clock that clocks most of the controller including the control logic and datapath.
- Output `c1` drives the write clock that lags the system clock.
- Output `c2` optionally drives the address and command clock.
- Output `c3` drives the DQS DLL clock.

The recommended configuration for implementing the RLDRAM II controller in Stratix II series and HardCopy II devices is to use a single enhanced PLL to produce all the required clock signals. No external clock buffer is required as the Altera® device can generate clock signals for the RLDRAM II devices.

Figure 3–9 on page 3–14 shows the recommended PLL configuration.
**Example Design**

IP Toolbench creates an example design that shows you how to instantiate and connect up the RLDRAM II controller to an example driver. The example design is a working system that can be compiled and used for both static timing checks and board tests. It also instantiates an example PLL that generates all the required clocks for the controller. In DQS mode, a DLL is instantiated that controls the DQS capture delay phase. In non-DQS mode, the example design instantiates a feedback PLL. The output of the feedback PLL is a phase-shifted `rldramii_qk[]` data strobe, which captures the read data.

The example driver is a self-checking test generator for the RLDRAM II controller. It uses a state machine to write data patterns to all memory banks. It then reads back the data and checks that the data matches. If any read data fails the comparison, the `pnf_per_byte` output transitions low for one cycle and the `pnfPersist` permanent output transitions low and stays low.

Figure 3–10 shows a testbench and an example design.
Table 3–2 describes the files that are associated with the example design and the testbench.

**Table 3–2. Example Design & Testbench Files**

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;top-level name&gt;_tb.v or .vhd (1)</code></td>
<td>Testbench for the example design.</td>
</tr>
<tr>
<td><code>&lt;top-level name&gt;.vhd or .v (1)</code></td>
<td>Example design.</td>
</tr>
<tr>
<td><code>rldramii_pll_&lt;device name&gt;.vhd or .v</code></td>
<td>Example PLL, which you should configure to match your frequency.</td>
</tr>
<tr>
<td><code>rldramii_fbpll_&lt;device name&gt;.vhd or .v</code></td>
<td>Feedback PLL</td>
</tr>
<tr>
<td><code>&lt;variation name&gt;_example_driver.v or .vhd (2)</code></td>
<td>Example driver.</td>
</tr>
<tr>
<td><code>&lt;variation name&gt;.v or .vhd (2)</code></td>
<td>RLDRAM II controller.</td>
</tr>
</tbody>
</table>

**Notes to Table 3–2:**
(1) `<top-level name>` is the name of the Quartus® II project top-level entity.
(2) `<variation name>` is the variation name.

The testbench instantiates an RLDRAM II model and generates a reference clock for the PLL.

> **Note** Altera does not provide a memory simulation model. You must download one or use your own.
For more details on how to run the simulation script, see “Simulate the Example Design” on page 2–18.

Constraints

The constraints scripts set the following constraints:

- Sets IO standards:
  - 1.5 or 1.8-V HSTL voltage selection
  - Address and command—HSTL Class I
  - Data CIO mode—HSTL Class II
  - Data SIO mode—HSTL Class I
- Sets output capacitance
- Places data pins as per selection in pin placement constraints floor plan. Allows automatic placement for DQS and non-DQS modes
- Places all DM pins
- Sets up correct output enable groups
- Sets rldramii_a_0, rldramii_ba_0, rldramii_cs_n_0, rldramii_ref_n_0 and rldrainii_we_n_0 as fast output registers (see note 1 in Table 3–5).
- Sets rldramii_qk[] non-global signal in DQS capture mode

Interfaces

This section describes the following RLDRAM II commands:

- Initialization
- Writes
- Reads
- Refreshes

Initialization

The control logic initializes the RLDRAM II devices. During initialization the mode register is set and each bank is refreshed in turn. IP Toolbench sets the following RLDRAM II initialization features:

- Termination
- Impedance matching resistor
- DLL enable
- RLDRAM II configuration

Figure 3–11 shows the initialization sequence.
### Figure 3–11. RLDRAM II Initialization Sequence

![RLDRAM II Initialization Sequence Diagram]

The mode register set (MRS) command configures the RLDRAM II devices. In the three-cycle MRS sequence, the first two MRS commands are dummy commands; the final MRS command configures the memory. The RLDRAM II configuration data (CFG) is output on the rldramii_a_0[] bus during all three MRS commands. The following memory parameters are setup during the MRS command cycles:

- RLDRAM II termination
- Impedance matching resistor
- DLL enable/disable
- RLDRAM II configuration

### Writes

When you assert local_write_req, the control logic issues the write transaction immediately at the memory interface. The control logic then requests write data by asserting local_wdata_req, so that the RLDRAM II tWL period is satisfied during write transactions. This functionality means that the write request is decoupled from the write data.

Figure 3–12 shows three write requests at the local and SIO RLDRAM II interface. In this example, the memory burst length is set to eight beats. The RLDRAM II device is setup with a tRC of six-clock cycles (configuration two).
Figure 3–12 shows the transactions at the local interface are separated by the correct number of clock cycles for the target RLDRAM II device configuration. If transaction requests are supplied to the RLDRAM II controller with the incorrect spacing the controller executes these transactions as requested, which can result in incorrect behavior.

Figure 3–13 shows an example of a write following a read at a CIO RLDRAM II interface. In this example, the memory burst length is set to two beats. The RLDRAM II device is setup with a tRC of six-clock cycles (configuration two).

For more information about bus turnaround timing calculations with CIO devices, refer to AN 325: Interfacing RLDRAM II with Stratix II, Stratix & Stratix GX Devices.
Reads

When you assert `local_read_req`, the control logic issues the read transaction immediately at the memory interface.

In DQS mode the read data, `rldramii_dq[]` (CIO devices) or `rldramii_q[]` (SIO devices), and the QVLD signals, `rldramii_qvld[]`, are captured using the delayed `rldramii_qk[]` data strobes that have been phase shifted using the dedicated DQS delay circuitry.

In non-DQS mode the read data, `rldramii_dq[]` or `rldramii_q[]`, and the QVLD signals, `rldramii_qvld[]`, are captured using an external capture clock.
During reads, the local interface indicates that read data is valid by asserting the `local_rdata_valid[]` signal. All captured read data is clocked off the clock that captures the RLDRAM II read data. In DQS mode, this clock is the delayed DQS signal, `capture_clk[]`, sourced from the dedicated DQS delay circuitry. In non-DQS mode this clock is the external capture clock, `non_dqs_capture_clk`.

Figure 3–14 shows an example of a read at an SIO RLDRAM II interface. In this example, the memory burst length is set to eight beats. The RLDRAM II device is setup with a $t_{RC}$ of six-clock cycles (configuration two).

**Figure 3–14. Read Example**

Figure 3–15 shows an example of a read following a write at a CIO RLDRAM II interface. In this example, the memory burst length is set to eight beats. The RLDRAM II device is setup with a $t_{RC}$ of six-clock cycles (configuration two).

For more information about bus turnaround timing calculations with CIO devices, refer to AN 325: Interfacing RLDRAM II with Stratix II, Stratix & Stratix GX Devices.
**Specifications**

**Figure 3–15. Read Following a Write**

You must issue refreshes to the RLDRAM II devices at periodic intervals. When a refresh is required, assert `local_refresh_req` and the RLDRAM II controller issues the refresh command immediately to the requested bank address on `local_bank_addr[]` input. You must correctly insert the refresh request and ensure that the \( t_{RC} \) timing parameter is not violated. You can issue single or ganged refreshes. For ganged refreshes assert `local_refresh_req` for \( X \) clock cycles, where \( X \) is the number of refreshes that you require.

Figure 3–16 shows a single refresh command:
Figure 3–16. Single Refresh Command

Table 3–3 shows the system signals.

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
<td>Input</td>
<td>System clock for the control logic and datapath.</td>
</tr>
<tr>
<td>write_clk</td>
<td>1</td>
<td>Input</td>
<td>Shifted clock that center aligns write data to the memory.</td>
</tr>
<tr>
<td>addr_cmd_clk</td>
<td>1</td>
<td>Input</td>
<td>Address and command output register clock.</td>
</tr>
</tbody>
</table>
### Table 3–3. System Signals (Part 2 of 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dqs_delay_ctrl[]</td>
<td>6</td>
<td>Input</td>
<td>Delay bus for DLL to shift DQS inputs. DQS mode only.</td>
</tr>
<tr>
<td>non_dqs_capture_clk</td>
<td>1</td>
<td>Input</td>
<td>Optional clock that captures read data and clocks read data logic. Non-DQS mode only.</td>
</tr>
<tr>
<td>reset_clk_n</td>
<td>1</td>
<td>Input</td>
<td>Reset input for logic on the system clock domain. The <code>reset_clk_n</code> can be asserted asynchronously but must be deasserted synchronous to the rising edge of the system clock.</td>
</tr>
<tr>
<td>reset_addr_cmd_clk_n</td>
<td>1</td>
<td>Input</td>
<td>Reset input for logic on the address and command clock domain. The <code>reset_addr_cmd_clk_n</code> can be asserted asynchronously but must be deasserted synchronous to the rising edge of the address and command clock.</td>
</tr>
</tbody>
</table>
| reset_read clk_n[]            | DQS mode: the number of RLDRAM II devices attached to the memory interface  
Non-DQS mode: 1 | Input     | Reset input for logic on the capture clock domain. In DQS mode, the capture clock domain is `capture_clk[]`; in non-DQS mode, it is `non_dqs_capture_clk`. In DQS mode, each `reset_read clk_n[]` is associated with the corresponding `capture_clk[]` clock domain. The `reset_read clk_n[]` can be asserted asynchronously but must be deasserted synchronous to the rising edge of the capture clock. |
| capture_clk[]                 | The number of RLDRAM II devices attached to memory interface | Output    | Delayed DQS clock used by capture circuitry to capture RLDRAM II read data. There is one `capture_clk[]` per attached RLDRAM II device. DQS mode only. |
Table 3–4 shows the local interface signals.

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_addr[]</td>
<td>Device dependant</td>
<td>Input</td>
<td>RLDRAM II address. IP Toolbench refers to the memory.dat file and selects the address width appropriate to the device.</td>
</tr>
<tr>
<td>local_bank_addr[]</td>
<td>3</td>
<td>–</td>
<td>RLDRAM II bank address.</td>
</tr>
<tr>
<td>local_dm[]</td>
<td>The number of RLDRAM II devices attached to the memory interface × 2</td>
<td>Input</td>
<td>Optional local data mask (DM). Twice the width of the memory rldramii_dm[] bus. When all high, all writes are masked.</td>
</tr>
<tr>
<td>local_read_req</td>
<td>1</td>
<td>Input</td>
<td>Read request signal.</td>
</tr>
<tr>
<td>local_refresh_req</td>
<td>1</td>
<td>Input</td>
<td>User controlled refresh request. This allows complete control over when refreshes are issued to the memory. The refresh is issued to the bank address on local_bank_addr[].</td>
</tr>
<tr>
<td>local_wdata[]</td>
<td>Data-bus width × 2</td>
<td>Input</td>
<td>Write data bus. The local interface must request local_wdata[] over multiple clock cycles to construct the write data for any requested write bursts. If the memory burst length is set to two beats, the write data is requested in a single clock cycle at the local interface.</td>
</tr>
<tr>
<td>local_write_req</td>
<td>1</td>
<td>Input</td>
<td>Write request signal.</td>
</tr>
<tr>
<td>local_init_done</td>
<td>1</td>
<td>Output</td>
<td>Memory initialization complete signal which is asserted when the controller has completed its initialization of the memory. Reads and writes should not be requested until local_init_done is asserted.</td>
</tr>
<tr>
<td>local_rdata[]</td>
<td>Data-bus width × 2</td>
<td>Output</td>
<td>Read data bus. The controller returns local_rdata[] over multiple clock cycles for any requested read transactions. If the memory burst length is set to two beats, the read data is returned in a single clock cycle at the local interface.</td>
</tr>
</tbody>
</table>

Table 3–4. Local Interface Signals  (Part 1 of 2)
Table 3–4. Local Interface Signals  (Part 2 of 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_rdata_valid[]</td>
<td>The number of RLDRAM II devices attached to memory interface</td>
<td>Output</td>
<td>Read data valid signal, which indicates that valid data is present on the read data bus. The <code>local_rdata_valid[]</code> signal is aligned with the local read data, <code>local_rdata[]</code>. There is only one <code>local_rdata_valid[]</code> per attached RLDRAM II device.</td>
</tr>
<tr>
<td>local_wdata_req</td>
<td>1</td>
<td>Output</td>
<td>Write data request signal. When the local interface asserts <code>local_wdata_req</code>, all the write data for the burst should be available in contiguous clock cycles.</td>
</tr>
</tbody>
</table>

Table 3–5 shows the memory interface signals.

Table 3–5. Memory Interface Signals  (Part 1 of 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rldramii_dq[]</td>
<td>Data-bus width</td>
<td>Bidirectional</td>
<td>Memory data bus. CIO devices only.</td>
</tr>
<tr>
<td>rldramii_qk[]</td>
<td>1 to 9</td>
<td>Bidirectional</td>
<td>In DQS mode, the memory data strobe signal that captures read data into the Altera device; in non-DQS mode, the RLDRAM II controller does not use <code>rldramii_qk[]</code>.</td>
</tr>
<tr>
<td>rldramii_q[]</td>
<td>Data-bus width</td>
<td>Input</td>
<td>Memory read data bus. SIO devices only.</td>
</tr>
<tr>
<td>rldramii_qvld[]</td>
<td></td>
<td>Input</td>
<td>Read data valid flag.</td>
</tr>
<tr>
<td>rldramii_a_0[] rldramii_a_1[] (1)</td>
<td>local_addr[]</td>
<td>Output</td>
<td>Memory address signals.</td>
</tr>
<tr>
<td>rldramii_ba_0[] rldramii_ba_1[] (1)</td>
<td>3</td>
<td>Output</td>
<td>Memory bank address signals.</td>
</tr>
<tr>
<td>rldramii_clk[], rldramii_clk_n[]</td>
<td>1 to 4</td>
<td>Output</td>
<td>Memory command output clock.</td>
</tr>
<tr>
<td>rldramii_cs_n_0 rldramii_cs_n_1 (1)</td>
<td>1</td>
<td>Output</td>
<td>Memory chip select signal.</td>
</tr>
<tr>
<td>rldramii_d[]</td>
<td>Data-bus width</td>
<td>Output</td>
<td>Memory write data bus. SIO devices only.</td>
</tr>
</tbody>
</table>
Table 3–5. Memory Interface Signals (Part 2 of 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rldramii_dm[]</td>
<td></td>
<td>Output</td>
<td>Memory DM (optional).</td>
</tr>
<tr>
<td>rldramii_ref_n_0</td>
<td>1</td>
<td>Output</td>
<td>Memory refresh request signal.</td>
</tr>
<tr>
<td>rldramii_ref_n_1</td>
<td>1</td>
<td>Output</td>
<td>Memory refresh request signal.</td>
</tr>
<tr>
<td>rldramii_we_n_0</td>
<td>1</td>
<td>Output</td>
<td>Memory write enable signal.</td>
</tr>
<tr>
<td>rldramii_we_n_1</td>
<td>1</td>
<td>Output</td>
<td>Memory write enable signal.</td>
</tr>
</tbody>
</table>

Note to Table 3–5:
(1) The default signal is <signal>_0. When you specify additional address and command busses, both <signal>_0 and <signal>_1 are present.

Table 3–6 shows the datapath interface signals.

Table 3–6. Datapath Interface Signals (Part 1 of 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Width (Bits)</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>control_a[]</td>
<td>local_addr[]</td>
<td>Input</td>
<td>Address bits.</td>
</tr>
<tr>
<td>control_ba[]</td>
<td>3</td>
<td>Input</td>
<td>Bank address bits.</td>
</tr>
<tr>
<td>control_cs_n</td>
<td>1</td>
<td>Input</td>
<td>Chip select signal.</td>
</tr>
<tr>
<td>control_dm[]</td>
<td>The number of RLDRAM II devices attached to the memory interface × 2</td>
<td>Input</td>
<td>The DM bus, which has valid data in the same clock cycles that control_wdata_valid is asserted.</td>
</tr>
<tr>
<td>control_doing_wr</td>
<td>1</td>
<td>Input</td>
<td>Control_doing_wr is asserted when the controller is writing to the RLDRAM II devices and controls the output enables on rldramii_dq[] or rldramii_d[].</td>
</tr>
<tr>
<td>control_ref_n</td>
<td>1</td>
<td>Input</td>
<td>Refresh signal.</td>
</tr>
<tr>
<td>control_wdata[]</td>
<td>Data-bus width × 2</td>
<td>Input</td>
<td>The write data bus, which has valid data in the same clock cycles that control_wdata_valid is asserted.</td>
</tr>
<tr>
<td>control_wdata_valid</td>
<td>1</td>
<td>Input</td>
<td>Enables the write data bus and DM enable registers so that they are only updated when valid data and enables are available.</td>
</tr>
</tbody>
</table>
Parameters

The parameters can only be set in IP Toolbench (see “Step 1: Parameterize” on page 2–7).

Memory

Table 3–7 shows the memory type parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLDRAM II device</td>
<td>Part number</td>
<td>–</td>
<td>A part number for a particular memory device. Choosing an entry sets many of the parameters in the wizard to the correct value for the specified part. You can add your own devices to this list by editing the memory_types.dat file in the \constraints directory.</td>
</tr>
<tr>
<td>Clock speed</td>
<td>100 to 400 MHz</td>
<td></td>
<td>The memory controller clock frequency. The constraints script and the datapath use this clock speed. It must be set to the value that you intend to use. The first time you use IP Toolbench or if you turn on Update example design system PLL, it uses this value for the IP Toolbench-generated PLL's input and output clocks.</td>
</tr>
<tr>
<td>Interface voltage</td>
<td>1.5 or 1.8 V</td>
<td></td>
<td>The RLDRAM II interface voltage.</td>
</tr>
<tr>
<td>DQ per DQS</td>
<td>8, 9, 16, 18 Bits</td>
<td></td>
<td>Number of DQ bits per DQS input pin. CIO devices only.</td>
</tr>
<tr>
<td>Q per DQS</td>
<td>8, 9, 16, 18 Bits</td>
<td></td>
<td>Number of Q bits per DQS input pin. SIO devices only.</td>
</tr>
</tbody>
</table>
| Data-bus width          | Device dependent | Bits | The width of the memory interface. For more information about supported interface data widths, refer to AN 325: Interfacing RLDRAM II with Stratix II, Stratix & Stratix GX Devices.
Table 3–8 shows the memory initialization options.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory configuration</td>
<td>1, 2, or 3</td>
<td>Refer to your RLDRAM II data sheet.</td>
</tr>
<tr>
<td>Burst length</td>
<td>2, 4, or 8</td>
<td>Number of beats in the burst at the memory interface. The number of beats at the local interface is half this value.</td>
</tr>
<tr>
<td>Manually enter initialization clock cycles</td>
<td>On or off</td>
<td>The wizard takes the number of initialization clock cycles from the memory.dat file in the constraints directory. The number is calculated from the initialization entry time and the clock speed. You can manually enter a number for the initialization clock cycles if you turn on Manually enter initialization clock cycles.</td>
</tr>
<tr>
<td>Number of initialization clock cycles</td>
<td>16 to 80,000</td>
<td></td>
</tr>
<tr>
<td>Enable on-die termination</td>
<td>On or off</td>
<td>Refer to your RLDRAM II data sheet.</td>
</tr>
<tr>
<td>Enable external impedance matching</td>
<td>On or off</td>
<td>Refer to your RLDRAM II data sheet.</td>
</tr>
<tr>
<td>Enable memory device DLL</td>
<td>On or off</td>
<td>Refer to your RLDRAM II data sheet.</td>
</tr>
</tbody>
</table>

Table 3–9 shows the memory interface parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of clock pairs from FPGA to memory</td>
<td>1, 2, 3, or 4</td>
<td>–</td>
<td>The number of RLDRAM II clock output pairs generated in the datapath.</td>
</tr>
<tr>
<td>Number of address and command busses from FPGA to memory for multiple devices</td>
<td>1 or 2</td>
<td>–</td>
<td>Depends on the number of devices. If you connect only one device there can be only one address and command bus. (1)</td>
</tr>
<tr>
<td>Generate DM pins</td>
<td>On or off</td>
<td>–</td>
<td>Adds DM pins and logic to the design.</td>
</tr>
</tbody>
</table>

Note to Table 3–9:
(1) The default signal is <signal>_0. When you specify additional address and command busses, both <signal>_0 and <signal>_1 are present.
Specifications

Timing

Table 3–10 shows the pipeline options.

### Table 3–10. Pipeline Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of address and command and write data pipeline registers</td>
<td>0, 1, 2 or 3</td>
<td>When you choose 1, 2, or 3 the wizard inserts 1, 2, or 3 pipeline registers between the memory controller and the command and address output registers and the write data output registers. These registers may help to achieve the required performance at higher frequencies.</td>
</tr>
<tr>
<td>Number of read data pipeline registers</td>
<td>0, 1, 2 or 3</td>
<td>When you choose 1, 2, or 3 the wizard inserts 1, 2, or 3 pipeline registers between the read capture registers and the memory controller. These registers may help to achieve the required performance at higher frequencies.</td>
</tr>
</tbody>
</table>

Table 3–11 shows the clocking modes.

### Table 3–11. Clocking Modes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address and command clock</td>
<td>System, write, or dedicated</td>
<td>The clock for the address and command output registers. For system_clk choose System; for write_clk, choose Write, and for a separate clock, choose Dedicated. If you choose Dedicated for the clock, ensure the clock phase allows the Quartus II software to meet the setup time on the address and command output registers.</td>
</tr>
<tr>
<td>Address and command clock edge</td>
<td>Falling or rising</td>
<td>The clock edge on which the addresses and commands are output.</td>
</tr>
<tr>
<td>Dedicated address and command clock PLL phase offset</td>
<td>± 180°</td>
<td>Sets the dedicated address and command clock PLL phase for better timing.</td>
</tr>
<tr>
<td>Enable DQS mode</td>
<td>On or off</td>
<td>Turn on for DQS mode; otherwise the controller is in non-DQS mode (Stratix II and Stratix II GX devices only). HardCopy II devices allow DQS mode only.</td>
</tr>
<tr>
<td>Use migratable byte groups</td>
<td>On or off</td>
<td>When turned on, you can migrate the design to a migration device. When turned off the wizard allows much greater flexibility in the placement of byte groups. You can only turn on this option when Enable DQS mode is turned off.</td>
</tr>
<tr>
<td>Feedback PLL phase offset</td>
<td>± 180°</td>
<td>Sets the feedback clock PLL phase for read capture (non-DQS mode only).</td>
</tr>
</tbody>
</table>
Table 3–12 shows the pin loading parameters.

**Table 3–12. Pin Loading Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range (pF)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin loading on FPGA DQ/DQS</td>
<td>0 to 100</td>
<td>Enter the pin loading to match your board and memory devices.</td>
</tr>
<tr>
<td>address and command pins</td>
<td>0 to 100</td>
<td>Enter the pin loading to match your board and memory devices.</td>
</tr>
<tr>
<td>clock pins</td>
<td>0 to 100</td>
<td>Enter the pin loading to match your board and memory devices.</td>
</tr>
</tbody>
</table>

**Project Settings**

Table 3–13 shows the example design settings.

**Table 3–13. Example Design Settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically apply RLDRAM II controller-specific constraints to the Quartus II project</td>
<td>When this option is turned on, the next time you compile, the Quartus II software automatically runs the add constraints script. Turn off this option if you do not want the script to run automatically.</td>
</tr>
<tr>
<td>Update the example design file that instantiates the RLDRAM II controller variation</td>
<td>When this option is turned on, IP Toolbench parses and updates the example design file. It only updates sections that are between the following markers: &lt;&lt;START MEGAWIZARD INSERT &lt;tagname&gt; &lt;&lt;END MEGAWIZARD INSERT &lt;tagname&gt; If you edit the example design file, ensure that your changes are outside of the markers or remove the markers. Once you remove the markers, you must keep the file updated, because IP Toolbench can no longer update the file.</td>
</tr>
<tr>
<td>Update example design system PLL</td>
<td>When this option is turned on, IP Toolbench automatically overwrites the PLL. Turn off this option, if you do not want the wizard to overwrite the PLL.</td>
</tr>
</tbody>
</table>
Table 3–14 shows the variation path parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable hierarchy control</td>
<td>The constraints script analyzes your design, to automatically extract the hierarchy to your variation. To prevent the constraints script analyzing your design, turn on Enable Hierarchy Control, and enter the correct hierarchy path to your datapath.</td>
</tr>
<tr>
<td>Hierarchy path to RLDRAM II controller datapath</td>
<td>The hierarchy path is the path to your RLDRAM II controller datapath, minus the top-level name. The hierarchy entered in the wizard must match your design, because the constraints scripts rely on this path for correct operation.</td>
</tr>
</tbody>
</table>

Table 3–15 shows the device pin prefixes parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix all RLDRAM II pins on the device with</td>
<td>This string prefixes the pin names for the FPGA pins that are connected to the RLDRAM II controller.</td>
</tr>
</tbody>
</table>

MegaCore Verification

MegaCore verification involves simulation testing and hardware testing.

Simulation Environment

Altera has carried out extensive functional tests using industry-standard models to ensure the functionality of the RLDRAM II controller. In addition, Altera has carried out a wide variety of gate-level tests on the RLDRAM II controller to verify the post-compilation functionality of the controller.

Hardware Testing

Table 3–16 shows the Altera development board on which Altera hardware tested the RLDRAM II controller.

<table>
<thead>
<tr>
<th>Development Board</th>
<th>Altera Device</th>
<th>Memory Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix II Memory Demonstration Board 1</td>
<td>EP2S60F1020C3</td>
<td>Micron 18-bit CIO and SIO RLDRAM II devices</td>
</tr>
</tbody>
</table>